



# Card-Based Cryptographic Protocols with Malicious Players Using Private Operations

Yoshifumi Manabe<sup>1</sup> · Hibiki Ono<sup>1</sup>

Received: 18 August 2021 / Accepted: 15 December 2021 / Published online: 3 February 2022  
© The Author(s) 2022

## Abstract

This paper shows new card-based cryptographic protocols using private operations that are secure against malicious players. Physical cards are used in card-based cryptographic protocols instead of computers. Operations that a player executes in a place where the other players cannot see are called private operations. Using several private operations, calculations of two variable Boolean functions and copy operations were realized with the minimum number of cards. Though private operations are very powerful in card-based cryptographic protocols, there is a problem that it is very hard to prevent malicious actions during private operations. Though most card-based protocols are discussed in the semi-honest model, there might be cases when the semi-honest model is not enough. Thus, this paper shows new protocols that are secure against malicious players. We show logical XOR, logical AND,  $n$ -variable Boolean function, and copy protocols. We can execute any logical computations with a combination of these protocols. We use envelopes as an additional tool that can be easily prepared and used by people.

**Keywords** Multi-party secure computation · Card-based cryptographic protocols · Private operations · Logical computations · Copy · Malicious model

---

Preliminary version of the paper was presented as Yoshifumi Manabe and Hibiki Ono: “Secure Card-based Cryptographic Protocols Using Private Operations Against Malicious Players,” Proc. of 13th International Conference on Information Technology and Communications Security (SecITC 2020), LNCS Vol. 12596, pp.55–70 (Nov. 2020).

---

✉ Yoshifumi Manabe  
manabe@cc.kogakuin.ac.jp

<sup>1</sup> Faculty of Informatics, Kogakuin University, 1-24-2, Nishisinjuku, Shinjuku, Tokyo 163-8677, Japan

## Introduction

Card-based cryptographic protocols [15, 34, 36] were proposed in which physical cards are used instead of computers to securely calculate values. They can be used when computers cannot be used or users cannot trust the software on the computer. Also, the protocols are easy to understand; thus, the protocols can be used to teach the basics of cryptography [4, 30]. den Boer [2] first showed a five-card protocol to securely calculate logical AND of two inputs. Since then, many protocols have been proposed to realize primitives to calculate any logical functions [14, 18, 24, 37, 42, 48, 49, 62, 63] and specific computations such as a specific class of logical functions [1, 7, 13, 19, 23, 25, 31, 33, 43, 46, 54, 58, 61, 68], universal computation such as Turing machines [6, 16], millionaires' problem [27, 40, 47], voting [32, 41, 44, 69, 70], random permutation [8, 10, 11, 39], grouping [9], ranking [66], lottery [64], proof of knowledge of a puzzle solution [3, 5, 12, 21, 26, 28, 29, 50–53, 55–57, 59], and so on. This paper considers calculations of logical functions and a copy operation under the malicious model since any logical function can be realized with a combination of these calculations.

Operations that a player executes in a place where the other players cannot see are called private operations. These operations are considered to be executed under the table or in the back. Private operations are shown to be the most powerful primitives in card-based cryptographic protocols. They were first introduced to solve millionaires' problem [40]. Using three private operations called private random bisection cuts, private reverse cuts, and private reveals, committed-input and committed-output logical AND, logical XOR, and copy protocols can be achieved with the minimum number of cards [48]. Another class of private operations is private input operations that are used when a player inputs a private value [20, 47, 65]. These operations are not discussed in this paper since it is impossible to prevent false input from a malicious player. If the input values are honestly given, the players can use the protocols shown in this paper.

The largest problem of protocols using private operations is malicious actions. Most of the card-based protocols assume the semi-honest model, in which the players obey the rule of the protocols but try to obtain private information. However, there are many cases when we must consider the malicious model. When we allow malicious actions, protocols using private operations are not secure. Since private operations are executed where the other player cannot see, any malicious operation is possible during the private operations, for example, watching the marks of face-down cards or changing the positions of cards.

One countermeasure to malicious actions is setting a watch person. When the protocols are executed by more than two players, it is possible to detect malicious actions by the following rule: whenever a player executes a private operation, another player watches the execution and reports incorrect behavior. The XOR, AND, and copy protocols can be executed securely against a malicious player when the protocols are executed by more than two players [48]. However, when the protocols are executed by two players, Alice and Bob, it is impossible to use the above method. If Bob watches Alice's private operations, Bob knows all

operations; thus, the relation between input data and output data is known to Bob. When the output card is opened, the secure input data are known to Bob using the relation between the input data and the output data.

Thus we need new protocols for the two-player case. Since Bob cannot watch Alice's private operations, some additional mechanism is necessary to prevent illegally watching the marks of face-down cards during private operations. This paper introduces envelopes to prevent illegally watching the marks of face-down cards. Cards that must not be seen are publicly put into an envelope. If an envelope is illegally opened, it can be detected by anyone. Envelopes are used in [38] to realize cryptographic protocols that do not use physical cards. In card-based cryptographic protocols, envelopes are used in [8, 45, 58, 63] to realize some kind of shuffles that are not easy to be executed by people.

This paper shows new card-based cryptographic protocols that are secure against malicious players using envelopes as an additional tool. Some malicious actions during private operations are prevented by adding extra cards for error correction. We show logical XOR, logical AND,  $n$ -variable Boolean function, and copy protocols since any logical functions can be obtained with a combination of these protocols.

As related works, protocols that use additional cards and prevent active attacks while a player executes a shuffle were shown [17]. The private operations used in the protocol are private shuffles; thus, the method does not consider the other types of private operations. This paper considers the other types of malicious actions in the protocols that use private operations. Another type of active attack is inputting a false value that is not 0 or 1. A protocol to detect such injection attacks was discussed in [35]. This paper assumes correct inputs since we need to consider input  $\bar{x}$  instead of correct input  $x$  but it is impossible to prevent or detect.

Protocols that prevent revealing face-down cards were discussed in [67]. The protocol uses the technique of secret-sharing to prevent information leakage by opening some numbers of cards. The protocol cannot be applied to the problem discussed in this paper since a malicious player might reveal all cards during a private operation.

Another usage of private operations is realizing a public shuffle by multiple private shuffles [37]. The protocols cannot be used as it is to solve the problem in this paper since a malicious player might not honestly execute a private shuffle. Preventing malicious actions for the protocols that use private random bisection cuts, private reverse cuts, and private reveals are not considered.

A protocol to detect malicious actions by executing two instances of a protocol and comparing the results was shown [60]. The protocol uses cases to prevent revealing face-down cards. The functionality of cases is just the same as envelopes in this paper. The protocol uses twice as many cards as the original protocols and it is impossible to correct some malicious actions. This paper's protocols use a smaller number of cards and can correct some malicious actions.

In Sect. 2, basic notations, the private operations introduced in [48], and notations related to the envelopes are shown. Section 3 shows the security model. Section 4 shows new protocols to prevent or detect malicious operations. Section 5 concludes the paper. The difference from the conference version [22] is as follows: (1) We updated all protocols to clarify the procedure to detect malicious actions. (2) We added a new protocol to calculate any  $n$ -variable Boolean functions. (3) We added formal descriptions

of the protocols that output multiple copies of the output value and obtain multiple copies of an input value.

## Preliminaries

### Basic Notations

This section gives the notations and basic definitions of card-based protocols. This paper is based on a standard two-color card model. In the two-color card model, there are two kinds of marks,  $\clubsuit$  and  $\heartsuit$ . Cards of the same marks cannot be distinguished. In addition, the back of both types of cards is  $\square$ . It is impossible to determine the mark on the back of a given card of  $\square$ .

One-bit data are represented by two cards as follows:  $\square\clubsuit\square\heartsuit = 0$  and  $\square\heartsuit\square\clubsuit = 1$ .

One pair of cards that represents one bit  $x \in \{0, 1\}$ , whose face is down, is called a commitment of  $x$ , and denoted as  $commit(x)$ . It is written as  $\underbrace{\square\square}_{x}$ . For a pair of cards, exchanging the position of the left card and the right card is called a swap. Note that when two cards of  $commit(x)$  are swapped,  $commit(\bar{x})$  can be obtained. Thus, logical negation can be calculated without private operations.

A set of cards placed in a row is called a sequence of cards. A sequence of cards  $S$  whose length is  $n$  is denoted as  $S = s_1, s_2, \dots, s_n$ , where  $s_i$  is  $i$ -th card of the sequence.  $S = \underbrace{\square}_{s_1} \underbrace{\square}_{s_2} \underbrace{\square}_{s_3} \dots \underbrace{\square}_{s_n}$ . A sequence whose length is even is

called an even sequence.  $S_1||S_2$  is a concatenation of sequence  $S_1$  and  $S_2$ . For two sequences of cards  $S_1$  and  $S_2$  ( $|S_1| = |S_2|$ ) that are put left and right, swapping  $S_1$  and  $S_2$  means exchanging the position of the two sequences.

All protocols are executed by two players, Alice and Bob. The players might be malicious, that is, they might not obey the rule of the protocol. There is no collusion between Alice and Bob, otherwise private input data can be easily revealed.

Throughout this paper, we assume that each input is given as a committed value. The output must also be given as a committed value so that the output can be used as an input to further computations. Though some multi-party secure calculation protocols assume that each player knows his/her private input, there are some cases when we cannot assume that. For example, suppose that  $x_1, x_2$  are Alice's private input values and  $y_1, y_2$  are Bob's private input values and they want to securely calculate  $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$ . After  $commit(x_1 \vee y_1)$  and  $commit(x_2 \vee y_2)$  are calculated, they need to calculate logical AND of two secret values. Thus, we need to calculate the logical functions of two committed inputs. Note that committed-input protocols can be used even when a player knows some inputs. For example, if Alice knows input value  $x$ , she first commits her input as  $commit(x)$ . Then the committed-input protocol can be executed.

### Private Operations

We show three private operations introduced in [48]: private random bisection cuts, private reverse cuts, and private reveals.

**Primitive 1** (Private random bisection cut) A private random bisection cut is the following operation on an even sequence  $S_0 = s_1, s_2, \dots, s_{2m}$ . A player selects a random bit  $b \in \{0, 1\}$  and outputs

$$S_1 = \begin{cases} S_0 & \text{if } b = 0 \\ s_{m+1}, s_{m+2}, \dots, s_{2m}, s_1, s_2, \dots, s_m & \text{if } b = 1. \end{cases}$$

The player executes this operation in a place where the other players cannot see. The player must not disclose the bit  $b$ .

Note that if the private random cut is executed when  $m = 1$  and  $S_0 = \text{commit}(x)$ , given  $S_0 = \underbrace{\boxed{?} \boxed{?}}_x$ , the player’s output  $S_1 = \underbrace{\boxed{?} \boxed{?}}_{x \oplus b}$ , which is  $\underbrace{\boxed{?} \boxed{?}}_x$  or  $\underbrace{\boxed{?} \boxed{?}}_{\bar{x}}$ .

We sometimes write the result of the private random bisection cut using a bit  $b$  to a sequence  $S_1 || S_2$  (where  $|S_1| = |S_2|$ ) as  $\text{swap}(b, S_1 || S_2)$ .  $\text{swap}(0, S_1 || S_2) = S_1 || S_2$  and  $\text{swap}(1, S_1 || S_2) = S_2 || S_1$  are satisfied. Note that  $\text{swap}(b, \text{swap}(b', S_1 || S_2)) = \text{swap}(b \oplus b', S_1 || S_2)$  holds for any  $b$  and  $b' \in \{0, 1\}$ .

**Primitive 2** (Private reverse cut, Private reverse selection) A private reverse cut is the following operation on an even sequence  $S_2 = s_1, s_2, \dots, s_{2m}$  and a bit  $b \in \{0, 1\}$ . A player outputs

$$S_3 = \begin{cases} S_2 & \text{if } b = 0 \\ s_{m+1}, s_{m+2}, \dots, s_{2m}, s_1, s_2, \dots, s_m & \text{if } b = 1. \end{cases}$$

The player executes this operation in a place where the other players cannot see. The player must not disclose  $b$ .

Note that the bit  $b$  is not newly selected by the player. This is the difference between the primitive in Primitive 1, where a random bit must be newly selected by the player.

Note that in some protocols below, selecting left  $m$  cards is executed after a private reverse cut. The sequence of these two operations is called a private reverse selection. A private reverse selection is the following procedure on an even sequence  $S_2 = s_1, s_2, \dots, s_{2m}$  and a bit  $b \in \{0, 1\}$ . A player outputs

$$S_3 = \begin{cases} s_1, s_2, \dots, s_m & \text{if } b = 0 \\ s_{m+1}, s_{m+2}, \dots, s_{2m} & \text{if } b = 1. \end{cases}$$

**Primitive 3** (Private reveal) A player privately opens a given committed bit. The player must not disclose the obtained value.

Using the obtained value, the player privately sets a sequence of cards.

Consider the case when Alice executes a private random bisection cut on  $commit(x)$  and Bob executes a private reveal on the bit. Since the committed bit is randomized by the bit  $b$  selected by Alice, the opened bit is  $x \oplus b$ . Even if Bob privately opens the cards, Bob obtains no information about  $x$  if  $b$  is randomly selected and not disclosed by Alice. Bob must not disclose the obtained value. If Bob discloses the obtained value to Alice, Alice knows the value of the committed bit.

## Space and Time Complexities

The space complexity of card-based protocols is evaluated by the number of cards. Minimizing the number of cards is discussed in many works.

The number of rounds was proposed as a criterion to evaluate the time complexity of card-based protocols using private operations [49]. The first round begins from the initial state. The first round is (possibly parallel) local executions by each player using the cards initially given to each player. It ends at the instant when no further local execution is possible without receiving cards from another player. The local executions in each round include sending cards to some other players but do not include receiving cards. The result of every private execution is known to the player. For example, shuffling whose result is unknown to the player him/herself is not executed. Since the private operations are executed in a place where the other players cannot see, it is hard to force the player to execute such operations whose result is unknown to the player. The  $i(> 1)$ -th round begins with receiving all the cards sent during the  $(i - 1)$ -th round. Each player executes local executions using the received cards and the cards left to the player at the end of the  $(i - 1)$ -th round. Each player executes local executions until no further local execution is possible without receiving cards from another player. The number of rounds of a protocol is the maximum number of rounds necessary to output the result among all possible inputs and random values.

Let us show an example of a protocol execution and its space complexity and time complexity.

### Protocol 1 (AND protocol in [48])

Input:  $commit(x)$  and  $commit(y)$ .

Output:  $commit(x \wedge y)$ .

1. Alice executes a private random bisection cut on  $commit(x)$ . Let the output be  $commit(x')$ . Alice sends  $commit(x')$  and  $commit(y)$  to Bob.
2. Bob executes a private reveal on  $commit(x')$ . Bob sets

$$S_2 = \begin{cases} commit(y) || commit(0) & \text{if } x' = 1 \\ commit(0) || commit(y) & \text{if } x' = 0 \end{cases}$$

and sends  $S_2$  to Alice.

3. Alice executes a private reverse selection on  $S_2$  using the bit  $b$  generated in the private random bisection cut. Let the obtained sequence be  $S_3$ . Alice outputs  $S_3$ .

The correctness of the protocol is shown in [48]. The number of cards is four, since the cards of  $commit(x')$  are re-used to set  $commit(0)$ .

The first round ends at the instant when Alice sends  $commit(x')$  and  $commit(y)$  to Bob. The second round begins at receiving the cards by Bob. The second round ends at the instant when Bob sends  $S_2$  to Alice. The third round begins at receiving the cards by Alice. The number of rounds of this protocol is three.

Since each operation is relatively simple, the dominating time to execute protocols with private operations is the time to send cards between players and set up so that the cards are not seen by the other players. Thus, the number of rounds is the criterion to evaluate the time complexity of card-based protocols with private operations.

### Malicious Actions During Private Operations

We show examples of cheats by a malicious player for the AND protocol shown in Protocol 1. In the first round, Alice may reveal the cards of  $commit(x)$  (and/or  $commit(y)$ ) and read the secret input value  $x$  (and/or  $y$ , respectively). Alice might swap the two cards of  $commit(x)$  (and/or  $commit(y)$ ) and use  $\bar{x}$  (and/or  $\bar{y}$ , respectively) as the input value. In the second round, Bob might reveal the cards of  $commit(y)$  and read the secret input value  $y$ . Bob might set the cards incorrectly, for example, set

$$S_2 = \begin{cases} commit(1)||commit(y) & \text{if } x' = 1 \\ commit(y)||commit(1) & \text{if } x' = 0, \end{cases}$$

then the result becomes  $x \vee y$  instead of  $x \wedge y$ . Bob can set any other card sequences to obtain other incorrect results. In the third round, Alice might execute a private reverse selection using a value  $b' (\neq b)$ . Alice might reveal all the cards and see the values. To make the protocol secure against malicious players, all of the above cheats must be prohibited or detected.

### Envelopes

We need to prevent malicious reveal of committed input values. In the following protocols, we use envelopes as an additional tool. The cards can be put into an envelope and sealed. Opening the sealed envelope can be easily detected by anyone. Thus, a malicious player cannot irregularly open envelopes during private operations because it is detected by the other player. It is impossible to distinguish between two envelopes. No player can prepare the same envelopes in his/her pocket and exchange them for the envelopes used in the protocol. Such envelopes are used in some card-based protocols [8, 45, 58, 63].

We show some basic operations and notations related to the envelopes. The order of the cards put into an envelope is preserved when the cards are removed. For example, a card sequence  $S$  is put into an envelope, the output card sequence from the envelope must also be  $S$ . In the following protocols, two envelopes, the left and the right envelope are used. Swapping the right and left envelopes is exchanging the positions of the two envelopes.

We use the following notation  $[\cdot]$  for the representation of the state of two envelopes.  $[\cdot]$  means the sequence of cards when the cards are removed from the two envelopes by the same order of insertion. Note that in any case, the numbers of cards in the left and the right envelopes are the same. There are two types of insertion of cards. The first type is each one card from the two cards representing one bit is stored to the left and right envelope. One bit data is represented by two cards, for example, suppose that the card sequence  $s_1, s_2$  is  $commit(x)$ .  $s_1$  is put into the left envelope and  $s_2$  is put into the right envelope. When the cards are removed from the two envelopes by the same order of insertion, we can obtain  $commit(x)$ . The status of the two envelopes is represented as  $[commit(x)]$ , since  $commit(x)$  is obtained when the cards are removed. Now, consider the case that the left and the right envelopes are swapped. When the cards are removed from the two envelopes by the same order of the insertion, the card sequence becomes  $s_2, s_1$ , thus  $commit(\bar{x})$  is obtained, as shown in Fig. 1. The status of the swapped two envelopes is written as  $[commit(\bar{x})]$ .

The other type of insertion is putting a pair of cards to each of the envelopes. When we insert  $S = t_1, t_2$  to the left envelope and  $S' = t_3, t_4$  to the right envelope, the status of the two envelopes is written as  $[S||S']$ , since we can obtain  $S||S'$  when we remove the cards from the envelopes. When we swap the two envelopes, the status of the two envelopes is changed as  $[S'||S]$ , because we obtain  $S'||S$  when the cards are removed from the envelopes by the same order of the insertion, as shown in Fig. 2.

The above two types of insertions can be executed to a single pair of envelopes. Such combination is represented using a comma in  $[\cdot]$ . For example, the left (right) cards of  $commit(x)$  and  $commit(y)$  are inserted to the left (right) envelope,

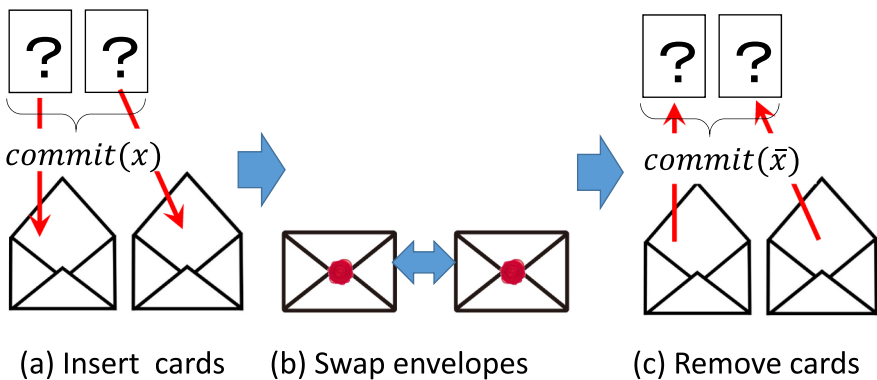


Fig. 1 Insertion and remove to a pair of envelopes(1)



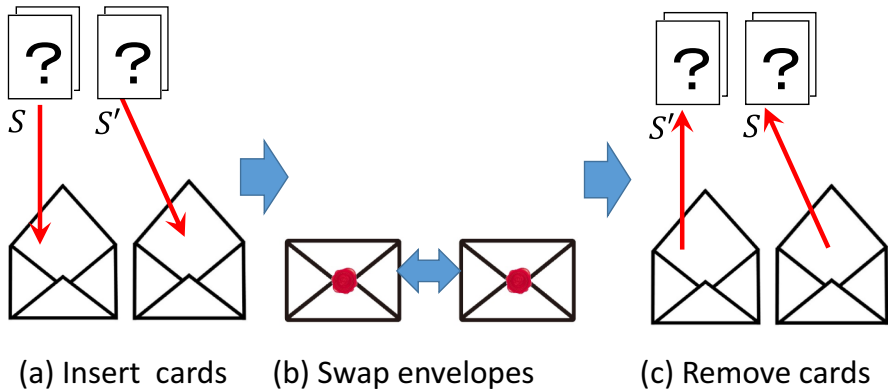


Fig. 2 Insertion and remove to a pair of envelopes(2)

respectively, and then  $S(S')$  is inserted to the left (right) envelope, respectively. The status of the two envelopes is written as  $[commit(x), commit(y), S||S']$ . When we swap the two envelopes, the status is changed to  $[commit(\bar{x}), commit(\bar{y}), S' || S]$ .

In this paper, private random bisection cuts are executed to these two envelopes. When Alice executes a private random bisection cut to the two envelopes that have  $[commit(x), commit(y)], [commit(x \oplus b), commit(y \oplus b)]$  is obtained, where  $b$  is the random bit selected by Alice. When Alice executes a private random bisection cut to the two envelopes that have  $[commit(x), S_1 || S_2], [commit(x \oplus b), swap(b, S_1 || S_2)]$  is obtained, where  $b$  is the random bit selected by Alice.

A private reveal by a player on an envelope means that the player opens the envelope, obtains a sequence of face-down cards, turns the face-down cards, and sees the marks of the cards.

### Security Model

This section first shows the possible malicious actions during protocol executions. Then, we show some additional assumptions related to security.

With the envelopes, the activities by a malicious player are as follows when the private primitives are private random bisection cuts, private reverse cuts, and private reveals on the envelopes.

#### Assumption 1 (Operations by malicious players)

- When a malicious player executes a private operation, he/she can swap some envelopes even if it is not allowed in the protocol. He/she can open some envelopes and see the marks on the cards.

- When a malicious player executes a private random bisection cut to two sets of envelopes  $A$  and  $B$  using the same random bit, he/she can use different bits to  $A$  and  $B$ .
- When a malicious player executes a private reveal on envelope  $A$ , he/she can open another envelope  $B$ . Also, he/she might not place envelopes according to the opened cards.
- When a malicious player executes a private reverse cut using a bit  $b$ , he/she might use  $b' (\neq b)$  instead of  $b$ .

Since there is no collusion of the two players, the public operations executed by both players together are correctly executed even if both players are malicious.

During the execution of a protocol, a player might find the other player's misbehavior. We need to define the malicious player's action about the detection of the other player's misbehavior.

**Assumption 2** (Misbehavior detection). A malicious player does not suppress to claim the other player's misbehavior.

As shown in the next section, an incorrect result might be obtained if a malicious player does not claim the other malicious player's misbehavior.

We add an assumption that for at least one input, say,  $x$  multiple copies of  $commit(x)$  are given as input. The reason for this assumption is as follows. When a player, say, Alice is given  $commit(x)$  and executes a private operation, there is no way for the other player to detect whether Alice maliciously executed swapping two cards of  $commit(x)$  and made  $commit(\bar{x})$ . Since Bob does not know  $x$ , Bob cannot claim that  $\bar{x}$  is used instead of  $x$ . To detect this type of malicious operation, another copy of  $commit(x)$  must be given. Using the copy of  $commit(x)$ , Bob can detect that Alice irregularly swapped  $commit(x)$ , as shown in the protocols in this paper. Note that a method to obtain multiple copies of inputs using envelopes is shown in Sect. 4.4.

## Secure Protocols Under Malicious Model

This section shows our new protocols that are secure against malicious behavior.

### XOR Protocol

First, we show an XOR protocol. Note that two copies of one input,  $commit(x)$ , are necessary to prevent malicious actions.

#### Protocol 2 (XOR protocol)

Input: two copies of  $commit(x)$  and one copy of  $commit(y)$ .

Output:  $commit(x \oplus y)$ .

1. Alice and Bob publicly put cards of one  $commit(x)$  and  $commit(y)$  into two envelopes. The left (right) cards of  $commit(x)$  and  $commit(y)$  are put into the left (right) envelope, respectively. The two envelopes have  $[commit(x), commit(y)]$ . The remaining two cards of  $commit(x)$  are put into two new envelopes so that the left (right) card is put into the left (right) envelope, respectively. The two envelopes have  $[commit(x)]$  (Fig. 3a). The envelopes that have  $[commit(x)]$  and  $[commit(x), commit(y)]$  are sent to Alice.

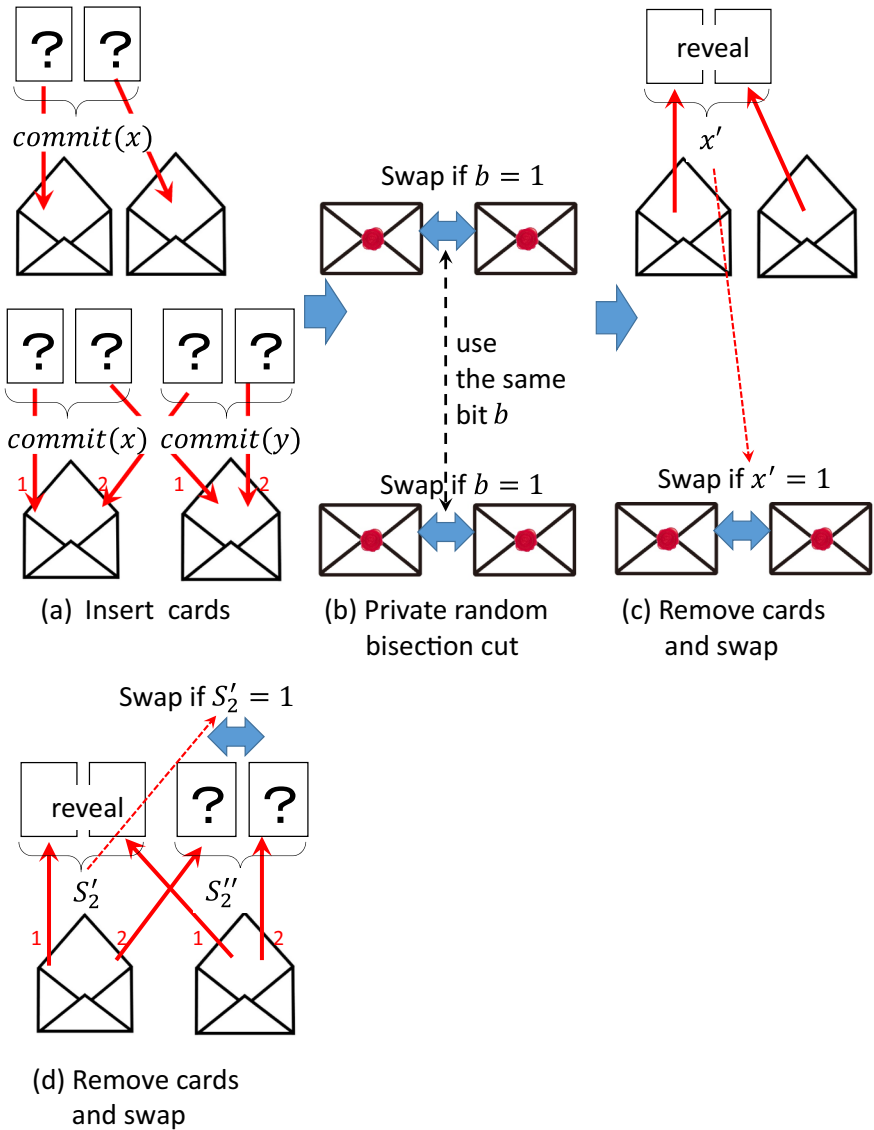


Fig. 3 XOR protocol

2. Alice executes private random bisection cuts on  $[commit(x)]$  and  $[commit(x), commit(y)]$  using the same random bit  $b$  (Fig. 3b). Let the output be  $[S_1]$  and  $[S'_1, S''_1]$ , respectively.  $S_1 = S'_1 = commit(x \oplus b)$  and  $S''_1 = commit(y \oplus b)$  are satisfied. Alice sends  $[S_1]$  and  $[S'_1, S''_1]$  to Bob.
3. Bob claims that Alice misbehaved and terminates the protocol if an envelope is opened. Bob executes a private reveal on  $[S_1 = commit(x')]$ . Bob claims that Alice misbehaved and terminates the protocol if the number of cards in an envelope is not one. Bob privately swaps the two envelopes of  $[S'_1, S''_1]$  if  $x' = 1$ , otherwise, does nothing (Fig. 3c). Bob makes the two envelopes public, which are denoted  $[S'_2, S''_2]$ .
4. Alice claims that Bob misbehaved and terminates the protocol if an envelope is opened. Alice and Bob open the envelopes together. If the number of cards in an envelope is one, Alice claims that Bob opened an incorrect envelope at Step 3 and terminates the protocol. Otherwise, they obtain  $S'_2$  and  $S''_2$ . They turns (that is, face-up)  $S'_2$ . If  $S'_2 = 0$ ,  $S''_2$  is the output of the protocol. If  $S'_2 = 1$ , swap the left and the right card of  $S''_2$  and the result is the output of the protocol (Fig. 3d).

The protocol is three rounds. The first round is the public execution by Alice and Bob. Note that the public operations can be executed by a player say, Alice, in front of the other player, Bob.

The second round is executed by Alice. The third round is executed by Bob. The last public executions at Step 4 by Alice and Bob do not need sending envelopes. Bob just makes the envelopes public and Bob can execute the operations in front of Alice. Thus no overhead is necessary for the public execution. Therefore, the number of rounds is considered to be three. The number of cards used in the protocol is six.

From Assumption 1, the malicious activities in this protocol are the follows. At Step 1 and Step 4, there are no malicious activities because the operations are publicly executed by the two players. At Step 2, Alice might open some envelopes during the execution. Alice might not execute the private random bisection cuts correctly, that is, Alice uses  $b$  for swapping  $[commit(x)]$  and  $b' (\neq b)$  for swapping  $[commit(x), commit(y)]$ . Alice might incorrectly send the envelopes to Bob, that is, Alice might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are given to Bob as the two envelopes of  $[S_1]$ ). At Step 3, Bob might open envelopes that are not allowed. Bob might not execute swapping  $[S'_1, S''_1]$  correctly. Bob might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are shown public as the two envelopes of  $[S'_2, S''_2]$ ). All of these malicious activities must be prevented or detected.

**Theorem 1** *The output of the XOR protocol is correct if the protocol is not terminated during execution even if Alice and/or Bob are malicious. The protocol does not reveal the input values to the players if the protocol is not terminated during execution.*

**Proof** First, we show the correctness when both Alice and Bob are honest.

Alice sends  $[S_1] = [commit(x \oplus b)]$  and  $[S'_1, S''_1] = [commit(x \oplus b), commit(y \oplus b)]$  to Bob. Bob swaps the pair of  $[S'_1, S''_1]$  if  $x \oplus b = 1$ . Thus  $[S'_2, S''_2] = [commit((x \oplus b) \oplus (x \oplus b)), commit((y \oplus b) \oplus (x \oplus b))] = [commit(0), commit(x \oplus y)]$ . Since  $S'_2 = commit(0)$ ,  $S''_2$  is not swapped and the output is  $commit(x \oplus y)$ . Therefore, the output is correct. The protocol is secure since Alice sees  $S'_2 = 0$  and Bob sees  $S'_2 = 0$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.

Next, consider the case when Alice is malicious and Bob is honest. If Alice opens an envelope during the private operation, Bob can detect misbehavior. Next, consider the case when Alice does not execute the private random bisection cut correctly. Since the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ, making an incorrect pair of envelopes (for example, the left envelope of  $[S_1]$  and the right envelope of  $[S'_1, S''_1]$  are sent to Bob as  $[S_1]$ ) is detected by Bob when Bob opens  $[S_1]$ . The only cheat that cannot be detected by Bob is incorrectly swapping each pair of envelopes. Though the cheat cannot be detected, the result becomes correct as shown below. Let  $b$  and  $b'$  be the random bits selected to swap the envelopes that have  $[commit(x)]$  and  $[commit(x), commit(y)]$ , respectively. The output by Alice is  $[S_1] = [commit(x \oplus b)]$  and  $[S'_1, S''_1] = [commit(x \oplus b'), commit(y \oplus b')]$ . After Bob opens  $[S_1] = [commit(x \oplus b)]$ , Bob swaps the envelopes if  $x \oplus b = 1$ , thus the result  $[S'_2, S''_2] = [commit(x \oplus b' \oplus x \oplus b), commit(y \oplus b' \oplus x \oplus b)] = [commit(b \oplus b'), commit(y \oplus b' \oplus x \oplus b)]$ . When the players open  $S'_2$ , the cards of  $S'_2$  are swapped if  $b \oplus b' = 1$ . Thus, the output is  $commit(y \oplus b' \oplus x \oplus b \oplus (b \oplus b')) = commit(y \oplus x)$ . The result is correct regardless of the selection of  $b$  and  $b'$ . The protocol is secure in this case since Alice sees  $S'_2 = b \oplus b'$  and Bob sees  $S'_2 = b \oplus b'$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.

Next, consider the case when Alice is honest or malicious and Bob is malicious. Alice might incorrectly execute the private random bisection cuts using  $b$  and  $b'$  as in the above case. The other cheats such as irregularly opening envelopes and making incorrect pairs of envelopes are detected by Bob as shown above. If Bob opens an envelope of  $[S'_1, S''_1]$ , the cheat can be detected by Alice. If Bob makes an incorrect pair of envelopes (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are shown public as the two envelopes of  $[S'_2, S''_2]$ ), the cheat can be detected by Alice when they open  $[S'_2, S''_2]$  because the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ. Next, consider the case when Bob does not set the envelopes correctly. When Bob sees  $x \oplus b$ , Bob does not swap the envelopes correctly, that is, Bob selects some value  $b'' (\neq x \oplus b) \in \{0, 1\}$  and swaps the envelopes of  $[S'_1, S''_1]$  using  $b''$ . If  $b'' = x \oplus b$ , the result is correct as shown above. Thus the only cheat selection of  $b''$  is  $b'' = x \oplus b = x \oplus b \oplus 1$ .

In this case, the result is  $[S'_2, S''_2] = [commit(x \oplus b' \oplus b''), commit(y \oplus b' \oplus b'')] = [commit(b' \oplus b \oplus 1), commit(y \oplus b' \oplus x \oplus b \oplus 1)]$ . When Alice and Bob open  $S'_2$ , the two cards of  $S''_2$  are swapped if  $b' \oplus b \oplus 1 = 1$ . Thus, the output is  $commit(y \oplus b' \oplus x \oplus b \oplus 1 \oplus (b' \oplus b \oplus 1)) = commit(y \oplus x)$  and the result is correct. The protocol is secure in this case since Alice sees  $S'_2 = b \oplus b' \oplus 1$  and Bob sees  $S'_2 = b \oplus b' \oplus 1$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.  $\square$

Note that the protocol achieves an error correction. Even if Alice and/or Bob make mistakes in swapping the left and the right of paired envelopes, the mistakes are automatically corrected. Consider a mistake by Alice when she executes a private random bisection cut on  $[commit(x)]$  and  $[commit(x), commit(y)]$ , that is, she swapped  $[commit(x)]$  but did not swap  $[commit(x), commit(y)]$  (or did not swap  $[commit(x)]$  but swapped  $[commit(x), commit(y)]$ ). The situation is just the same when malicious Alice used different  $b$  and  $b'$  to swap the pairs. The other mistake by Bob is at Step 3, Bob did not swap the two envelopes of  $[S'_1, S''_1]$  when  $x' = 1$  (or swapped  $[S'_1, S''_1]$  when  $x' = 0$ ). The situation is just the same when malicious Bob executed a false swap. Since the result is correct even with the malicious actions, these errors are corrected.

Note that a malicious player can output a false misbehavior detection even if the other player is honest. For example, at Step 3 of the above protocol, malicious Bob can claim that the envelopes are incorrectly sent by Alice even if Alice is honest, for example, Bob claims that the left envelope of  $[S_1]$  has two cards when Bob opened  $[S_1]$ . Actually, Bob is malicious and Bob swapped the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  and opened the left envelope of  $[S'_1, S''_1]$ . In this case, though Alice (and Bob) knows that Bob's claim is incorrect, the players just terminate the protocol, since there is no third party for Alice and Bob to appeal which player is malicious. Note that if we prepare envelopes with different colors,  $[commit(x)]$  is put into two white envelopes and  $[commit(x), commit(y)]$  is put into two yellow envelopes. Then, this kind of cheat becomes impossible.

From Assumption 2, a malicious player does not suppress to claim the other player's misbehavior. Suppose that malicious Bob does not claim malicious Alice's misbehavior. At Step 2, Alice opens the envelopes of  $[S_1, S'_1]$  and swap the cards so that  $[S_1, S'_1] = [commit(x \oplus b), commit(\bar{y} \oplus b)]$ . At Step 3, Bob does not claim that Alice opened the envelopes and continues the protocol. Then the output of the protocol is  $commit(x \oplus \bar{y})$ . This case is just the same as the case when Alice and Bob collude. Thus, Assumption 2 is necessary.

Note that there can be cases when both of the players are independently malicious. Let us consider the following case. Alice knows the secret value  $x$  and  $y$  by some irregular means in advance. Bob does not know the fact. Malicious Alice knows the correct value of  $x \oplus y$  and wants Bob to obtain a false result. On the other hand, Bob independently knows the secret value  $x$  and  $y$  by some other irregular means in advance. Alice does not know the fact. Malicious Bob wants Alice to obtain a false result. Even in this case, when the protocol is terminated, the result is correct, that is, the two independent malicious actions become void.

## AND Protocol

Next, we show an AND protocol that uses envelopes.

### Protocol 3 (AND protocol)

Input: two copies of  $commit(x)$  and one copy of  $commit(x)$ .

Output:  $commit(x \wedge y)$ .

1. Alice and Bob publicly put cards into envelopes. The left card of  $commit(x)$  and two new cards of  $commit(0)$  are put into the left envelope. The right card of  $commit(x)$  and the two cards of  $commit(y)$  are put into the right envelope. The envelopes have  $[commit(x), commit(0) || commit(y)]$ . The remaining two cards of  $commit(x)$  are put into two envelopes so that the left (right) card is put into the left (right) envelope, respectively. The envelopes have  $[commit(x)]$  (Fig. 4a). The

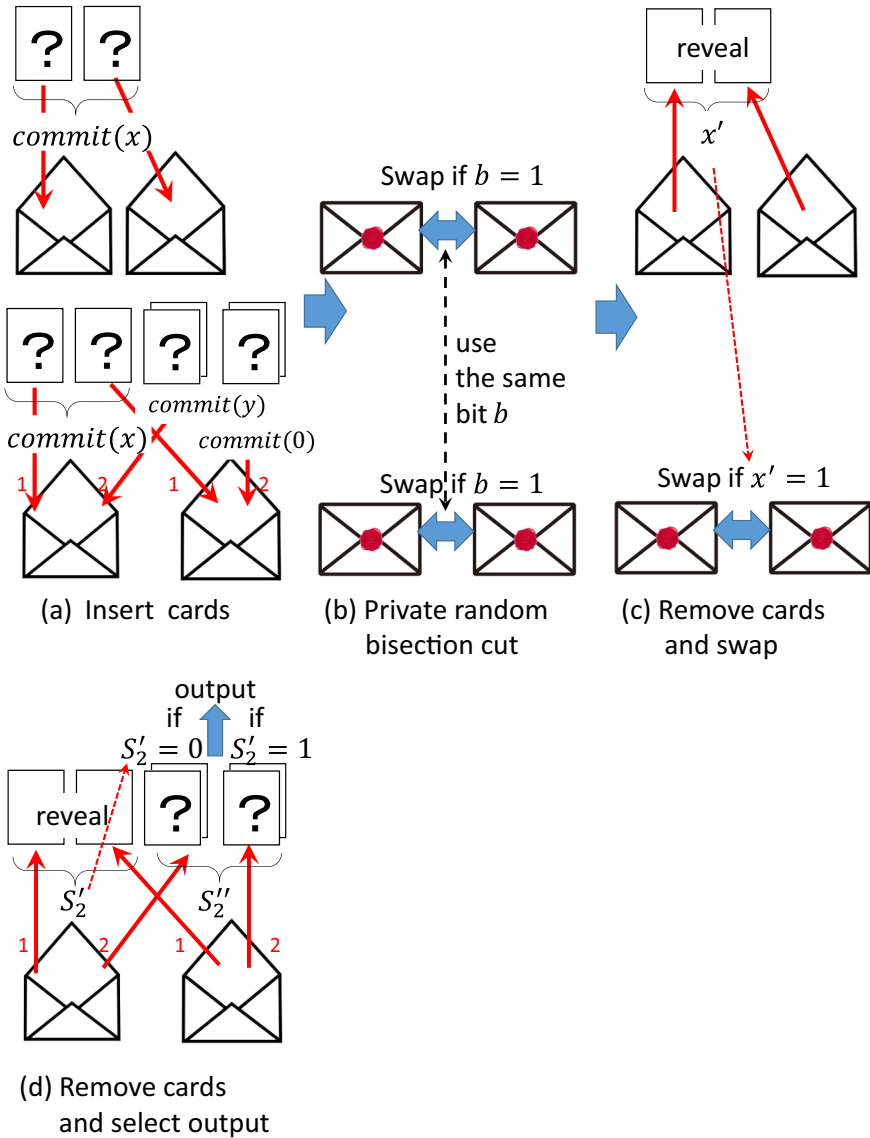


Fig. 4 AND protocol

- envelopes that have  $[commit(x)]$  and  $[commit(x), commit(0)||commit(y)]$  are sent to Alice.
- Alice executes private random bisection cuts on  $[commit(x)]$  and  $[commit(x), commit(0)||commit(y)]$  using the same random bit  $b$  (Fig. 4b). Let the output be  $[S_1]$  and  $[S'_1, S''_1]$ .  $S_1 = S'_1 = commit(x \oplus b)$  and  $S''_1 = swap(b, commit(0)||commit(y))$  are satisfied. Alice sends  $[S_1]$  and  $[S'_1, S''_1]$  to Bob.
  - Bob claims that Alice misbehaved and terminates the protocol if an envelope is opened. Bob executes a private reveal on  $[S_1 = commit(x')]$ . Bob claims that Alice misbehaved and terminates the protocol if the number of cards in an envelope is not one. Bob privately swaps two envelopes of  $[S'_1, S''_1]$  if  $x' = 1$ , otherwise, does nothing (Fig. 4c). Bob makes the two envelopes public, which are denoted  $[S'_2, S''_2]$ .
  - Alice claims that Bob misbehaved and terminates the protocol if an envelope is opened. Alice and Bob open the envelopes together. If the number of cards in an envelope is one, Alice claims that Bob opened an incorrect envelope at Step 3 and terminates the protocol. Otherwise, they obtain  $S'_2$  and  $S''_2$ . They turn (that is, face-up)  $S'_2$ . If  $S'_2 = 0$ , the left two cards of  $S''_2$  are the output of the protocol. If  $S'_2 = 1$ , the right two cards of  $S''_2$  are the output of the protocol (Fig. 4d).

The protocol is three rounds. The protocol uses eight cards since two new cards are used to set  $commit(0)$ .

From Assumption 1, the malicious activities in this protocol are the follows. At Step 1 and Step 4, there are no malicious activities because the operations are publicly executed by the two players. At Step 2, Alice might open some envelopes during the execution. Alice might not execute the private random bisection cuts correctly, that is, Alice uses  $b$  for swapping  $[commit(x)]$  and  $b'(\neq b)$  for swapping  $[commit(x), commit(0)||commit(y)]$ . Alice might incorrectly send the envelopes to Bob, that is, Alice might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are given to Bob as the two envelopes of  $[S_1]$ ). At Step 3, Bob might open envelopes that are not allowed. Bob might not execute swapping  $[S'_1, S''_1]$  correctly. Bob might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are shown public as the two envelopes of  $[S'_2, S''_2]$ ). All of these malicious activities must be prevented or detected.

**Theorem 2** *The output of the AND protocol is correct if the protocol is not terminated during execution even if Alice and/or Bob are malicious. The protocol does not reveal the input values to the players if the protocol is not terminated during execution.*

**Proof** The desired output can be represented as follows.

$$x \wedge y = \begin{cases} y & \text{if } x = 1 \\ 0 & \text{if } x = 0. \end{cases}$$

First, we show the correctness when both Alice and Bob are honest.



Alice sends  $[S_1] = [commit(x \oplus b)]$  and  $[S'_1, S''_1] = [commit(x \oplus b), swap(b, commit(0)||commit(y))]$  to Bob. Bob swaps the pair of  $[S'_1, S''_1]$  if  $x \oplus b = 1$ . Thus,  $[S'_2, S''_2] = [commit((x \oplus b) \oplus (x \oplus b)), swap(x \oplus b, swap(b, commit(0)||commit(y)))] = [commit(0), swap(x, commit(0)||commit(y))]$ . Thus the players select the left pair of  $swap(x, commit(0)||commit(y))$ . The selected cards are  $commit(y)$  if  $x = 1$  and  $commit(0)$  if  $x = 0$ . Thus, the output is correct.

The protocol is secure since Alice sees  $S'_2 = 0$  and Bob sees  $S'_2 = 0$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.

Next, consider the case when Alice is malicious and Bob is honest. If Alice opens an envelope during the private operation, Bob can detect misbehavior. Next, consider the case when Alice does not execute the private random bisection cut correctly. Since the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ, making an incorrect pair of envelopes (for example, the left envelope of  $[S_1]$  and the right envelope of  $[S'_1, S''_1]$  are sent to Bob as  $[S_1]$ ) is detected by Bob when Bob opens  $[S_1]$ . The only cheat that cannot be detected by Bob is incorrectly swapping each pair of envelopes. Though the cheat cannot be detected, the result becomes correct as shown below. Let  $b$  and  $b'$  be the random bits selected to swap the envelopes that have  $[commit(x)]$  and  $[commit(x), commit(0)||commit(x)]$ , respectively. The output by Alice is  $[commit(x \oplus b)]$  and  $[commit(x \oplus b'), swap(b', commit(0)||commit(y))]$ . After Bob opens  $[commit(x \oplus b)]$ , Bob swaps the envelopes if  $x \oplus b = 1$ , thus the result  $[S'_2, S''_2] = [commit(x \oplus b' \oplus x \oplus b), swap(x \oplus b, swap(b', commit(0)||commit(y)))] = [commit(b \oplus b'), swap(x \oplus b \oplus b', commit(0)||commit(y))]$ . When the players open  $S'_2$ , the left pair of  $S'_2$  is the output if  $b \oplus b' = 0$ . The right pair of  $S'_2$  is the output if  $b \oplus b' = 1$ . This is equivalent to execute  $swap(b \oplus b', S'_2)$  and select the left pair. Since  $swap(b \oplus b', S'_2) = swap(b \oplus b', swap(x \oplus b \oplus b', commit(0)||commit(y))) = swap(x, commit(0)||commit(y))$ , the output is  $commit(0)$  if  $x = 0$ , otherwise the output is  $commit(y)$ . Therefore, the output is correct regardless of the selection of  $b$  and  $b'$ .

The protocol is secure in this case since Alice sees  $S'_2 = b \oplus b'$  and Bob sees  $S'_2 = b \oplus b'$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.

Next, consider the case when Alice is honest or malicious and Bob is malicious. Alice might incorrectly execute the private random bisection cuts using  $b$  and  $b'$  as in the above case. If Bob opens an envelope of  $[S'_1, S''_1]$ , the cheat can be detected by Alice. If Bob makes an incorrect pair of envelopes (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are shown public as the two envelopes of  $[S'_2, S''_2]$ ), the cheat can be detected by Alice when they open  $[S'_2, S''_2]$  because the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ. Next, consider the case when Bob does not set the envelopes correctly. When Bob sees  $x \oplus b$ , Bob does not swap the envelopes correctly, that is, Bob selects some value  $b'' (\neq x \oplus b) \in \{0, 1\}$  and swaps the envelopes of  $[S'_1, S''_1]$  using  $b''$ . When  $b'' = x \oplus b$ , the output is correct since it is the correct value. Thus the only cheat selection of  $b''$  is  $b'' = x \oplus b = x \oplus b \oplus 1$ .

In this case, the result is  $[S'_2, S''_2] = [commit(x \oplus b' \oplus b''), swap(b'', swap(b, commit(0)||commit(y)))] = [commit(b \oplus b' \oplus 1), swap(x \oplus b \oplus b' \oplus 1, commit(0)||commit(y))]$ . When Alice and Bob open  $S'_2$ , the right pair of  $S'_2$  are used as the output if  $b \oplus b' \oplus 1 = 1$  otherwise, the left pair of  $S'_2$  are used as the output. This is equivalent to execute  $swap(b \oplus b' \oplus 1, S'_2)$  and select the left pair. Since  $swap(b \oplus b' \oplus 1, S'_2) = swap(b \oplus b' \oplus 1, swap(x \oplus b \oplus b' \oplus 1, commit(0)||commit(y)))$

$= \text{swap}(x, \text{commit}(0) || \text{commit}(y))$ , the output is  $\text{commit}(0)$  if  $x = 0$ , otherwise the output is  $\text{commit}(y)$ . Therefore, the output is correct regardless of the selection of  $b$  and  $b'$ .

The protocol is secure in this case since Alice sees  $S'_2 = b \oplus b' \oplus 1$  and Bob sees  $S'_2 = b \oplus b' \oplus 1$  and  $S_1 = x \oplus b$  but  $b$  is an unknown random value for Bob.  $\square$

Note that even if Alice and/or Bob make mistakes in swapping the left and the right of paired envelopes, the mistakes are automatically corrected. The reason is just the same as the one for the XOR protocol.

Note that using an argument similar to the one in [48], any 2-variable Boolean function can be calculated by a protocol similar to the XOR protocol or the AND protocol.

### n-variable Boolean Functions

We show a protocol that calculates any  $n$ -variable Boolean function

$$f(x_1, x_2, \dots, x_n).$$

#### Protocol 4 (Protocol for $n$ -variable Boolean function)

Input: two copies of  $\text{commit}(x_i) (i = 1, 2, \dots, n)$ .

Output:  $\text{commit}(f(x_1, x_2, \dots, x_n))$ .

1. Alice and Bob publicly make  $2^n$  commitments  $C_{a_1, a_2, \dots, a_n} = \text{commit}(f(a_1, a_2, \dots, a_n)) (a_i \in \{0, 1\}, i = 1, 2, \dots, n)$ . Alice and Bob makes one sequence  $T$  from the sequences  $C_{a_1, a_2, \dots, a_n} (a_i \in \{0, 1\}, i = 1, 2, \dots, n)$ . by lining up them using the lexicographical order of  $(a_1, a_2, \dots, a_n)$ . That is,  $T = C_{0,0,\dots,0,0} || C_{0,0,\dots,0,1} || C_{0,0,\dots,1,0} || \dots || C_{1,1,\dots,1,1}$ .
2. For  $i = 1, 2, \dots, n$ , execute the following procedure.
  - (a) Alice and Bob publicly divide  $T$  into the two equal size sequence  $T_0$  and  $T_1$ , that is,  $T_0 = C_{x_1, x_2, \dots, x_{i-1}, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, 0, 1, 1, \dots, 1, 1}$  and  $T_1 = C_{x_1, x_2, \dots, x_{i-1}, 1, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, 1, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, 1, 1, 1, \dots, 1, 1}$ . Note that when  $i = 1$ ,  $T_0 = C_{0,0,\dots,0,0} || C_{0,0,\dots,0,1} || \dots || C_{0,1,\dots,1,1}$  and  $T_1 = C_{1,0,\dots,0,0} || C_{1,0,\dots,0,1} || \dots || C_{1,1,\dots,1,1}$ . Alice and Bob publicly put cards into envelopes. The left card of  $\text{commit}(x_i)$  and  $T_0$  are put into the left envelope. The right card of  $\text{commit}(x_i)$  and  $T_1$  are put into the right envelope. The two envelopes have  $[\text{commit}(x_i), T_0 || T_1]$ . The remaining two cards of  $\text{commit}(x_i)$  are put into two new envelopes so that the left (right) card is put into the left (right) envelope, respectively. The envelopes have  $[\text{commit}(x_i)]$ . The envelopes that have  $[\text{commit}(x_i)]$  and  $[\text{commit}(x_i), T_0 || T_1]$  are sent to Alice.
  - (b) Alice executes private random bisection cuts on  $[\text{commit}(x_i)]$  and  $[\text{commit}(x_i), T_0 || T_1]$  using the same new random bit  $b_i$ . Let the output be  $[S_i]$

- and  $[S'_1, S''_1]$ .  $S_1 = S'_1 = \text{commit}(x_i \oplus b_i)$ . and  $S''_1 = \text{swap}(b_i, T_0 || T_1)$ . Alice sends  $[S_1]$  and  $[S'_1, S''_1]$  to Bob.
- (c) Bob claims that Alice misbehaved and terminates the protocol if an envelope is opened. Bob executes a private reveal on  $[S_1 = \text{commit}(x'_i)]$ . Bob claims that Alice misbehaved and terminates the protocol if the number of cards in an envelope is not one. Bob privately swaps two envelopes of  $[S'_1, S''_1]$  if  $x'_i = 1$ , otherwise, does nothing. Bob makes the two envelopes public, which are denoted  $[S'_2, S''_2]$ .
  - (d) Alice claims that Bob misbehaved and terminates the protocol if an envelope is opened. Alice and Bob open the envelopes together. If the number of cards in an envelope is one, Alice claims that Bob opened an incorrect envelope at Step 2 (c) and terminates the protocol. Otherwise, they obtain  $S'_2$  and  $S''_2$ . They turn (that is, face-up)  $S'_2$ . If  $S'_2 = 0$ , let  $T$  be the left half cards of  $S''_2$ . If  $S'_2 = 1$ , let  $T$  be the right half cards of  $S''_2$ . As shown in the proof of Theorem 3,  $T = C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, x_i, 1, 1, \dots, 1, 1}$  is satisfied.

3. At the end of the protocol,  $T = C_{x_1, x_2, \dots, x_n} = \text{commit}(f(x_1, x_2, \dots, x_n))$  is the output.

From Assumption 1, the malicious activities in this protocol are the follows. At Step 1, Step 2 (a), and Step 2 (d), there are no malicious activities because the operations are publicly executed by the two players. At Step 2 (b), Alice might open some envelopes during the execution. Alice might not execute the private random bisection cuts correctly, that is, Alice uses  $b_i$  for swapping  $[\text{commit}(x_i)]$  and  $b'_i (\neq b_i)$  for swapping  $[\text{commit}(x_i), T_0 || T_1]$ . Alice might incorrectly send the envelopes to Bob, that is, Alice might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are given to Bob as the two envelopes of  $[S_1]$ ). At Step 2 (c), Bob might open envelopes that are not allowed. Bob might not execute swapping  $[S'_1, S''_1]$  correctly. Bob might swap the four envelopes of  $[S_1]$  and  $[S'_1, S''_1]$  (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$  are shown public as the two envelopes of  $[S'_2, S''_2]$ ). All of these malicious activities must be prevented or detected.

The correctness of the protocol is shown as follows.

**Theorem 3** *The output of Protocol 4 is correct if the protocol is not terminated during execution even if Alice and/or Bob are malicious. The protocol does not reveal input values to the players if the protocol is not terminated during execution.*

**Proof** The correctness of the protocol is shown by proving the following property. Given inputs  $T_0 = C_{x_1, x_2, \dots, x_{i-1}, 0, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, 0, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, 0, 1, 1, \dots, 1, 1}$ ,  $T_1 = C_{x_1, x_2, \dots, x_{i-1}, 1, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, 1, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, 1, 1, 1, \dots, 1, 1}$ , and two copies of  $\text{commit}(x_i)$  at the beginning of Step 2 (a),  $T = C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, x_i, 1, 1, \dots, 1, 1}$  is obtained at the end of Step 2 (d) if the protocol is not terminated during execution.

First, we show the correctness when both Alice and Bob are honest. Alice sends  $[S_1] = [commit(x_i \oplus b_i)]$  and  $[S'_1, S''_1] = [commit(x_i \oplus b_i), swap(b_i, T_0 || T_1)]$  to Bob. Bob swaps the pair of  $[S'_1, S''_1]$  if  $x_i \oplus b_i = 1$ . Thus  $[S'_2, S''_2] = [commit((x_i \oplus b_i) \oplus (x_i \oplus b_i)), swap(x_i \oplus b_i, swap(b_i, T_0 || T_1))] = [commit(0), swap(x_i, T_0 || T_1)]$ . Since  $S'_2 = commit(0)$ ,  $S''_2$  is not swapped and the output is the left half of  $swap(x_i, T_0 || T_1)$ , which is  $C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, x_i, 1, 1, \dots, 1, 1}$ . Therefore, the output is correct.

The protocol is secure since Alice sees  $S'_2 = 0$  and Bob sees  $S'_2 = 0$  and  $S_1 = x_i \oplus b_i$  but  $b_i$  is an unknown random value for Bob.

Next, consider the case when Alice is malicious and Bob is honest. If Alice opens an envelope during the private operation, Bob can detect misbehavior. Next, consider the case when Alice does not execute the private random bisection cut correctly. Since the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ, making incorrect pairs of envelopes (for example, the left envelope of  $[S_1]$  and the right envelope of  $[S'_1, S''_1]$ ) are sent to Bob as  $[S_1]$  is detected by Bob when Bob opens  $[S_1]$ . The only cheat that cannot be detected by Bob is incorrectly swapping each pair of envelopes. Though the cheat cannot be detected, the result becomes correct as shown below. Let  $b_i$  and  $b'_i$  be the random bits selected to swap the envelopes that have  $[commit(x_i)]$  and  $[commit(x_i), T_0 || T_1]$ , respectively. The output by Alice is  $[S_1] = [commit(x_i \oplus b_i)]$  and  $[S'_1, S''_1] = [commit(x_i \oplus b'_i), swap(b'_i, T_0 || T_1)]$ . After Bob opens  $[S_1] = [commit(x_i \oplus b_i)]$ , Bob swaps the envelopes if  $x_i \oplus b_i = 1$ , thus the result  $[S'_2, S''_2] = [commit(x_i \oplus b'_i \oplus x_i \oplus b_i), swap(x_i \oplus b_i, swap(b'_i, T_0 || T_1))] = [commit(b_i \oplus b'_i), swap(x_i \oplus b_i \oplus b'_i, T_0 || T_1)]$ . When the players open  $S'_2$ , the left half cards of  $S'_2$  is the output if  $b_i \oplus b'_i = 0$ . The right half cards of  $S'_2$  is the output if  $b_i \oplus b'_i = 1$ . This is equivalent to execute  $swap(b_i \oplus b'_i, S''_2)$  and select the left half cards. Since  $swap(b_i \oplus b'_i, swap(x_i \oplus b_i \oplus b'_i, T_0 || T_1)) = swap(x_i, T_0 || T_1)$ , the output  $T = T_0$  if  $x_i = 0$  and  $T = T_1$  if  $x_i = 1$ . Thus,  $T = C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, x_i, 1, 1, \dots, 1, 1}$ . The result is correct regardless of the selection of  $b_i$  and  $b'_i$ .

The protocol is also secure in this case since Alice sees  $S'_2 = b_i \oplus b'_i$  and Bob sees  $S'_2 = b_i \oplus b'_i$  and  $S_1 = x_i \oplus b_i$  but  $b_i$  is an unknown random value for Bob.

Next, consider the case Alice is honest or malicious and Bob is malicious. Alice might incorrectly execute the private random bisection cuts using  $b_i$  and  $b'_i$  as in the above case. The other cheats such as irregularly opening envelopes and making incorrect pairs of envelopes are detected by Bob as shown above. If Bob opens an envelope of  $[S'_1, S''_1]$ , the cheat can be detected by Alice. If Bob makes an incorrect pair of envelopes (for example, the left envelope of  $[S_1]$  and the left envelope of  $[S'_1, S''_1]$ ) are shown public as the two envelopes of  $[S'_2, S''_2]$ , the cheat can be detected by Alice when they open  $[S'_2, S''_2]$  because the numbers of cards in  $[S_1]$  and  $[S'_1, S''_1]$  differ. Next, consider the case when Bob does not set the envelopes correctly. When Bob sees  $x_i \oplus b_i$ , Bob does not swap the envelopes correctly, that is, Bob selects some value  $b''_i (\neq x_i \oplus b_i) \in \{0, 1\}$  and swaps the envelopes of  $[S'_1, S''_1]$  using  $b''_i$ . If  $b''_i = x_i \oplus b_i$ , the result is correct as shown above. Thus the only cheat selection of  $b''_i$  is  $b''_i = x_i \oplus b_i = x_i \oplus b_i \oplus 1$ .

In this case, the result is  $[S'_2, S''_2] = [commit(x_i \oplus b_i \oplus b''_i), swap(b''_i, swap(b'_i, T_0 || T_1))] = [commit(b'_i \oplus b_i \oplus 1), swap(x_i \oplus b_i \oplus b'_i \oplus 1, T_0 || T_1)]$ . When Alice

and Bob open  $S'_2$ , the right half cards of  $S''_2$  are used as the output if  $b_i \oplus b'_i \oplus 1 = 1$ , otherwise the left half cards of  $S''_2$  are used as the output. This is equivalent to execute  $swap(b_i \oplus b'_i \oplus 1, S''_2)$  and select the left half of cards. Since  $swap(b_i \oplus b'_i \oplus 1, S''_2) = swap(b_i \oplus b'_i \oplus 1, swap(x_i \oplus b_i \oplus b'_i \oplus 1, T_0 || T_1)) = swap(x_i, T_0 || T_1)$ , the output is  $T_0$  if  $x_i = 0$ , otherwise the output is  $T_1$ , that is,  $T = C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 0} || C_{x_1, x_2, \dots, x_{i-1}, x_i, 0, 0, \dots, 0, 1} || \dots || C_{x_1, x_2, \dots, x_{i-1}, x_i, 1, 1, \dots, 1, 1}$ . Therefore, the output is correct regardless of the selection of  $b_i$  and  $b'_i$ .

The protocol is secure in this case since Alice sees  $S'_2 = b_i \oplus b'_i \oplus 1$  and Bob sees  $S'_2 = b_i \oplus b'_i \oplus 1$  and  $S_1 = x_i \oplus b_i$  but  $b_i$  is an unknown random value for Bob.  $\square$

The protocol is  $2n + 1$  rounds since Step (d) of  $i$ -th iteration of the loop and Step (a) of  $(i + 1)$ -th iteration of the loop can be executed in the same round. The protocol uses  $4n + 2^{n+1}$  cards.

### Multiple Output Protocol

Multiple copies of output data of computation might be needed in some cases. For example, consider the case when we calculate  $w = (x \oplus y) \wedge (x' \oplus y')$ . We calculate  $z = x \oplus y$ ,  $z' = x' \oplus y'$ , and then  $w = z \wedge z'$ . To calculate  $w$ , two copies of output value  $z$  are necessary at calculating  $z = x \oplus y$ . Thus, multiple output protocols are necessary. A method to obtain  $m (> 1)$  copies of the output is preparing  $m$  copies of  $commit(y)$ . We show XOR and AND protocols that output multiple copies of the result.

#### Protocol 5 (multiple output XOR protocol)

Input: two copies of  $commit(x)$  and  $m (> 1)$  copies of  $commit(y)$ .

Output:  $m$  copies of  $commit(x \oplus y)$ .

The differences from Protocol 2 are as follows:

Step 1: Alice and Bob publicly put cards of one  $commit(x)$  and  $m$   $commit(y)$  into two envelopes. The left (right) cards of  $commit(x)$  and  $m$  copies of  $commit(y)$  are put into the left (right) envelope, respectively. The two envelopes have [ $commit(x), commit(y), commit(y), \dots, commit(y)$ ]. The remaining  $commit(x)$  is put into two new envelopes as in Protocol 2.

Step 3: At the end of Step 3, the cards in the envelopes are denoted [ $S'_2, S''_{2,1}, S''_{2,2}, \dots, S''_{2,m}$ ]

Step 4: The procedure until opening the cards of  $S'_2$  is the same as in Protocol 2.

If  $S'_2 = 0$ ,  $S''_{2,1}, S''_{2,2}, \dots, S''_{2,m}$  are the outputs of the protocol. If  $S'_2 = 1$ , swap the left and the right cards of  $S''_{2,i} (i = 1, 2, \dots, m)$  and the results are the outputs of the protocol.

The proof of correctness and security is just the same as the one for Protocol 2.

**Protocol 6** (multiple output AND protocol)

Input: two copies of  $commit(x)$  and  $m(> 1)$  copies of  $commit(y)$ .

Output:  $m$  copies of  $commit(x \wedge y)$ .

The differences from Protocol 3 are as follows:

Step 1: Alice and Bob publicly put cards of one  $commit(x)$  and  $m$   $commit(y)$  and  $commit(0)$  into two envelopes. The left card of  $commit(x)$  and  $m$  copies of  $commit(0)$  are put into the left envelope. The right card of  $commit(x)$  and  $m$  copies of  $commit(y)$  are put into the right envelope. The two envelopes have  $[commit(x), commit(0) || commit(0) || \dots || commit(0) || commit(y) || commit(y) || \dots || commit(y)]$ . The remaining  $commit(x)$  is put into two new envelopes as in Protocol 3.

Step 3: At the end of Step 3, the cards in the envelopes are denoted  $[S'_2, S''_{2,1} || S''_{2,2} || \dots || S''_{2,2m}]$ , where  $S''_{2,i}$  ( $i = 1, 2, \dots, 2m$ ) is one pair of cards.

Step 4: The procedure until opening the cards of  $S'_2$  is the same as in Protocol 3. If  $S'_2 = 0$ ,  $S''_{2,1}, S''_{2,2}, \dots, S''_{2,m}$  are the outputs of the protocol. If  $S'_2 = 1$ ,  $S''_{2,m+1}, S''_{2,m+2}, \dots, S''_{2,2m}$  are the outputs of the protocol.

The proof of correctness and security is just the same as the one for Protocol 3.

We can obtain an  $n$ -variable Boolean function calculation protocol that outputs  $m$  copies using the same idea using  $4n + 2^{n+1}m$  cards.

There might be some cases when the number of necessary copies of some value is changed during execution. For example, the players securely calculate Boolean function  $z, f_0$ , and  $f_1$ . The intermediate result  $z$  is opened, which is 0 or 1. Next, the players need to calculate  $f_z \wedge y_i$  ( $i = 1, \dots, n$ ). Since many copies of  $f_0$  ( $f_1$ ) are necessary only if  $z = 0$  ( $z = 1$ ) respectively, preparing many copies of  $f_0$  and  $f_1$  in advance is wasteful. In such cases, a copy protocol is used during execution. We show another protocol that directly increases the number of copies of input data using the XOR protocol.

**Protocol 7** (copy protocol)

Input: two copies of  $commit(x)$ .



Output:  $m(> 2)$  copies of  $commit(x)$ .

Execute Protocol 5 with input  $x$  and  $m$  copies of  $y = 0$ .

Since  $x \oplus 0 = x$ ,  $m$  copies of  $x$  are obtained.

Last, we show a method to obtain multiple copies of input  $x$  using envelopes.

**Protocol 8** (multiple input protocol)

Input:  $m(> 1)$   cards and  $m$   cards.

Output:  $m$  copies of  $commit(x)$ .

1. The players publicly put  $m$  face-down  () cards into the left (right) envelope, respectively.

2. The input player who has the private input value  $x$  privately swaps the two envelopes if  $x = 1$ , otherwise, does nothing. The input player makes the two envelopes public.
3. A player claims that the input player misbehaved and terminates the protocol if an envelope is opened. The players open the envelopes together. Two piles of face-down cards are obtained. When the players select one card from each of the piles, a copy of  $commit(x)$  can be obtained, thus  $m$  copies of  $commit(x)$  can be obtained.

Note that a malicious input player might input  $\bar{x}$  instead of  $x$ , but it is impossible to prevent such a cheat since no other player knows the secret value  $x$ .

When we calculate general logical functions using XOR, AND, and copy protocols, we need to prepare two copies of each input. Any number of copies of a value can be obtained by using the copy protocol at any time, if there are two copies of each value. Obtaining two copies of an intermediate output value can be realized by the above protocols; thus, any logical functions can be calculated securely using these protocols.

## Conclusion

This paper proposed new protocols using private operations that are secure against malicious players. We show logical XOR, logical AND, copy, and  $n$ -variable Boolean function calculation protocols that use envelopes as an additional tool. Since the envelopes are a very powerful tool to restrict swap executions, some malicious executions are corrected in the protocols.

We can consider weak tools for preventing the illegal opening of face-down cards, for example, seals on the marks of the cards. They cannot restrict swap executions. One of the open problems is considering secure protocols with such weak tools.

**Acknowledgements** The authors would like to thank anonymous referees who gave us valuable comments to improve 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. Abe, Y., Hayashi, Y.I., Mizuki, T., Sone, H.: Five-card and computations in committed format using only uniform cyclic shuffles. *N. Gener. Comput.* **39**(1), 97–114 (2021)

2. den Boer, B.: More efficient match-making and satisfiability the five card trick. In: Proc. of EURO-CRYPT '89, LNCS Vol. 434, pp. 208–217 (1990)
3. Bultel, X., Dreier, J., Dumas, J.G., Lafourcade, P., Miyahara, D., Mizuki, T., Nagao, A., Sasaki, T., Shinagawa, K., Sone, H.: Physical zero-knowledge proof for makaro. In: Proc. of 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2018), LNCS Vol.11201, pp. 111–125 (2018)
4. Cheung, E., Hawthorne, C., Lee, P.: Cs 758 project: Secure computation with playing cards (2013). [http://cdhawthorne.com/writings/secure\\_playing\\_cards.pdf](http://cdhawthorne.com/writings/secure_playing_cards.pdf)
5. Dumas, J.G., Lafourcade, P., Miyahara, D., Mizuki, T., Sasaki, T., Sone, H.: Interactive physical zero-knowledge proof for norinori. In: Proc. of 25th International Computing and Combinatorics Conference(COCOON 2019), LNCS Vol. 11653, pp. 166–177. Springer (2019)
6. Dvořák, P., Koucký, M.: Barrington plays cards: The complexity of card-based protocols. arXiv preprint [arXiv:2010.08445](https://arxiv.org/abs/2010.08445) (2020)
7. Francis, D., Aljunied, S.R., Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Necessary and sufficient numbers of cards for securely computing two-bit output functions. In: Proc. of Second International Conference on Cryptology and Malicious Security(Mycrypt 2016), LNCS Vol. 10311, pp. 193–211 (2017)
8. Hashimoto, Y., Nuida, K., Shinagawa, K., Inamura, M., Hanaoka, G.: Toward finite-runtime card-based protocol for generating hidden random permutation without fixed points. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **101**–A(9), 1503–1511 (2018)
9. Hashimoto, Y., Shinagawa, K., Nuida, K., Inamura, M., Hanaoka, G.: Secure grouping protocol using a deck of cards. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **101**(9), 1512–1524 (2018)
10. Ibaraki, T., Manabe, Y.: A more efficient card-based protocol for generating a random permutation without fixed points. In: Proc. of 3rd Int. Conf. on Mathematics and Computers in Sciences and in Industry (MCSI 2016), pp. 252–257 (2016)
11. Ishikawa, R., Chida, E., Mizuki, T.: Efficient card-based protocols for generating a hidden random permutation without fixed points. In: Proc. of 14th International Conference on Unconventional Computation and Natural Computation(UCNC 2015), LNCS Vol. 9252, pp. 215–226 (2015)
12. Isuzugawa, R., Miyahara, D., Mizuki, T.: Zero-knowledge proof protocol for cryptarithmic using dihedral cards. In: Proc. of 19th International Conference on Unconventional Computation and Natural Computation (UCNC 2021), LNCS Vol. 12984, pp. 51–67. Springer (2021)
13. Isuzugawa, R., Toyoda, K., Sasaki, Y., Miyahara, D., Mizuki, T.: A card-minimal three-input and protocol using two shuffles. In: Proc. of 27th International Computing and Combinatorics Conference (COCOON 2021), LNCS Vol. 13025, pp. 668–679. Springer (2021)
14. Kastner, J., Koch, A., Walzer, S., Miyahara, D., Hayashi, Y., Mizuki, T., Sone, H.: The minimum number of cards in practical card-based protocols. In: Proc. of Asiacrypt 2017, Part III, LNCS Vol. 10626, pp. 126–155 (2017)
15. Koch, A.: The landscape of optimal card-based protocols. IACR Cryptology ePrint Archive, Report 2018/951 (2018)
16. Koch, A., Walzer, S.: Private function evaluation with cards. Cryptology ePrint Archive, Report 2018/1113 (2018). <https://eprint.iacr.org/2018/1113>
17. Koch, A., Walzer, S.: Foundations for actively secure card-based cryptography. In: Proc. of 10th International Conference on Fun with Algorithms (FUN 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
18. Koch, A., Walzer, S., Härtel, K.: Card-based cryptographic protocols using a minimal number of cards. In: Proc. of Asiacrypt 2015, LNCS Vol. 9452, pp. 783–807 (2015)
19. Koyama, H., Toyoda, K., Miyahara, D., Mizuki, T.: New card-based copy protocols using only random cuts. In: Proceedings of the 8th ACM on ASIA Public-Key Cryptography Workshop, APKC '21, p. 13?22. Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3457338.3458297>
20. Kurosawa, K., Shinozaki, T.: Compact card protocol. In: Proc. of 2017 Symposium on Cryptography and Information Security(SCIS 2017), pp. 1A2–6 (2017). (In Japanese)
21. Lafourcade, P., Miyahara, D., Mizuki, T., Robert, L., Sasaki, T., Sone, H.: How to construct physical zero-knowledge proofs for puzzles with a “single loop” condition. Theoretical Computer Science **888**, 41–55 (2021) <https://doi.org/10.1016/j.tcs.2021.07.019>. <https://www.sciencedirect.com/science/article/pii/S0304397521004242>



22. Manabe, Y., Ono, H.: Secure card-based cryptographic protocols using private operations against malicious players. In: Proc. of 13th International Conference on Information Technology and Communications Security (SecITC 2020), LNCS Vol. 12596, pp. 55–70. Springer (2020)
23. Manabe, Y., Ono, H.: Card-based cryptographic protocols for three-input functions using private operations. In: Proc. of 32nd International Workshop on Combinatorial Algorithms (IWCOA 2021), LNCS Vol. 12757, pp. 469–484. Springer (2021)
24. Manabe, Y., Ono, H.: Card-based cryptographic protocols with a standard deck of cards using private operations. In: Proc. of 18th International Colloquium on Theoretical Aspects of Computing (ICTAC 2021), LNCS Vol.12819. Springer (2021)
25. Marcedone, A., Wen, Z., Shi, E.: Secure dating with four or fewer cards. IACR Cryptology ePrint Archive, Report 2015/1031 (2015)
26. Miyahara, D., Haneda, H., Mizuki, T.: Card-based zero-knowledge proof protocols for graph problems and their computational model. In: Proc. of 15th International Conference on Provable and Practical Security (ProvSec 2021), LNCS Vol.13059. Springer (2021)
27. Miyahara, D., Hayashi, Y.I., Mizuki, T., Sone, H.: Practical card-based implementations of yao's millionaire protocol. Theoret. Comput. Sci. **803**, 207–221 (2020)
28. Miyahara, D., Robert, L., Lafourcade, P., Takeshige, S., Mizuki, T., Shinagawa, K., Nagao, A., Sone, H.: Card-based zkp protocols for takuzu and juosan. In: Proc. of 10th International Conference on Fun with Algorithms (FUN 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
29. Miyahara, D., Sasaki, T., Mizuki, T., Sone, H.: Card-based physical zero-knowledge proof for kakuro. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **102**(9), 1072–1078 (2019)
30. Mizuki, T.: Applications of card-based cryptography to education. In: IEICE Technical Report ISEC2016-53, pp. 13–17 (2016). (In Japanese)
31. Mizuki, T.: Card-based protocols for securely computing the conjunction of multiple variables. Theoret. Comput. Sci. **622**, 34–44 (2016)
32. Mizuki, T., Asiedu, I.K., Sone, H.: Voting with a logarithmic number of cards. In: Proc. of 12th International Conference on Unconventional Computing and Natural Computation (UCNC 2013), LNCS Vol. 7956, pp. 162–173 (2013)
33. Mizuki, T., Kumamoto, M., Sone, H.: The five-card trick can be done with four cards. In: Proc. of Asiacrypt 2012, LNCS Vol.7658, pp. 598–606 (2012)
34. Mizuki, T., Shizuya, H.: A formalization of card-based cryptographic protocols via abstract machine. Int. J. Inf. Secur. **13**(1), 15–23 (2014)
35. Mizuki, T., Shizuya, H.: Practical card-based cryptography. In: Proc. of 7th International Conference on Fun with Algorithms (FUN2014), LNCS Vol. 8496, pp. 313–324 (2014)
36. Mizuki, T., Shizuya, H.: Computational model of card-based cryptographic protocols and its applications. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **100**(1), 3–11 (2017)
37. Mizuki, T., Sone, H.: Six-card secure and and four-card secure xor. In: Proc. of 3rd International Workshop on Frontiers in Algorithms (FAW 2009), LNCS Vol. 5598, pp. 358–369 (2009)
38. Moran, T., Naor, M.: Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In: Proc. of EUROCRYPT 2006, LNCS Vol. 4004, pp. 88–108. Springer (2006)
39. Murata, S., Miyahara, D., Mizuki, T., Sone, H.: Efficient generation of a card-based uniformly distributed random derangement. In: Proc. of 15th International Workshop on Algorithms and Computation (WALCOM 2021), LNCS Vol. 12635, pp. 78–89. Springer International Publishing, Cham (2021)
40. Nakai, T., Misawa, Y., Tokushige, Y., Iwamoto, M., Ohta, K.: How to solve millionaires' problem with two kinds of cards. N. Gener. Comput. **39**(1), 73–96 (2021)
41. Nakai, T., Shirouchi, S., Iwamoto, M., Ohta, K.: Four cards are sufficient for a card-based three-input voting protocol utilizing private sends. In: Proc. of 10th International Conference on Information Theoretic Security (ICITS 2017), LNCS Vol. 10681, pp. 153–165 (2017)
42. Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Card-based protocols for any boolean function. In: Proc. of 15th International Conference on Theory and Applications of Models of Computation (TAMC 2015), LNCS Vol. 9076, pp. 110–121 (2015)
43. Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Securely computing three-input functions with eight cards. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **98**(6), 1145–1152 (2015)
44. Nishida, T., Mizuki, T., Sone, H.: Securely computing the three-input majority function with eight cards. In: Proc. of 2nd International Conference on Theory and Practice of Natural Computing (TPNC 2013), LNCS Vol. 8273, pp. 193–204 (2013)

45. Nishimura, A., Hayashi, Y.I., Mizuki, T., Sone, H.: Pile-shifting scramble for card-based protocols. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **101**(9), 1494–1502 (2018)
46. Nishimura, A., Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Card-based protocols using unequal division shuffles. *Soft. Comput.* **22**(2), 361–371 (2018)
47. Ono, H., Manabe, Y.: Efficient card-based cryptographic protocols for the millionaires' problem using private input operations. In: *Proc. of 13th Asia Joint Conference on Information Security(AsiaJCIS 2018)*, pp. 23–28 (2018)
48. Ono, H., Manabe, Y.: Card-based cryptographic logical computations using private operations. *N. Gener. Comput.* **39**(1), 19–40 (2021)
49. Ono, H., Manabe, Y.: Minimum round card-based cryptographic protocols using private operations. *Cryptography* **5**(3) (2021)
50. Robert, L., Miyahara, D., Lafourcade, P., Mizuki, T.: Physical zero-knowledge proof for suguru puzzle. In: *Proc. of 22th International Symposium on Stabilizing, Safety, and Security of Distributed Systems(SSS 2020)*, LNCS Vol. 12514, pp. 235–247. Springer (2020)
51. Robert, L., Miyahara, D., Lafourcade, P., Mizuki, T.: Interactive physical zkp for connectivity:applications to nurikabe and hitori. In: *Proc. of 17th International Conference on Computability in Europe(CiE 2021)*, LNCS Vol. 12813, pp. 373–384. Springer International Publishing, Cham (2021)
52. Ruangwises, S.: An improved physical zkp for nonogram. arXiv preprint [arXiv:2106.14020](https://arxiv.org/abs/2106.14020) (2021)
53. Ruangwises, S.: Two standard decks of playing cards are sufficient for a zkp for sudoku. arXiv preprint [arXiv:2106.13646](https://arxiv.org/abs/2106.13646) (2021)
54. Ruangwises, S., Itoh, T.: And protocols using only uniform shuffles. In: *Proc. of 14th International Computer Science Symposium in Russia(CSR 2019)*, LNCS Vol. 11532, pp. 349–358 (2019)
55. Ruangwises, S., Itoh, T.: Physical zero-knowledge proof for numberlink puzzle and k vertex-disjoint paths problem. *N. Gener. Comput.* **39**(1), 3–17 (2021)
56. Ruangwises, S., Itoh, T.: Physical zero-knowledge proof for ripple effect. *Theoretical Computer Science* (2021) <https://doi.org/10.1016/j.tcs.2021.09.034>. <https://www.sciencedirect.com/science/article/pii/S0304397521005557>
57. Ruangwises, S., Itoh, T.: Physical zkp for connected spanning subgraph: applications to bridges puzzle and other problems. In: *Proc. of 19th International Conference on Unconventional Computation and Natural Computation (UCNC 2021)*, LNCS Vol. 12984, pp. 149–163. Springer (2021)
58. Ruangwises, S., Itoh, T.: Securely computing the n-variable equality function with 2n cards. *Theoretical Computer Science* **887**, 99–110 (2021) <https://doi.org/10.1016/j.tcs.2021.07.007>. <https://www.sciencedirect.com/science/article/pii/S0304397521004126>
59. Sasaki, T., Miyahara, D., Mizuki, T., Sone, H.: Efficient card-based zero-knowledge proof for sudoku. *Theoret. Comput. Sci.* **839**, 135–142 (2020)
60. Shimizu, Y., Kishi, Y., Sasaki, T., Fujioka, A.: Card-based cryptographic protocols with private operations which can prevent malicious behaviors. In: *IEICE Technical Report ISEC2017-113*, pp. 129–135 (2018). (In Japanese)
61. Shinagawa, K., Mizuki, T.: The six-card trick:secure computation of three-input equality. In: *Proc. of 21st International Conference on Information Security and Cryptology (ICISC 2018)*, LNCS Vol. 11396, pp. 123–131 (2018)
62. Shinagawa, K., Mizuki, T.: Secure computation of any boolean function based on any deck of cards. In: *Proc. of 13th International Workshop on Frontiers in Algorithmics (FAW 2019)*, LNCS Vol. 11458, pp. 63–75. Springer (2019)
63. Shinagawa, K., Nuida, K.: A single shuffle is enough for secure card-based computation of any boolean circuit. *Discret. Appl. Math.* **289**, 248–261 (2021)
64. Shinoda, Y., Miyahara, D., Shinagawa, K., Mizuki, T., Sone, H.: Card-based covert lottery. In: *Proc. of 13th International Conference on Information Technology and Communications Security(SecITC 2020)*, LNCS Vol. 12596, pp. 257–270. Springer (2020)
65. Shirouchi, S., Nakai, T., Iwamoto, M., Ohta, K.: Efficient card-based cryptographic protocols for logic gates utilizing private permutations. In: *Proc. of 2017 Symposium on Cryptography and Information Security(SCIS 2017)*, pp. 1A2–2 (2017). (In Japanese)
66. Takashima, K., Abe, Y., Sasaki, T., Miyahara, D., Shinagawa, K., Mizuki, T., Sone, H.: Card-based protocols for secure ranking computations. *Theoret. Comput. Sci.* **845**, 122–135 (2020)
67. Takashima, K., Miyahara, D., Mizuki, T., Sone, H.: Actively revealing card attack on card-based protocols. *Natural Computing* pp. 1–14 (2021)
68. Toyoda, K., Miyahara, D., Mizuki, T., Sone, H.: Six-card finite-runtime xor protocol with only random cut. In: *Proc. of the 7th ACM Workshop on ASIA Public-Key Cryptography*, pp. 2–8 (2020)

69. Watanabe, Y., Kuroki, Y., Suzuki, S., Koga, Y., Iwamoto, M., Ohta, K.: Card-based majority voting protocols with three inputs using three cards. In: Proc. of 2018 International Symposium on Information Theory and Its Applications (ISITA), pp. 218–222. IEEE (2018)
70. Yasunaga, K.: Practical card-based protocol for three-input majority. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E103.A**(11), 1296–1298 (2020). <https://doi.org/10.1587/transfun.2020EAL2025>

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