



NP-Completeness and Physical Zero-Knowledge Proofs for Sumplete, a Puzzle Generated by ChatGPT

Kyosuke Hatsugai¹ · Suthee Ruangwises¹ · Kyoichi Asano¹ · Yoshiki Abe^{1,2}

Received: 14 December 2023 / Accepted: 29 May 2024 / Published online: 31 July 2024
© The Author(s) 2024

Abstract

Sumplete is a logic puzzle generated by ChatGPT in March 2023. The puzzle consists of a rectangular grid, with each cell containing an integer. Each row and column also has an integer called *target value* assigned to it. The objective of this puzzle is to cross out some numbers in the grid such that the sum of uncrossed numbers in each row and column is equal to the corresponding target value. In this paper, we prove that Sumplete is NP-complete. We also propose a physical zero-knowledge proof protocol for the puzzle using physical cards.

Keywords Physical zero-knowledge proof · Card-based cryptography · NP-completeness · Sumplete · Puzzle

1 Introduction

1.1 Background

Chat Generative Pre-trained Transformer (ChatGPT) is an artificial intelligence chatbot developed by OpenAI [2]. ChatGPT is famous for its ability to answer questions

A preliminary version of this paper [1] was published in the proceedings of COCOON 2023.

✉ Kyosuke Hatsugai
hatsugai@uec.ac.jp
Suthee Ruangwises
ruangwises@uec.ac.jp
Kyoichi Asano
k.asano@uec.ac.jp
Yoshiki Abe
yoshiki@uec.ac.jp

¹ The University of Electro-Communications, 1–5–1 Chofugaoka, Chofu, Tokyo 182–8585, Japan

² National Institute of Advanced Industrial Science and Technology, 2–3–26 Aomi, Koto-ku, Tokyo 135–0064, Japan

| | | | | | |
|----|----|----|---|----|----|
| 3 | 5 | 5 | 7 | 1 | 13 |
| 5 | 1 | 4 | 1 | 8 | 14 |
| 4 | 7 | 2 | 5 | 2 | 11 |
| 6 | 2 | 4 | 9 | 4 | 6 |
| 3 | 3 | 4 | 9 | 6 | 15 |
| 11 | 18 | 11 | 9 | 10 | |

| | | | | | |
|--------------|----|--------------|--------------|--------------|----|
| 3 | 5 | 5 | 7 | 1 | 13 |
| 5 | 1 | 4 | 1 | 8 | 14 |
| 4 | 7 | 2 | 5 | 2 | 11 |
| 6 | 2 | 4 | 9 | 4 | 6 |
| 3 | 3 | 4 | 9 | 6 | 15 |
| 11 | 18 | 11 | 9 | 10 | |

Fig. 1 An example of a 5×5 Sumplete puzzle (left) and its solution (right)

more naturally than other chatbots. Furthermore, it can perform generative tasks, including writing essays, drawing pictures, coding, and making puzzles.

In March 2023, ChatGPT created a puzzle named *Sumplete*¹ [3], which is a logic puzzle similar to Sudoku, Kakuro, and Hitori. The Sumplete puzzle consists of an $m \times n$ rectangular grid, with each cell containing an integer. Each row and column also has an integer called *target value* assigned to it. The objective of this puzzle is to cross out some numbers in the grid such that the sum of uncrossed numbers in each row and column is equal to the corresponding target value [4]. See Fig. 1.

The official rule does not have any constraint on the number in each cell, which means it can be any integer. However, in all example puzzles displayed on the official website, all numbers x are limited to $0 < |x| < 20$ [4]. We call a variant with this constraint *restricted Sumplete*, and a variant without this constraint *general Sumplete* (or simply just Sumplete).

As there are 2^{mn} possible combinations of cells to cross out, it seems difficult to determine whether a given Sumplete puzzle has a solution for a large puzzle. In fact, deciding the solvability of Sumplete is NP-complete, as will be shown later.

1.2 Physical Zero-Knowledge Proof Protocols

Consider a situation where a contestant wants to convince a challenger that a given Sumplete puzzle has a solution but does not want to reveal the solution to them. Although these requirements may seem contradictory, they can be satisfied simultaneously using a cryptographic technique called *zero-knowledge proof (ZKP)*.

ZKP is an interactive protocol introduced by Goldwasser et al. in 1989 [5]. A ZKP allows a prover who knows the solution of a problem to convince a verifier of the existence of the problem's solution without revealing the solution itself. Typically, ZKP protocols are implemented using computers. However, ZKP protocols using physical objects such as playing cards are also widely studied. The first physical ZKP protocol was developed by Gradwohl et al. in 2007 for a pencil puzzle Sudoku [6]. Since then, physical ZKP protocols for many other puzzles have been proposed, including Sudoku [7, 8] and Kakuro [9, 10].

¹ This name was also generated by ChatGPT.

Most operations used in physical ZKP protocols come from previous work in *card-based cryptography*, a study in secure multi-party computation using cards. This area of study began with the protocols to compute the logical AND function by den Boar in 1989 [11] and the logical XOR function by Crepeau and Kilian in 1993 [12]. After their work, shuffling operations called *scramble shuffle*, *pile-shifting shuffle* [13], and *pile-scramble shuffle* [14] were proposed to compute more complex functions. These operations will be used in this paper.

1.3 Our Contribution

Neither an NP-completeness proof of Sumplete nor a ZKP protocol for Sumplete has been proposed before. In this paper, we prove that Sumplete is NP-complete in two ways. The NP-completeness of general Sumplete can be proven by a reduction from the Subset Sum problem (SSP). Moreover, even if the grid contains only two different integers, the NP-completeness still holds (which implies restricted Sumplete is NP-complete). This fact cannot be proven by the reduction from SSP. Instead of SSP, we show NP-completeness of the restricted sumplete by a reduction from the XSAT problem for 3-CNF_+^3 , which is a satisfiability problem for special logic circuits. Both the SSP and the XSAT problem for 3-CNF_+^3 are known to be NP-complete.

We also propose a physical card-based ZKP protocol for Sumplete. Because of the NP-completeness of the puzzle, it is worth developing a ZKP protocol for it.

The main difference from the conference version [1] is the addition of the second NP-completeness proof of Sumplete. The second proof is stronger than the first one; it shows that even restricted Sumplete is also NP-complete.

1.4 Organization of Paper

The rest of this paper is organized as follows. In Sect. 2, we introduce zero-knowledge proof and card operations. Section 3 is devoted to the NP-completeness proofs of Sumplete. In Sect. 4, we propose a physical ZKP protocol for Sumplete. Then, in Sect. 5, we show that our proposed protocol satisfies all ZKP properties. Finally, Sect. 6 concludes this paper.

2 Preliminaries

2.1 Notation

(Multi)sets are denoted by calligraphic uppercase letters, e.g., $\mathcal{A} = \{a_1, a_2, a_3\}$. In particular, let $\{\cdot\}$ denote a set without duplication, and $\{\{\cdot\}\}$ denote a multiset. Vectors are denoted by boldfaced lowercase letters, and the i -th element of a vector \mathbf{v} is denoted by v_i , e.g., $\mathbf{v} = (v_1, v_2, v_3)$. Matrices are denoted by boldfaced uppercase letters, and

the element of the i -th row and j -th column of a matrix \mathbf{M} is denoted by $m_{i,j}$, e.g.,

$$\mathbf{M} = \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix}.$$

2.2 Zero-Knowledge Proof








A zero-knowledge proof (ZKP) is an interactive protocol between a prover P and a verifier V . We assume that P is a probabilistic Turing machine with unbounded computational ability and V is a probabilistic polynomial-time Turing machine. Let x be an instance of an NP language. For a given x which has the witness, we suppose that P can calculate the witness from x , but V cannot. Note that x has the witness if and only if P knows the witness, as the computational ability of P is unbounded. In ZKP, P interacts with V and finally convinces V that the problem x has the witness without revealing the witness. ZKP protocols must achieve the following three properties.

Completeness. If P knows the witness, V is always convinced.

Soundness. If P does not know the witness, V is not convinced with more than negligible probability. The probability that V is convinced when P does not know the witness is called *soundness error*. If the soundness error is less than 1, it approaches asymptotically to 0 by executing proof many times. However, repeating physical protocols with human hands is hard. Therefore, it is desirable that the soundness error of physical zero-knowledge proof is 0.

Zero-Knowledge. V cannot obtain any information about the witness. Formally, for every V , there exists a probabilistic polynomial-time algorithm S that does not know the witness. If the output of S is indistinguishable from the output of the interaction of P and V , no information about the solution is leaked during the interaction.

2.3 Card Operations

In this paper, we use cards whose front side is either  or  and whose back side is . We assume that the front sides of cards with the same suit are identical. The back sides of all cards are also assumed to be indistinguishable. For understanding, we denote a face-down card  (resp., ) by  (resp., ).

2.3.1 Cyclic Shift

Let $\mathbf{c} := (c_1, c_2, \dots, c_k)$ be a sequence of k cards. A *left cyclic shift* over \mathbf{c} is an operation that outputs

$$(c_{\rho^{-1}(1)}, c_{\rho^{-1}(2)}, \dots, c_{\rho^{-1}(k)}),$$

where ρ is a cyclic permutation $\rho := (1 \ k \ k-1 \ \dots \ 3 \ 2)$.

Analogously, a *right cyclic shift* over \mathbf{c} is an operation that outputs

$$(c_{\sigma^{-1}(1)}, c_{\sigma^{-1}(2)}, \dots, c_{\sigma^{-1}(k)}),$$

where σ is a cyclic permutation $\sigma := (1\ 2\ 3 \dots k - 1\ k)$.

2.3.2 Scramble Shuffle

Let $\mathbf{c} := (c_1, c_2, \dots, c_k)$ be a sequence of k cards. A *scramble shuffle* over \mathbf{c} is an operation that outputs

$$(c_{\pi^{-1}(1)}, c_{\pi^{-1}(2)}, \dots, c_{\pi^{-1}(k)}),$$

where a permutation π is an element of S_k , the symmetric group of degree k . We assume that the permutation π is kept secret from every party.

2.3.3 Pile-Scramble Shuffle

Let $\mathbf{p} := (p_1, p_2, \dots, p_k)$ be a sequence of k piles of cards, with each pile having the same number of cards. A *pile-scramble shuffle* [14] over \mathbf{p} is an operation that outputs

$$(P_{\pi^{-1}(1)}, P_{\pi^{-1}(2)}, \dots, P_{\pi^{-1}(k)}),$$

where a permutation π is an element of S_k , the symmetric group of degree k . We assume that the permutation π is kept secret to every party.

2.4 Representation of an Integer

In this subsection, we will show how to represent an integer using cards. In card-based cryptography protocols, e.g., [15, 16], each integer from 1 to k is represented by a sequence of k cards, which consists of a \heartsuit and $k - 1$ \clubsuit s. Specifically, an integer i ($1 \leq i \leq k$) is represented by the position of the \heartsuit in the sequence: if the \heartsuit is at the i -th leftmost position, the sequence represents the integer i . For example, 1 and 3 are represented as follows.

$$\begin{aligned} 1 &= \heartsuit \clubsuit \clubsuit \clubsuit \clubsuit \\ 3 &= \clubsuit \clubsuit \heartsuit \clubsuit \clubsuit \end{aligned}$$

In this paper, we apply this method to represent any integer, including zero and negative integers.

Definition 1 (Integer Counter) Let α and β be positive integers. An integer i ($-\alpha \leq i \leq \beta$) is represented by a sequence of $\alpha + \beta + 1$ cards, consisting of one \heartsuit card and $\alpha + \beta$ \clubsuit cards. Specifically, if \heartsuit is at the i -th leftmost position of the sequence, the sequence represents the integer $-\alpha + i - 1$. We call such card sequence an *integer counter*.

For example, 0 and -2 are represented as follows when $\alpha = 3$ and $\beta = 2$.

$$\begin{aligned}
 0 &= \boxed{\clubsuit \clubsuit \clubsuit \heartsuit \clubsuit \clubsuit} \\
 -2 &= \boxed{\clubsuit \heartsuit \clubsuit \clubsuit \clubsuit \clubsuit}
 \end{aligned}$$

We execute the addition by shifting the counter. This method is used by Shinagawa et al. [13] and by Ruangwises and Itoh [17]. Suppose there is an integer counter representing $x \in \mathbb{Z}$, where the number of cards (i.e., the values α and β for the counter) is large enough so that \heartsuit does not overflow. Then, we can obtain the counter representing $x + y$ ($y \in \mathbb{Z}$) as follows: if $y < 0$, we execute the *left* cyclic shift over the card sequence $|y|$ times; otherwise (i.e., if $y \geq 0$), we execute the *right* cyclic shift over the card sequence $|y|$ times. Since these cyclic shift operations can be performed even if the cards of the counter are face-down, the addition of y can be performed without revealing the value of x .

3 NP-Completeness of Sumplete

In this section, we provide two NP-completeness proofs of Sumplete via polynomial-time reductions from NP-complete problems. In Sect. 3.1, we prove that general Sumplete is NP-complete via a reduction from SSP. In Sect. 3.2, we prove that restricted Sumplete is NP-complete via a reduction from XSAT for 3-CNF_+^3 .

3.1 General Sumplete

First, we will prove that general Sumplete is NP-complete. To do so, we will show that every SSP instance can be reduced to a Sumplete instance in polynomial time.

3.1.1 Formal Definitions of Problems

Before proving Sumplete’s NP-completeness, we formally define the decisional version of SSP and general Sumplete.

Definition 2 (Subset Sum Problem) The Subset Sum Problem (SSP) consists of a multiset $\mathcal{A} \subseteq \mathbb{Z}$ and an integer $N \in \mathbb{Z}$. The answer of the instance $\text{SSP} = (\mathcal{A}, N)$ is *Yes* if there exists a subset $\mathcal{A}' \subseteq \mathcal{A}$ that satisfies $\sum_{a \in \mathcal{A}'} a = N$; if there is no such \mathcal{A}' , the answer is *No*. We call such subset \mathcal{A}' a solution of the instance SSP.

For example, consider an instance $\text{SSP} = (\mathcal{A}, N)$ where $\mathcal{A} := \{-3, 3, 2, -7, 9\}$ and $N := -8$. The answer of this instance is *Yes* since there exists a solution $\mathcal{A}' = \{-3, 2, -7\} \subseteq \mathcal{A}$.

Definition 3 (General Sumplete) A (general) Sumplete instance consists of three elements: an $m \times n$ matrix $\mathbf{G} \in \mathbb{Z}^{m \times n}$ representing each number in the grid, a vector $\mathbf{r} \in \mathbb{Z}^m$ representing each row’s target value, and a vector $\mathbf{c} \in \mathbb{Z}^n$ representing each

column’s target value. The answer of the instance $S = (\mathbf{G}, \mathbf{r}, \mathbf{c})$ is *Yes* if there exists an $m \times n$ matrix $\hat{\mathbf{G}} \in \{0, 1\}^{m \times n}$ satisfying the following equations:

$$r_i = \sum_{j=1}^n g_{i,j} \hat{g}_{i,j} \quad \text{for all } i \in \{1, \dots, m\}, \tag{1}$$

$$c_j = \sum_{i=1}^m g_{i,j} \hat{g}_{i,j} \quad \text{for all } j \in \{1, \dots, n\}; \tag{2}$$

if there is no such $\hat{\mathbf{G}}$, the answer is *No*. We call such matrix $\hat{\mathbf{G}}$ a solution of the instance S .

For example, the puzzle in Fig. 1 can be represented as follows:

$$\mathbf{G} = \begin{pmatrix} 3 & 5 & 5 & 7 & 1 \\ 5 & 1 & 4 & 1 & 8 \\ 4 & 7 & 2 & 5 & 2 \\ 6 & 2 & 4 & 9 & 4 \\ 3 & 3 & 4 & 9 & 6 \end{pmatrix},$$

$$\mathbf{r} = (13 \ 14 \ 11 \ 6 \ 15),$$

$$\mathbf{c} = (11 \ 18 \ 11 \ 9 \ 10).$$

The answer of this instance is *Yes* since there exists the following solution $\hat{\mathbf{G}}$:

$$\hat{\mathbf{G}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

3.1.2 NP-Completeness Proof of General Sumplete

Theorem 1 *General Sumplete is NP-complete.*

Proof To prove the NP-completeness, we will show that the following two conditions hold.

- (1) Sumplete is in NP.
- (2) Sumplete is polynomial-time reductive from SSP.

First, we will prove (1) by showing the existence of a non-deterministic polynomial-time algorithm that can decide whether a given Sumplete instance is *Yes* or *No*. Consider a non-deterministic algorithm M that works as follows.

- 1. M chooses some cells non-deterministically.
- 2. M crosses out numbers in the chosen cells.

Fig. 2 A Sumplete instance constructed from an SSP instance

| | | | | |
|----------|----------|----------|-----------|----------|
| a_1 | a_2 | \dots | a_n | N |
| a_2 | a_3 | \dots | a_1 | N |
| \vdots | \vdots | \ddots | \vdots | \vdots |
| a_n | a_1 | \dots | a_{n-1} | N |
| N | N | \dots | N | |

3. For every row and column, M calculates the sum of uncrossed numbers and compares it with the target value. If there is a row or column whose sum is not equal to the target value, M rejects the instance; otherwise, M accepts.

Since each operation terminates in polynomial time, M halts in polynomial time, so (1) holds.

Next, We will prove (2) by showing that the following three conditions hold.

- (i) There exists a polynomial-time reduction f from SSP to Sumplete.
- (ii) For an arbitrary SSP instance, say SSP, if the answer of SSP is Yes, then the answer of the reduced Sumplete instance $S' := f(\text{SSP})$ is Yes.
- (iii) For an arbitrary SSP instance, SSP, if the answer of $S' := f(\text{SSP})$ is Yes, then the answer of SSP is Yes.

First, we will show that (i) holds. Consider the following polynomial-time reduction f . See Fig. 2.

- 1. f receives an SSP instance $\text{SSP} := (\mathcal{A}, N)$, where $\mathcal{A} := \{a_1, a_2, \dots, a_n\} \subseteq \mathbb{Z}$ and $N \in \mathbb{Z}$.
- 2. f outputs a Sumplete instance $S' := (\mathbf{G}', \mathbf{r}', \mathbf{c}')$, where \mathbf{G}' , \mathbf{r}' , and \mathbf{c}' are defined as follows:

$$\mathbf{G}' := \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ a_{\rho^{-1}(1)} & a_{\rho^{-1}(2)} & \dots & a_{\rho^{-1}(n)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\rho^{-(n-1)}(1)} & a_{\rho^{-(n-1)}(2)} & \dots & a_{\rho^{-(n-1)}(n)} \end{pmatrix},$$

$$\mathbf{r}' := (N \ N \ \dots \ N),$$

$$\mathbf{c}' := (N \ N \ \dots \ N),$$

where $\rho := (1 \ n \ n - 1 \ \dots \ 3 \ 2)$ is a cyclic permutation.

Note that all the target values in S' are N . In addition, each element of \mathcal{A} appears once for each row and column. During the reduction, $n^2 + 2n$ times of writing numbers and n times of shifting are executed, so the running time of f is polynomial in n . Therefore, (ii) holds.

Next, we will show that (ii) holds. Suppose the answer of SSP is Yes. From the solution \mathcal{A}' of SSP, construct a matrix $\hat{\mathbf{G}}' \in \{0, 1\}^{n \times n}$, whose (i, j) -element $\hat{g}'_{i,j}$ is defined as follows:

$$\hat{g}'_{i,j} = \begin{cases} 1, & \text{if } g'_{i,j} \in \mathcal{A}'; \\ 0, & \text{if } g'_{i,j} \notin \mathcal{A}'. \end{cases}$$

We will show that the following equations hold:

$$r'_i = \sum_{j=1}^n g'_{i,j} \hat{g}'_{i,j} \quad \text{for all } i \in \{1, \dots, n\}, \tag{1}'$$

$$c'_j = \sum_{i=1}^n g'_{i,j} \hat{g}'_{i,j} \quad \text{for all } j \in \{1, \dots, n\}. \tag{2}'$$

As the integers in i -th row of \mathbf{G}' are $a_{\rho^{-(i-1)}(1)}, a_{\rho^{-(i-1)}(2)}, \dots, a_{\rho^{-(i-1)}(n)}$, all elements of \mathcal{A} appear exactly once. Moreover, $\hat{g}'_{i,j}$ is 1 (resp., 0) when $g'_{i,j} \in \mathcal{A}'$ (resp., $g'_{i,j} \notin \mathcal{A}'$). Thus, for all $1 \leq i \leq n$ we have

$$\sum_{j=1}^n g'_{i,j} \hat{g}'_{i,j} = \sum_{a \in \mathcal{A}'} a = N = r'_i.$$

As the integers in j -th column of \mathbf{G}' are $a_j, a_{\rho^{-1}(j)}, \dots, a_{\rho^{-(n-1)}(j)}$, all elements of \mathcal{A} appear exactly once. Moreover, $\hat{g}'_{i,j}$ is 1 (resp., 0) when $g'_{i,j} \in \mathcal{A}'$ (resp., $g'_{i,j} \notin \mathcal{A}'$). Thus, for all $1 \leq j \leq n$ we have

$$\sum_{i=1}^n g'_{i,j} \hat{g}'_{i,j} = \sum_{a \in \mathcal{A}'} a = N = c'_j.$$

From above, equations (1)' and (2)' hold, which implies the answer of $f(\text{SSP})$ is Yes as it has $\hat{\mathbf{G}}'$ as a solution. Therefore, (ii) holds.

Finally, we will show that (iii) holds. Suppose the answer of $f(\text{SSP})$ is Yes. From the solution $\hat{\mathbf{G}}'$ of $f(\text{SSP})$, construct a multiset $\mathcal{A}' \subseteq \mathcal{A}$ as follows:

$$\mathcal{A}' = \{g'_{1,j} \mid \text{for } 1 \leq j \leq n, \hat{g}'_{1,j} = 1\}.$$

We will show that the equation $\sum_{a \in \mathcal{A}'} a = N$ holds. As mentioned above, in the first row of \mathbf{G}' , all elements of \mathcal{A} appear exactly once. From the definition of $\hat{\mathbf{G}}'$, we have

$$\sum_{j=1}^n g'_{1,j} \hat{g}'_{1,j} = r'_1 = N,$$

which means the summation of $g'_{1,j}$ such that $\hat{g}'_{1,j}$ is 1 is equal to N , the target value of SSP. Since \mathcal{A}' is exactly the set of $g'_{1,j}$ such that $\hat{g}'_{1,j}$ is 1, the equation $\sum_{a \in \mathcal{A}'} a = \sum_{j=1}^n g'_{1,j} \hat{g}'_{1,j} = N$ holds. Therefore, since the subset \mathcal{A}' can be regarded as the solution of SSP, (iii) holds.

From (i)–(iii), the condition (2) holds.

From (1) and (2), we can conclude that general Sumplete is NP-complete. □

3.2 Restricted Sumplete

It is more challenging to prove the NP-completeness of restricted Sumplete. In particular, the technique in Sect. 3.1 cannot be applied to restricted Sumplete, as SSP in the setting where each element in the set is bounded by a constant is solvable in linear time (e.g., by dynamic programming). Therefore, we have to develop a new technique to prove the NP-completeness of restricted Sumplete.

In this subsection, we will prove a stronger statement that Sumplete is NP-complete even if the grid contains only two different integers, p and $3p$, where p is an arbitrary nonzero integer. We call such instance a $(p, 3p)$ -Sumplete instance.

3.2.1 Formal Definitions of Problems

XSAT problem for 3-CNF_+^3 is the satisfiability problem for the special kind of logic circuits, 3-CNF_+^3 . 3-CNF_+^3 is a set of logic circuits where each clause contains exactly three positive literals, and each variable appears in exactly three clauses.

Definition 4 (3-CNF_+^3) Suppose that x_1, x_2, \dots, x_n denote n literals and $\#_\alpha(x_i)$ denotes the number of literal x_i in the logic circuit α . 3-CNF_+^3 is the following set.

$$3\text{-CNF}_+^3 = \left\{ \alpha = \bigwedge_{i=1}^n \bigvee_{j=1}^3 c_{i,j} \mid \begin{array}{l} (c_{i,j} \in \{x_1, x_2, \dots, x_n\}) \\ \wedge (\text{for } 1 \leq i \leq n, 1 \leq k, l \leq 3, k \neq l \Leftrightarrow c_{i,k} \neq c_{i,l}) \\ \wedge (\text{for } 1 \leq i \leq n, \#_\alpha(x_i) = 3) \end{array} \right\}$$

For $1 \leq i \leq n$, $c_{i,1} \vee c_{i,2} \vee c_{i,3}$ is called clause and denoted by C_i .

XSAT problem for 3-CNF_+^3 is a decision problem of whether there is a Boolean assignment of the circuit in 3-CNF_+^3 , in which every clause has exactly one literal that evaluates to true. This problem is known to be NP-complete [18].

Definition 5 (XSAT for 3-CNF_+^3) An instance of XSAT for 3-CNF_+^3 is an n -input logic circuit $\bigwedge_{i=1}^n C_i$ which belongs to 3-CNF_+^3 .

The answer of an XSAT instance is *Yes* if there exists a truth assignment that for arbitrary clause C_i , $\{c_{i,1}, c_{i,2}, c_{i,3}\}$ are assigned to $\{\{\text{TRUE}, \text{FALSE}, \text{FALSE}\}\}$; if there does not exist such truth assignment, the answer is *No*. We call such assignment a solution of the instance XSAT.

Definition 6 ($(p, 3p)$ -Sumplete) Let p be a nonzero integer. A $(p, 3p)$ -Sumplete instance consists of three elements: an $m \times n$ matrix $\mathbf{G} \in \{p, 3p\}^{m \times n}$ representing each number in the grid, a vector $\mathbf{r} \in \mathbb{Z}^m$ representing each row’s target value, and a vector $\mathbf{c} \in \mathbb{Z}^n$ representing each column’s target value. The answer of the instance $S = (\mathbf{G}, \mathbf{r}, \mathbf{c})$ is *Yes* if there exists an $m \times n$ matrix $\hat{\mathbf{G}} \in \{0, 1\}^{m \times n}$ satisfying the following equations:

$$r_i = \sum_{j=1}^n g_{i,j} \hat{g}_{i,j} \quad \text{for all } i \in \{1, \dots, m\}, \tag{3}$$

$$c_j = \sum_{i=1}^m g_{i,j} \hat{g}_{i,j} \quad \text{for all } j \in \{1, \dots, n\}; \tag{4}$$

if there is no such $\hat{\mathbf{G}}$, the answer is *No*. We call such matrix $\hat{\mathbf{G}}$ a solution of the instance RS.

3.2.2 NP-Completeness Proof of Restricted Sumplete

Since $(p, 3p)$ -Sumplete for $-6 \leq p \leq 6$ are special cases of restricted Sumplete, we can prove NP-completeness of restricted Sumplete by proving that of $(p, 3p)$ -Sumplete for arbitrary nonzero integer p . Note that we can prove that $(p, 3p)$ -Sumplete is in NP as well as we prove that for general Sumplete. Therefore, it is sufficient to prove that $(p, 3p)$ -Sumplete is NP-hard. We will do so by showing that it is polynomial reducible from 3-CNF_+^3 .

Theorem 2 For any given nonzero integer p , $(p, 3p)$ -Sumplete is NP-complete.

Proof Consider for any nonzero integer p . To prove the NP-completeness, we will show that the following three conditions hold.

- (i) There exists a polynomial-time reduction f' from XSAT for 3-CNF_+^3 to $(p, 3p)$ -Sumplete.
- (ii) For an arbitrary instance of XSAT for 3-CNF_+^3 , say XSAT, if the answer of XSAT is Yes, then the answer of the reduced $(p, 3p)$ -Sumplete instance $RS' := f(\text{XSAT})$ is Yes.
- (iii) For an arbitrary instance of XSAT for 3-CNF_+^3 , XSAT, if the answer of $RS' := f(\text{XSAT})$ is Yes, then the answer of XSAT is Yes.

First, we will show (i). Consider the following polynomial-time reduction f' . See Figs. 3 and 4.

1. f' receives an instance of XSAT for 3-CNF_+^3 $\text{XSAT} := \bigwedge_{i=1}^n \bigvee_{j=1}^3 c_{i,j}$, where $c_{i,j} \in \{x_1, x_2, \dots, x_n\}$.

$$\begin{array}{ll}
 C_1 = x_1 \vee x_2 \vee x_3 & x_1 = \text{FALSE} \\
 C_2 = x_2 \vee x_3 \vee x_6 & x_2 = \text{TRUE} \\
 C_3 = x_1 \vee x_4 \vee x_6 & x_3 = \text{FALSE} \\
 C_4 = x_2 \vee x_5 \vee x_6 & x_4 = \text{TRUE} \\
 C_5 = x_1 \vee x_4 \vee x_5 & x_5 = \text{FALSE} \\
 C_6 = x_3 \vee x_4 \vee x_5 & x_6 = \text{FALSE}
 \end{array}$$

Fig. 3 An XSAT problem for 3-CNF₊³ (left) and its solution (right)

2. f outputs a $(p, 3p)$ -Sumplete instance $S' := (\mathbf{G}', \mathbf{r}', \mathbf{c}')$, where \mathbf{G}' , \mathbf{r}' , and \mathbf{c}' are defined as follows:

$$\begin{aligned}
 \mathbf{G}' &:= \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \\ 3p & 3p & \cdots & 3p \end{pmatrix}, \\
 \mathbf{r}' &:= (p \ p \ \cdots \ p \ 2pn), \\
 \mathbf{c}' &:= (3p \ 3p \ \cdots \ 3p),
 \end{aligned}$$

where each $b_{i,j}$ is defined as follows:

$$b_{i,j} = \begin{cases} p, & \text{if } x_j \text{ appears in clause } C_i; \\ 3p, & \text{if } x_j \text{ does not appear in clause } C_i. \end{cases}$$

During the reduction, $n(n + 1) + n + (n + 1) = n^2 + 3n + 1$ times of writing numbers are executed, so the running time of f' is polynomial in n . Therefore, (i) holds.

Next, we will show that (ii) holds. Suppose the answer of XSAT is Yes. From the solution of XSAT, construct a matrix $\hat{\mathbf{G}}' \in \{0, 1\}^{(n+1) \times n}$, whose (i, j) -element $\hat{g}'_{i,j}$ is defined as follows:

$$\hat{g}'_{i,j} = \begin{cases} 1, & \text{if } \left((1 \leq i \leq n) \wedge (x_j \text{ is assigned to TRUE.}) \wedge (g'_{i,j} = p) \right) \\ & \vee \left((i = n + 1) \wedge (x_j \text{ is assigned to FALSE.}) \right); \\ 0, & \text{otherwise.} \end{cases}$$

We will show that the following equations hold:

$$r'_i = \sum_{j=1}^n g'_{i,j} \hat{g}'_{i,j} \quad \text{for all } i \in \{1, \dots, n\}, \tag{3}'$$

| | | | | | | |
|------|------|------|------|------|------|-------|
| p | p | p | $3p$ | $3p$ | $3p$ | p |
| $3p$ | p | p | $3p$ | $3p$ | p | p |
| p | $3p$ | $3p$ | p | $3p$ | p | p |
| $3p$ | p | $3p$ | $3p$ | p | p | p |
| p | $3p$ | $3p$ | p | p | $3p$ | p |
| $3p$ | $3p$ | p | p | p | $3p$ | p |
| $3p$ | $3p$ | $3p$ | $3p$ | $3p$ | $3p$ | $2pn$ |

| | | | | | | |
|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|-------|
| p | p | p | $3p$ | $3p$ | $3p$ | p |
| $3p$ | p | p | $3p$ | $3p$ | p | p |
| p | $3p$ | $3p$ | p | $3p$ | p | p |
| $3p$ | p | $3p$ | $3p$ | p | p | p |
| p | $3p$ | $3p$ | p | p | $3p$ | p |
| $3p$ | $3p$ | p | p | p | $3p$ | p |
| $3p$ | $3p$ | $3p$ | $3p$ | $3p$ | $3p$ | $2pn$ |

$3p \quad 3p \quad 3p \quad 3p \quad 3p \quad 3p$ $3p \quad 3p \quad 3p \quad 3p \quad 3p \quad 3p$

Fig. 4 A $(p, 3p)$ -Sumplace instance constructed from the XSAT problem in Fig. 3 (left) and its solution (right)

$$c'_j = \sum_{i=1}^n g'_{i,j} \hat{g}'_{i,j} \quad \text{for all } j \in \{1, \dots, n\}. \tag{4}'$$

First, we will show that (4)' holds. For each j ($1 \leq j \leq n$), since x_j appears in exactly three clauses, in the j -th column of \mathbf{G}' there must be exactly three elements in the first n row being p , with the rest being $3p$. Suppose p appears in the $d_{j,1}, d_{j,2}$, and $d_{j,3}$ -th row.

If x_j is assigned to TRUE, the $(d_{j,1}, j)$, $(d_{j,2}, j)$ and $(d_{j,3}, j)$ -elements are 1 and all other elements are 0. Thus,

$$\sum_{i=1}^{n+1} g'_{i,j} \hat{g}'_{i,j} = \sum_{k=1}^3 g'_{d_{j,k},j} = \sum_{k=1}^3 p = 3p = c'_j.$$

On the other hand, if x_j is assigned to FALSE, then $(n + 1, j)$ -element is 1 and all other elements are 0. Thus,

$$\sum_{i=1}^{n+1} g'_{i,j} \hat{g}'_{i,j} = g'_{n+1,j} = 3p = c'_j.$$

Therefore, (4)' holds for all $1 \leq j \leq n$.

Next, we will show that (3)' holds. Recall that for each $1 \leq i, j \leq n$, the (i, j) -element of \mathbf{G}' is p (resp., $3p$) when x_j appears (resp., does not appear) in C_i . For each i ($1 \leq i \leq n$), since exactly one literal in C_i is assigned to TRUE, there is exactly one 1 in the i -th row of $\hat{\mathbf{G}}'$, and its corresponding element of \mathbf{G}' is p . Thus, we have

$$\sum_{j=1}^n g'_{i,j} \hat{g}'_{i,j} = p = r'_i.$$

Now consider the sum of the $(n + 1)$ -st row of \mathbf{G}' . From the obtained equation and (4)', we have

$$\begin{aligned} \sum_{j=1}^n g'_{n+1,j} \hat{g}'_{n+1,j} &= \sum_{j=1}^n \left(\sum_{i=1}^{n+1} g'_{i,j} \hat{g}'_{i,j} \right) - \sum_{i=1}^n \left(\sum_{j=1}^n g'_{i,j} \hat{g}'_{i,j} \right) \\ &= n \cdot 3p - n \cdot p \\ &= 2pn = r'_{n+1}. \end{aligned}$$

Therefore, (3)' holds for all $1 \leq i \leq n + 1$.²

From (3)' and (4)', the answer of $f(\text{XSAT})$ is Yes as it has $\hat{\mathbf{G}}'$ as a solution. Therefore, (ii) holds.

Finally, we will show that (iii) holds. Suppose the answer of $f(\text{XSAT})$ is Yes. From the solution $\hat{\mathbf{G}}'$ of $f(\text{XSAT})$, construct an assignment T for XSAT, which is defined as follows:

$$x_j = \begin{cases} \text{TRUE}, & \text{if there are exactly three 1s in the } j\text{-th column of } \hat{\mathbf{G}}'; \\ \text{FALSE}, & \text{if there is exactly one 1 in the } j\text{-th column of } \hat{\mathbf{G}}'. \end{cases}$$

We will show that exactly one literal is assigned to TRUE in every clause of XSAT. For each i ($1 \leq i \leq n$), since the target value of i -th row of $f(\text{XSAT})$ is p , exactly one element (let it be the (i, e_i) -element) in the i -th row of $\hat{\mathbf{G}}'$ is 1, and its corresponding element of \mathbf{G}' is p . This means x_{e_i} appears in C_i and is assigned to TRUE, and all other literals in C_i are assigned to FALSE. Thus, only one literal in C_i is assigned to TRUE. For each j ($1 \leq j \leq n$), since the target value of j -th column of $f(\text{XSAT})$ is $3p$, if there exists one 1 in j -th column of $\hat{\mathbf{G}}'$ (let it be the (e_j, j) -element) and its corresponding element of \mathbf{G}' is p , then there exists two 1s in j -th column of $\hat{\mathbf{G}}'$ other than the (e_j, j) -element and their corresponding elements of \mathbf{G}' are p . This means that x_j appears three times in XSAT and all three x_j are assigned to the same value. Thus, since the assignment T can be regarded as the solution of XSAT, the condition (iii) holds.

From (i)–(iii), there exists a polynomial reduction from XSAT problem for 3-CNF^3_+ to $(p, 3p)$ -Sumplete.

Hence, we can conclude that $(p, 3p)$ -Sumplete is NP-complete for any given nonzero integer p . □

² In $(n + 1)$ -st row, the equation (3)' holds only when n is a multiple of 3. Since our proof assumes that the answer of XSAT is Yes, the equation (3)' always holds in $(n + 1)$ -st row. It is because if the answer of XSAT is Yes, then n is a multiple of 3. Since there exists three x_j ($1 \leq j \leq n$) in XSAT, there exists $3n$ literals in XSAT. From the condition of the solution of XSAT problem for 3-CNF^3_+ , if the answer of XSAT is Yes, then n out of $3n$ literals are assigned to TRUE. Since all three x_j must be assigned to the same value, if it is not a multiple of 3, then there does not exist the assignment that can be regarded as the solution of XSAT. This means that if n is not a multiple of 3, the answer of XSAT is No. From the contraposition, the statement “if the answer of XSAT is Yes, then n is a multiple of 3” holds.

Recall that in restricted sumplete, the integer of cells, say x , is constrained to be $-20 < |x| < 20$. Since (1, 3)-Sumplete (where $p = 1$) is a special case of restricted Sumplete, restricted Sumplete is also NP-complete.

4 Physical ZKP Protocol for Sumplete

4.1 Idea of Proposed Protocol

In this subsection, we will construct a physical ZKP protocol for general Sumplete (which clearly works for restricted Sumplete and $(p, 3p)$ -Sumplete as well). In our proposed protocol, the prover represents the solution of a Sumplete instance using cards, and calculates the sum of uncrossed numbers in each row and column of the grid using an integer counter. Then, the verifier checks whether the sum of uncrossed numbers in each row and column is equal to the target value. If the sum is equal to the target value for every row and column, the verifier accepts.

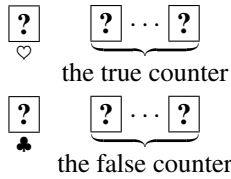
Decoy Technique

To realize the above protocol without revealing the solution, i.e., the locations of cells whose numbers are crossed out, we prepare two integer counters called a *true counter* and a *false counter* for each row and column. The true (resp., false) counter is used to calculate the sum of uncrossed (resp., crossed) numbers in a row or column. For each number in a row or column, the prover adds it to either the true or false counter depending on its status in the solution: if it is uncrossed (resp., crossed), it is added to the true (resp., false) counter. By using a technique called *decoy technique*, we can add integers while hiding the solution, i.e., which counter they were added to. Similar techniques are widely seen in physical cryptography, e.g., [19, 20].

4.2 Proposed Protocol

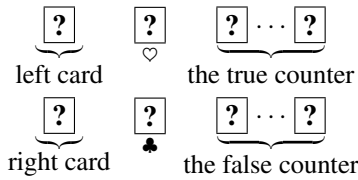
The proposed protocol proceeds as follows.

1. On each cell, the prover places a pair of face-down cards $\begin{matrix} \boxed{?} & \boxed{?} \\ \heartsuit & \clubsuit \end{matrix}$ if the number in that cell is uncrossed or $\begin{matrix} \boxed{?} & \boxed{?} \\ \clubsuit & \heartsuit \end{matrix}$ if the number in that cell is crossed out.
2. For each row and column, the prover and the verifier execute the following operations.
 - (a) The prover computes α (resp., β), an absolute value of the sum of all negative (resp., positive) integers in that row or column. The prover also makes a true counter and a false counter, which can represent an integer i ($-\alpha \leq i \leq \beta$). Then, the prover places the false counter below the true counter and the verifier checks that both counters indicate 0.
 - (b) The prover places \heartsuit on the left of the true counter and \clubsuit on the left of the false counter. Then, the prover makes all the cards face-down. We call these two sequences of cards a *card matrix*.

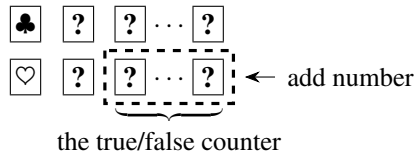


(c) For each cell in the row or column, the prover and the verifier execute the following operations.

- (i) The prover picks a pair of cards on the cell and places the left (resp., right) card of the pair to the left of the upper (resp., lower) sequence of the card matrix made in Step 2(b).



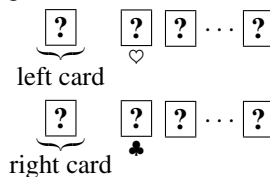
- (ii) The prover regards each of the upper and lower rows of the matrix made in Step 2(c)i as a pile and applies the pile-scramble shuffle to these two piles.
- (iii) The prover turns over the leftmost card of each row and adds the number in the cell to the counter in the row whose leftmost card is \heartsuit . Note that this addition must be executed by keeping the cards off the counter face-down.



- (iv) The prover turns all cards face-down, regards each of the upper and lower rows as a pile, and applies the pile-scramble shuffle in the same way as in Step 2(c)ii.
- (v) The verifier opens the second leftmost card of each row and swaps the two rows (if necessary) such that the row having an opened \heartsuit becomes the upper row.



- (vi) The prover turns all cards face-down and returns the pair of leftmost cards of both rows to their original cells.



- (d) The verifier turns over the leftmost card of each row and reveals the counter in the row whose leftmost card is \heartsuit . If the value indicated by the counter is not equal to the target value, the verifier rejects the instance. If the verifier does not reject the instance, the prover applies scramble shuffle to both truth and decoy counter. After the scramble shuffles, the prover opens all the counters and resets the value to 0.

3. If the verifier did not reject the instance for all rows and columns, the verifier accepts.

4.3 Decoy Technique

The decoy technique is used in Steps 2(c)ii and 2(c)iii. In Step 2(c)iii, the counter in the same row as \heartsuit is the true (resp., false) counter if the number is uncrossed (resp., crossed). Because of the pile-scramble shuffle in Step 2(c)ii, the location of \heartsuit in the leftmost column is independent of the card matrix made in Step 2(c)i. Therefore, the verifier does not know which counter appears on the right of \heartsuit .

4.4 Numbers of Cards and Shuffles

Consider the number of cards used in our protocol. In Step 1, the prover uses mn pairs of \heartsuit and \clubsuit cards. In Step 2, the prover and the verifier check m rows and n columns. Let k be the max value of $\alpha + \beta$ in each row and column, where α and β are those in Step 2(a). Then, Step 2(a) requires one \heartsuit card and at most k \clubsuit cards for each counter. In addition, Step 2(b) needs one \heartsuit card and one \clubsuit card. Since we can reuse the cards required for Steps 2(a) and 2(b), it is sufficient to consider the maximum number of cards required for the counter within the $m + n$ checks. Therefore, we need 3 \heartsuit cards and $2k + 1$ \clubsuit cards in Step 2.

In total, our protocol uses $2mn + 2k + 4 = O(mn + k)$ cards ($mn + 3$ \heartsuit cards and $mn + 2k + 1$ \clubsuit cards). As the input size is measured by m , n , and $\log_2 k$ (the grid size and the binary size of the integers in the grid), the number of cards is actually pseudo-polynomial (exponential in terms of $\log_2 k$). Therefore, we need to reduce the number of cards further.

We can lower the number of cards to polynomials of m , n , and $\log_2 k$ by representing each integer in two’s complements. By encoding 0 and 1 as $\clubsuit\heartsuit$ and $\heartsuit\clubsuit$, respectively, integers $(10)_{10} = (01010)_2$ and $(-6)_{10} = (11010)_2$ digits can be encoded as follows.

$$\begin{aligned}
 (10)_2 &= \heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit \\
 (-6)_2 &= \heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit
 \end{aligned}$$

The addition of the counter to the number of cells in two’s complements can be executed by card-based half-adder and card-based AND protocol with two additional cards [21]. When using two’s complements, we need $2\lceil\log_2 k\rceil \cdot 2$ cards for two

counters instead of $2k + 2$ cards, additional $2\lceil \log_2 k \rceil$ cards for the number of cells, and additional 4 cards for the addition operation. Hence, the total number of cards is $2mn + 6\lceil \log_2 k \rceil + 6$.

Next, consider the number of shuffles in our protocol. Since pile-scramble shuffles are executed in Steps 2(c)ii and 2(c)iv, $2n$ (resp., $2m$) shuffles are executed in Step 2(c) for each row (resp., column). Moreover, one scramble shuffle is applied to each of the true and false counters in Step 2(d). Also, both of Step 2(c) and step 2(d) are executed m times for all rows and n times for all columns in Step 2. In total, our protocol requires $m \cdot (2n + 2) + n \cdot (2m + 2) = 4mn + 2m + 2n$ shuffles. When using two's complements, we need additional $7\lceil \log_2 k \rceil$ shuffles for each addition operation. Since the addition operation is executed for $2mn$ times in the proposed protocol, the total number of shuffles is $14mn\lceil \log_2 k \rceil + 4mn + 2m + 2n$.

5 Proof of Security

In this section, we will show that our protocol satisfies the three properties of ZKP.

5.1 Completeness

Lemma 1 *If the prover knows the solution, then the verifier always accepts.*

Proof If the prover knows the solution, then the sum of the numbers in cells where two cards $\boxed{\heartsuit \clubsuit}$ are placed in this order is equal to the target value for all rows and columns. Therefore, the verifier does not reject the instance for all rows and columns. Hence, the verifier always accepts. □

5.2 Soundness

Lemma 2 *If the prover does not know the solution, then the verifier always rejects the instance, i.e., the soundness error is 0.*

Proof We will prove a contraposition of this statement: if the verifier accepts, then the prover knows the solution of the given Sumplete instance.

If the verifier accepts, then for every row and column, the sum of cells where $\boxed{\heartsuit \clubsuit}$ are placed is equal to the target value. Consider the $m \times n$ matrix $\hat{\mathbf{G}} \in \{0, 1\}^{m \times n}$ such that

$$\hat{g}_{i,j} = \begin{cases} 1 & (\boxed{\heartsuit \clubsuit} \text{ are placed in } (i, j)\text{-cell.}) \\ 0 & (\boxed{\clubsuit \heartsuit} \text{ are placed in } (i, j)\text{-cell.}) \end{cases}$$

$\hat{\mathbf{G}}$ is the solution of given instance of Sumplete. Thus, the prover knows the solution $\hat{\mathbf{G}}$. Since the contraposition holds, the soundness error is 0. □

5.3 Zero-Knowledge

Lemma 3 *During the protocol, the verifier learns nothing about the solution.*

Proof To prove the zero-knowledge property, it is sufficient to show that all distributions of cards opened during the protocol execution can be simulated without the solution.

- In Step 2(c)iii, we open the leftmost column in the card matrix. Before the operation, we apply the pile-scramble shuffle to the two piles of the upper row and the lower row of the matrix. Thus, \heartsuit appears at each row with probability $1/2$. Therefore, the distribution of cards opened in Step 2(c)iii can be simulated without the solution.
- In Step 2(c)v, we open the second leftmost column in the card matrix. Before the operation, we apply the pile-scramble shuffle to the two piles of the upper row and the lower row of the matrix. Thus, \heartsuit appears at each row with the probability $1/2$. Therefore, the distribution of cards opened in Step 2(c)v can be simulated without the solution.
- In Step 2(d), we open the true counter after adding numbers in the row or column. If the prover knows the solution, the value represented by the true counter is equal to the target value of that row or column. Therefore, the distribution of the true counter's cards opened in Step 2(d) can be simulated without the solution.

Hence, the verifier learns nothing about the solution. \square

6 Conclusion

We proved that Sumplete, a puzzle generated by ChatGPT, is NP-complete and proposed a physical ZKP protocol for it. In our ZKP protocol, we realized the addition of not only positive integers but also negative integers by expanding the usual technique. Moreover, we use the decoy technique to conceal the solution from the verifier.

Acknowledgements This work was supported by JSPS KAKENHI Grant Numbers JP22KJ1362 and JP23KJ0968.

Data availability No data was used for the research described in the article.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Hatsugai, K., Asano, K., Abe, Y.: A physical zero-knowledge proof for Sumplete, a puzzle generated by chatgpts. In: COCOON (2023)
2. OpenAI: GPT-4 Technical Report (2023)
3. Puzzled Penguin: ChatGPT invented its own puzzle game. <https://puzzledpenguin.substack.com/p/chatgpt-invented-its-own-puzzle-game> (2023)
4. Hey, Good Game: Sumplete. <https://sumplete.com/>
5. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
6. Gradwohl, R., Naor, M., Pinkas, B., Rothblum, G.N.: Cryptographic and physical zero-knowledge proof systems for solutions of Sudoku puzzles. *Theory Comput. Syst.* **44**(2), 245–268 (2009)
7. Ruangwises, S.: Two standard decks of playing cards are sufficient for a ZKP for sudoku. *New Gener. Comput.* **40**(1), 49–65 (2022)
8. Sasaki, T., Miyahara, D., Mizuki, T., Sone, H.: Efficient card-based zero-knowledge proof for sudoku. *Theor. Comput. Sci.* **839**, 135–142 (2020)
9. Bultel, X., Dreier, J., Dumas, J., Lafourcade, P.: Physical zero-knowledge proofs for Akari, Takuzu, Kakuro and KenKen. In: FUN, vol. 49, pp. 8–1820 (2016)
10. Miyahara, D., Sasaki, T., Mizuki, T., Sone, H.: Card-based physical zero-knowledge proof for Kakuro. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **102-A**(9), 1072–1078 (2019)
11. Boer, B.: More efficient match-making and satisfiability: the five card trick. In: EUROCRYPT, vol. 434, pp. 208–217 (1989)
12. Crépeau, C., Kilian, J.: Discreet solitary games. In: CRYPTO, pp. 319–330 (1993)
13. Shinagawa, K., Mizuki, T., Schuldt, J.C.N., Nuida, K., Kanayama, N., Nishide, T., Hanaoka, G., Okamoto, E.: Card-based protocols using regular polygon cards. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **100-A**(9), 1900–1909 (2017)
14. Ishikawa, R., Chida, E., Mizuki, T.: Efficient card-based protocols for generating a hidden random permutation without fixed points. In: UCNC, vol. 9252, pp. 215–226 (2015)
15. Miyahara, D., Hayashi, Y., Mizuki, T., Sone, H.: Practical card-based implementations of Yao’s millionaire protocol. *Theor. Comput. Sci.* **803**, 207–221 (2020)
16. Takashima, K., Abe, Y., Sasaki, T., Miyahara, D., Shinagawa, K., Mizuki, T., Sone, H.: Card-based protocols for secure ranking computations. *Theor. Comput. Sci.* **845**, 122–135 (2020)
17. Ruangwises, S., Itoh, T.: How to physically verify a rectangle in a grid: a physical ZKP for Shikaku. In: FUN, pp. 24–12412 (2022)
18. Porschen, S., Schmidt, T., Specknmeyer, E., Wotzlaw, A.: XSAT and NAE-SAT of linear CNF classes. *Discrete Appl. Math.* **167**, 1–14 (2014)
19. Mizuki, T., Sone, H.: Six-card secure AND and four-card secure XOR. In: FAW, pp. 358–369 (2009)
20. Nakai, T., Tokushige, Y., Misawa, Y., Iwamoto, M., Ohta, K.: Efficient card-based cryptographic protocols for millionaires’ problem utilizing private permutations. In: CANS, pp. 500–517 (2016)
21. Nishida, T., Hayashi, Y.-i., Mizuki, T., Sone, H.: Card-based protocols for any Boolean function. In: Theory and Applications of Models of Computation: 12th Annual Conference, TAMC 2015, Singapore, 18–20 May 2015, Proceedings 12, pp. 110–121. Springer (2015)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.