Card-based Cryptographic Protocols for Three-input Functions with a Standard Deck of Cards Using Private Operations

Naoki Kobayashi[1] and Yoshifumi Manabe[1][0000−0002−6312−257X]

School of Informatics, Kogakuin University,
Shinjuku, Tokyo 163-8677 Japan. manabe@cc.kogakuin.ac.jp

Abstract. This paper shows card-based cryptographic protocols to calculate several Boolean functions with a standard deck of cards using private operations. They are multi-party secure computations executed by multiple semi-honest players without computers. The protocols use private operations that are executed by a player at a place where the other players cannot see. Most card-based cryptographic protocols use a special deck of cards that consists of many cards with two kinds of marks. Though these protocols are simple and efficient, the users need to prepare such special cards. Few protocols were shown that use a standard deck of playing cards, though the protocols with a standard deck of cards can be easily executed in our daily lives. It was shown that logical AND, logical XOR, and copy protocols can be executed with the minimum number of cards. However, the protocols for complicated functions are not known. This paper shows that by using private operations, all of the following Boolean functions can be calculated without additional cards other than the input cards: (1) any three input Boolean functions, (2) half adder and full adder, and (3) any n-input symmetric Boolean functions. The results show the effectiveness of private operations in card-based cryptographic protocols.

Keywords: card-based cryptographic protocols · multi-party secure computation · Boolean functions · half adder · symmetric functions · private operations · standard deck of cards.

1 Introduction

1.1 Overview of Card-based Cryptographic Protocols

Card-based cryptographic protocols [26, 28] were proposed in which physical cards are used instead of computers to securely compute 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, 21] to accelerate the social implementation of advanced cryptography [6]. den Boer [3] first showed a five-card protocol to securely compute the logical AND of two inputs. Since then,

many protocols have been proposed to realize primitives to compute any Boolean functions [8, 11, 29, 34, 37, 38, 47, 48] and computate a specific class of Boolean functions [1, 2, 5, 7, 13–15, 18, 22, 25, 35, 36, 40, 41, 44–46, 50, 51].

This paper considers computations of (1) any three input Boolean functions, (2) half adder and full adder, and (3) any n-input symmetric Boolean functions. No additional cards are necessary to calculate these functions with a standard deck of cards when we use private operations.

Note that in this paper, all players are assumed to be semi-honest. Few works are done for the case when some players are malicious or make mistakes [10, 16, 24, 27, 30, 49].

1.2 Standard Deck of Cards

Most of the above works are based on a two-color card model. In the two-color card model, there are two kinds of cards, $\boxed{\clubsuit}$ and $\boxed{\heartsuit}$. Cards of the same marks cannot be distinguished. In addition, the back of both types of cards is $\boxed{?}$. It is impossible to determine the mark in the back of a given card of $\boxed{?}$. Though the model is simple, such special cards are not available in our daily lives.

To solve the problem, card-based cryptographic protocols using a standard deck of playing cards were shown [9, 12, 13, 19, 20, 23, 30, 31, 33, 47]. Playing cards are available at many houses and are easy to buy. The standard deck of playing cards is also used for zero-knowledge proof of puzzle solutions [39, 42]. This paper discusses protocols to calculate logical functions. Niemi and Renvall first showed protocols that use a standard deck of playing cards [33]. They showed logical XOR, logical AND, and copy protocols since any Boolean functions can be realized by a combination of these protocols. Their protocols are 'Las Vegas' type protocols, that is, the execution times of the protocols are not limited. The protocols are expected to terminate within a finite time and the efficiency of these protocols is evaluated by the expected execution time. However, if the sequence of the random numbers is bad, the protocols do not terminate forever. Mizuki showed fixed time logical XOR, logical AND, and copy protocols [23]. Though the number of cards used by the XOR protocol is the minimum, the ones used by the logical AND and copy protocols are not the minimum. Koch et al. showed a four-card 'Las Vegas' type AND protocol and it is impossible to obtain a four-card finite time protocol with the model without private operations [9]. Koyama et al. showed a three-input 'Las Vegas' type AND protocol with the minimum number of cards [12]. Koyama et al. showed an efficient 'Las Vegas' type copy protocol [13]. Shinagawa and Mizuki showed protocols to compute any n-variable function using a standard deck of playing cards and a deck of UNO[1] cards [47]. Miyahara et al. showed a protocol that solves Yao's millionares' problem using a standard deck of playing cards [19]. Miyahara and Mizuki showed new protocols that use a special primitive that opens the suit of a playing card [20]. This paper discusses protocols that publically or privately open the cards. Another class of protocols is considered, in which each player knows his/her private data, and

[1] https://www.letsplayuno.com

the player privately inputs the data to the protocol. Nakai et al. showed AND, XOR, and majority protocols [31].

1.3 Private Operations

Randomization or a private operation is the most important primitive in these card-based protocols. If every primitive executed in a card-based protocol is deterministic and public, the relationship between the private input values and the output values is known to the players. When the output values are disclosed, the private input values can be known to the players from the relationship. Thus, all protocols need some random or private operation.

First, public randomization primitives have been discussed then recently, private operations are considered. Many protocols use random bisection cuts [29], which randomly execute swapping two decks of cards or not swapping. If the random value used in the randomization is disclosed, the secret input value is known to the players. If some player privately brings a high-speed camera, the random value selected by the randomization might be known by analyzing the image. Though the size of a high-speed camera is very large, the size might become very small shortly. To prepare for the situation, we need to consider using private operations.

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 the millionaires' problem [32]. Using three private operations shown later, committed-input and committed-output logical AND, logical XOR, and copy protocols can be achieved with the minimum number of cards on the two-color card model [37].

For the primitives of logical AND, logical XOR, and copy operation, the minimum number of cards is achieved with a standard deck of cards using private operations [17]. So the research question is whether we can achieve the minimum number of cards for complicated calculations.

1.4 Our Results

This paper shows new card-based protocols with a standard deck of cards using private operations to calculate (1) any three input Boolean functions, (2) half adder and full adder, and (3) any n-input symmetric Boolean functions. All of these protocols need no additional cards other than the input cards. Thus these protocols are optimal in regard to the number of cards.

In Section 2 basic definitions and the private operations introduced by [37] are shown. Then, the sub-protocols shown in [17] and used in this paper are stated. Section 3 shows protocols to calculate three input Boolean functions. Section 4 shows protocols to calculate half and full adder, and n-input symmetric Boolean functions. Section 5 concludes the paper.

2 Preliminaries

2.1 Basic Notations

This section gives the notations and basic definitions of card-based protocols with a standard deck of cards. A deck of playing cards consists of 52 distinct mark cards, which are named as 1 to 52. The number of each card (for example, 1 is the ace of spade and 52 is the king of club) is common knowledge among the players. The back of all cards is the same $\boxed{?}$. It is impossible to determine the mark in the back of a given card of $\boxed{?}$.

One-bit data is represented by two cards as follows: $\boxed{\mathbf{i}}\boxed{\mathbf{j}} = 0$ and $\boxed{\mathbf{j}}\boxed{\mathbf{i}} = 1$ if $i < j$.

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 $\underbrace{\boxed{?}\boxed{?}}_{x}$. The base of a commitment is the pair of cards used for the commitment. If card i and $j (i < j)$ are used to set $commit(x)$ (That is, set $\boxed{\mathbf{i}}\boxed{\mathbf{j}}$ if $x = 0$ and set $\boxed{\mathbf{j}}\boxed{\mathbf{i}}$ if $x = 1$), the commitment is written as $commit(x)^{\{i,j\}}$ and written as $\underbrace{\boxed{?}\boxed{?}}_{x^{\{i,j\}}}$. When the base information is obvious or unnecessary, it is not written.

Note that when these two cards are swapped, $commit(\bar{x})^{\{i,j\}}$ can be obtained. Thus, logical negation can be computed 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 $S = s_1, s_2, \ldots, s_n$, where s_i is i-th card of the sequence. $S = \underbrace{\boxed{?}}_{s_1}\underbrace{\boxed{?}}_{s_2}\underbrace{\boxed{?}}_{s_3}\ldots,\underbrace{\boxed{?}}_{s_n}$. 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.

All protocols are executed by two players, Alice and Bob. The players are semi-honest, that is, they obey the rules of the protocols but try to obtain secret values.

The inputs of the protocols are given in a committed manner, that is, the players do not know the input values. If a player knows his secure input value x, the player just makes a commitment of x, and the protocols in this paper can be used. The output of the protocol must be given in a committed format so that the result can be used as an input to further computation. If the players need to obtain the output value, they just open the committed output. Thus committed output is desirable.

A protocol is secure when the following two conditions are satisfied: (1) If the output cards are not opened, each player obtains no information about the private input values from the view of the protocol for the player (the sequence of the cards opened to the player). (2) When the output cards are opened, each player obtains no additional information about the private input values other than the information by the output of the protocol. For example, if the output cards of an AND protocol for input x and y are opened and the value is 1, the

players can know that $x = 1$ and $y = 1$. If the output value is 0, the players must not know whether the input (x, y) is $(0, 0)$, $(0, 1)$, or $(1, 0)$.

The following protocols use random numbers. Random numbers can be generated without computers using coin-flipping or some similar methods. During the protocol executions, cards are sent and received between the players. The communication is executed by sending the cards between the players to avoid information leakage during the communication. If the players are not in the same place during the protocol execution, a trusted third party (for example, the post office) is necessary to send and receive cards between players.

2.2 Private Operations

We show three private operations introduced in [37]: 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 $S_0 = s_1, s_2, \ldots, s_{2m}$. *A player selects a random bit* $b \in \{0, 1\}$ *and outputs*

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

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 $S_0 = commit(x)$, 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 $\underbrace{\boxed{?}\boxed{?}}_{\overline{x}}$.

Note that a private random bisection cut is the same as the random bisection cut [29], but the operation is executed in a hidden place.

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

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

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 $S_2 = s_1, s_2, \ldots, s_{2m}$ and a bit $b \in \{0, 1\}$. A player outputs

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

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.

2.3 Opaque Commitment Pair

An opaque commitment pair is defined as a useful situation to design a secure protocol using a standard deck of cards [23]. It is a pair of commitments whose bases are unknown to a player. Let us consider the following two commitments using cards i, j, i' and j'. The left (right) commitment has value x (y), respectively, but it is unknown that (1) the left (right) commitment is made using i and j (i' and j'), respectively, or (2) the left (right) commitment is made using i' and j' (i and j), respectively. Such a pair of commitments is called an opaque commitment pair and written as $commit(x)^{\{i,j\},\{i',j'\}}||commit(y)^{\{i,j\},\{i',j'\}}$.

The protocols in this paper use a little different kind of pair, called semi-opaque commitment pair. A player thinks a pair is an opaque commitment pair but another player knows the bases of the commitments. Let us consider the case when a protocol is executed by Alice and Bob. Bob privately makes the pair of commitments with the knowledge of x and y. For example, Bob randomly selects a bit $b \in \{0, 1\}$ and

$$S = \begin{cases} commit(x)^{\{i,j\}}||commit(y)^{\{i',j'\}} & \text{if } b = 0 \\ commit(x)^{\{i',j'\}}||commit(y)^{\{i,j\}} & \text{if } b = 1 \end{cases}$$

then $S = commit(x)^{\{i,j\},\{i',j'\}}||commit(y)^{\{i,j\},\{i',j'\}}$ for Alice. Such a pair is called a semi-opaque commitment pair and written as $commit(x)^{\{i,j\},\{i',j'\}|Alice}||$ $commit(y)^{\{i,j\},\{i',j'\}|Alice}$, where the name(s) of the players who think the pair is a opaque commitment pair is written. Note that a name is not written does not mean the player knows the bases of the commitments. For example, the above example says nothing about whether Bob knows the bases or not. Note that the name of the player is written with the initial when it is not ambiguous.

2.4 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 [38]. 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 himself 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. If the local execution needs many operations, for example, $O(n)$ operations where n is the size of the problem, we might need another criterion to consider the cost of local executions.

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

Protocol 1 *(XOR protocol) [17]*
Input: $commit(x)^{\{1,2\}}$ *and* $commit(y)^{\{3,4\}}$.
Output: $commit(x \oplus y)^{\{1,2\}}$.

1. *Alice executes a private random bisection cut on* $commit(x)^{\{1,2\}}$ *and* $commit(y)^{\{3,4\}}$ *using the same random bit* $b \in \{0,1\}$. *The result is* $commit(x \oplus b)^{\{1,2\}}$ *and* $commit(y \oplus b)^{\{3,4\}}$. *Alice sends these cards to Bob.*
2. *Bob executes a private reveal on* $commit(y \oplus b)^{\{3,4\}}$. *Bob sees* $y \oplus b$. *Bob executes a private reverse cut on* $commit(x \oplus b)^{\{1,2\}}$ *using* $y \oplus b$. *The result is* $commit((x \oplus b) \oplus (y \oplus b))^{\{1,2\}} = commit(x \oplus y)^{\{1,2\}}$.

The number of cards is four since no cards are used other than the inputs.

Let us consider the time complexity of the protocol. The first round ends at the instant when Alice sends $commit(x \oplus b)^{\{1,2\}}$ and $commit(y \oplus b)^{\{3,4\}}$ to Bob. The second round begins with receiving the cards by Bob. The number of rounds of this protocol is two.

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.

2.5 Protocols for AND, Copy, and Other Boolean Functions

This subsection shows the sub-protocols presented in [17] used in this paper's protocols. The correctness proof is shown in [17].

AND Protocol Before showing the AND protocol, a subprotocol to fix the base of commitments is shown.

Protocol 2 *(Base-fixed protocol) [17]*
Input: $commit(x)^{\{1,2\},\{3,4\}|A}||commit(y)^{\{1,2\},\{3,4\}|A}$.
(Note: y is a private value that must not be known to the players)
Output: $commit(x)^{\{1,2\}}$.

1. *Bob executes a private random bisection cut on both pairs using two distinct random bits* $br_1, br_2 \in \{0,1\}$. *The result* $S_1 = commit(x \oplus br_1)^{\{1,2\},\{3,4\}|A}||commit(y \oplus br_2)^{\{1,2\},\{3,4\}|A}$. *Bob sends* S_1 *to Alice.*
2. *Alice executes a private reveal on* S_1. *Alice sees* $x \oplus br_1$ *and* $y \oplus br_2$. *If the base of the left pair is* $\{1,2\}$, *Alice just faces down the left pair and the cards,* S_2, *are the result. Otherwise, the base of the right pair is* $\{1,2\}$. *Alice makes* $S_2 = commit(x \oplus br_1)^{\{1,2\}}$ *using the right cards. Alice sends* S_2 *to Bob.*
3. *Bob executes a private reverse cut using* br_1 *on* S_2. *The result is* $commit(x)^{\{1,2\}}$.

Using the base-fixed protocol, the AND protocol in [17] is shown below.

Protocol 3 *(AND protocol) [17]*
Input: $commit(x)^{\{1,2\}}$ *and* $commit(y)^{\{3,4\}}$.
Output: $commit(x \wedge y)^{\{1,2\}}$.

1. *Alice executes a private random bisection cut on* $commit(x)^{\{1,2\}}$ *using random bit* a_1. *Alice sends the results,* $S_1 = commit(x \oplus a_1)^{\{1,2\}}$ *and* $S_2 = commit(y)^{\{3,4\}}$ *to Bob.*
2. *Bob executes a private reveal on* S_1. *Bob sees* $x \oplus a_1$. *Bob privately sets*

$$S_{3,0} = \begin{cases} commit(0)^{\{1,2\}}||commit(y)^{\{3,4\}} & \text{if } x \oplus a_1 = 0 \\ commit(y)^{\{3,4\}}||commit(0)^{\{1,2\}} & \text{if } x \oplus a_1 = 1 \end{cases}$$

Bob sends $S_{3,0}$ *to Alice.*
3. *Alice executes private random bisection cuts on each of pairs in* $S_{3,0}$ *using two distinct random bits* a_2 *and* a_3. *Let the result be* $S_{3,1}$.

$$S_{3,1} = \begin{cases} commit(0 \oplus a_2)^{\{1,2\}}||commit(y \oplus a_3)^{\{3,4\}} & \text{if } x \oplus a_1 = 0 \\ commit(y \oplus a_2)^{\{3,4\}}||commit(0 \oplus a_3)^{\{1,2\}} & \text{if } x \oplus a_1 = 1 \end{cases}$$

Alice sends $S_{3,1}$ *to Bob.*
4. *Bob randomly selects bit* $b_1 \in \{0,1\}$. *Bob reveals* $S_{3,1}$ *and exchanges the bases of the two commitments if* $b_1 = 1$. *Let the result be* $S_{3,2}$.

$$S_{3,2} = \begin{cases} commit(0 \oplus a_2)^{\{1,2\},\{3,4\}|A}||commit(y \oplus a_3)^{\{1,2\},\{3,4\}|A} & \text{if } x \oplus a_1 = 0 \\ commit(y \oplus a_2)^{\{1,2\},\{3,4\}|A}||commit(0 \oplus a_3)^{\{1,2\},\{3,4\}|A} & \text{if } x \oplus a_1 = 1 \end{cases}$$

Bob sends $S_{3,2}$ *to Alice.*

5. *Alice executes private reverse cuts on the two pairs of $S_{3,2}$ using a_2 and a_3, respectively. Let the result be S_4.*

$$S_4 = \begin{cases} commit(0)^{\{1,2\},\{3,4\}|A} || commit(y)^{\{1,2\},\{3,4\}|A} \text{ if } x \oplus a_1 = 0 \\ commit(y)^{\{1,2\},\{3,4\}|A} || commit(0)^{\{1,2\},\{3,4\}|A} \text{ if } x \oplus a_1 = 1 \end{cases}$$

Alice then executes a private reverse selection on S_4 using a_1. Let S_5 be the result and the remaining two cards be S_6. The result $S_5 = commit(y)^{\{1,2\},\{3,4\}|A}$ if ($a_1 = 0$ and $x \oplus a_1 = 1$) or ($a_1 = 1$ and $x \oplus a_1 = 0$). The condition equals $x = 1$.
$S_5 = commit(0)^{\{1,2\},\{3,4\}|A}$ if ($a_1 = 0$ and $x \oplus a_1 = 0$) or ($a_1 = 1$ and $x \oplus a_1 = 1$). The condition equals $x = 0$. Thus,

$$S_5 = \begin{cases} commit(y)^{\{1,2\},\{3,4\}|A} \text{ if } x = 1 \\ commit(0)^{\{1,2\},\{3,4\}|A} \text{ if } x = 0 \end{cases}$$

$$= commit(x \wedge y)^{\{1,2\},\{3,4\}|A}$$

Alice sends S_5 and S_6 to Bob.
6. *Bob and Alice execute Protocol 2 (Base-fixed protocol) to $S_5||S_6$. Then they obtain $commit(x \wedge y)^{\{1,2\}}$.*

COPY Protocol

Protocol 4 *(Copy protocol) [17]*
Input: $commit(x)^{\{1,2\}}$ and two new cards 3 and 4.
Output: $commit(x)^{\{1,2\}}$ and $commit(x)^{\{3,4\}}$

1. *Alice executes a private random bisection cut on $commit(x)^{\{1,2\}}$. Let b be the random bit Alice selects. Alice sends the result, $commit(x \oplus b)^{\{1,2\}}$, to Bob.*
2. *Bob executes a private reveal on $commit(x \oplus b)^{\{1,2\}}$ and sees $x \oplus b$. Bob privately makes $commit(x \oplus b)^{\{3,4\}}$. Bob sends $commit(x \oplus b)^{\{1,2\}}$ and $commit(x \oplus b)^{\{3,4\}}$ to Alice.*
3. *Alice executes a private reverse cut on each of the pairs using b. The result is $commit(x)^{\{1,2\}}$ and $commit(x)^{\{3,4\}}$.*

The protocol is three rounds.

Preserving an Input In the above protocols to calculate Boolean functions, the input commitment values are lost. If the input is not lost, the input commitment can be used as an input to another calculation. Thus input preserving calculation is discussed [34, 37].

In the XOR protocol, $commit(y \oplus b)^{\{3,4\}}$ is no longer necessary after Bob sets the result. Thus, Bob can send back $commit(y \oplus b)^{\{3,4\}}$ to Alice. Then, Alice can recover $commit(y)^{\{3,4\}}$ using the private reverse cut. In this modified protocol, the output is $commit(x \oplus y)^{\{1,2\}}$ and $commit(y)^{\{3,4\}}$ without additional cards.

By exchanging the roles of x and y, the output can be $commit(x \oplus y)^{\{3,4\}}$ and $commit(x)^{\{1,2\}}$.

An input preserving AND protocol can be obtained using the idea in [34]. When we execute the AND protocol, two cards are selected by Alice at the final step. The remaining two cards are used to recover an input value. The unused two cards' value is

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

thus the output is $commit(\overline{x} \wedge y)$.

Execute the above input preserving XOR protocol for these two output values so that the input $x \wedge y$ is preserved. The output of the XOR protocol is $(x \wedge y) \oplus (\overline{x} \wedge y) = y$. Thus, input y can be recovered without additional cards. By executing a base-fixed protocol, the output can be $commit(x \wedge y)^{\{1,2\}}$ and $commit(y)^{\{3,4\}}$.

***n*-input Boolean Functions** Since any 2-input Boolean function, NOT, and COPY can be executed, any n-input Boolean function can be calculated by the combination of the above protocols using $2n + 4$ cards by the idea in [34, 37].

Any Boolean function $f(x_1, x_2, \ldots, x_n)$ can be represented as follows: $f(x_1, x_2, \ldots, x_n) = \bar{x_1} \wedge \bar{x_2} \wedge \cdots \bar{x_n} \wedge f(0, 0, \ldots, 0) \oplus x_1 \wedge \bar{x_2} \wedge \cdots \bar{x_n} \wedge f(1, 0, \ldots, 0) \oplus \bar{x_1} \wedge x_2 \wedge \cdots \bar{x_n} \wedge f(0, 1, \ldots, 0) \oplus \cdots \oplus x_1 \wedge x_2 \wedge \cdots x_n \wedge f(1, 1, \ldots, 1)$.

Since the terms with $f(i_1, i_2, \ldots, i_n) = 0$ can be removed, this function f can be written as $f = \bigoplus_{i=1}^{k} v_1^i \wedge v_2^i \wedge \cdots \wedge v_n^i$, where $v_j^i = x_j$ or $\bar{x_j}$. Let us write $T_i = v_1^i \wedge v_2^i \wedge \cdots \wedge v_n^i$. The number of terms $k (< 2^n)$ depends on f.

Protocol 5 *(Protocol for any n-variable Boolean function [17]*

Input: $commit(x_i)^{\{2i+3,2i+4\}} (i = 1, 2, \ldots, n)$.

Output: $commit(f(x_1, x_2, \ldots, x_n))^{\{1,2\}}$.

The additional four cards (two pairs of cards) 1,2,3, and 4 are used as follows.

1 and 2 store the intermediate value to compute f.

3 and 4 store the intermediate value to compute T_i.

Execute the following steps for $i = 1, 2, \ldots, k$.

1. *Copy* v_1^i *from the input* $commit(x_1)$ *as* $commit(v_1^i)^{\{3,4\}}$. *(Note that if* v_1^i *is* $\bar{x_1}$, *NOT is taken after the copy).*
2. *For* $j = 2, \ldots, n$, *execute the following procedure: Execute the input preserving AND protocol to* $commit(\cdot)^{\{3,4\}}$ *and* $commit(v_j^i)$ *so that input* $commit(v_j^i)$ *is preserved. The result is stored as* $commit(\cdot)^{\{3,4\}}$. *(Note that if* v_j^i *is* $\bar{x_j}$, *NOT is taken before the AND protocol, and NOT is taken again for the preserved input.)*
   At the end of this step, T_i *is obtained as* $commit(v_1^i \wedge v_2^i \wedge \cdots \wedge v_n^i)^{\{3,4\}}$.
3. *If* $i = 1$, *copy* $commit(\cdot)^{\{3,4\}}$ *to* $commit(\cdot)^{\{1,2\}}$. *If* $i > 1$, *apply the XOR protocol between* $commit(\cdot)^{\{3,4\}}$ *and* $commit(\cdot)^{\{1,2\}}$. *The result is stored as* $commit(\cdot)^{\{1,2\}}$.

At the end of the protocol, $commit(f(x_1, x_2, \ldots x_n))^{\{1,2\}}$ *is obtained.*

3 Protocols for Three-input Boolean Functions

This section shows protocols for three input Boolean functions. The arguments to show the protocols with six cards are just the same as the one in [35]. The main difference is that logical AND can be calculated by four cards using private operations. In our protocols, no additional cards are necessary other than the cards for inputs.

There are $2^{2^3} = 256$ different functions with three inputs. However, some of these functions are equivalent by replacing variables and taking negations. NPN-classification [43] was considered to reduce the number of different functions considering the equivalence class of functions. The rules of NPN-classification are as follows.

1. Negation of input variables (Example: $x_i \leftrightarrow \overline{x_i}$).
2. Permutations of input variables (Example: $x_i \leftrightarrow x_j$).
3. Negation of the output ($f \leftrightarrow \overline{f}$).

For example, consider $f_1(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee x_3$. Several functions in the same equivalence class that includes f_1 are: $f_2 = (\overline{x_1} \wedge \overline{x_2}) \vee x_3$, $f_3 = (\overline{x_1} \wedge \overline{x_3}) \vee x_2$, $f_4 = \overline{f_3}$, and so on.

Input negation and output negation can be executed by card-based protocols without increasing the number of cards. They are executed by just swapping input cards or output cards. Permutations of input variables can also be executed without increasing the number of cards. They can be achieved by just changing the positions of the input values. Therefore, all functions in the same NPN equivalence class can be calculated with the same number of cards.

Theorem 1. *Any three input Boolean functions can be securely calculated without additional cards other than the input cards with a standard deck of cards when we use private operations.*

Proof. When the number of inputs is 3, there are the following 14 NPN-representative functions [43]. (Note that x, y, and z are used to represent input variables.)

1. $NPN_1 = 1$
2. $NPN_2 = x$
3. $NPN_3 = x \vee y$
4. $NPN_4 = x \oplus y$
5. $NPN_5 = x \wedge y \wedge z$
6. $NPN_6 = (x \wedge y \wedge z) \vee (\overline{x} \wedge \overline{y} \wedge \overline{z})$
7. $NPN_7 = (x \wedge y) \vee (x \wedge z)$
8. $NPN_8 = (x \wedge y) \vee (\overline{x} \wedge \overline{y} \wedge z)$
9. $NPN_9 = (x \wedge y \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge z) \vee (\overline{x} \wedge y \wedge z)$
10. $NPN_{10} = (x \wedge \overline{y} \wedge \overline{z}) \vee (\overline{x} \wedge y \wedge \overline{z}) \vee (\overline{x} \wedge \overline{y} \wedge z) \vee (x \wedge y \wedge z) = x \oplus y \oplus z.$
11. $NPN_{11} = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
12. $NPN_{12} = (x \wedge \overline{z}) \vee (y \wedge z)$
13. $NPN_{13} = (x \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z})$
14. $NPN_{14} = (x \wedge y) \vee (x \wedge z) \vee (\overline{x} \wedge \overline{y} \wedge \overline{z})$

Among these 14 functions, NPN_1 - NPN_4 depend on less than three inputs. These functions can be calculated without additional cards [17]. We show a calculation protocol for each of the remaining functions. Note that the output is $commit(f)^{\{1,2\}}$ when the inputs are $commit(x)^{\{1,2\}}$, $commit(y)^{\{3,4\}}$, and $commit(z)^{\{5,6\}}$.

For NPN_5, $x \wedge y$ can be calculated without additional cards. Then $x \wedge y \wedge z$ can be calculated without additional cards other than the input cards, $x \wedge y$ and z.

NPN_7 can be represented as $NPN_7 = x \wedge (y \vee z)$, thus this function can also be calculated without additional cards.

NPN_{10} can be calculated as $(x \oplus y) \oplus z$ without additional cards.

NPN_{13} can be represented as $NPN_{13} = x \wedge (\overline{y \oplus z})$, thus this function can also be calculated without additional cards.

NPN_{14} can be represented as $NPN_{14} = \overline{x} \oplus (y \vee z)$, thus this function can also be calculated without additional cards.

NPN_6 can be represented as $NPN_6 = (\overline{x \oplus y}) \wedge (\overline{x \oplus z})$. First, calculate $commit(x \oplus y)^{\{3,4\}}$ with preserving input $commit(x)^{\{1,2\}}$. Then calculate $commit(x \oplus z)^{\{1,2\}}$. Then NOT is applied to each result. Next, calculate AND to these results.

NPN_8 can be represented as $NPN_8 = (\overline{x \oplus y}) \wedge (y \vee z)$. First, calculate $commit(x \oplus y)^{\{1,2\}}$ with preserving input $commit(y)^{\{3,4\}}$. Then NOT is applied to the result. Then calculate $commit(y \vee z)^{\{3,4\}}$. Next, calculate AND to these results.

NPN_9 can be represented as $NPN_9 = (\overline{x \oplus y \oplus z}) \wedge (x \vee z)$. First, calculate $commit(x \oplus y)^{\{3,4\}}$ with preserving input $commit(x)^{\{1,2\}}$. Next, calculate $commit((x \oplus y) \oplus z)^{\{3,4\}}$ with preserving $commit(z)^{\{5,6\}}$. Then NOT is applied to the result. Next, calculate $commit(x \vee z)^{\{1,2\}}$. Next, calculate AND to these results.

NPN_{12} can be calculated as follows. First, calculate $commit(x \wedge \overline{z})^{\{1,2\}}$ with preserving input $commit(z)^{\{5,6\}}$. Next, calculate $commit(y \wedge z)^{\{3,4\}}$. Then, calculate OR to these results by using the AND protocol.

NPN_{11} can be represented as

$$NPN_{11} = \begin{cases} z & \text{if } x \oplus y = 1 \\ x & \text{if } x \oplus y = 0 \end{cases}$$

Since this equation is similar to the AND equation, the function can be calculated by modifying the AND protocol as follows.

1. Alice and Bob calculate $commit(x \oplus y)^{\{3,4\}}$ with preserving input $commit(x)^{\{1,2\}}$.
2. Alice executes private random bisection cut on $commit(x \oplus y)^{\{3,4\}}$, $commit(x)^{\{1,2\}}$, and $commit(z)^{\{5,6\}}$ using different random bit $a_1, a_2, a_3 \in \{0, 1\}$. Alice sends the result $commit(x \oplus y \oplus a_1)^{\{3,4\}}$, $commit(x \oplus a_2)^{\{1,2\}}$, and $commit(z \oplus a_3)^{\{5,6\}}$ to Bob.
3. Bob privately select a bit $b_1 \in \{0, 1\}$ and exchanges the bases of $commit(x \oplus a_2)^{\{1,2\}}$ and $commit(z \oplus a_3)^{\{5,6\}}$ if $b_1 = 1$. Though Bob sees the committed values, Bob obtains no information about x and z since Alice randomized

the values. Bob sends $commit(x \oplus a_2)^{\{1,2\},\{5,6\}|A} || commit(z \oplus a_3)^{\{1,2\},\{5,6\}|A}$ to Alice.

4. Alice executes private reverse cuts to the sequence using a_2 and a_3. Alice sends the result $commit(x)^{\{1,2\},\{5,6\}|A} || commit(z)^{\{1,2\},\{5,6\}|A}$ to Bob.
5. Bob executes private reveal on $commit(x \oplus y \oplus a_1)$. Bob sets

$$S_2 = \begin{cases} commit(z)^{\{1,2\},\{5,6\}|A} || commit(x)^{\{1,2\},\{5,6\}|A} \text{ if } x \oplus y \oplus a_1 = 1 \\ commit(x)^{\{1,2\},\{5,6\}|A} || commit(z)^{\{1,2\},\{5,6\}|A} \text{ if } x \oplus y \oplus a_1 = 0 \end{cases}$$

6. Alice executes a private reverse cut on S_2 using the bit a_1 generated in the private random bisection cut. Let the obtained sequence be S_3. S_3 is $commit(z)^{\{1,2\},\{5,6\}|A} || commit(x)^{\{1,2\},\{5,6\}|A}$ if ($x \oplus y \oplus a_1 = 1$ and $a_1 = 0$) or ($x \oplus y \oplus a_1 = 0$ and $a_1 = 1$). The case equals to $x \oplus y = 1$. The output is $commit(x)^{\{1,2\},\{5,6\}|A} || commit(z)^{\{1,2\},\{5,6\}|A}$ if ($x \oplus y \oplus a_1 = 1$ and $a_1 = 1$) or ($x \oplus y \oplus a_1 = 0$ and $a_1 = 0$). The case equals to $x \oplus y = 0$. Thus the result is

$$S_3 = \begin{cases} commit(z)^{\{1,2\},\{5,6\}|A} || commit(x)^{\{1,2\},\{5,6\}|A} \text{ if } x \oplus y = 1 \\ commit(x)^{\{1,2\},\{5,6\}|A} || commit(z)^{\{1,2\},\{5,6\}|A} \text{ if } x \oplus y = 0 \end{cases}$$

   Note that the left pair has the value of the result.
7. Alice and Bob execute base-fixed protocol on S_3. They obtain

$$\begin{cases} commit(z)^{\{1,2\}} \text{ if } x \oplus y = 1 \\ commit(x)^{\{1,2\}} \text{ if } x \oplus y = 0 \end{cases}$$

Therefore, NPN_{11} can also be calculated without additional cards. □

4 Half Adder, Full Adder, and Symmetric Functions

This section first shows a realization of half adder and full adder. In our protocols, no additional cards are necessary other than the cards for inputs.

The input and output of the secure half adder are as follows:

- Input: $commit(x)^{\{1,2\}}$ and $commit(y)^{\{3,4\}}$
- Output: $S = commit(x \oplus y)^{\{3,4\}}$ and $C = commit(x \wedge y)^{\{1,2\}}$

The half adder is realized by the following steps, whose idea is just the same as the one in [34].

1. Execute XOR protocol with preserving input x. Thus $commit(x)^{\{1,2\}}$ and $commit(x \oplus y)^{\{3,4\}}$ are obtained.
2. Obtain $commit(\overline{x \oplus y})^{\{3,4\}}$ by swapping the two cards of $commot(x \oplus y)^{\{3,4\}}$.
3. Execute AND protocol to $commit(x)^{\{1,2\}}$ and $commit(\overline{x \oplus y})^{\{3,4\}}$ with preserving input $commit(\overline{x \oplus y})^{\{3,4\}}$. Thus $commit(\overline{x \oplus y})^{\{3,4\}}$ and $commit(x \wedge \overline{(x \oplus y)})^{\{1,2\}} = commit(x \wedge y)^{\{1,2\}}$ are obtained.
4. Obtain $commit(x \oplus y)^{\{1,2\}}$ by swapping the two cards of $commit(\overline{x \oplus y})^{\{1,2\}}$.

No additional cards are necessary other than the four input cards.

The input and output of the secure full adder are as follows:

- Input: $commit(C_I)^{\{1,2\}}$, $commit(x)^{\{3,4\}}$, and $commit(y)^{\{5,6\}}$.
- Output: $C_O = commit((x \wedge y) \vee (x \wedge C_I) \vee (y \wedge C_I))^{\{1,2\}}$ and $S = commit(x \oplus y \oplus C_I)^{\{3,4\}}$.

Since the half adder can be calculated without additional cards, the full adder can also be calculated without additional cards by the following protocol.

1. Add C_I and x using the half adder. The outputs are $commit(x \oplus c_I)^{\{3,4\}}$ and $commit(x \wedge C_I)^{\{1,2\}}$.
2. Add $commit(y)^{\{5,6\}}$ to the result $commit(x \oplus C_I)^{\{3,4\}}$ using the half adder. The outputs are $commit(x \oplus y \oplus C_I)^{\{3,4\}}$ and $commit(y \wedge (x \oplus C_I))^{\{5,6\}}$.
3. Execute OR protocol to $commit(y \wedge (x \oplus C_I))^{\{5,6\}}$ and $commit(x \wedge C_I)^{\{1,2\}}$. Since $(y \wedge (x \oplus C_I)) \vee (x \wedge C_I) = (x \wedge y) \vee (x \wedge C_I) \vee (y \wedge C_I)$, the carry C_O is obtained by the base of $\{1, 2\}$.

Using the half adder and full adder, calculation of symmetric function can be done by the technique in [34]. n-input symmetric function $f(x_1, x_2, \ldots, x_n)$ depends only on the number of variables such that $x_i = 1$. Let $Y = \sum_{i=1}^{n} x_i$. Then the function f can be written as $f(x_1, x_2, \ldots, x_n) = g(Y)$. When Y is given by a binary representation, $Y = y_k y_{k-1} ....y_1$, g can be written as $g(y_1, y_2, \ldots, y_k)$, where $k = \lfloor \log n \rfloor + 1$.

Given input $x_1, x_2, \ldots, x_n$, first, obtain the sum of these inputs using the half adder and full adder protocols without additional cards. The sum is obtained as $y_1, y_2, \ldots, y_k$. Then, calculate g using y_i. When $n \leq 7$, $k \leq 3$, thus any three input Boolean function g can be calculated without additional cards. When $n \geq 8$, Y is represented with $k = \lfloor \log n \rfloor + 1$ bits. Since $n - k \geq 4$, at least 8 input cards are unused after y_is are calculated. Any Boolean function can be calculated with four additional cards, thus g can be calculated without additional cards other than the input cards.

Theorem 2. *Any symmetric Boolean function can be securely calculated without additional cards other than the input cards when we use private operations.*

5 Conclusion

This paper showed card-based cryptographic protocols to calculate three input Boolean functions, half adder, full adder, and symmetric functions with a standard deck of cards using private operations. These results show the effectiveness of private operations.

One of the important open problems is obtaining another class of Boolean functions that can be calculated without additional cards using private operations. However, it seems very difficult to achieve all four-input Boolean functions without additional cards.

References

1. Abe, Y., Hayashi, Y.i., Mizuki, T., Sone, H.: Five-card and computations in committed format using only uniform cyclic shuffles. New Generation Computing **39**(1), 97–114 (2021)
2. Abe, Y., Mizuki, T., Sone, H.: Committed-format and protocol using only random cuts. Natural Computing pp. 1–7 (2021)
3. den Boer, B.: More efficient match-making and satisfiability the five card trick. In: Proc. of EUROCRYPT '89, LNCS Vol. 434. pp. 208–217 (1990)
4. Cheung, E., Hawthorne, C., Lee, P.: Cs 758 project: Secure computation with playing cards (2013), `http://cdchawthorne.com/writings/secure\_playing\_cards.pdf`
5. Francis, D., Aljunid, 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)
6. Hanaoka, G., Iwamoto, M., Watanabe, Y., Mizuki, T., Abe, Y., Shinagawa, K., Arai, M., Yanai, N.: Physical and visual cryptography to accelerate social implementation of advanced cryptographic technologies. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences pp. 214–228 (2023), (In Japanese)
7. 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)
8. 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)
9. Koch, A., Schrempp, M., Kirsten, M.: Card-based cryptography meets formal verification. New Generation Computing **39**(1), 115–158 (2021)
10. 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)
11. 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)
12. Koyama, H., Miyahara, D., Mizuki, T., Sone, H.: A secure three-input and protocol with a standard deck of minimal cards. In: Santhanam, R., Musatov, D. (eds.) Proc. of 16th International Computer Science Symposium in Russia (CSR 2021), LNCS Vol. 12730. pp. 242–256. Springer International Publishing, Cham (2021)
13. 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. pp. 13–22. APKC '21, Association for Computing Machinery, New York, NY, USA (2021)
14. Kuzuma, T., Isuzugawa, R., Toyoda, K., Miyahara, D., Mizuki, T.: Card-based single-shuffle protocols for secure multiple-input and and xor computations. In: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop. pp. 51–58 (2022)
15. Manabe, Y., Ono, H.: Card-based cryptographic protocols for three-input functions using private operations. In: Proc. of 32nd International Workshop on Combinatorial Algorithms (IWOCA 2021), LNCS Vol. 12757. pp. 469–484. Springer (2021)

16. Manabe, Y., Ono, H.: Card-based cryptographic protocols with malicious players using private operations. New Generation Computing **40**(1), 67–93 (2022)
17. Manabe, Y., Ono, H.: Card-based cryptographic protocols with a standard deck of cards using private operations. New Generation Computing (2024)
18. Marcedone, A., Wen, Z., Shi, E.: Secure dating with four or fewer cards. IACR Cryptology ePrint Archive, Report 2015/1031 (2015)
19. Miyahara, D., Hayashi, Y.i., Mizuki, T., Sone, H.: Practical card-based implementations of yao's millionaire protocol. Theoretical Computer Science **803**, 207–221 (2020)
20. Miyahara, D., Mizuki, T.: Secure computations through checking suits of playing cards. In: Proc. of International Workshop on Frontiers in Algorithmic Wisdom (IJTCS-FAW 2022), LNCS Vol. 13461. pp. 110–128. Springer (2022)
21. Mizuki, T.: Applications of card-based cryptography to education. In: IEICE Techinical Report ISEC2016-53. pp. 13–17 (2016), (In Japanese)
22. Mizuki, T.: Card-based protocols for securely computing the conjunction of multiple variables. Theoretical Computer Science **622**, 34–44 (2016)
23. Mizuki, T.: Efficient and secure multiparty computations using a standard deck of playing cards. In: Proc. of 15th International Conference on Cryptology and Network Security(CANS 2016), LNCS Vol.10052. pp. 484–499. Springer (2016)
24. Mizuki, T., Komano, Y.: Information leakage due to operative errors in card-based protocols. Information and Computation **285**, 104910 (2022)
25. 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)
26. Mizuki, T., Shizuya, H.: A formalization of card-based cryptographic protocols via abstract machine. International Journal of Information Security **13**(1), 15–23 (2014)
27. 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)
28. Mizuki, T., Shizuya, H.: Computational model of card-based cryptographic protocols and its applications. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **100**(1), 3–11 (2017)
29. 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)
30. Morooka, T., Manabe, Y., Shinagawa, K.: Malicious player card-based cryptographic protocols with a standard deck of cards using private operations. In: Proc. of 18th International Conference on Information Security Practice and Experience (ISPEC 2023), LNCS vol. 14341. pp. 332–346. Springer Nature Singapore (2023)
31. Nakai, T., Iwanari, K., Ono, T., Abe, Y., Watanabe, Y., Iwamoto, M.: Card-based cryptography with a standard deck of cards, revisited: Efficient protocols in the private model. New Generation Computing (2024)
32. Nakai, T., Misawa, Y., Tokushige, Y., Iwamoto, M., Ohta, K.: How to solve millionaires' problem with two kinds of cards. New Generation Computing **39**(1), 73–96 (2021)
33. Niemi, V., Renvall, A.: Solitaire zero-knowledge. Fundamenta Informaticae **38**(1, 2), 181–188 (1999)
34. 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)

35. Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Securely computing three-input functions with eight cards. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **98**(6), 1145–1152 (2015)
36. Nishimura, A., Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Card-based protocols using unequal division shuffles. Soft Computing **22**(2), 361–371 (2018)
37. Ono, H., Manabe, Y.: Card-based cryptographic logical computations using private operations. New Generation Computing **39**(1), 19–40 (2021)
38. Ono, H., Manabe, Y.: Minimum round card-based cryptographic protocols using private operations. Cryptography **5**(3) (2021)
39. Ruangwises, S.: Two standard decks of playing cards are sufficient for a zkp for sudoku. New Generation Computing **40**(1), 49–65 (2022)
40. 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)
41. Ruangwises, S., Itoh, T.: Securely computing the n-variable equality function with 2n cards. Theoretical Computer Science **887**, 99–110 (2021)
42. Ruangwises, S., Itoh, T.: Physical zkp for makaro using a standard deck of cards. In: Proc. of 17th International Conference on Theory and Applications of Models of Computation (TAMC 2022), LNCS Vol. 13571. pp. 43–54. Springer (2022)
43. Sasao, T., Butler, J.T.: Progress in Applications of Boolean Functions. Morgan and Claypool Publishers (2010)
44. Shikata, H., Miyahara, D., Mizuki, T.: Few-helping-card protocols for some wider class of symmetric boolean functions with arbitrary ranges. In: Proceedings of the 10th ACM International Workshop on ASIA Public-Key Cryptography (APKC). pp. 33–41 (2023)
45. Shikata, H., Toyoda, K., Miyahara, D., Mizuki, T.: Card-minimal protocols for symmetric boolean functions of more than seven inputs. In: Seidl, H., Liu, Z., Pasareanu, C.S. (eds.) Proc. of 18th International Conference on Theoretical Aspects of Computing (ICTAC 2022) LNCS Vol. 13572. pp. 388–406. Springer International Publishing, Cham (2022)
46. 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)
47. 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)
48. Shinagawa, K., Nuida, K.: A single shuffle is enough for secure card-based computation of any boolean circuit. Discrete Applied Mathematics **289**, 248–261 (2021)
49. Takashima, K., Miyahara, D., Mizuki, T., Sone, H.: Actively revealing card attack on card-based protocols. Natural Computing **21**(4), 615–628 (2022)
50. 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)
51. Yoshida, T., Tanaka, K., Nakabayashi, K., Chida, E., Mizuki, T.: Upper bounds on the number of shuffles for two-helping-card multi-input and protocols. In: Proc. of 22nd International Conference on Cryptology and Network Security(CANS 2023), LNCS vol. 14342. pp. 211–231. Springer (2023)