Zero-Knowledge Proof for Norinori

A Thesis
Presented to
The Established Interdisciplinary Committee for Mathematics and Computer Science
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Roscoe Elings-Haynie

May 2023

Approved for the Division
(Mathematics and Computer Science)

Chanathip Namprempre	Marcus Robinson

Acknowledgements

Meaw - Thank you for your feedback through this process. Thank you for the countless resources and instruction you gave me in learning about my topic. Finally, you truly made me think about how I can move forward in this field. Thank you.

Marcus - Thank you for all your feedback and edits. You have made me a significantly better academic writer, both in showing me how to keep others engaged on complex topics and by showing the precision of language and notation necessary for those topics. Thank you.

Mom and Dad - No words could sum up how much I owe you both. Thank you for my education, thank you for all the support you have given me, and thank you for making me into a person who is curious about the world. I love you both so much.

Altana - It's kind of funny that I followed you here, academics was always a realm where we were very different, but I could not be happier that I did. Knowing that you had been through the same process made me feel like I could make it through as well. Love you, Atlanta.

Table of Contents

;

List of Figures

;

Abstract

In this thesis, we establish a formal notion of a puzzle. We discuss categorizing the varying degrees of difficulty of problems. We also explore the idea of zero-knowledge proofs. A zero-knowledge proof is a proof structure where the solution to the relevant problem is not revealed. All of this is done through the puzzle Norinori. We will prove that Norinori is NP-complete and show a zero-knowledge protocol for Norinori. We will also prove that this protocol meets all conditions of zero-knowledge proofs.

Introduction

The central focus of this thesis is zero-knowledge proofs, this is the idea of convincing someone you know something without revealing the information that you know. For example, imagine you know the lottery numbers and you prove to all of your friends that you know the lottery numbers. You can't just tell them the numbers because then they will buy tickets and take a portion of your winnings. You can instead write the numbers down and lock the note in a safe until after the drawing. This was there is proof that you knew the numbers in advance. This is an example of a tool used in zero-knowledge proofs.

In the real world, zero-knowledge proofs can be used to strike a balance between accuracy of information and privacy. For instance, they are used in blockchain to ensure private transfers of information, they could be used in electronic voting systems so that the privacy of each individual vote can be preserved while maintaining confidence that each vote is counted correctly.

This thesis explores zero-knowledge proofs through the pencil-and-paper game Norinori. Norinori is a rectangular grid that is divided into subsections called rooms, the goal of the game is to mark two squares in each room as black in a way that each black square is next to exactly one other black square. It is non-trivial to find a solution for a Norinori board, making it an excellent case study in zero-knowledge proofs. The game is very similar to other popular games such as Sudoku and LITs. An example is shown in Figure 1.

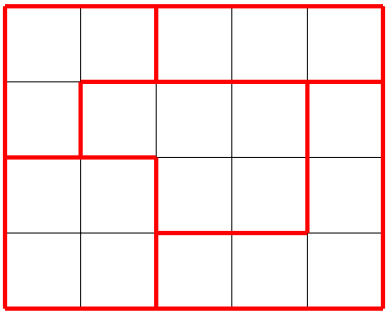


Figure 1: An example Norinori board.

Problems can be categorized into several different classes that represent the difficulty in solving said problem. We will explore how Norinori compares to other

problems in its class. The class containing Norinori is called NP. We will show an algorithm that equates solving other problems in NP to solving boards within Norinori.

The first chapter will define the necessary machinery, including zero-knowledge proofs, commitment schemes, languages, etc... The second will focus on showing that Norinori is NP-complete, meaning it is similar in difficulty to other NP problems. The third will demonstrate a zero-knowledge protocol for Norinori and prove that the protocol meets the conditions of zero-knowledge.

The solution to Figure 1 is shown in Figure 2.

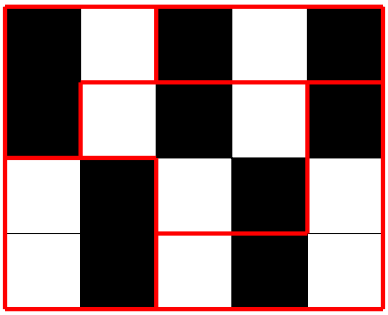


Figure 2: A solution to Figure 1.

Chapter 1

Background

In order to look at zero-knowledge proofs, we need to establish what a zero-knowledge proof is. We will also define some of the underlying machinery and a tool that is commonly used in zero-knowledge proofs.

1.1 Language

Imagine a jigsaw puzzle, the box contains a set of pieces and a photo that is meant to show their final orientation. If this puzzle was to be solved by a computer, there would need to be a method of representation that the computer could interact with, thus not a physical one. The initial building block in representation is called an alphabet.

Definition 1.1.1. An *alphabet* is a non-empty finite set. The members of an alphabet are referred to as *symbols*. An alphabet is commonly represented by Σ.

Back to thinking about the puzzle, only having two symbols to represent the puzzle pieces and the final solution is not very helpful. Sequences of symbols can be connected to hold different meanings, these are called strings.

Definition 1.1.2. A *string* is a finite sequence of symbols.

For example consider the alphabet Σ, then 01, 111, and 0 are strings over Σ. The string of length 0 is referred to as the empty string and is denoted ε. With strings containing 0 and 1 the puzzle above can be represented. The representation of the pieces and the final solution are called an *instance*, and the final orientation that matches the image is called a *witness*.

Next, we need a way of grouping together pieces and images so that each set of pieces can be organized to form their final image, this grouping is called a language.

Definition 1.1.3. A *language* is a set of strings.

An example of a language would be the set of puzzle pieces that can be organized to form a specific image. Another example would be the set of strings that contain a

0.
There is a difference in the resources used to determine if a string is an instance of one of these languages, problems that take a similar amount of resources to compute can be categorized together.

1.2 NP

Determine if a string is in a language is referred to as *deciding* a language. The difficulty in deciding a language is often called the harness of a language. Complexity classes categorize the hardness of languages. Many exist but the only one relevant to our work is the class Non-deterministic Polynomial Time (NP). The description of NP relies on another class called P, for Polynomial time. This means that, given a language $L \in \mathrm{P}$ and a string s, you can decide if $s \in L$ using a polynomial time algorithm. Polynomial time means any algorithm that has a time complexity in $O(n^k)$ where n is the length of the input and k is some constant.

Definition 1.2.1. P is the class of languages that are decidable in Polynomial time by a Turing Machine [Sip12].

Consider the language that contains all strings that have a 0, this is decidable in polynomial time by checking all symbols in the string. Non-deterministic Polynomial time NP is the class of languages where a witness can be verified in polynomial time.

Definition 1.2.2. Let L be a language. We say that $L \in \mathrm{NP}$ if and only if there exists a polynomial time verifier algorithm

$$\mathsf{VF} : \Sigma^* \times \Sigma^* \mapsto \{0, 1\}$$

such that the following conditions are met:

1. For all $x \in L$, there exists w such that $\mathsf{VF}(x, w) = 1$.
2. For all $x \notin L$, there exists no w such that $\mathsf{VF}(x, w) = 1$.

Consider the language Quadratic Residuosity.

Definition 1.2.3. *Quadratic Residuosity* (QR) is the language of pairs (N, W), $N \in \mathbb{Z}, W \in \mathbb{Z}_N^*$ where there exists some $w \in \mathbb{Z}_N^*$ such that $w^2 \equiv W \mod N$.

There is a function that can verify an instance of QR in Polynomial time, thus QR $\in$ NP.

Theorem 1.2.4. The language QR $\in$ NP.

Proof. Let $N \in \mathbb{Z}, W \in \mathbb{Z}_N^*$. Define a verifier VF for QR given a pair (N, W) and a certificate $w \in \mathbb{Z}_N^*$ as follows,

$\mathsf{VF}((N, W), w)$:
if $w^2 = W \mod N$ return 1
else return 0

Since multiplication and equality in the multiplicative group $\mathbb{Z}_N^*$ are both computable in polynomial time, V is a polynomial time algorithm. If $(N, W) \in \text{QR}$, then by Definition 1.2.3, there exists some $w \in \mathbb{Z}_N^*$ such that $w^2 \equiv W \mod N$. Thus, $V((N, W), w) = 1$. If $(N, W) \notin \text{QR}$, then W does not have a square root in mod N, and there exists no $w \in \mathbb{Z}_N^*$ for which $V((N, W), w) = 1$. Thus, $\text{QR} \in \text{NP}$. □

1.3 NP-Completeness

Completeness is used to compare the hardness of different languages within a specific class. Specifically, NP-completeness means that there is some Polynomial time algorithm that when given a string in one NP language, maps to a string in another NP language, and when given a string not in the first language maps to a string not in the second language.

Definition 1.3.1. A language A is *polynomial time reducible* to a language B if there exists a polynomial time function $f : \Sigma^* \mapsto \Sigma^*$ such that for all $x \in \Sigma^*$ it holds that $x \in A$ if and only if $f(x) \in B$. The function f is called a polynomial time reduction [Sip12].

If there is a polynomial time reduction between two languages, it follows that deciding if a string is in one language is at most as hard as deciding if a string is in the other language. While reduction exists in multiple complexity classes, the following only looks at reductions between languages in NP.

Definition 1.3.2. A language A is *NP-hard* if for all languages $B \in \text{NP}$ there exists a polynomial time reduction from B to A [Sip12].

NP-hardness if necessary for defining NP-completeness.

Definition 1.3.3. A language L is *NP-complete* if $L \in \text{NP}$ and L is NP-hard [Sip12].

The easiest way to show that a language is known to be NP-complete is to show that it is reducible to all other languages in NP. A standard language for reduction is the language SAT and 3-SAT.

Definition 1.3.4. The language SAT contains the set of satisfiable boolean formulas in which boolean variables are connected using the operators and, or, and not. A formula is satisfiable if it evaluates to true with some truth value assignment.

For example the formula, $x \wedge y$ is satisfiable with the assignment $x = 1, y = 1$. Thus $x \wedge y \in \text{SAT}$. The Cook-Levin theorem proved that SAT is NP-complete [Sip12]. Michael Biro and Christiane Schmidt used a variant of SAT to prove that Norinori is NP-complete [BS17a].

1.4 Zero-Knowledge Proofs

An interactive proof is a procedure that allows one party called a prover to convince another party called a verifier that a specific string is in a language. The simplest

method would be for the prover to send a witness to the verifier. A zero-knowledge proof is an interactive proof with the added condition that the witness cannot be discerned from the procedure.

1.4.1 Interactive Proofs

Interactive proofs are a way for one party to convince another party of information. An interactive proof for an NP-language L consists of two interactive functions, a prover P and an interactive verifier IV. An interactive function is a function that can send to another function and receive a response. The response history is then used as additional input to the function, you can think of it as if the functions are mutually recursive. You can think of the message history as an additional input.

Definition 1.4.1. A *prover*

$$P : \Sigma^* \times \Sigma^* \times \Sigma^* \mapsto \Sigma^*,$$

takes as input two strings x and w as well as a message history that contains its previous interaction with a verifier and returns the next message to the verifier. A prover P may have unlimited computational capacity and can choose values randomly [Sip12].

The prover intends to convince the verifier that a string $x \in L$. The interactive verifier attempts to verify whether $x \in L$.

Definition 1.4.2. An *interactive verifier*

$$\mathsf{IV} : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$$

takes as input a string x and a message history containing its previous interactions with a prover. It returns either the next message to the prover or a bit. The verifier IV is limited to polynomial time computation but can choose values randomly [Sip12].

A protocol is the combination of a prover and an interactive verifier.

Definition 1.4.3. Let P be a prover, and IV be an interactive verifier. A *protocol* is the pair (P, IV). Let $x, w \in \Sigma^*$. A protocol being run on inputs x and w is denoted $P(x, w) \leftrightarrow \mathsf{IV}(x)$. The value $P(x, w) \leftrightarrow \mathsf{IV}(x)$ is equal to the output of the verifier after the interaction. The information sent between P and IV during an interaction is called a transcript.

When describing protocols, it is common to ignore the input to both functions that represent the message history and assume that they remember previous interaction. There is also some notation commonly used in protocols that will be useful. Let A be a set then $a \twoheadleftarrow A$ means that an element a is chosen uniformly at random from A. The notation $a \leftarrow b$ means that the variable a is assigned to the value of b.

Example 1.4.4. The following protocol is an interactive proof for the language QR (Definition 1.2.3) where the public input is the instance represented by the pair $(N, W), N \in \mathbb{Z}, W \in \mathbb{Z}_N^*$ and the auxiliary input for the prover is the witness $w \in \mathbb{Z}_N^*$. First, the prover P selects $y \twoheadleftarrow \mathbb{Z}_N^*$ uniformly at random then computes $Y \leftarrow y^2 \mod N$. Then P then sends Y to the verifier IV, afterwards IV selects a bit $c \twoheadleftarrow \{0,1\}$ at random and sends it back to P. Next, P computes $z \leftarrow yw^c \mod N$ and sends z to IV. Finally, IV returns 1 if $z \in \mathbb{Z}_N^*$ and $z^2 = YW^c \mod N$, otherwise IV returns 0. This is illustrated in Figure 1.1.

$\underline{P((N,W),w)}$		$\underline{\mathsf{IV}((N,W))}$
$y \twoheadleftarrow \mathbb{Z}_N^*$		
$Y \leftarrow y^2$	$\xrightarrow{Y}$	$c \twoheadleftarrow \{0,1\}$
$z \leftarrow yw^c$	$\xleftarrow{c}$	
	$\xrightarrow{z}$	return 1 if $z \in \mathbb{Z}_N^* \wedge$ $z^2 = YW^c$

Figure 1.1: Zero-Knowledge Protocol for QR

Through this process, the prover is able to convince the verifier that it knows w such that $w^2 = W \mod N$ and the verifier knows with some certainty that the prover is behaving honestly. These are the conditions that make and interactive proof effective, they are called completeness and soundness.

1.4.2 Completeness and Soundness

Completeness and soundness are the two components necessary for an interactive proof. A protocol being complete means that the verifier is willing to accept a correct witness. A protocol being sound means that the verifier will reject an incorrect witness some of the time. Both definitions rely on a witness relation.

Definition 1.4.5. Let $R(x, y)$ be a function $R : \Sigma^* \times \Sigma^* \mapsto \{0,1\}$ called a *witness relation*. Let L_R be the language $\{x \mid \exists y : R(x,y) = 1\}$. Let $R(x)$ denote the set $\{y \mid R(x,y) = 1\}$ [GMR89].

For an interaction with the language L and the string x, completeness means that the verifier will accept if $x \in L$. Soundness means that if $x \notin L$ the verifier will accept some percentage of the time. The verifier only needs to reject some probability of the time because the process can be repeated. If it rejects with probability $\frac{1}{2}$ and the process is iterated n times, then the interaction will reject with probability $\frac{1}{2^n}$.

Definition 1.4.6. Let R be a relation as in Definition 1.4.5. Let $L_R \in$ NP be a language. We say that L_R has an *interactive proof of language membership* if and only if there exists an interactive function IV such that

1. (*Completeness*) There exists an interactive function P such that for all $x \in L_R$, there exists $w \in R(x)$ such that the probability that the interaction between $P(x, w)$ and $\mathsf{IV}(x)$ returns 1 is equal to 1 [GMR89].

2. (*Soundness*) For all interactive functions P', for all $x \notin L_R, w \in \Sigma^*$, the probability that the interaction between $P'(x, w)$ and $\mathsf{IV}(x)$ returns 1 is negligible [GMR89].

A probability δ is called *negligible* if for all $k \in \mathbb{R}_{>0}$ it holds that $\lim_{n\to\infty} \delta^n n^k = 0$ [BS17b].

Considering this definition, we take a second look at the protocol in Example 1.4.4.

Theorem 1.4.7. The interactive function IV defined in Example 1.4.4 for QR is complete.

Proof. Let R be a relation for QR, the verifier VF described in the proof of Theorem 1.2.4 is an example of one such relation. Let P, IV be the prover and verifier defined in Example 1.4.4. For all $(N, W) \in$ QR, for all $w \in R((N, W))$, we will show that the interaction between $P(x, w)$ and $\mathsf{IV}(x)$ returns 1. P computes $z \equiv yw^c, Y \equiv y^2$, then it follows that

$$\begin{aligned} z^2 &\equiv (yw^c)^2 \mod N \\ &\equiv y^2 w^{2c} \mod N && \text{(By commutativity of multiplication in } \mathbb{Z}_N^*) \\ &\equiv YW^c \mod N && \text{(By the definitions of } Y \text{ and } W). \end{aligned}$$

Since $z^2 = YW^c$ and by definition $z \in \mathbb{Z}_N^*$, the conditions for IV to return 1 are met. Thus, $P(x, w) \leftrightarrow \mathsf{IV}(x) = 1$ with probability 1, and the verifier defined in Example 1.4.4 is complete over QR. □

Theorem 1.4.8. The interactive function IV defined in Example 1.4.4 for QR is sound.

Proof. Let IV be the verifier defined in Example 1.4.4. For all provers P', for all $(N, W) \notin$ QR, for all $w \in \Sigma^*$, we will show that the interaction between $P'(x, w)$ and $\mathsf{IV}(x)$ returns 1 with negligible probability. From number theory, we know that if some ab is a perfect square then either a and b are perfect squares or neither a nor b are perfect squares. Let $Pr[\cdot]$ be the probability of an event. Let p be the probability that Y is a square in mod N, then the probability that $P'(x, w) \leftrightarrow \mathsf{IV}(x) = 1$ can be split into two cases, one where $c = 0$ and one where $c = 1$. Thus,

$$\begin{aligned} Pr[P'(x, w) \leftrightarrow \mathsf{IV}(x) = 1] = \\ = & Pr[P'(x, w) \leftrightarrow \mathsf{IV}(x) = 1 \mid c = 0] \cdot Pr[c = 0] + \\ & Pr[P'(x, w) \leftrightarrow \mathsf{IV}(x) = 1 \mid c = 1] \cdot Pr[c = 1]. \end{aligned}$$

Since $(N, W) \notin QR$ it follows that W is not a square in mod N. By the number theory result from earlier, it follows that $Pr[P'(x, w) \leftrightarrow V(x) = 1 \mid c = 0] = p$ since

YW^0 is a perfect square if Y is a perfect square. Furthermore, $Pr[P'(x,w) \leftrightarrow V(x) = 1 \mid c = 1] = 1 - p$. Given that $Pr[c = 0] = \frac{1}{2}$ and $Pr[c = 1] = \frac{1}{2}$, it follows that

$$\Pr[P'(x,w) \leftrightarrow V(x) = 1] = p \cdot \frac{1}{2} + (1-p) \cdot \frac{1}{2} = \frac{1}{2}.$$

Now we will show that for n iterations of the protocol, $\frac{1}{2}^n$ is negligible as in Definition 1.4.6. Let $k \in \mathbb{R}_{>0}$. Consider

$$\lim_{n\to\infty} \left(\frac{1}{2}\right)^n n^k = \lim_{n\to\infty} \frac{n^k}{2^n}.$$

Now, applying L'Hopital's Rule, we obtain

$$\begin{aligned} \lim_{n\to\infty} \frac{n^k}{2^n} &= \lim_{n\to\infty} \frac{\frac{d^c}{dn^k}(n^k)}{\frac{d^k}{dn^k}(2^n)} \\ &= \lim_{n\to\infty} \frac{k!}{\ln^c(2)2^n} \\ &= 0. \end{aligned}$$

Thus, $\lim_{n\to\infty} \frac{1}{2}^n n^k = 0$. Since $\frac{1}{2}^n$ is negligible, it follows that the verifier in Example 1.4.4 is sound. □

This confirms that the above protocol is an interactive proof.

1.4.3 Zero-Knowledge

Zero-knowledge means that the verifier learns no new information through their interaction with the prover. In order to prove this we need some way to refer to the messages sent between a prover and an interactive verifier.

Definition 1.4.9. Let P be a prover, IV be an interactive verifier, and let $x, w \in \Sigma^*$. A *transcript* is the history of messages sent during the interaction between $P(x,w)$ and $\mathsf{IV}(x)$, this is referred to as $\text{Conv}^{P,w}_{\mathsf{IV},x}$.

Next, there needs to be a piece of machinery called a simulator, functionally a simulator just takes a string and outputs another string.

Definition 1.4.10. A *simulator* S is a function $S : \sigma^* \mapsto \Sigma^*$.

In order to show something is zero-knowledge, we need to show that no information was learned. To do this we create a simulator that takes a string and creates an output that is identical to the transcript between a prover and a verifier on the same string.

Definition 1.4.11. Let R be a relation, let $L_R \in \text{NP}$ be a language, and let P be an interactive function. For all interactive functions IV', the interaction between P and IV' is *zero-knowledge* if and only if there exists a function S called a simulator such that for all $x \in \Sigma^*$, for all $w \in R(x)$, $S(x)$ is perfectly indistinguishable from $\text{Conv}^{P,w}_{\mathsf{IV}',x}$ [GMR89].

The term *perfectly indistinguishable* means that any string has the same probability of appearing in $S(x)$ as it does in $\text{Conv}^{P,w}_{\mathsf{IV},x}$. The protocol outlined for QR does not give any information about the witness to the verifier.

Theorem 1.4.12. The interactive function P defined in Example 1.4.4 for QR is zero-knowledge.

Proof. Define the following simulator S:

$S((N, W))$:
$\overline{c} \twoheadleftarrow \{0, 1\}$;
$\overline{z} \twoheadleftarrow \mathbb{Z}^*_N$;
$\overline{Y} \leftarrow \overline{z}^2 / W^{\overline{c}} \mod N$;
return$(\overline{Y}, \overline{c}, \overline{z})$

The transcript for the protocol is generated based on the selection of two elements: c and y. Since W is fixed before both the protocol and the simulator are run it can cause no difference in the probability distribution. Let $\phi(N)$ represent the Euler totient function of N. When the protocol runs the prover uniformly selects some $y \in \mathbb{Z}^*_N$. Since it is a uniform distribution the probability of getting selecting any y is

$$\frac{1}{|\mathbb{Z}^*_N|} = \frac{1}{\phi(N)}.$$

Then the verifier uniformly picks a value of $c \in \{0, 1\}$, thus any c is picked with probability $\frac{1}{2}$. The transcript for the protocol is comprised of three elements, Y, c, and z. Since c and y are chosen and Y and z are generated from c and y the probability of generating any transcript is

$$\frac{1}{2 \cdot \phi(N)}.$$

Now for the simulator, choose $\overline{c} \in \{0, 1\}$ since this is chosen from the same set as c is has the same probability. Then choose $\overline{z} \in \mathbb{Z}^*_N$, the value $\overline{z}$ is chosen uniformly at random from the same set as y, thus any $\overline{z}$ will also be chosen with probability, $\frac{1}{\phi(N)}$. Since $\overline{Y}$ is determined by $\overline{c}$ and $\overline{z}$ we have all elements of the simulator's transcript and each transcript can be generated with probability

$$\frac{1}{2 \cdot \phi(N)}.$$

Since the protocol and the simulator generate any transcript with the same probability it follows that the prover is zero-knowledge. □

1.5 Commitment Schemes

An often useful tool for zero-knowledge protocols is for a prover to commit to information and allow the verifier to randomly select what portion of the information to

reveal. To conceptualize this, imagine the prover writing information down on a piece of paper and placing that paper into a safe, then handing the safe to the verifier and giving the verifier the combination. Since the prover can no longer alter the information in the safe once the verifier has it, the verifier knows the information hasn't been tampered with based on the information they requested. A commitment scheme is a cryptographic primitive that formalizes safes.

Definition 1.5.1. A *commitment scheme* $\mathcal{CS} = (\mathcal{P}, \mathcal{C}, \mathcal{V})$ is a collection of three algorithms that operate in the following way:

- A parameter generation algorithm $\mathcal{P}$ to generate a public parameter π.
- A commitment algorithm $\mathcal{C}_\pi$ that takes as input a message M and outputs (K, C) where C is a committed message and K is a decommitment key.
- A verification algorithm $\mathcal{V}_\pi$ that takes a commitment C, a message M, and a decommitment key K and outputs 1 if C is a commitment to M and outputs 0 otherwise.

The commitment scheme $\mathcal{CS}$ is correct if for all π that may be an output of $\mathcal{P}$, for all M in the message space, the function $\mathcal{V}_\pi(C, M, K)$ is always equal to 1, where$(K, C) = \mathcal{C}_\pi(M)$ [Gol01].

A commitment scheme has two qualities hiding security and binding security. The first is hiding security, this states that given only a commitment, nothing can be known about the message by any adversarial party. The definition of hiding relies on what is called an oracle, an oracle is a function that is aware of all global variables.

Definition 1.5.2. Let $\mathcal{CS} = (\mathcal{P}, \mathcal{C}, \mathcal{V})$ be a commitment scheme and A be an adversary. Define a game $\text{HIDE}_{\mathcal{CS}}$, initialize $\pi \leftarrow \mathcal{P}$ and $b \twoheadleftarrow \{0, 1\}$, these values cannot be seen by the adversary. Define an oracle $LR(M_0, M_1)$ which computes $(K, C) \leftarrow \mathcal{C}_\pi(M_b)$ and returns C. The adversary is given π and the ability to run the oracle. The adversary A attempts to generate b' such that $b = b'$, the game returns 1 if $b' = b$ and returns 0 otherwise [Gol01]. The hiding advantage of A is defined as

$$\text{Adv}^{\text{hide}}_{\mathcal{CS}} = 2 \cdot \Pr\left[\text{HIDE}_{\mathcal{CS}} = 1\right] - 1.$$

This means that an adversary has no hiding advantage if its probability of guessing a bit correctly is $\frac{1}{2}$. The next component is called binding security, this asks that an adversary cannot create a commitment that can open to two different messages.

Definition 1.5.3. Let $\mathcal{CS} = (\mathcal{P}, \mathcal{C}, \mathcal{V})$ be a commitment scheme and A be an adversary. Define a game $\text{BIND}_{\mathcal{CS}}$, initialize $\pi \leftarrow \mathcal{P}$. The adversary A is then given access to some cipher text C, two messages M_0, M_1, and two keys K_0, K_1. Then compute $v_0 = \mathcal{V}_\pi(C, M_0, K_0), v_1 = \mathcal{V}_\pi(C, M_1, K_1)$ and return 1 if $v_0 = v_1 = 1$ and $M_0 \neq M_1$. [Gol01] The binding advantage of A is

$$\text{BIND}_{\mathcal{CS}}(A) = \Pr\left[\text{BIND}_{\mathcal{CS}} = 1\right].$$

There is no binding advantage if a commitment can only be opened to reveal a single message.

Chapter 2

Norinori

Norinori is a pencil puzzle invented by Nikoli in 1986. The same company created similar grid-based paper games such as Sudoku, Nurikabe, and LITS.

2.1 Rules

The puzzle consists of an $n \times m$ rectangular grid split into t sections, called rooms. A cell must be horizontally or vertically adjacent to at least one other cell in the same room, rooms must be contiguous. An example of an instance can be seen in Figure 2.1.

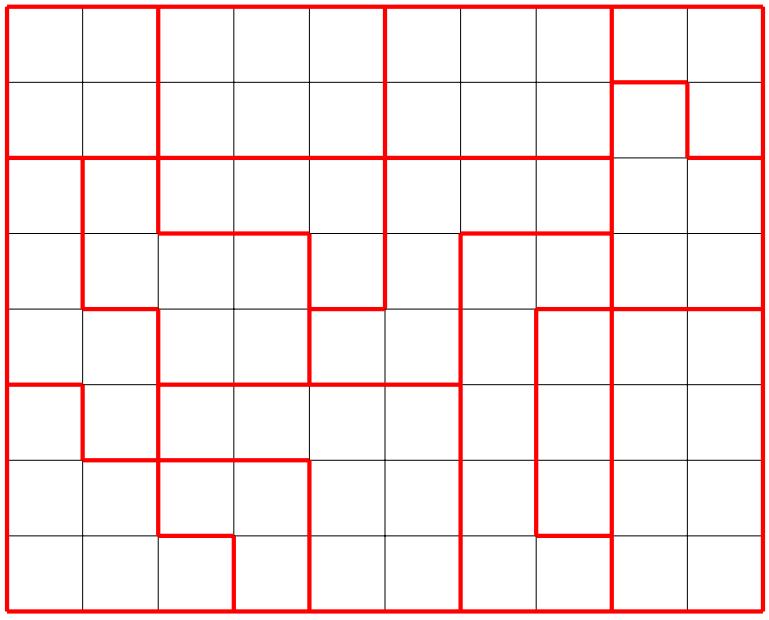


Figure 2.1: An empty 10x8 sample board

The puzzle is solved by filling in squares within the grid so that the following conditions are met:

1. Each room contains exactly two black squares.
2. Each black square is horizontally or vertically adjacent to exactly one other black square.

The board above is solvable with the assignment in Figure 2.2.

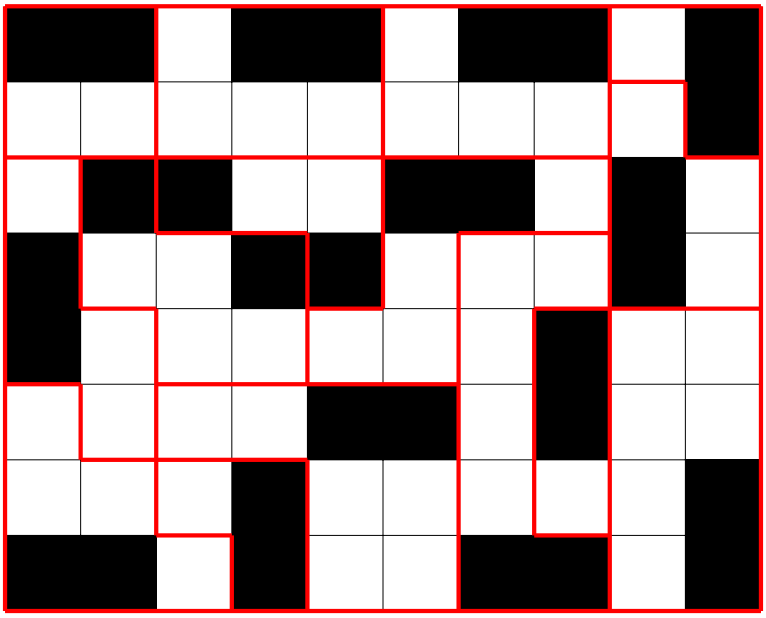


Figure 2.2: A solved 10x8 sample board from Figure 2.1

Not every board is solvable, for example, any board that contains the sub-board shown in Figure 2.3 is unsolvable.

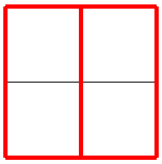


Figure 2.3: Unsolvable sub-board

It is also possible for a board to have multiple solutions, trivially three horizontal squares in one room can be filled in two valid ways.

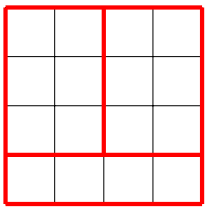
2.2 NP-Completeness of Norinori

Norinori can be represented as a language. The language representing Norinori is NP-complete. The categorizes Norinori with every other NP-complete language in terms of computational difficulty and guarantees that there exists a zero-knowledge proof for it.

2.2.1 Language

Norinori can be represented by a language NN that includes matrices of integers where two cells that contain the same integer are in the same room. This represents an empty board but it does not say that that board has a solution. If the board can be filled in according to the criteria of Norinori, a corresponding witness can be represented by a matrix with a bit in each cell.

Definition 2.2.1. The language NN is comprised of matrices of integers where any cell containing an integer x is orthogonally adjacent to at least one other cell containing x and all cells containing x are connected. Cells with the same integer can be referred to as rooms. An instance described above is in the language if each cell can also be assigned a bit such that each room has two 1-bits and each 1-bit is next to exactly one other 1-bit. The 1-bit represents black squares in the puzzle while 0-bits represent white squares.



(a)

$$\begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

(c)

Figure 2.4: An empty Norinori board (a) along with corresponding instance (b) and a valid witness (c).

Figures 2.4b and 2.4c show the instance and witness for the board in Figure 2.4a.

2.2.2 NP

To show that NN $\in$ NP, we define a verifier VF that checks that each room has two black squares and that each black square is next to exactly one other black square. For an instance x and witness w, VF(x, w) checks that the witness describes a valid solution. First, VF(x, w) first iterates over each cell of the witness and checks that if the cell has a 1-bit then it is adjacent to exactly one other 1-bit. If any are not, VF(x, w) returns 0. After that, it keeps a counter of 1-bits in each room and iterates over the witness increasing the counter as it goes. If all counters are at 2, the machine VF(x, w) returns 1, otherwise returns 0. The following algorithm describes this. The function adj(i) returns the number of bits with value 1 horizontally or vertically adjacent to the i^{th} cell.

VF(x, w) :
    blackCellsPerRoom $= [0, 0, ..., 0]$ where $|\text{blackCellsPerRoom}| = t$
    for i in the length of x
        if $w_i = 1$ and adj$(i) \neq 1$

```
            return 0
        if w_i = 1
            blackCellsPerRoom[x_i] += 1
    if blackCellsPerRoom ≠ [2, 2, ..., 2]
        return 0
    return 1
```

The verifier will return 0 if there is a section that fails the condition to be a part of the language and will return 1 otherwise.

Theorem 2.2.2. The language NN ∈ NP.

Proof. Let $R(x) = \{w \mid w$ describes a valid solution for the board represented by $x.\}$. Let VF be the verifier defined above, and let $x \in \Sigma^*$ be an $n \times m$ matrix that describes a Norinori board.

1. If $x \in$ NN, there exists $w \in R(x)$. It follows that for each i where $w_i = 1$ the function adj$(i) = 1$. In addition the count of squares with value 1 in each room will be 2. Since both conditions are met $\mathsf{VF}(x, w) = 1$.

2. If $x \notin$ NN, let $w \in \Sigma^*$. It follows that either for some i where $w_i = 1$, adj$(i) \neq 1$ or one of the rooms does not contain exactly two black squares. In either case $\mathsf{VF}(x, w) = 0$.

Since VF checks every cell in x and w once, it follows that $\mathsf{VF} \in O(nm)$. □

2.2.3 NP-Hardness

The final step in proving NN is NP-complete is proving that it is NP-hard. In order to do that, we need another NP-complete language to reduce from. For this, we will use the language Planar-3SAT. An instance in this language is in Figure 2.5.

Definition 2.2.3. Planar-3SAT is an extension of SAT where a Boolean statement is embedded in a plane [Dem15]. A boolean statement is in 3SAT if it is made up of clauses containing three literals connected by an and operator and each clause is connected by an or operator. A boolean statement $x \in$ PLANAR-3SAT if both of the following conditions are met:

1. $x \in$ 3SAT.

2. Each variable and clause in x can be embedded in a plane such that there is a graph with edges between each clause and the variables contained by it where no edge intersects with any other edge.

An instance of Planar-SAT is shown in Figure 2.5.

$$(x_1 \vee x_2 \vee \overline{x}_1) \wedge (x_2 \vee x_2 \vee x_3)$$

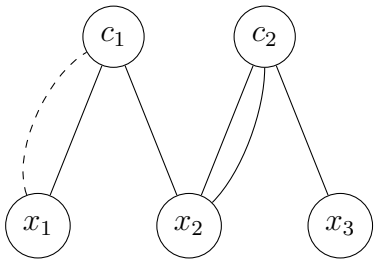


Figure 2.5: Planar-3SAT example. Nodes x_1, x_2, and x_3 correspond to variables in the formula. Nodes c_1 and c_2 capture the clauses. The solid line indicates that a literal, e.g. x_1, is in a clause. The dotted line indicates a negated literal, e.g. $\overline{x_1}$, within a clause. The graph represents the boolean formula at the top of the Figure.

Then Biro and Schmidt show a polynomial time algorithm that converts an instance of Planar-3SAT to an instance of Norinori [BS17a]. The final board created after reduction is very large, even for simple examples. We will show the resulting board piecewise.

Theorem 2.2.4. Planar-3SAT is polynomial time reducible to Norinori.

Proof. Consider a graph for an instance of Planar-3SAT where all lines are made either horizontal or vertical and have 90-degree angles, also known as a rectilinear embedding. We can refer to nodes of a graph of Planar-SAT that represent a variable as variable nodes similarly, nodes that represent a clause can be referred to as clause nodes. Then, each component of the graph needs to be mapped, to a section of a Norinori board, these sections are called gadgets. Figure 2.6 is a variable gadget.

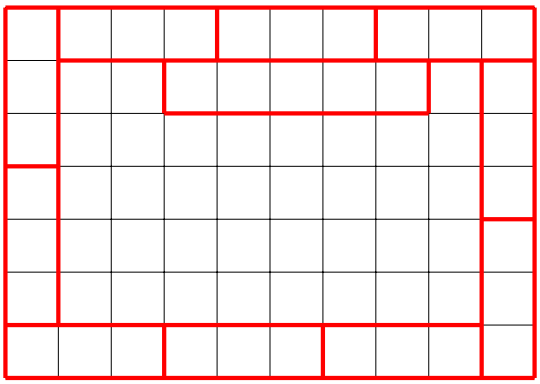
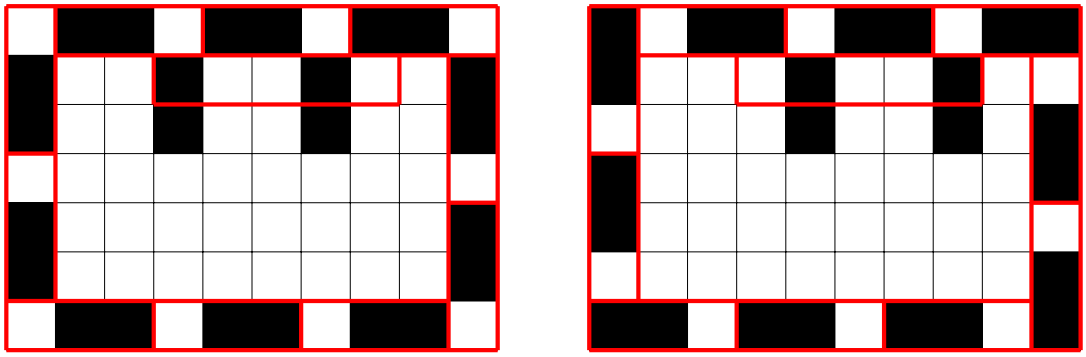


Figure 2.6: A variable Gadget

This subsection can be filled in exactly two ways so it can be used to represent a boolean. Figure 2.7a shows how it should be filled in if it is true and Figure 2.7b

show the filling for false. The gadget can also be expanded by lengthening any two opposing sides so more connections can be made. Replace each variable node in the original instance with a variable gadget.



(a) (b)

Figure 2.7: Assignments of black squares for variable gadgets corresponding to boolean variables with true (a) and false (b) assignments.

Now there needs to be a way to replace all of the edges of the graph. We use the corridor gadget in Figure 2.8 for this purpose. While the edges in the initial graph have no concept of length, the corresponding board does. If there are many variables or clauses the corridor gadget can be extended to reach further away variables.

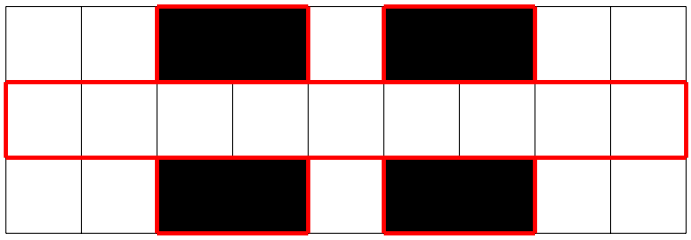
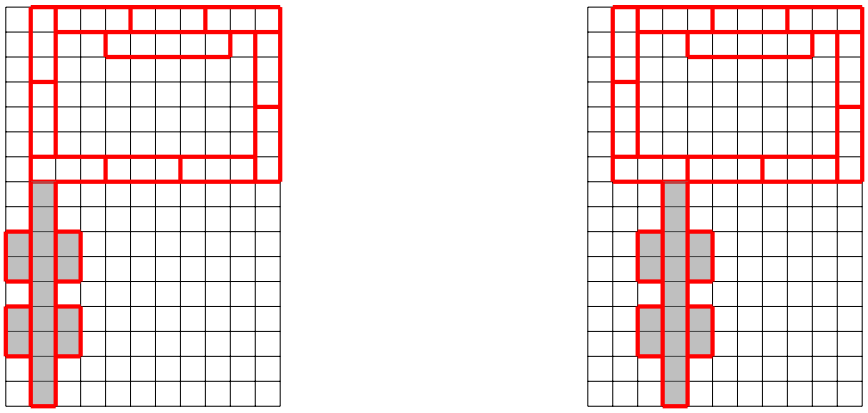


Figure 2.8: A corridor gadget corresponding to an edge in the graph along with the black squares guaranteed to be filled in.

Since the edges in Planar-3SAT are different for positive and negative literals, the resulting board in NN needs to represent that as well. If the literal is positive in the clause it is being connected to, it can be connected to a variable gadget by every third position around the edge starting from a corner going clockwise as these spaces are empty when the variable is true. If the literal is negative, it must connect to one of the squares that are every third starting one square clockwise from the corner. This becomes relevant for the clause gadget that will be introduced later. The connections can be seen in Figures 2.9a and 2.9b.

This gadget can be used to represent the edges in the original graph. However it currently only works for straight edges, Figure 2.10 shows how a corridor could be turned at a right angle with a bend gadget.



(a) (b)

Figure 2.9: A corridor gadget connected to a variable gadget for a positive (a) and negative (b) literal. The gray area is the corridor gadget.

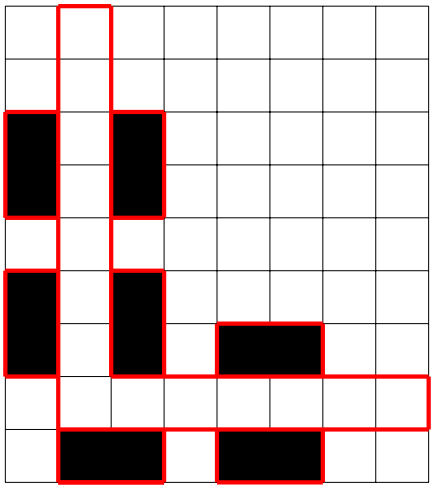


Figure 2.10: A bend gadget. This is used along with corridor gadgets to replace edges instance of Planar-SAT that have a 90-degree angle.

This is just a bent corridor. It works in the same way as the original corridor, in that it can only be filled at either end depending on the value of the variable. Now there is a conversion for every edge in the graph. Next, replace each clause node in the Planar-3SAT instance with the clause gadget in Figure 2.11.

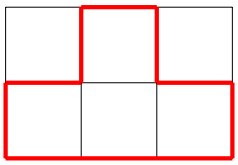
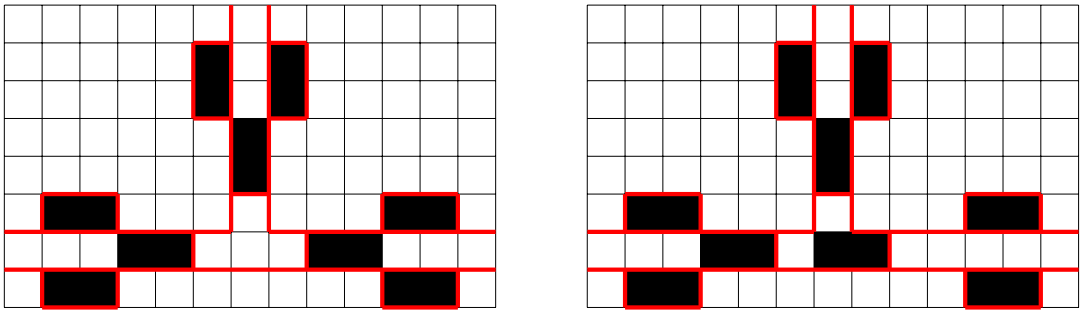


Figure 2.11: A Clause Gadget. This replaces all clause nodes in the instance of Planar-SAT

The corridor for each given variable is connected to the clause gadget by one of the three protruding points. This is set up so that if every variable in the clause is false then there is no valid way to fill the clause gadget.



(a) (b)

Figure 2.12: All corridors in (a) are connected to a false variable gadget. Thus, there is no valid way to fill the clause gadget. In (b), the left and top corridors are connected to a false variable gadget, and the right corridor is connected to a true variable gadget.

It can be seen in Figure 2.12a that there is no way to fill in the room correctly if all of the connections are false. In Figure 2.12b the rightmost connection is a true value, thus the clause can be filled in. Since the original graph is planar, it can always be embedded in a board without sections overlapping.

If the initial formula has a satisfying assignment, then each clause has at least one true statement. It follows that each gadget can be filled in at least one valid way. Thus, the resulting board is in NN. If the initial formula is not satisfiable, then at least one clause has three false variables. It follows that at least one clause cannot be filled in. Thus, the resulting board is not in NN. It follows that Planar-SAT is polynomial time reducible to NN and thus NN is NP-hard. Since NN $\in$ NP, it follows that NN is NP-complete. $\square$

Chapter 3

Zero-Knowledge Proof for Norinori

The protocol described in this chapter is based on zero-knowledge proofs from "Interactive Physical Zero-Knowledge Proof for Norinori" [DLM+19] and "Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles" [GNPR09].

3.1 Protocol

First, define a commitment scheme $\mathcal{CS} = (\mathcal{P}, \mathcal{C}, \mathcal{V})$, with $\pi \leftarrow \mathcal{P}$ as in Definition 1.5.1. Then, consider a Norinori board with room markings represented as shown, for example, in Figure 2.4b. Let x represent the board and let w represent the witness. Next, both the prover and the verifier consider the following modifications to the instance x and witness w: there is an additional row added to the top and an additional column added to both the left and right. All added squares are in a new room. You can also treat the grid as if it was wrapped onto a torus, so the left column is adjacent to the right column and the top row is adjacent to the bottom row. This is done so that each square has the same number of adjacent squares. The witness has the same modifications where the top left and top right corners are marked as 1 and the other additional squares are marked as 0. The matrices are then flattened to list. These modifications are done by the algorithm pad,

pad(x, w):
    for row in m
        $x_{\text{row}} = [t+1, x_{\text{row}}, t+1]$
        $w_{\text{row}} = [0, w_{\text{row}}, 0]$
    $x = [t+1, t+1, ...t+1, x]$
    $w = [1, 0, 0, ..., 0, 1, w]$
    n += 2
    m += 1
    t += 1

Consider the modified instance and witness as the new x and w respectively. The new instance and witness are $n \times m$ grids with t rooms.

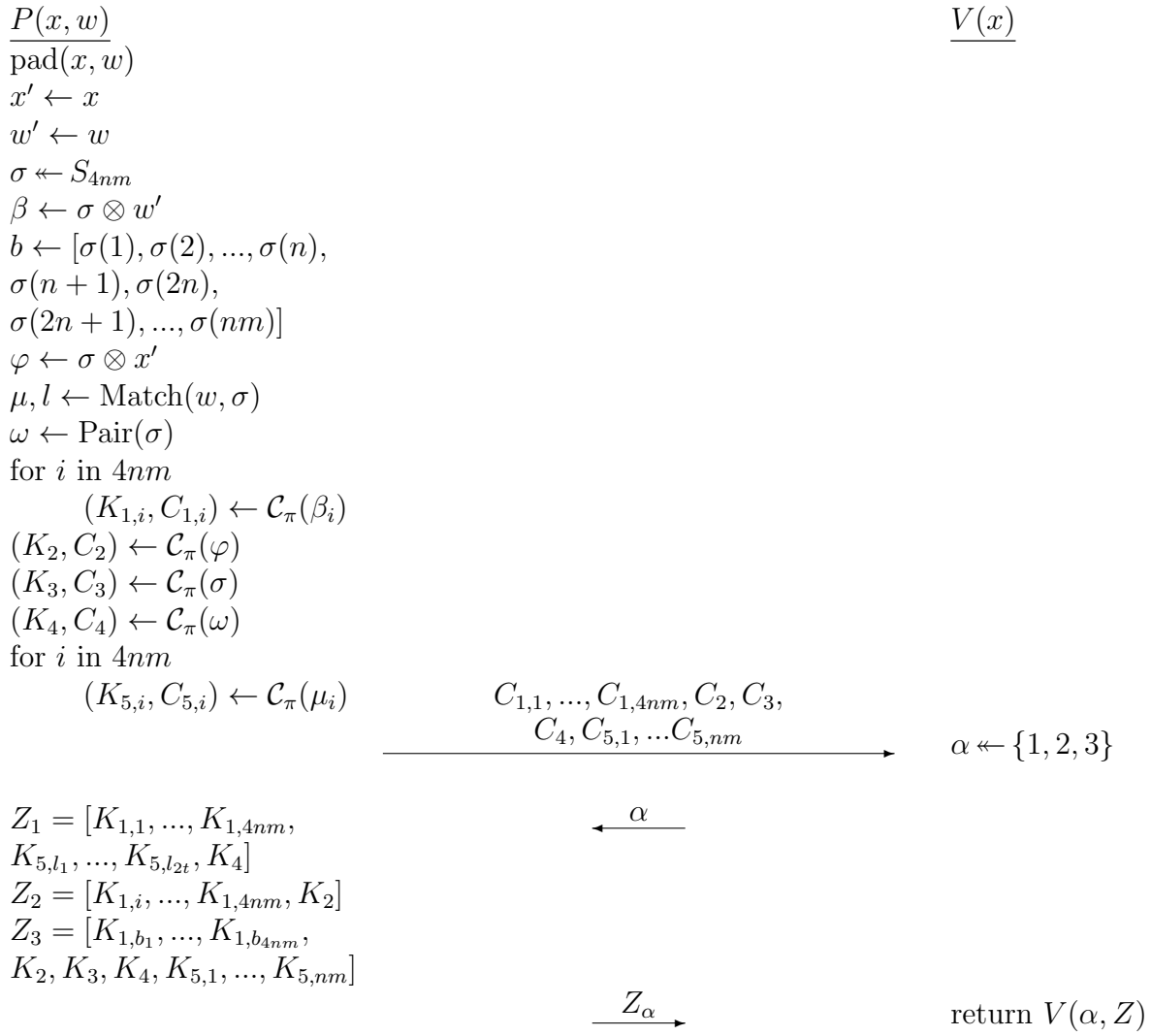


Figure 3.1: Zero-Knowledge protocol for Norinori

With the buffer added, consider the protocol shown in Figure 3.1. The prover creates w' and x' which are copies of w and x where each value is repeated 4 times. This is necessary to maintain zero-knowledge during pair verification. The prover then computes

$$\begin{aligned} w &= [w_1, w_2, ..., w_{nm}] \\ x &= [x_1, x_2, ..., x_{nm}] \\ w' &= [w_1, w_1, w_1, w_1, w_2, w_2, w_2, w_2, ..., w_{nm}, w_{nm}, w_{nm}, w_{nm}] \\ x' &= [x_1, x_1, x_1, x_1, x_2, x_2, x_2, x_2, ..., x_{nm}, x_{nm}, x_{nm}, x_{nm}] \end{aligned}$$

Let S_{4nm} be the symmetric group of size $4nm$, the prover randomly selects $\sigma \in S_{4nm}$. This is used to create two new lists with the operator $\otimes$ defined as follows

$$\begin{aligned} \beta = \sigma \otimes w' &= \left[w'_{\sigma(1)}, w'_{\sigma(2)}, ..., w'_{\sigma(4nm)}\right] \\ \varphi = \sigma \otimes x' &= \left[x'_{\sigma(1)}, x'_{\sigma(2)}, ..., x'_{\sigma(4nm)}\right]. \end{aligned}$$

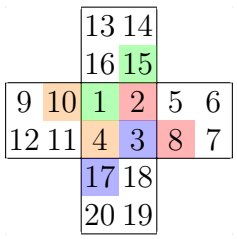
The lists β and φ permute the values so that the verifer is unable to determine anything about the witness. Next, create a list of tuples μ that contains the indexes of β that represent the same cell, there also needs to be a set l that keeps track of which indexes of μ represent a black square. For all $a, b, c, d \in [1, 4nm]$ where $b = a + 1, c = b + 1, d = c + 1$

$$\begin{aligned} \mu &= [(\sigma(a), \sigma(b), \sigma(c), \sigma(d)) \mid a = 1 \mod 4] \\ l &= \{i \mid \text{for each } j \text{ in } \mu_i, \beta_j = 1\} \end{aligned}$$

Each value in each element of μ correspond to the same square in β. The set l keeps track of which tuples in μ correspond to a black square. To see how μ is created see the following pseudo-code

```
Match(w, σ):
    μ = [ ]
    l = { }
    for i in length of w
        j = i * 4
        append μ with (σ(j), σ(j + 1), σ(j + 2), σ(j + 3))
        if w_i = 1
            add i to l
    return μ, l
```

The last thing needed is a way to show which squares are horizontally or vertically adjacent to each other. In order to avoid giving over information, there cannot be any sense of how the squares are connected beyond pairs. This is the reason why 4 copies of each cell were created earlier so that one copy could be used per pair that the cell is included in. Figure 3.2 shows how the middle square would be paired with the outer ones.



Pairs: $(1, 15), (2, 8), (3, 17), (4, 10)$

Figure 3.2: Shows how the middle cell would be paired to the surrounding cells

The set of these pairs is defined as follows:

$$\omega = \{(a, b) \mid a \text{ is horizontally or vertically adjacent to } b\}$$

The prover could generate ω with the following algorithm:

Pair(σ):
$\omega = [\,]$
for i in nm
left $= 4 * (((i-1) \mod n) + n * \lfloor \frac{i}{n} \rfloor) + 1$
right $= 4 * (((i-1) \mod n) + n * \lfloor \frac{i}{n} \rfloor) + 3$
up $= 4 * ((i-n) \mod nm) + 2$
down $= 4 * ((i+n) \mod nm)$
for j, direction in [up, right, down, left]
append ω with $(\sigma(i * 4 + j), \sigma(\text{direction}))$
return ω

Now using the defined commitment scheme $\mathcal{CS}$ commit the values as follows,

$$\begin{aligned}
&\text{for } i \text{ in } 4nm \\
&\qquad (K_{1,i}, C_{1,i}) \leftarrow \mathcal{C}_\pi(\beta_i) \\
&(K_2, C_2) \leftarrow \mathcal{C}_\pi(\varphi) \\
&(K_3, C_3) \leftarrow \mathcal{C}_\pi(\sigma) \\
&(K_4, C_4) \leftarrow \mathcal{C}_\pi(\omega) \\
&\text{for } i \text{ in } 4nm \\
&\qquad (K_{5,i}, C_{5,i}) \leftarrow \mathcal{C}_\pi(\mu_i)
\end{aligned}$$

The commitments are sent to the verifier. The verifier then randomly chooses $\alpha \in \{1, 2, 3\}$ and sends α back to the prover. The prover generates

$$\begin{aligned}
Z_1 &= [K_{1,1}, ..., K_{1,4nm}, K_{5,l_1}, ..., K_{5,l_{2t}}, K_4] \\
Z_2 &= [K_{1,i}, ..., K_{1,4nm}, K_2] \\
Z_3 &= [K_{1,b_1}, ..., K_{1,b_{4nm}}, K_2, K_3, K_4, K_{5,1}, ..., K_{5,nm}]
\end{aligned}$$

and send Z_α to the verifier. Afterward, the verifier runs the function $V(\alpha, Z)$. The general idea is that if $\alpha = 1$ then the verifier checks that each black square is next to exactly one other black square, if $\alpha = 2$ the verifier checks if each room contains two black squares, and if $\alpha = 3$ the verifier checks that all of the pairs given are actually next to each other and that all of the cells given are in the correct room. The first two are meant to check if there is a valid witness and the second just prevents the prover from lying about the witness. More precisely, $V(\alpha, Z)$ operates as follows:

1. If $\alpha = 1$, the verifier iterates over all the bits in β and counts that there are $8t$ bits with value 1. Iterates over all pairs in ω and counts that there are t pairs with value 1, it also uses μ the verify that each black square is only used in one pair. If all conditions are met return 1, otherwise return 0. The following algorithm details this,

   blackBits = 0
   for i in the length of β
       if $\beta_i = 1$
           blackBits += 1
   if blackBits $\neq 8 * t$
       return 0

   blackPairs = 0
   usedCells = [ ]
   for (i, j) in ω
       if $\beta_i = 1 = \beta_j$
           blackPairs += 1
           each element of the tuples of μ that contains i, or j appended to usedCells
   if blackPairs $\neq t$ or usedCells contains duplicates
       return 0
   else
       return 1

2. If $\alpha = 2$, the verifier iterates over all the bits in β and keep count of the number of bits with value 1 and the number of 1 bits in each room. If there are not $8t$ bits with value 1 overall or 8 bits with value 1 bits in each room, return 0, otherwise return 1. The following algorithm details this,

   blackBits = 0
   roomCounts = $[0, 0, ..., 0]$ where |roomCounts| $= t$
   for i in β
       if $\beta_i = 1$
           blackBits += 1
           roomCounts[φ_i] += 1
   if blackBits $\neq 8 * t$
       return 0

for count in roomCounts
    if count $\neq 8$
        return 0
return 1

3. If $\alpha = 3$, the verifier takes the inverse of σ as σ^{-1} and computes $\sigma^{-1}\varphi$ to verify that the rooms are in the correct positions relative to the x. Then, applies σ^{-1} to the elements of each pair in ω to verify that the pairs are indeed adjacent. Next, checks that the values of β within the buffer have the agreed-upon values. Finally, checks that each element in μ represents a single cell. If both conditions are met return 1, else return 0. The following algorithm details this,

if $\sigma^{-1}\otimes\varphi \neq x'$
        return 0
if Pair$(\sigma) \neq \omega$
        return 0
if $\sigma^{-1}(\mu) \neq [(1,2,3,4),(5,6,7,8),...,(4nm-3,4nm-2,4nm-1,4nm)]$
        return 0
if $\sigma^{-1} \otimes \beta \neq [1,0,0,...,1,0,0,...,0]$
        return 0
return 1

Figure 3.1 illustrates the protocol. Example 3.1.1 will illustrate the computation done by the prover.

Example 3.1.1. Figure 3.3 shows a simple board that will be used for this example.

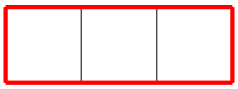


Figure 3.3: Shows a simple Norinori board

The instance x known to both the prover and the verifier is represented in Figure 3.3, the witness w will have both left squares filled in. The prover and the verifier will both use the pad algorithm to add a buffer to x and w, the result is shown in Figure3.4a.

(a) (b)

Figure 3.4: Instance (a) and witness (b) after modification by the pad algorithm

The modified board considers the edges to wrap around so that the left column is adjacent to the right column and the top is adjacent to the bottom, for instance, the top left corner and top right corner are now adjacent. The prover would then makes in new witness and instance,

$$
\begin{aligned}
w =&[1,0,0,0,1,0,1,1,0,0]\\
x =&[1,1,1,1,1,1,2,2,2,1]\\
w' =&[1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,0,\\
&0,0,0,0,0,0,0]\\
x' =&[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,\\
&2,2,2,1,1,1,1]
\end{aligned}
$$

Again, the first four numbers in both w' and x' all represent the first cell and the same is true for every following set of four numbers. This is done so that the verifier cannot determine the distance between any black cells in the witness unless they are next to each other. Then randomly select $\sigma \in S_{40}$ and generate,

$$
\begin{aligned}
\sigma =&(4,15,40,29)(27,39,35,34)(1,37,9,30,18,6,\\
&22,2,19,38,13,3,24,21,17,23,28,5,14,10,8,11,36,26,32,7,31,20,16,33,12)\\
\beta =&[0,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,1,0,\\
&0,0,0,1,1,1,0]\\
\varphi =&[1,1,1,2,2,1,2,1,1,1,1,2,1,1,1,1,1,2,1,2,1,1,1,1,2,2,2,1,1,1,1,2,1,\\
&2,1,1,1,1,2,1]\\
\mu =&[(14,22,31,11),(37,19,24,15),(30,8,36,1),(23,6,38,16),(25,32,39,5),\\
&(3,10,40,33),(17,2,28,21),(12,27,34,26),(4,18,20,7),(9,13,35,29)]\\
l =&\{2,4,5,9\}\\
\omega =&[(6,15),(28,37),(1,22),(7,32),(13,21),(17,24),(2,5),(14,39),(25,31),(23,35),\\
&(11,19),(10,16),(8,33),(3,34),(12,40),(20,30),(9,38),(4,36),(18,26),(27,29)].
\end{aligned}
$$

These are the computations done by the prover. The value σ is used to randomize the lists of bits so that the verifier cannot discern anything about the witness.

Now, we will show why the above protocol has completeness, soundness, and zero-knowledgeness.

3.2 Completeness

The verifier V in the protocol will accept correct answers presented by the prover P, this is referred to as completeness.

Theorem 3.2.1. Let R be the relation for Norinori, let $L_R = \text{NN}$. Let P and V be the prover and verifier defined in Figure 3.1. The interaction between P and V is complete as defined in Definition 1.4.6.

Proof. Let $x \in \text{NN}$ and let $w \in R(x)$. The interaction between $P(x, w)$ and $V(x)$ proceeds in one of the following ways depending on the selection of α,

1. Suppose $\alpha = 1$. Since $x \in L$ and $w \in R(x)$, it follows that there will be t pairs in ω where each corresponding bit in β has value 1. It also follows that the counter *usedCells* will contain no duplicates. Thus, V returns 1.

2. Suppose $\alpha = 2$. Since $x \in L$ and $w \in R(x)$, it follows that the list *roomCounts* will contain 2 at every index, thus V will return 1.

3. Suppose $\alpha = 3$. Since $x \in L$ and $w \in R(x)$, it follows that $\varphi = \sigma \otimes x'$ thus, $\sigma^{-1} \otimes \varphi = x'$. It also follows that $\text{Pair}(\sigma) = \omega$. Further, $\mu = \sigma(\{(1,2,3,4),(5,6,7,8),...,(4nm-3,4nm-2,4nm-1,4nm)\})$ thus, $\sigma^{-1}(\mu) = \{(1,2,3,4),(5,6,7,8),...,(4nm-3,4nm-2,4nm-1,4nm)\}$. Similar logic follows for the revealed part of β. Thus, V returns 1.

Thus, $P(x, w) \leftrightarrow V(x)$ returns 1 in all cases. In other words, with probability 1. Therefore, the protocol is complete. □

3.3 Soundness

We will show that the verifier V rejects a wrong answer from the prover P some percentage of the time, this is referred to as soundness.

Theorem 3.3.1. Let R be the relation for Norinori, let $L_R = \text{NN}$. Let V be the verifier defined in Figure 3.1. For all provers P', the interaction between P' and V is sound with probability $\frac{2}{3}$ as in Definition 1.4.6.

Proof. Let $x \notin \text{NN}$ and let $a \in \Sigma^*$. Let $\delta = \frac{2}{3}$. Since $x \notin L$ it follows that at least one of the following it true,

1. The prover P is cheating in some way. This could mean not giving proper pairs in ω, not giving proper rooms in φ, incorrectly placing the buffer squares of β, or matching cells with different representations together in μ.

2. The witness a contains a number of bits with value 1 that is not equal to $2t$.

3. The witness a contains a room that does not contain two bits with value 1.

4. There exists a bit with value 1 paired with more than one other bit with value 1.

5. There exists a bit with value 1 paired with no other bits with value 1.

Each of these will be found in the protocol in the following ways,

1. If V selects $\alpha = 3$ and the pairs are incorrect then,

$$\text{Pair}(\sigma) \neq \omega.$$

   If the rooms are incorrect then,

$$\sigma^{-1} \otimes \varphi \neq x'.$$

   If μ does not refer to the correct cells then

$$\sigma^{-1}(\mu) \neq [(1,2,3,4),(5,6,7,8),...,(4nm-3,4nm-2,4nm-1,4nm)].$$

   If the buffer squares are incorrect,

$$\sigma^{-1} \otimes \beta \neq [1,0,0,...,1,0,0,...,0].$$

   Each of these will return 0 when $\alpha = 3$ thus, V will return 1 with probability $\frac{2}{3}$.

2. If V chooses $\alpha = 1, 2$, the count of bits with value 1 will not be correct. Thus, V returns 1 with probability $\frac{1}{3}$.

3. If V chooses $\alpha = 2$, the list *roomCounts* that contains the number of black squares in each room will contain one value that is not equal to 2, thus V will return 0. It follows that V will return 1 with probability $\frac{2}{3}$.

4. If V chooses $\alpha = 1$, the list *usedCells* will contain a duplicate index and will return 0. Thus, V returns 1 with probability $\frac{2}{3}$.

5. If V chooses $\alpha = 1$, the counter *blackPairs* will not be equal to t, thus the V returns 0. It follows that V returns 1 with probability $\frac{2}{3}$.

Since the existence of any of the above conditions causes V to return 1 less than $\frac{2}{3}$ of the time. Now we will show is this process is repeated n times, $\frac{2}{3}$ is negligible. Let $k \in \mathbb{R}_{>0}$. It follows that $\lim_{n\to\infty} \left(\frac{2}{3}\right)^n n^k = \lim_{n\to\infty} \frac{2^n n^k}{3^n}$. By L'Hopital's Rule, we have

$$\begin{aligned}
\lim_{n\to\infty} \frac{2^n n^k}{3^n} &= \lim_{n\to\infty} \frac{\frac{d^k}{dn^k}(2^n n^k)}{\frac{d^k}{dn^k}(3^n)} \\
&= \lim_{n\to\infty} \frac{k! 2^n}{3^n \log^k(2)} \\
&= 0.
\end{aligned}$$

Thus, $\left(\frac{2}{3}\right)^n$ is negligible and it follows that $P'(x,a) \leftrightarrow V(x)$ is sound. □

3.4 Zero-Knowledgeness

Next, we will show that the prover P reveals no information to the verifier V. The protocol is zero-knowledge if there exists a simulator that creates a transcript indistinguishable from one generated by interaction. The simulator S takes an instance of Norinori x as input. Then, S randomly chooses some value $\alpha \in \{1, 2, 3\}$. Based on α it generates the following:

1. If $\alpha = 1$, the transcript should match the transcript of an interaction where the verifier checked the pairs. The simulator generates the list of pairs before permutation,
$$\omega = \{(1, 2), (3, 4), ..., (4nm - 1, 4nm)\}.$$
Then generate the list of bits before permutation,
$$\beta = [\underbrace{1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, ..., 1, 0}_{14t}, 0, 0, 0, ..., 0].$$
The first 14 bits in β are repeated t times so there are the correct number of each type of pair, note that $|\beta| = 4nm$. Next, create the list of which bits with value 1 belong to the same cell,
$$\mu = [(1, 3, 5, 7), (2, 9, 11, 13), (15, 17, 19, 21), ..., (14t-12, 14t-5, 14t-3, 14t-1)].$$
Due to the ordering of β these will always represent four different black squares that are in the correct pairings. Then, randomly choose some $\sigma \in S_{4nm}$ and create the following,
$$\overline{\omega} = \sigma(\omega)$$
$$\overline{\beta} = \sigma \otimes \beta$$
$$\overline{\mu} = \sigma(\mu).$$
The simulator S returns the the transcript $(\overline{\beta}, \overline{\omega}, \overline{\mu})$.

2. If $\alpha = 2$, the transcript should match room verification. First, initialize a list $\beta = [0, 0, ..., 0]$ where $|\beta| = nm$. Then, S changes the values corresponding to the first two squares of each room to 1. Next, randomly choose $\sigma \in S_{4nm}$ and generate
$$\overline{\beta} = \sigma \otimes \beta'$$
$$\overline{\varphi} = \sigma \otimes x'.$$
Then, return the transcript $(\overline{\beta}, \overline{\varphi})$.

3. If $\alpha = 3$, the transcript will resemble a cheating check. Initialize β to contain the values in the added buffer around x, no other values need to be initialized cause they will not be revealed. Initialize the set
$$\mu = [(1, 2, 3, 4), (5, 6, 7, 8), ..., (4nm - 3, 4nm - 2, 4nm - 1, 4nm)].$$

Next, randomly choose $\overline{\sigma} \in S_{4nm}$ and using the Pair algorithm from the protocol create the set

$$\overline{\omega} = \text{Pair}(\overline{\sigma}).$$

Then generate

$$\overline{\beta} = \overline{\sigma} \otimes \beta'$$
$$\overline{\varphi} = \overline{\sigma} \otimes x'$$
$$\overline{\mu} = \overline{\sigma}(\mu).$$

Finally, return the transcript $(\overline{\sigma}, \overline{\beta}, \overline{\omega}, \overline{\varphi}, \overline{\mu})$.

The description above used some caveats. First, the initial ordering of each list is chosen arbitrarily, there are many other ways the initial lists could be chosen. Second, since the ciphertext of any commitment cannot be discerned without a corresponding key any information where the verifier was not given the key is ignored. The simulator S is represented in Pseudo-code in Figure 3.5.

Theorem 3.4.1. Let R be the relation for Norinori and let $L_R = \text{NN}$. Let P be the prover defined in Figure 3.1. For all verifiers V', the interaction between P and V' is zero-knowledge.

Proof. Let $x \in \Sigma^*$ and let $w \in \Sigma^*$ be a witness for x. Let S be the simulator defined in Figure 3.5. The first portion of generating the transcript is choosing α, both the protocol and the simulator choose α randomly from the set $\{1, 2, 3\}$. Since both are chosen randomly from the same set, it follows that any alpha can be chosen with probability $\frac{1}{3}$ in both transcripts. Regardless of the α chosen, the simulator and the protocol compute several fixed lists and sets and permute them according to an element of S_{4nm}. Since both elements are chosen randomly from the same set any one can be chosen with probability $\frac{1}{4nm!}$. Next see that, if a, b are lists of integers where $|a| = |b|$ and the count of each integer in the list is the same, for all $\tau \in S_{|a|}$ there exists $\rho \in S_{|a|}$ such that $\tau(a) = \rho(b)$. This means that to show that $\text{Conv}^{P,w}_{\mathsf{IV},x}$ is indistinguishable from $S(x)$ all that is left is to show that the corresponding lists and sets have the same length and contain the same elements. This can be split into cases depending on which α is chosen.

1. If $\alpha = 1$, the list β from $\text{Conv}^{P,w}_{\mathsf{IV},x}$ has length $4nm$ and is made up of $8t$ entries of value 1 and $4nm - 8t$ entries of value 0. The list $\overline{\beta}$ in $S(x)$ also has length $4nm$ and starts by repeating a sequence of 8 bits with value 1 and 6 bits with value 0 a total of t times. Thus, there are $8t$ bits with value 1, and since the only other value is 0, there are $4nm - 8t$ bits with value 0. Next, μ is a list of $2t$ length 4 tuples of unique integers in $[1, 4nm]$. It is plain to see that $\overline{\mu}$ is the same. Similarly, ω is a set of $2nm$ length 2 tuples of unique integers in $[1, 4nm]$. The simulator's $\overline{\omega}$ is generated as such.

2. If $\alpha = 2$, the same argument for β and $\overline{\beta}$ is true from the previous point. The list φ and $\overline{\varphi}$ are both permutations of x' thus, they have the same length and the same elements.

$S(x)$:
- $\alpha \twoheadleftarrow \{1, 2, 3\}$
- if $\alpha = 1$
  - $\omega = \{(1, 2), (3, 4), ..., (4nm - 1, 4nm)\}$
  - $\beta = [\underbrace{1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, ..., 1, 0}_{14t}, 0, 0, ..., 0]$
  - $\mu = [(1, 3, 5, 7), (2, 9, 11, 13), ..., (14t - 12, 14t - 5, 14t - 3, 14t - 1)]$
  - $\sigma \twoheadleftarrow S_{4nm}$
  - $\overline{\omega} = \sigma(\omega)$
  - $\overline{\beta} = \sigma \otimes \beta$
  - $\overline{\mu} = \sigma(\mu)$
  - return $(\overline{\beta}, \overline{\omega}, \overline{\mu})$
- if $\alpha = 2$
  - roomCounts $= [0, 0, ..., 0]$ where $|\text{roomCounts}| = t$
  - $\beta = [0, 0, ..., 0]$ where $|\beta| = nm$
  - for i in x
    - if $\text{roomCounts}_{x_i} < 2$
      - $\beta_i = 1$
      - $\text{roomCounts}_{x_i} += 1$
  - $\sigma \twoheadleftarrow S_{4nm}$
  - $\overline{\beta} = \sigma \otimes \beta'$
  - $\overline{\varphi} = \sigma \otimes x'$
  - return $(\overline{\beta}, \overline{\varphi})$
- if $\alpha = 3$
  - $\beta = [1, 0, ..., 0]$
  - $\mu = [(1, 2, 3, 4), (5, 6, 7, 8), ..., (4nm - 3, 4nm - 2, 4nm - 1, 4nm)]$
  - $\overline{\sigma} \twoheadleftarrow S_{4nm}$
  - $\overline{\omega} = \text{Pair}(\overline{\sigma})$
  - $\overline{\beta} = \overline{\sigma} \otimes \beta'$
  - $\overline{\varphi} = \overline{\sigma} \otimes x'$
  - $\overline{\mu} = \overline{\sigma}(\mu)$
  - return $(\overline{\sigma}, \overline{\beta}, \overline{\omega}, \overline{\varphi}, \overline{\mu})$

Figure 3.5: Zero-knowledge simulator for Norinori

3. If $\alpha = 3$, the arguments for φ, $\overline{\varphi}$, ω, and $\overline{\omega}$ follow from above. The permutation elements σ and $\overline{\sigma}$ are chosen randomly from the same set. The argument for β and $\overline{\beta}$ is similar to the above except that it only contains the buffer elements of w, in this case, $|\beta| = n + 2m$ and only two bits have value 1. The list $\overline{\beta}$ is created in the same way using information from x. Finally, μ is similar to above except for in this case $|\mu| = nm$, again the same is true for this case of $\overline{\mu}$.

Since all of the corresponding fixed elements have the same length and values and are permuted by an element of the same group, it follows that any transcript that results from the interaction can result from the simulator with probability $\frac{1}{3(4nm)!}$, thus $\text{Conv}^{P,w}_{\mathsf{IV},x}$ is indistinguishable from $S(x)$. It follows that $P(x, w) \leftrightarrow V(x)$ is zero-knowledge. □

References

[BS17a] Michael Biro and Christiane Schmidt. Computational complexity and bounds for norinori and lits. In *Proceedings of the 33rd European Workshop on Computational Geometry (EuroCG 2017), Malmö, Sweden*, pages 29–32, 2017.

[BS17b] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Crypto textbook, 2017.

[Dem15] Erik Demaine. Planar sat. YouTube video, 2015.

[DLM+19] Jean-Guillaume Dumas, Pascal Lafourcade, Daiki Miyahara, Takaaki Mizuki, Tatsuya Sasaki, and Hideaki Sone. Interactive physical zero-knowledge proof for norinori. In *Proceedings of the 25th International Computing and Combinatorics Conference (COCOON)*, pages 138–151, Cham, 2019. Springer.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GNPR09] Ronen Gradwohl, Moni Naor, Benny Pinkas, and Guy N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of Sudoku puzzles. In *Advances in Cryptology – EUROCRYPT 2009*, pages 19–36, Berlin, Heidelberg, 2009. Springer.

[Gol01] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, Cambridge, UK, 2001.

[Sip12] Michael Sipser. *Theory of Computation*. Cengage Learning, Boston, MA, 3rd edition, 2012.