Single-shuffle Full-open Card-based Protocols Imply Private Simultaneous Messages Protocols

Kazumasa Shinagawa[1,3] Koji Nuida[2,3]

[1] Ibaraki University
[2] Institute of Mathematics for Industry (IMI), Kyushu University
[3] National Institute of Advanced Industrial Science and Technology (AIST)

October 1, 2022

Abstract

In this note, we introduce a class of card-based protocols called *single-shuffle full-open (SSFO) protocols* and show that any SSFO protocol for a function $f : \{0,1\}^n \to [d]$ using k cards is generically converted to a private simultaneous messages (PSM) protocol for f with (nk)-bit communication. As an example application, we obtain an 18-bit PSM protocol for the three-bit equality function from the six-card trick (Heather–Schneider–Teague, Formal Aspects of Computing 2014), which is an SSFO protocol in our terminology. We then generalize this result to another class of protocols which we name *single-shuffle single-branch (SSSB) protocols*, which contains SSFO protocols as a subclass. As an example application, we obtain an 8-bit PSM protocol for the two-bit AND function from the four-card trick (Mizuki–Kumamoto–Sone, ASIACRYPT 2012), which is an SSSB protocol in our terminology.

1 Introduction

Card-based protocols [1,2,8] are secure computation protocols using a deck of physical cards. They have been developed along with the conventional cryptography (i.e., cryptography without using physical objects), and some of them are inspired by the techniques from the conventional cryptography (e.g., the circuit evaluation technique [1,8], the garbled circuit technique [11], and the private permutation setting [9]). However, for the opposite direction, as far as we know, it has not been known what implies from card-based cryptography to the conventional cryptography. A natural question is: *Is there any implication from card-based cryptography to the conventional cryptography?*

To answer this question, in this note, we show that any card-based protocol with certain properties implies a *private simultaneous messages (PSM) protocol* [3, 5]. We first introduce a class of card-based protocols called *single-shuffle full-open (SSFO) protocols*. An SSFO protocol is a card-based protocol with the "single-shuffle" property, which requires a single shuffle only, and the "full-open" property, in which all cards are opened at the end of the protocol. Our first contribution is to show that any SSFO protocol for a function $f : \{0,1\}^n \to [d] = \{1, 2, \ldots, d\}$ using k cards is generically converted to a PSM protocol for f with (nk)-bit communication. We note that, starting from the same SSFO protocol, our conversion can also generate a PSM protocol for a slightly more complicated function than the original function f. As applications, we obtain a 10-bit PSM protocol for the two-bit AND function from the five-card trick [2], an 18-bit PSM protocol for the three-bit equality function from the six-card trick [4], and an 18-bit PSM protocol for the three-bit majority function from the six-card majority protocol in [12].

Although our protocol compiler supports a natural but limited class of card-based protocols, we hope that a deeper connection from card-based protocols to other cryptographic primitives will be discovered in future research. Towards this direction, we generalize our result to another class of protocols which we name *single-shuffle single-branch (SSSB) protocols*. An SSSB protocol is a card-based protocol with the single-shuffle property and the "single-branch" property as follows. Right after applying a shuffle, it first opens a single card and then opens some other cards, where the positions of the second opening depends on the opened value of the first opening. Our second contribution is to show that any SSSB protocol for a function $f : \{0,1\}^n \to [d]$ using k cards implies a PSM protocol for f with nk bits (and also a PSM protocol for a slightly more complicated function). As an application, we obtain an 8-bit PSM protocol for the two-bit AND function from the four-card trick [6]. We remark that any SSFO protocol can be viewed as an SSSB protocol. Thus, our second compiler for SSSB protocols can be also applied to an SSFO protocol with k cards, and it also generates a PSM protocol with nk bits, the same communication complexity as our first compiler for SSFO protocols. The advantages of our first compiler are the simplicity of the construction and the efficiency of the randomness, where it saves $(k-1)$ random bits compared to our second compiler.

It is worthwhile to note that our results capture card-based protocols with *non-uniform shuffles* (i.e., shuffles whose probability distributions are not uniform) or *non-closed shuffles* (i.e., shuffles whose permutation sets are not closed under composition). In card-based cryptography, they are considered to be difficult to implement physically, and thus considered to be less practical. In contrast, our results do not matter whether the underlying shuffle is non-uniform/non-closed since the permutations in the converted PSM protocol are to be performed on an electronic computer which can feasibly choose permutations by even non-uniform/non-closed distributions. We believe that our results shed light on non-uniform/non-closed shuffles, and provide a new motivation to improve the efficiency of card-based protocols by using non-uniform/non-closed shuffles.

We note that the aim of our result is to show a connection from card-based protocols to the conventional cryptography for the first time. We do not claim that our protocols obtained from card-based protocols are the most efficient among the existing PSM protocols. Indeed, for example, there exists a $(2\log_2 3)$-bit PSM protocol for the two-bit AND protocol [3], which is more efficient than our 8-bit PSM protocol from the four-card trick. Intuitively speaking, the less efficiency of the protocol by our conversion comes mainly from the protocol design that an input of each party is encoded as a sparse bit string. Improvement of the efficiency will be an important future research topic.

This note is organized as follows. In Section 2, we introduce card-based protocols and PSM protocols. In Section 3, we define SSFO protocols, show that SSFO protocols imply PSM protocols, and give concrete PSM protocols from some existing SSFO protocols. In Section 4, we define SSSB protocols, show that SSSB protocols imply PSM protocols, and give concrete PSM protocols from some existing SSSB protocols. In Appendix, we give the protocol descriptions of these SSFO and SSSB protocols.

2 Preliminaries

In Section 2.1, we define the basic notations. In Section 2.2, we briefly introduce card-based protocols. See Mizuki–Shizuya's paper [7] for the details. In Section 2.3, we give the definition of PSM protocols.

2.1 Notations

For an integer $n \geq 2$, we denote $[n] = \{1,2,\dots,n\}$. For an integer $k \geq 1$, we denote the k-th symmetric group by S_k. For a k-bit string $s = (s_1, s_2, \dots, s_k) \in \{0,1\}^k$ and a permutation $\pi \in S_k$, we define the permuted string of s by π, denoted by $\pi(s)$, as follows:

$$\pi(s) = (s_{\pi^{-1}(1)}, s_{\pi^{-1}(2)}, \dots, s_{\pi^{-1}(k)}).$$

For example, for a string $s = 111000 \in \{0,1\}^6$ and a cyclic permutation $\sigma = (1\ 2\ 3\ 4\ 5\ 6) \in S_6$, we have $\sigma(s) = 011100$.

2.2 Card-based Protocols

We use cards having two face-up symbols $\boxed{\clubsuit}$ and $\boxed{\heartsuit}$ whose backs are both $\boxed{?}$. We assume that all cards having the same symbol are indistinguishable, and all face-down cards are indistinguishable regardless of the face-up symbols. By using the encoding $\clubsuit = 0$ and $\heartsuit = 1$, we identify a sequence of k cards with a k-bit string $s \in \{0,1\}^k$. For a bit $x \in \{0,1\}$, a pair of face-down cards corresponding to $(x, \overline{x}) \in \{0,1\}^2$ is called a *commitment to* x, and depicted as follows:

$$\underbrace{\boxed{?}\boxed{?}}_{x} \text{ or } \underset{x}{\boxed{?}}\underset{\overline{x}}{\boxed{?}}.$$

A card-based protocol for a function $f : \{0,1\}^n \to \{0,1\}$ is a protocol, which takes n commitments to $x_1, x_2, \ldots, x_n \in \{0,1\}$ with some helping cards, and outputs either a commitment to $f(x_1, x_2, \ldots, x_n)$ or the value $f(x_1, x_2, \ldots, x_n)$ itself. A protocol of the former type is called a *committed-format protocol*, and of the latter type is called a *non-committed-format protocol*. In this paper, we deal with non-committed format protocols only, and also deal with functions $f : \{0,1\}^n \to [d]$ whose range is possibly non-binary.

In card-based protocols, a sequence of operations are applied to a card sequence. Let k be the number of cards. A *perm* is an operation that arranges a card sequence along with a publicly known permutation $\pi \in S_k$. A *turn* is an operation that turns over a set of cards on some position $T \subseteq [k]$. A *shuffle* is an operation that arranges a card sequence along with a permutation $\pi \in \Pi \subseteq S_k$, where π is drawn from a probability distribution $\mathcal{F}$ over a subset Π. Here, no player should know which permutation is chosen by the shuffle when it is applied to a face-down card sequence $\boxed{?}\boxed{?}\cdots\boxed{?}$. We denote a shuffle with Π and $\mathcal{F}$ by $(\mathsf{shuffle}, \Pi, \mathcal{F})$.

2.3 PSM Protocols

Let Y, R, X_i, M_i $(1 \le i \le n)$ be finite sets. Set $X = X_1 \times X_2 \times \cdots \times X_n$ and $M = M_1 \times M_2 \times \cdots \times M_n$. Let $\mathcal{R}$ be a probability distribution over R. Let $f : X \to Y$, $\mathsf{Enc}_i : X_i \times R \to M_i$ $(1 \le i \le n)$, and $\mathsf{Dec} : M \to Y$ be functions. A *private simultaneous messages (PSM) protocol* P for f is a tuple $(n, X, Y, M, R, \mathcal{R}, (\mathsf{Enc}_i)_{1 \le i \le n}, \mathsf{Dec})$. It is said to be *correct* if for any $(x_1, \ldots, x_n) \in X$ and any $r \in R$ such that $\Pr[r \leftarrow \mathcal{R}] > 0$, it holds that

$$\mathsf{Dec}(\mathsf{Enc}_1(x_1, r), \ldots, \mathsf{Enc}_n(x_n, r)) = f(x_1, \ldots, x_n).$$

It is said to be *secure* if there exists an algorithm $\mathcal{S}$ called a simulator such that for any $x \in \{0,1\}^n$, the distribution $\mathcal{S}(y)$ for $y = f(x)$ and the message distribution $(\mathsf{Enc}_1(x_1, r), \ldots, \mathsf{Enc}_n(x_n, r))$ for $r \leftarrow \mathcal{R}$ are the same distribution. The communication complexity of the protocol P is defined by $\sum_{i=1}^{n} \log_2 |M_i|$. The randomness complexity of the protocol P is defined by the Shannon entropy $H(\mathcal{R})$ of $\mathcal{R}$.

We note that the randomness distribution $\mathcal{R}$ is often restricted to be uniform in the standard definition of PSM protocols. The reason why we allow a non-uniform distribution is to capture non-uniform shuffles.

3 PSM Protocols from Single-shuffle Full-open Protocols

In Section 3.1, we define single-shuffle full-open (SSFO) protocols. In Section 3.2, we show that any SSFO protocol is converted into a PSM protocol. In Section 3.3, we give concrete PSM protocols from some existing SSFO protocols as applications.

3.1 Single-shuffle Full-open Protocols

We define a class of protocols which we name *single-shuffle full-open (SSFO) protocols*. An SSFO protocol is a card-based protocol consisting of the following three Steps:

1. Arrange the cards to the *input card sequence* $s(x)$ (see below for the detail), which is a face-down card sequence determined by the input $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, as follows:

$$\underbrace{\boxed{?}\boxed{?}\boxed{?}\cdots\boxed{?}\boxed{?}\boxed{?}}_{s(x)}.$$

2. Apply a shuffle $(\mathsf{shuffle}, \Pi, \mathcal{F})$ to the card sequence as follows:

$$\underbrace{\boxed{?}\boxed{?}\boxed{?}\cdots\boxed{?}\boxed{?}\boxed{?}}_{s(x)} \xrightarrow{\mathsf{shuffle}} \underbrace{\boxed{?}\boxed{?}\boxed{?}\cdots\boxed{?}\boxed{?}\boxed{?}}_{\pi(s(x))},$$

   where $\pi \in \Pi$ is a permutation drawn from the probability distribution $\mathcal{F}$.

3. Open all cards and obtain the bit string $\pi(s(x))$ as follows:

$$\underbrace{\boxed{?}\boxed{?}\boxed{?}\cdots\boxed{?}\boxed{?}\boxed{?}}_{\pi(s(x))} \xrightarrow{\mathsf{turn}} \underbrace{\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}\cdots\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}}_{\pi(s(x))}.$$

   Determine the output value $y \in [d]$ from $\pi(s(x))$.

We explain the detail of the input card sequence $s(x)$. Let $\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n$ denote variable symbols, each corresponding to an input of each player. For an SSFO protocol with k cards, an *input template* $s(\mathsf{x})$ is defined as a sequence of length k of the following form;

$$s(\mathsf{x}) \in \{0, 1, \mathsf{x}_1, \overline{\mathsf{x}_1}, \mathsf{x}_2, \overline{\mathsf{x}_2}, \ldots, \mathsf{x}_n, \overline{\mathsf{x}_n}\}^k .$$

Given an input template $s(\mathsf{x}) = (s_1, \ldots, s_k)$ and players' concrete input values $x_1, \ldots, x_n \in \{0, 1\}$, the input card sequence $s(x)$ (where $x = (x_1, \ldots, x_n)$) consists of k face-down cards where the face-up symbol of the j-th card in $s(x)$ ($j \in [k]$) is

$$\begin{cases} \clubsuit & \text{if } s_j = 0\text{; or } s_j = \mathsf{x}_i \text{ and } x_i = 0\text{; or } s_j = \overline{\mathsf{x}_i} \text{ and } x_i = 1\,, \\ \heartsuit & \text{if } s_j = 1\text{; or } s_j = \mathsf{x}_i \text{ and } x_i = 1\text{; or } s_j = \overline{\mathsf{x}_i} \text{ and } x_i = 0\,. \end{cases}$$

Intuitively, $s(x)$ is obtained by first substituting each input value $x_i \in \{0, 1\}$ into the variable x_i appearing in the input template $s(\mathsf{x})$ and then encoding the resulting k-bit string as k face-down cards. More precisely, if $s_j = \mathsf{x}_i$ (respectively, $\overline{\mathsf{x}_i}$), then the j-th card in $s(x)$ encodes the input value $x_i \in \{0, 1\}$ (respectively, the negation of the input value $\overline{x_i} \in \{0, 1\}$) according to the encoding rule $\clubsuit = 0$ and $\heartsuit = 1$, therefore this card can be chosen (and put into the card sequence) solely by the i-th player. On the other hand, if $s_j = 0$ (respectively, 1), then the j-th card in $s(x)$ is a helping card whose face-up symbol is always $\clubsuit$ (respectively, $\heartsuit$), therefore this card can be chosen (and put into the card sequence) in public. By abusing the notation, the bit string corresponding to the sequence of face-up symbols for the cards in $s(x)$ is also denoted by $s(x)$. For example, the input template for the five-card trick [2] can be expressed as $s(\mathsf{x}) = (\overline{\mathsf{x}_1}, \mathsf{x}_1, 1, \mathsf{x}_2, \overline{\mathsf{x}_2})$, and the face-up symbols of the input card sequence $s(x)$ for input $x = (1, 0)$ is $\clubsuit\heartsuit\heartsuit\clubsuit\heartsuit$, which we also write as $s(x) = s(1, 0) = 01101$. We note that though a formalization of card-based protocols in the literature often strictly supposes that the initial card sequence is a sequence of commitments to $x_1, \ldots, x_n$ followed by some helping cards, our definition of the input card sequence $s(x)$ is more flexible and convenient for protocol description while not compromising the feasibility of physical implementation.

Consequently, an SSFO protocol with k cards is defined as a tuple $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$, where $s(\mathsf{x})$ is an input template as above, $(\mathsf{shuffle}, \Pi, \mathcal{F})$ represents a shuffle with $\Pi \subseteq S_k$, and $g \colon \{0, 1\}^k \to [d]$ is a partial function called the *output function* indicating the rule of determining the output value $y \in [d]$ from the face-up card sequence $\pi(s(x))$ (identified with a bit string in $\{0, 1\}^k$ via the encoding rule $\clubsuit = 0$ and $\heartsuit = 1$) at the end of the protocol (it is a partial function because some bit sequence that never appears as $\pi(s(x))$ is often ignored in a concrete protocol description).

Let $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$ be an SSFO protocol for a function $f \colon \{0, 1\}^n \to [d]$. The protocol P is *correct* if $g(\pi(s(x))) = f(x)$ for any input $x \in \{0, 1\}^n$ and any permutation $\pi \in \Pi$ such that $\Pr[\pi \leftarrow \mathcal{F}] > 0$. The protocol P is *secure* if there exists an algorithm $\mathcal{S}$ (called a simulator) such that for any $x \in \{0, 1\}^n$, the output distribution of $\mathcal{S}(y)$ for $y = f(x)$ and the distribution of $\pi(s(x)) \in \{0, 1\}^k$ for $\pi \leftarrow \mathcal{F}$ are the same distribution.

The five-card trick [2] is an example of an SSFO protocol for the two-bit AND function $f(x_1, x_2) = x_1 \wedge x_2$. It is described as $(5, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$, where $s(\mathsf{x}) = (\overline{\mathsf{x}_1}, \mathsf{x}_1, 1, \mathsf{x}_2, \overline{\mathsf{x}_2})$, $\Pi = \langle(1\ 2\ 3\ 4\ 5)\rangle$, $\mathcal{F}$ is the uniform distribution over Π, and $g : \{0,1\}^5 \to \{0,1\}$ is a partial function defined as follows:

$$g(x) = \begin{cases} 1 & \text{if } x \in \{00111, 10011, 11001, 11100, 01110\}; \\ 0 & \text{if } x \in \{01011, 10101, 11010, 01101, 10110\}. \end{cases}$$

3.2 Single-shuffle Full-open Protocols Imply PSM Protocols

A central idea for our generic conversion from an SSFO protocol to a PSM protocol is based on the following observation. For example, let $s(\mathsf{x}) = (\overline{\mathsf{x}_1}, \mathsf{x}_1, 1, \mathsf{x}_2, \overline{\mathsf{x}_2})$ be the input template for the five-card trick. Then for an input $x = (x_1, x_2) \in \{0,1\}^2$, the bit string $s(x) = s(x_1, x_2)$ can be written as

$$s(x) = (x_1 \oplus 1, x_1, 1, x_2, x_2 \oplus 1) = (1,0,1,0,1) \oplus (x_1, x_1, 0, 0, 0) \oplus (0, 0, 0, x_2, x_2)\,.$$

Moreover, any shuffle for $s(x)$ is compatible with this decomposition; namely, for any permutation $\pi \in S_5$, we have

$$\pi(s(x)) = \pi(1,0,1,0,1) \oplus \pi(x_1, x_1, 0, 0, 0) \oplus \pi(0, 0, 0, x_2, x_2)\,. \quad (1)$$

This observation also motivates us to introduce the following class of functions associated to a given function $f \colon \{0,1\}^n \to [d]$ (which also includes the original function f). Let $n' \geq 1$, and for each $i \in [n]$, let I_i be a subset of $[n']$. Let $I = (I_1, \ldots, I_n)$. Then we define a function $f^I \colon \{0,1\}^{n'} \to [d]$ as follows, where $x = (x_1, \ldots, x_{n'})$:

$$f^I(x) := f(x[I_1], \ldots, x[I_n]) \quad \text{where} \quad x[J] := \bigoplus_{h \in J} x_h \text{ for } J \subseteq [n'] \ (x[\emptyset] = 0)\,.$$

We note that f^I with $n' = n$ and $I_i = \{i\}$ ($i \in [n]$) coincides with the original function f. Moreover, for any input template $s(\mathsf{x}) = (s_1, \ldots, s_k)$ (with variables $\mathsf{x}_1, \overline{\mathsf{x}_1}, \ldots, \mathsf{x}_n, \overline{\mathsf{x}_n}$), we define the *local decomposition* $(s_0^I, s_1^I(\mathsf{x}_1), \ldots, s_{n'}^I(\mathsf{x}_{n'}))$ of $s(\mathsf{x})$ associated to I by $s_0^I := (s_{0,1}^I, \ldots, s_{0,k}^I)$ and $s_h^I(\mathsf{x}_h) := (s_{h,1}^I, \ldots, s_{h,k}^I)$ for $h \in [n']$, where for $j \in [k]$,

$$s_{0,j}^I := \begin{cases} 0 & \text{if } s_j \in \{0, \mathsf{x}_1, \ldots, \mathsf{x}_n\}\,, \\ 1 & \text{if } s_j \in \{1, \overline{\mathsf{x}_1}, \ldots, \overline{\mathsf{x}_n}\}\,, \end{cases} \quad s_{h,j}^I := \begin{cases} \mathsf{x}_h & \text{if } s_j \in \{\mathsf{x}_i, \overline{\mathsf{x}_i}\} \text{ and } h \in I_i \text{ for some } i \in [n]\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

For example, when $s(\mathsf{x}) = (\overline{\mathsf{x}_1}, \mathsf{x}_1, 1, \mathsf{x}_2, \overline{\mathsf{x}_2})$, $n' = 3$, $I_1 = \{1, 2\}$, and $I_2 = \{1, 3\}$, we have

$$s_0^I = (1,0,1,0,1)\,, s_1^I(\mathsf{x}_1) = (\mathsf{x}_1, \mathsf{x}_1, 0, \mathsf{x}_1, \mathsf{x}_1)\,, s_2^I(\mathsf{x}_2) = (\mathsf{x}_2, \mathsf{x}_2, 0, 0, 0)\,, s_3^I(\mathsf{x}_3) = (0, 0, 0, \mathsf{x}_3, \mathsf{x}_3)\,.$$

Now the definition of the local decomposition implies the following: for any $b \in \{0,1\}$, let $s_h^I(b)$ denote the bit string obtained by substituting b into the variable x_h in $s_h^I(\mathsf{x}_h)$. Then for any $x \in \{0,1\}^{n'}$, we have

$$s(x[I_1], \ldots, x[I_n]) = s_0^I \oplus \bigoplus_{h \in [n']} s_h^I(x_h)\,. \quad (2)$$

Let $f : \{0,1\}^n \to [d]$ be a function. Let $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$ be an SSFO protocol for f. Let $n' \geq 1$, and let $I = (I_1, \ldots, I_n)$ be a tuple of subsets $I_i \subseteq [n']$. Let $(s_0^I, s_1^I(\mathsf{x}_1), \ldots, s_{n'}^I(\mathsf{x}_{n'}))$ be the local decomposition of $s(\mathsf{x})$ associated to I. Then we construct a PSM protocol P' for the function $f^I \colon \{0,1\}^{n'} \to [d]$ as in Figure 1. We may omit the superscripts I when $n' = n$ and $I_i = \{i\}$ for all $i \in [n]$, i.e., in the case $f^I = f$. The correctness and security for P' are shown as follows.

Randomness $R := \Pi \times (\{0,1\}^k)^{n'}$

$\mathcal{R}$ outputs $r = (\pi, r_1, \ldots, r_{n'})$ where $\pi \leftarrow \mathcal{F}$, $r_h \in \{0,1\}^k$ $(1 \leq h \leq n'-1)$ is uniformly random, $r_{n'} = \bigoplus_{h=1}^{n'-1} r_h$

Messages $M_h := \{0,1\}^k$ $(h \in [n'])$

$m_1 := \pi(s_0^I \oplus s_1^I(x_1)) \oplus r_1$, $m_h := \pi(s_h^I(x_h)) \oplus r_h$ $(2 \leq h \leq n')$

Decoding $\mathsf{Dec}(m_1, \ldots, m_{n'})$ computes $z \leftarrow \bigoplus_{h=1}^{n'} m_h$ and outputs $g(z) \in [d]$

Communication Complexity $\sum_{h=1}^{n'} \log_2 |M_h| = n'k$

Randomness Complexity $H(\mathcal{R}) = H(\mathcal{F}) + (n'-1)k$

Figure 1: Our conversion from an SSFO protocol $(k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$ for function $f \colon \{0,1\}^n \to [d]$ to a PSM protocol for function $f^I \colon \{0,1\}^{n'} \to [d]$ (here $x = (x_1, \ldots, x_{n'}) \in \{0,1\}^{n'}$ denotes an input tuple, and we put $m_h := \mathsf{Enc}_h(x_h, r)$ for $h \in [n']$)

Correctness. By the compatibility of permutations and XOR operations such as in Eq.(1), we have

$$\begin{aligned} z &= m_1 \oplus m_2 \oplus \cdots \oplus m_{n'} \\ &= (\pi(s_0^I \oplus s_1^I(x_1)) \oplus r_1) \oplus (\pi(s_2^I(x_2)) \oplus r_2) \oplus \cdots \oplus (\pi(s_{n'}^I(x_{n'})) \oplus r_{n'}) \\ &= \pi(s_0^I \oplus s_1^I(x_1) \oplus s_2^I(x_2) \oplus \cdots \oplus s_{n'}^I(x_{n'})) \oplus r_1 \oplus r_2 \oplus \cdots \oplus r_{n'} \\ &= \pi(s(x[I_1], \ldots, x[I_n])) \end{aligned}$$

where we used Eq.(2) at the last equality. From the correctness of P, it holds that

$$g(z) = g(\pi(s(x[I_1], \ldots, x[I_n]))) = f(x[I_1], \ldots, x[I_n]) = f^I(x)\,.$$

Therefore, P' is correct.

Security. It is sufficient to construct a simulator $\mathcal{S}'$ for P' from the simulator $\mathcal{S}$ for P. Given an output value $y \in [d]$ corresponding to an input $x = (x_1, \ldots, x_{n'})$, the simulator $\mathcal{S}'$ invokes $\mathcal{S}(y)$ and obtains a string $s' \in \{0,1\}^k$ corresponding to the opened values in P for input $x^I := (x[I_1], \ldots, x[I_n])$ (note that $f(x^I) = f^I(x) = y$). Then, the simulator $\mathcal{S}'$ uniformly samples $r'_h \in \{0,1\}^k$ $(1 \leq h \leq n'-1)$. The simulator $\mathcal{S}'$ outputs $(r'_1, r'_2, \ldots, r'_{n'-1}, s' \oplus \bigoplus_{h=1}^{n'-1} r'_h)$ as the simulated messages. It is easily seen that the simulated messages and the real messages follow the same distribution due to the choice of masking vectors $r_1, \ldots, r_{n'}$ in P'. Therefore, P' is secure.

Remark 3.1. In our PSM protocol P', when some player holds multiple input bits, say $x_{h_1}, \ldots, x_{h_\mu}$, a naive operation for the protocol is to let the player receive multiple components $r_{h_1}, \ldots, r_{h_\mu}$ of randomness and output multiple messages $m_{h_1}, \ldots, m_{h_\mu}$. Here we note that the efficiency can be improved by letting the player output, instead, the XOR of the original messages $m_{h_1} \oplus \cdots \oplus m_{h_\mu}$ at once. Such a change of gathering some messages preserves the security of the protocol in general, while it also preserves the correctness due to the structure of the protocol P'. Moreover, the components $r_{h_1}, \ldots, r_{h_\mu}$ of randomness can also be gathered into a single component $r_{h_1} \oplus \cdots \oplus r_{h_\mu}$, which also reduces the randomness complexity of the protocol.

We also note that negation of some input value in the protocol P' can be handled by introducing an additional input component (which is owned by some player as in the previous paragraph) and then setting the additional component to be 1. For example, we have $f(\overline{x_1}, x_2, \overline{x_3}) = f^I(x_1, x_2, x_3, 1)$ where $f^I(x_1, x_2, x_3, x_4) = f(x_1 \oplus x_4, x_2, x_3 \oplus x_4)$ (i.e., $I = (I_1, I_2, I_3)$, $I_1 = \{1, 4\}$, $I_2 = \{2\}$, $I_3 = \{3, 4\}$).

3.3 Applications

3.3.1 AND Protocol

Five-Card Trick. The five-card trick [2] (see also Appendix) is an SSFO protocol for the two-bit AND function. It is described as $(5, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$, where $s(\mathsf{x}) = (\overline{\mathsf{x}_1}, \mathsf{x}_1, 1, \mathsf{x}_2, \overline{\mathsf{x}_2})$, $\Pi = \langle (1\ 2\ 3\ 4\ 5) \rangle$, $\mathcal{F}$ is the uniform distribution over Π, and $g : \{0,1\}^5 \to \{0,1\}$ is a partial function defined as follows:

$$g(z) = \begin{cases} 1 & \text{if } z \in \{11100, 01110, 00111, 10011, 11001\}; \\ 0 & \text{if } z \in \{11010, 01101, 10110, 01011, 10101\}. \end{cases}$$

The local decomposition of $s(\mathsf{x})$ is given as follows:

$$\begin{aligned} s_0 &= (1, 0, 1, 0, 1), \\ s_1(\mathsf{x}_1) &= (\mathsf{x}_1, \mathsf{x}_1, 0, 0, 0), \\ s_2(\mathsf{x}_2) &= (0, 0, 0, \mathsf{x}_2, \mathsf{x}_2). \end{aligned}$$

Construction. The shared randomness r is a pair $r = (\pi, r')$, where $\pi \in \Pi$ is a permutation drawn from $\mathcal{F}$, and $r' \in \{0,1\}^5$ is a uniformly random bit string. The first player holding x_1 computes $m_1 = \pi(s_0 \oplus s_1(x_1)) \oplus r'$. The second player holding x_2 computes $m_2 = \pi(s_2(x_2)) \oplus r'$. On receiving m_1, m_2, the referee computes $z = m_1 \oplus m_2$ and outputs $g(z)$.

Complexity. The communication complexity of the protocol is 10. The randomness complexity of the protocol is $5 + \log_2 5$.

Extension. For any $n' \geq 1$, any $I_1, I_2 \subseteq [n']$, and any $b_1, b_2 \in \{0,1\}$, we also have a PSM protocol with $(5n')$-bit communication for the function $f' \colon \{0,1\}^{n'} \to \{0,1\}$ defined as follows:

$$f'(x_1, x_2, \ldots, x_{n'}) = \left(b_1 \oplus \bigoplus_{i \in I_1} x_i \right) \wedge \left(b_2 \oplus \bigoplus_{i \in I_2} x_i \right).$$

3.3.2 Equality Protocol

Six-Card Trick. The six-card trick [4] (see also Appendix) is an SSFO protocol for $\mathsf{EQ}_3 : \{0,1\}^3 \to \{0,1\}$ as follows:

$$\mathsf{EQ}_3(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_1 = x_2 = x_3; \\ 0 & \text{otherwise.} \end{cases}$$

It is described as $(6, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$, where $s(\mathsf{x}) = (\mathsf{x}_1, \overline{\mathsf{x}_2}, \mathsf{x}_3, \overline{\mathsf{x}_1}, \mathsf{x}_2, \overline{\mathsf{x}_3})$, $\Pi = \langle (1\ 2\ 3\ 4\ 5\ 6) \rangle$, $\mathcal{F}$ is the uniform distribution over Π, and $g : \{0,1\}^6 \to \{0,1\}$ is a partial function defined as follows:

$$g(z) = \begin{cases} 1 & \text{if } z \in \{010101, 101010\}; \\ 0 & \text{if } z \in \{000111, 100011, 110001, 111000, 011100, 001110\}. \end{cases}$$

The local decomposition of $s(\mathsf{x})$ is given as follows:

$$\begin{aligned} s_0 &= (0, 1, 0, 1, 0, 1), \\ s_1(\mathsf{x}_1) &= (\mathsf{x}_1, 0, 0, \mathsf{x}_1, 0, 0), \\ s_2(\mathsf{x}_2) &= (0, \mathsf{x}_2, 0, 0, \mathsf{x}_2, 0), \\ s_3(\mathsf{x}_3) &= (0, 0, \mathsf{x}_3, 0, 0, \mathsf{x}_3). \end{aligned}$$

Construction. The shared randomness r is a tuple $r = (\pi, r_1, r_2, r_3)$, where $\pi \in \Pi$ is a permutation drawn from $\mathcal{F}$, $r_1, r_2 \in \{0,1\}^6$ are uniformly random bit strings, and $r_3 = r_1 \oplus r_2$ so that $r_1 \oplus r_2 \oplus r_3 = 000000$. The first player holding x_1 computes $m_1 = \pi(s_0 \oplus s_1(x_1)) \oplus r_1$. Each of the other players holding x_i ($i \in \{2,3\}$) computes $m_i = \pi(s_i(x_i)) \oplus r_i$. On receiving m_1, m_2, m_3, the referee computes $z = m_1 \oplus m_2 \oplus m_3$ and outputs $g(z)$.

Complexity. The communication complexity of the protocol is 18. The randomness complexity of the protocol is $12 + \log_2 6$.

Extension. For any $n' \geq 1$, any $I_1, I_2, I_3 \subseteq [n']$, and any $b_1, b_2, b_3 \in \{0,1\}$, we also have a PSM protocol with $(6n')$-bit communication for the function $f' : \{0,1\}^{n'} \to \{0,1\}$ defined as follows:

$$f'(x_1, x_2, \ldots, x_{n'}) = \mathsf{EQ}_3\left(b_1 \oplus \bigoplus_{i \in I_1} x_i, b_2 \oplus \bigoplus_{i \in I_2} x_i, b_3 \oplus \bigoplus_{i \in I_3} x_i\right).$$

3.3.3 Majority Protocol

Six-card Majority Protocol. The six-card majority protocol in [12] (see also Appendix) is an SSFO protocol for $\mathsf{MAJ}_3 : \{0,1\}^3 \to \{0,1\}$ as follows:

$$\mathsf{MAJ}_3(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_1 + x_2 + x_3 \geq 2; \\ 0 & \text{otherwise.} \end{cases}$$

It is described as $(6, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), g)$, which is defined as follows. The input template is $s(\mathsf{x}) = (\mathsf{x}_1, \overline{\mathsf{x}_1}, \mathsf{x}_2, \overline{\mathsf{x}_2}, \overline{\mathsf{x}_3}, \mathsf{x}_3)$. For $\tau = (1\ 2)(3\ 6)$ and $\sigma = (2\ 3\ 4\ 5\ 6)$, the permutation set Π is defined as $\Pi = \{\sigma^b \circ \tau^a \mid a \in \{0,1\}, b \in \{0,1,2,3,4\}\}$, and $\mathcal{F}$ is the uniform distribution over Π. The output function $g : \{0,1\}^6 \to \{0,1\}$ is a partial function defined as follows:

$$g(z) = \begin{cases} 1 & \text{if } z \in \{011100, 001110, 000111, 010011, 011001, 100101, 110010, 101001, 110100, 101010\}; \\ 0 & \text{if } z \in \{011010, 001101, 010110, 001011, 010101, 100011, 110001, 111000, 101100, 100110\}. \end{cases}$$

The local decomposition of $s(\mathsf{x})$ is given as follows:

$$\begin{aligned} s_0 &= (0,1,0,1,1,0), \\ s_1(\mathsf{x}_1) &= (\mathsf{x}_1, \mathsf{x}_1, 0, 0, 0, 0), \\ s_2(\mathsf{x}_2) &= (0, 0, \mathsf{x}_2, \mathsf{x}_2, 0, 0), \\ s_3(\mathsf{x}_3) &= (0, 0, 0, 0, \mathsf{x}_3, \mathsf{x}_3). \end{aligned}$$

Construction. The shared randomness r is a tuple $r = (\pi, r_1, r_2, r_3)$, where $\pi \in \Pi$ is a permutation drawn from $\mathcal{F}$, $r_1, r_2 \in \{0,1\}^6$ are uniformly random bit strings, and $r_3 = r_1 \oplus r_2$ so that $r_1 \oplus r_2 \oplus r_3 = 000000$. The first player holding x_1 computes $m_1 = \pi(s_0 \oplus s_1(x_1)) \oplus r_1$. Each of the other players holding x_i ($i \in \{2,3\}$) computes $m_i = \pi(s_i(x_i)) \oplus r_i$. On receiving m_1, m_2, m_3, the referee computes $z = m_1 \oplus m_2 \oplus m_3$ and outputs $g(z)$.

Complexity. The communication complexity of the protocol is 18. The randomness complexity of the protocol is $13 + \log_2 5$.

Extension. For any $n' \geq 1$, any $I_1, I_2, I_3 \subseteq [n']$, and any $b_1, b_2, b_3 \in \{0,1\}$, we also have a PSM protocol with $(6n')$-bit communication for the function $f' : \{0,1\}^{n'} \to \{0,1\}$ defined as follows:

$$f'(x_1, x_2, \ldots, x_{n'}) = \mathsf{MAJ}_3\left(b_1 \oplus \bigoplus_{i \in I_1} x_i, b_2 \oplus \bigoplus_{i \in I_2} x_i, b_3 \oplus \bigoplus_{i \in I_3} x_i\right).$$

4 PSM Protocols from Single-shuffle Single-branch Protocols

In Section 4.1, we define single-shuffle single-branch (SSSB) protocols. In Section 4.2, we show that any SSSB protocol is converted into a PSM protocol. In Section 4.3, we give an 8-bit PSM protocol from the four-card trick, as an application.

4.1 Single-shuffle Single-branch Protocols

We define a class of protocols which we name *single-shuffle single-branch (SSSB) protocols*. An SSSB protocol is a card-based protocol consisting of the following four Steps:

1. Arrange the cards to the input card sequence $s(x)$ in the same way as an SSFO protocol.

2. Apply a shuffle $(\mathsf{shuffle}, \Pi, \mathcal{F})$ to the card sequence.

3. Open a single card at position $j^* \in [k]$. Let $b \in \{0,1\}$ be the opened value.

4. Let $S_0, S_1 \subseteq [k] \setminus \{j^*\}$ be the (predetermined) sets of positions. Depending on b, open the cards at positions in S_b. Determine the output value $y \in [d]$ from the opened values.

An SSSB protocol with k cards is defined as a tuple $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), j^*, S_0, S_1, g)$, where $s(\mathsf{x})$ and $(\mathsf{shuffle}, \Pi, \mathcal{F})$ are an input template and a shuffle, respectively, similar to the case of SSFO protocols, and g is an output function. Here the partially opened card sequence at the end of the protocol is identified with a string $\alpha \in \{0,1,?\}^k$ where each face-up card is regarded as 0 or 1 as usual and each face-down card is regarded as the symbol '?'. Then g is a partial function $\{0,1,?\}^k \to [d]$. For simplifying the notation, for any $w = (w_1, \dots, w_k) \in \{0,1\}^k$ and any $S \subseteq [k]$, we define $w|_S := (w'_1, \dots, w'_k) \in \{0,1,?\}^k$ where $w'_j = w_j \in \{0,1\}$ if $j \in S$ and $w'_j = ?$ if $j \notin S$. Using this notation, an input for g that may appear in the protocol is of the form $\pi(s(x))|_{S_b \cup \{j^*\}}$ with $\pi \in \Pi$ and $b \in \{0,1\}$.

Let $f : \{0,1\}^n \to [d]$ be a function. Let $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), j^*, S_0, S_1, g)$ be an SSSB protocol for f. For any $x \in \{0,1\}^n$ and any $\pi \in \Pi$, we define the string $\alpha_{x,\pi} \in \{0,1,?\}^k$ as follows, where γ_{j^*} denotes the j^*-th component of $\pi(s(\mathsf{x}))$:

$$\alpha_{x,\pi} = \begin{cases} \pi(s(x))|_{S_0 \cup \{j^*\}} & \text{if } \gamma_{j^*} = 0; \text{ or } \gamma_{j^*} = \mathsf{x}_i \text{ and } x_i = 0; \text{ or } \gamma_{j^*} = \overline{\mathsf{x}_i} \text{ and } x_i = 1; \\ \pi(s(x))|_{S_1 \cup \{j^*\}} & \text{if } \gamma_{j^*} = 1; \text{ or } \gamma_{j^*} = \mathsf{x}_i \text{ and } x_i = 1; \text{ or } \gamma_{j^*} = \overline{\mathsf{x}_i} \text{ and } x_i = 0. \end{cases}$$

The protocol P is *correct* if $g(\alpha_{x,\pi})) = f(x)$ for any input $x \in \{0,1\}^n$ and any permutation $\pi \in \Pi$ such that $\Pr[\pi \leftarrow \mathcal{F}] > 0$. The protocol P is *secure* if there exists an algorithm $\mathcal{S}$ (called a simulator) such that for any $x \in \{0,1\}^n$, the output distribution of $\mathcal{S}(y)$ for $y = f(x)$ and the distribution of $\alpha_{x,\pi} \in \{0,1,?\}^k$ for $\pi \leftarrow \mathcal{F}$ are the same distribution.

4.2 Single-shuffle Single-branch Protocols Imply PSM Protocols

Let $f : \{0,1\}^n \to [d]$ be a function. Let $P = (k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), j^*, S_0, S_1, g)$ be an SSSB protocol for f. In the same way as the case of SSFO protocols, let $n' \geq 1$, and let $I = (I_1, \dots, I_n)$ be a tuple of subsets $I_i \subseteq [n']$. Let $(s^I_0, s^I_1(\mathsf{x}_1), \dots, s^I_{n'}(\mathsf{x}_{n'}))$ be the local decomposition of $s(\mathsf{x})$ associated to I. Then we construct a PSM protocol P' for the function $f^I : \{0,1\}^{n'} \to [d]$ as in Figure 2. We may omit the superscripts I when $n' = n$ and $I_i = \{i\}$ for all $i \in [n]$, i.e., in the case $f^I = f$.

In the following, we prove the correctness and security of our protocol P'. For the purpose, we investigate the properties of the masking vectors $c^{(0)}_h \oplus c^{(1)}_h \oplus r_h$ used in the protocol.

Lemma 4.1. *In the protocol P' for input $x = (x_1, \dots, x_{n'}) \in \{0,1\}^{n'}$, let $b \in \{0,1\}$ and $j \in S_b$. Let $b' \in \{0,1\}$ be the j^*-th component of $\pi(s(x[I_1], \dots, x[I_n]))$. Let δ be the number of indices $h \in [n']$ satisfying $c^{(b)}_{h,j} = r^{(j)}$. Then $\delta \equiv b \oplus b' \oplus 1 \pmod 2$.*

Randomness $R := \Pi \times (\{0,1\}^k)^{n'}$

$\mathcal{R}$ outputs $r = (\pi, r_1, \dots, r_{n'})$ where $\pi \leftarrow \mathcal{F}$, $r_h = (r_{h,1}, \dots, r_{h,k}) \in \{0,1\}^k$ $(h \in [n'])$ are uniformly random subject to $\bigoplus_{h=1}^{n'} r_{h,j^*} = 0$

Messages $M_h := \{0,1\}^k$ $(h \in [n'])$

Let γ_{j^*} be the j^*-th component of $\pi(s(\mathsf{x}))$. For $j \in [k]$, let $r^{(j)} = \bigoplus_{h=1}^{n'} r_{h,j} \in \{0,1\}$. For $i \in [n]$ and $h \in [n']$, put $\varepsilon_{i,h} := 1$ if $h \in I_i$, and $\varepsilon_{i,h} := 0$ if $h \notin I_i$.

$$m_1 := \pi(s_0^I \oplus s_1^I(x_1)) \oplus c_1^{(0)} \oplus c_1^{(1)} \oplus r_1 \,,$$

where for each $b \in \{0,1\}$, $c_1^{(b)} = (c_{1,1}^{(b)}, \dots, c_{1,k}^{(b)}) \in \{0,1\}^k$ satisfies that $c_{1,j}^{(b)} = 0$ if $j \notin S_b$, and for $j \in S_b$,

$$c_{1,j}^{(b)} = \begin{cases} b \cdot r^{(j)} & \text{if } \gamma_{j^*} = 1\text{; or } \gamma_{j^*} = \mathsf{x}_i \text{ and } \varepsilon_{i,1} x_1 = 1\text{; or } \gamma_{j^*} = \overline{\mathsf{x}_i} \text{ and } \varepsilon_{i,1} x_1 = 0; \\ (1-b) \cdot r^{(j)} & \text{otherwise.} \end{cases}$$

$$m_h := \pi(s_h^I(x_h)) \oplus c_h^{(0)} \oplus c_h^{(1)} \oplus r_h \quad (2 \le h \le n') \,,$$

where for each $b \in \{0,1\}$, $c_h^{(b)} = (c_{h,1}^{(b)}, \dots, c_{h,k}^{(b)}) \in \{0,1\}^k$ satisfies that $c_{h,j}^{(b)} = 0$ if $j \notin S_b$, and for $j \in S_b$,

$$c_{h,j}^{(b)} = \begin{cases} r^{(j)} & \text{if } \gamma_{j^*} \in \{\mathsf{x}_i, \overline{\mathsf{x}_i}\} \text{ and } \varepsilon_{i,h} x_h = 1; \\ 0 & \text{otherwise.} \end{cases}$$

Decoding $\mathsf{Dec}(m_1, \dots, m_{n'})$ computes $z = (z_1, \dots, z_k) \leftarrow \bigoplus_{h=1}^{n'} m_h$ and puts $b := z_{j^*}$. It defines $\alpha = (\alpha_1, \dots, \alpha_k) \in \{0,1,?\}^k$ by $\alpha_j := z_j$ if $j \in S_b \cup \{j^*\}$ and $\alpha_j := ?$ otherwise. Then it outputs $g(\alpha) \in [d]$.

Communication Complexity $\sum_{h=1}^{n'} \log_2 |M_h| = n'k$

Randomness Complexity $H(\mathcal{R}) = H(\mathcal{F}) + n'k - 1$

Figure 2: Our conversion from an SSSB protocol $(k, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), j^*, S_0, S_1, g)$ for function $f\colon \{0,1\}^n \to [d]$ to a PSM protocol for function $f^I\colon \{0,1\}^{n'} \to [d]$ (here $x = (x_1, \dots, x_{n'}) \in \{0,1\}^{n'}$ denotes an input tuple, and we put $m_h := \mathsf{Enc}_h(x_h, r)$ for $h \in [n']$)

Proof. We use case-by-case argument with respect to the j^*-th component γ_{j^*} of $\pi(s(\mathsf{x}))$. Let δ' be the number of indices $h \in [n'] \setminus \{1\}$ satisfying $c_{h,j}^{(b)} = r^{(j)}$. We note that $x[I_i] = \bigoplus_{h=1}^{n'} (\varepsilon_{i,h} x_h)$ for each $i \in [n]$.

Case $\gamma_{j^*} \in \{0,1\}$**:** Now $b' = \gamma_{j^*}$. For the first player's message, we have $c_{1,j}^{(b)} = r^{(j)}$ if $(b,b') = (0,0)$ or $(1,1)$ (i.e., $b = b'$), and $c_{1,j}^{(b)} = 0$ if $(b,b') = (0,1)$ or $(1,0)$ (i.e., $b \ne b'$). For the h-th player's message $(2 \le h \le n')$, we have $c_{h,j}^{(b)} = 0$. Hence $\delta = 1$ is odd if $b = b'$ (i.e., $b \oplus b' \oplus 1 = 1$) and $\delta = 0$ is even if $b \ne b'$ (i.e., $b \oplus b' \oplus 1 = 0$), as desired.

Case $\gamma_{j^*} = \mathsf{x}_i$ **and** $\varepsilon_{i,1} = 0$**:** Now $b' = x[I_i]$. For the h-th player's message $(2 \le h \le n')$, we have $c_{h,j}^{(b)} = r^{(j)}$ if $\varepsilon_{i,h} x_h = 1$, and $c_{h,j}^{(b)} = 0$ otherwise. This implies that $\delta' \equiv x[I_i] = b' \pmod 2$. Moreover, for the first player's message, we have $c_{1,j}^{(b)} = r^{(j)}$ if $b = 0$, and $c_{1,j}^{(b)} = 0$ if $b = 1$. Hence $\delta = \delta' + 1 \equiv b' + 1 \equiv b \oplus b' \oplus 1 \pmod 2$ if $b = 0$, and $\delta = \delta' \equiv b' \equiv b \oplus b' \oplus 1$ if $b = 1$, as desired.

Case $\gamma_{j^*} = \overline{\mathsf{x}_i}$ **and** $\varepsilon_{i,1} = 0$**:** Now $b' = \overline{x[I_i]}$. An argument as above implies that $\delta' \equiv x[I_i] = b' \oplus 1 \pmod 2$. Moreover, for the first player's message, we have $c_{1,j}^{(b)} = 0$ if $b = 0$, and $c_{1,j}^{(b)} = r^{(j)}$ if $b = 1$. Hence

$\delta = \delta' \equiv b' \oplus 1 \equiv b \oplus b' \oplus 1 \pmod 2$ if $b = 0$, and $\delta = \delta' + 1 \equiv (b' \oplus 1) + 1 \equiv b \oplus b' \oplus 1$ if $b = 1$, as desired.

Case $\gamma_{j^*} = \mathsf{x}_i$ and $\varepsilon_{i,1} = 1$: Now $b' = x[I_i] = x[I_i \setminus \{1\}] \oplus x_1$. An argument as above implies that $\delta' \equiv x[I_i \setminus \{1\}] = b' \oplus x_1 \pmod 2$. Moreover, for the first player's message, $c_{1,j}^{(b)} = r^{(j)}$ if $(b, x_1) = (0,0)$ or $(1,1)$, and $c_{1,j}^{(b)} = 0$ if $(b, x_1) = (0,1)$ or $(1,0)$. Hence, in the former case we have $\delta = \delta' + 1 \equiv b' \oplus x_1 \oplus 1 = b \oplus b' \oplus 1$, and in the latter case we have $\delta = \delta' \equiv b' \oplus x_1 \equiv b \oplus b' \oplus 1$, as desired.

Case $\gamma_{j^*} = \overline{\mathsf{x}_i}$ and $\varepsilon_{i,1} = 1$: Now $b' = \overline{x[I_i]} = x[I_i \setminus \{1\}] \oplus x_1 \oplus 1$. An argument as above implies that $\delta' \equiv x[I_i \setminus \{1\}] = b' \oplus x_1 \oplus 1 \pmod 2$. Moreover, for the first player's message, we have $c_{1,j}^{(b)} = r^{(j)}$ if $(b, x_1) = (0,1)$ or $(1,0)$, and $c_{1,j}^{(b)} = 0$ if $(b, x_1) = (0,0)$ or $(1,1)$. Hence, in the former case we have $\delta = \delta' + 1 \equiv b' \oplus x_1 = b \oplus b' \oplus 1$, and in the latter case we have $\delta = \delta' \equiv b' \oplus x_1 \oplus 1 \equiv b \oplus b' \oplus 1$, as desired.

Hence the claim holds in any case. □

Let $b \in \{0,1\}$ be the j^*-th component of $\pi(s(x[I_1], \dots, x[I_n]))$. For $j \in [k]$, let J be the set of all indices $h \in [n']$ satisfying $c_{h,j}^{(0)} \oplus c_{h,j}^{(1)} = r^{(j)}$. Then Lemma 4.1 implies that $|J| \equiv 1 \pmod 2$ if and only if $j \in S_b$. Let Id denote the $n' \times n'$ identity matrix over $\mathbb{F}_2$, and let A denote an $n' \times n'$ matrix over $\mathbb{F}_2$ whose components in the h-th row are all 1's if $h \in J$ and all 0's if $h \notin J$. Put $\rho_h := c_{h,j}^{(0)} \oplus c_{h,j}^{(1)} \oplus r_{h,j}$ for $h \in [n']$. Then we have

$$(\rho_1, \dots, \rho_{n'})^T = (\mathsf{Id} + A) \cdot (r_{1,j}, \dots, r_{n',j})^T$$

where $(\cdots)^T$ denotes the transposition of a vector. Now when $r_{1,j}, \dots, r_{n',j} \in \{0,1\}$ are uniformly random and independent of each other, the vector $(\rho_1, \dots, \rho_{n'})^T$ distributes uniformly at random over the image $\mathrm{Im}(\mathsf{Id} + A)$ of $\mathsf{Id} + A$ (as an $\mathbb{F}_2$-linear map $(\mathbb{F}_2)^{n'} \to (\mathbb{F}_2)^{n'}$). Then we have the following:

Lemma 4.2. *In the setting above, we have* $\mathrm{Im}(\mathsf{Id} + A) = (\mathbb{F}_2)^{n'}$ *if* $|J| \equiv 0 \pmod 2$*, and* $\mathrm{Im}(\mathsf{Id} + A) = V$ *if* $|J| \equiv 1 \pmod 2$*, where* $V := \{(v_1, \dots, v_{n'})^T \in (\mathbb{F}_2)^{n'} \mid v_1 + \cdots + v_{n'} = 0\}$.

Proof. Let M_1 and M_2 be the results of the following elementary transformations applied to Id and A, respectively: add the last column to each of the other $n' - 1$ columns. Then M_1 has components 1 at the diagonal and the last row, and components 0 at the other positions; and M_2 has components 1 at the positions (h, n') with $h \in J$, and components 0 at the other positions.

Let M_1' and M_2' be the results of the following elementary transformations applied to M_1' and M_2', respectively: add the other $n' - 1$ rows to the last row. Then $M_1' = \mathsf{Id}$, and the first $n' - 1$ columns of M_2' are the zero vectors. Moreover, the (n', n')-component of M_2' is equal to $|J| \bmod 2$. This implies that $M_1' + M_2'$ (which is obtained from $\mathsf{Id} + A$ by elementary transformations) is an upper triangular matrix with diagonal components being $(1, \dots, 1, 1)$ when $|J| \equiv 0 \pmod 2$ and $(1, \dots, 1, 0)$ when $|J| \equiv 1 \pmod 2$. In the former case, $M_1' + M_2'$ (hence $\mathsf{Id} + A$) is non-singular, as desired. In the latter case, $M_1' + M_2'$ (hence $\mathsf{Id} + A$) has rank $n' - 1$ and hence $\dim \mathrm{Im}(\mathsf{Id} + A) = n' - 1 = \dim V$, while $\mathrm{Im}(\mathsf{Id} + A) \subseteq V$ by the shape of $\mathsf{Id} + A$. Therefore we have $\mathrm{Im}(\mathsf{Id} + A) = V$, as desired. □

By this lemma, it follows that $\rho_1, \dots, \rho_{n'} \in \{0,1\}$ are uniformly random when $|J| \equiv 0 \pmod 2$ (or equivalently, $j \notin S_b$), and $\rho_1, \dots, \rho_{n'} \in \{0,1\}$ are uniformly random subject to $\rho_1 \oplus \cdots \oplus \rho_{n'} = 0$ when $|J| \equiv 1 \pmod 2$ (or equivalently, $j \in S_b$). Due to this and the following fact

$$\pi(s_0^I \oplus s_1^I(x_1)) \oplus \pi(s_2^I(x_2)) \oplus \cdots \oplus \pi(s_{n'}^I(x_{n'})) = \pi(s(x[I_1], \dots, x[I_n]))$$

used in the argument for SSFO protocols, the distribution of the tuple of messages $(m_1, \dots, m_{n'})$ in the protocol P' with input $x = (x_1, \dots, x_{n'})$ is equal to the superposition, conditioned on $\pi \leftarrow \mathcal{F}$, of the uniform distributions over the following sets, where $x^I := (x[I_1], \dots, x[I_n])$ and $\pi(s(x^I)) = (\sigma_1, \dots, \sigma_k) \in \{0,1\}^k$:

$$\left\{ (m_1', \dots, m_{n'}') \in (\{0,1\}^k)^{n'} \;\middle|\; \begin{array}{c} m_h' = (m_{h,1}', \dots, m_{h,k}') \text{ for } h \in [n'] \,, \\ \bigoplus_{h=1}^{n'} m_{h,j^*}' = \sigma_{j^*} \,,\; \bigoplus_{h=1}^{n'} m_{h,j}' = \sigma_j \text{ for } j \in S_{\sigma_{j^*}} \end{array} \right\} .$$

This implies that the string α computed in the decoding algorithm of P' coincides with the string $\alpha_{x^I,\pi}$ defined in Section 4.1, therefore we have $g(\alpha) = g(\alpha_{x^I,\pi}) = f(x^I) = f^I(x)$ by the correctness of P. Hence P' is correct. On the other hand, we construct the following simulator $\mathcal{S}'$ for P' by using the simulator $\mathcal{S}$ for P: Given $y \in [d]$, $\mathcal{S}'$ first invokes $\mathcal{S}(y)$ to obtain a string $\alpha = (\alpha_1, \ldots, \alpha_k) \in \{0, 1, ?\}^k$. Secondly, $\mathcal{S}'$ puts $b := \alpha_{j^*}$ (note that $b \in \{0, 1\}$ by the property of the simulator $\mathcal{S}$), and for each $j \in [k]$, $\mathcal{S}'$ generates $(m'_{1,j}, \ldots, m'_{n',j}) \in \{0, 1\}^{n'}$ uniformly at random if $j \notin S_b \cup \{j^*\}$, and uniformly at random subject to $\bigoplus_{h=1}^{n'} m'_{h,j} = \alpha_j$ if $j \in S_b \cup \{j^*\}$. Finally, $\mathcal{S}'$ outputs $(m'_1, \ldots, m'_{n'})$ where $m'_h := (m'_{h,1}, \ldots, m'_{h,k})$. Now by the fact that the distribution of $\mathcal{S}(y)$ for $y = f^I(x) = f(x^I)$ is the same as the distribution of $\alpha_{x^I,\pi}$ with $\pi \leftarrow \mathcal{F}$, it follows that the distribution of $\mathcal{S}'(y)$ is equal to the distribution of $(m_1, \ldots, m_{n'})$ described above. Hence P' is secure.

4.3 Applications

4.3.1 AND Protocol

Four-Card Trick. The four-card trick [6] (see also Appendix) is an SSSB protocol for the two-bit AND function. It is described as $(4, s(\mathsf{x}), (\mathsf{shuffle}, \Pi, \mathcal{F}), j^* = 2, S_0 = \{4\}, S_1 = \{1\}, g)$, which is defined as follows. The input template is $s(\mathsf{x}) = (\mathsf{x}_1, \overline{\mathsf{x}_1}, \mathsf{x}_2, \overline{\mathsf{x}_2})$. For $\tau = (1\ 3)(2\ 4)$ and $\sigma = (2\ 3)$, the permutation set Π is defined as $\Pi = \{\sigma^b \circ \tau^a \mid a, b \in \{0, 1\}\}$. The probability distribution $\mathcal{F}$ is the uniform distribution over Π. The output function $g : \{0, 1, ?\}^4 \to \{0, 1\}$ is a partial function defined as follows:

$$g(\alpha) = \begin{cases} 1 & \text{if } \alpha \in \{?0?0, 11??\}; \\ 0 & \text{if } \alpha \in \{?0?1, 01??\}. \end{cases}$$

The local decomposition of $s(\mathsf{x})$ is given as follows:

$$\begin{aligned} s_0 &= (0, 1, 0, 1), \\ s_1(\mathsf{x}_1) &= (\mathsf{x}_1, \mathsf{x}_1, 0, 0), \\ s_2(\mathsf{x}_2) &= (0, 0, \mathsf{x}_2, \mathsf{x}_2). \end{aligned}$$

Construction. The shared randomness r is a tuple $r = (\pi, r_1, r_2)$, where $\pi \in \Pi$ is a permutation drawn from $\mathcal{F}$, and $r_1 = r_{1,1}r_{1,2}r_{1,3}r_{1,4}, r_2 = r_{2,1}r_{2,2}r_{2,3}r_{2,4} \in \{0, 1\}^4$ are uniformly random bit strings with the condition $r_{1,2} \oplus r_{2,2} = 0$. Set $r^{(1)} = r_{1,1} \oplus r_{2,1}$ and $r^{(4)} = r_{1,4} \oplus r_{2,4}$. The i-th player holding $x_i \in \{0, 1\}$ first computes $c_i \in \{0, 1\}^4$ as follows, where γ_2 denotes the second component of $\pi(s(\mathsf{x}))$:

$$c_1 = \begin{cases} (0, 0, 0, r^{(4)}) & \text{if } \gamma_2 = \mathsf{x}_1 \text{ and } x_1 = 0; \\ (r^{(1)}, 0, 0, 0) & \text{if } \gamma_2 = \mathsf{x}_1 \text{ and } x_1 = 1; \\ (r^{(1)}, 0, 0, 0) & \text{if } \gamma_2 = \overline{\mathsf{x}_1} \text{ and } x_i = 0; \\ (0, 0, 0, r^{(4)}) & \text{if } \gamma_2 = \overline{\mathsf{x}_1} \text{ and } x_i = 1; \\ (0, 0, 0, r^{(4)}) & \text{if } \gamma_2 = \mathsf{x}_2; \\ (r^{(1)}, 0, 0, 0) & \text{if } \gamma_2 = \overline{\mathsf{x}_2}. \end{cases}$$

$$c_2 = \begin{cases} (r^{(1)}, 0, 0, r^{(4)}) & \text{if } \gamma_2 \in \{\mathsf{x}_2, \overline{\mathsf{x}_2}\} \text{ and } x_2 = 1; \\ (0, 0, 0, 0) & \text{otherwise.} \end{cases}$$

Then, the first player computes $m_1 = \pi(s_0 \oplus s_1(x_1)) \oplus c_1 \oplus r_1$ and the second player computes $m_2 = \pi(s_2(x_2)) \oplus c_2 \oplus r_2$. On receiving m_1, m_2, the referee computes $z = m_1 \oplus m_2 =: z_1z_2z_3z_4$. If $z_2 = 0$, then the referee outputs $g(?0?z_4)$. If $z_2 = 1$, then the referee outputs $g(z_11??)$.

Complexity. The communication complexity of the protocol is 8. The randomness complexity of the protocol is 9.

Extension. For any $n' \geq 1$, any $I_1, I_2 \subseteq [n']$, and any $b_1, b_2 \in \{0,1\}$, we also have a PSM protocol with $(4n')$-bit communication for the function $f' : \{0,1\}^{n'} \to \{0,1\}$ defined as follows:

$$f'(x_1, x_2, \ldots, x_{n'}) = \left(b_1 \oplus \bigoplus_{i \in I_1} x_i \right) \wedge \left(b_2 \oplus \bigoplus_{i \in I_2} x_i \right).$$

Acknowledgments

The first author was supported during this work by JSPS KAKENHI Grant Number JP21K17702. The second author was supported during this work by JSPS KAKENHI Grant Number JP19H01109, JST CREST Grant Number JPMJCR2113, and JST AIP Acceleration Research JPMJCR22U5. This work was supported by FY2022 IMI Joint Usage/Research Program Short-term Visiting Researcher.

References

[1] C. Crépeau and J. Kilian. Discreet solitary games. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 319–330. Springer, 1993.

[2] B. den Boer. More efficient match-making and satisfiability: *The Five Card Trick*. In J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 208–217. Springer, 1989.

[3] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In F. T. Leighton and M. T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994.

[4] J. Heather, S. Schneider, and V. Teague. Cryptographic protocols with everyday objects. *Formal Asp. Comput.*, 26(1):37–62, 2014.

[5] Y. Ishai and E. Kushilevitz. Private Simultaneous Messages Protocols with Applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184. IEEE Computer Society, 1997.

[6] T. Mizuki, M. Kumamoto, and H. Sone. The five-card trick can be done with four cards. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 598–606. Springer, 2012.

[7] T. Mizuki and H. Shizuya. A formalization of card-based cryptographic protocols via abstract machine. *Int. J. Inf. Sec.*, 13(1):15–23, 2014.

[8] T. Mizuki and H. Sone. Six-card secure AND and four-card secure XOR. In X. Deng, J. E. Hopcroft, and J. Xue, editors, *Frontiers in Algorithmics, Third International Workshop, FAW 2009, Hefei, China, June 20-23, 2009. Proceedings*, volume 5598 of *Lecture Notes in Computer Science*, pages 358–369. Springer, 2009.

[9] T. Nakai, Y. Tokushige, Y. Misawa, M. Iwamoto, and K. Ohta. Efficient card-based cryptographic protocols for millionaires' problem utilizing private permutations. In S. Foresti and G. Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 500–517, 2016.

[10] K. Shinagawa and T. Mizuki. The six-card trick: Secure computation of three-input equality. In K. Lee, editor, *Information Security and Cryptology - ICISC 2018 - 21st International Conference, Seoul, South Korea, November 28-30, 2018, Revised Selected Papers*, volume 11396 of *Lecture Notes in Computer Science*, pages 123–131. Springer, 2018.

[11] K. Shinagawa and K. Nuida. A single shuffle is enough for secure card-based computation of any boolean circuit. *Discret. Appl. Math.*, 289:248–261, 2021.

[12] K. Toyoda, D. Miyahara, and T. Mizuki. Another use of the five-card trick: Card-minimal secure three-input majority function evaluation. In A. Adhikari, R. Küsters, and B. Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 536–555. Springer, 2021.

Appendix

Five-Card Trick

The five-card trick [2] is a five-card two-party card-based protocol for the two-bit AND function. It proceeds as follows.

1. Arrange commitments to x_1, x_2 and a helping card $\boxed{\heartsuit}$ as follows:

$$\underset{\overline{x_1}}{\boxed{?}}\underset{x_1}{\boxed{?}}\;\underset{\heartsuit}{\boxed{?}}\;\underset{x_2}{\boxed{?}}\underset{\overline{x_2}}{\boxed{?}}.$$

2. Apply a random cut to the five cards as follows:

$$\left\langle \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \right\rangle \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}.$$

3. Turn over all cards. If we have $\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\clubsuit}$ up to cyclic shifts, then the output value is 1. If we have $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}$ up to cyclic shifts, then the output value is 0.

Six-Card Trick

The six-card trick [4] (independently rediscovered by [10]) is a six-card three-party card-based protocol for the three-bit equality function $\mathsf{EQ}_3 : \{0,1\}^3 \to \{0,1\}$. It proceeds as follows.

1. Arrange commitments to x_1, x_2, x_3 as follows:

$$\underset{x_1}{\boxed{?}}\underset{\overline{x_2}}{\boxed{?}}\underset{x_3}{\boxed{?}}\underset{\overline{x_1}}{\boxed{?}}\underset{x_2}{\boxed{?}}\underset{\overline{x_3}}{\boxed{?}}.$$

2. Apply a random cut to the six cards as follows:

$$\left\langle \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \right\rangle \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}.$$

3. Turn over all cards. If we have $\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\clubsuit}$ up to cyclic shits, then the output value is 0. If we have $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}$ up to cyclic shits, then the output value is 1.

Six-card Majority Protocol

The six-card majority protocol in [12] is a six-card three-party card-based protocol for the three-bit majority function. It proceeds as follows.

1. Arrange commitments to x_1, x_2, x_3 as follows:

$$\underset{x_1}{\boxed{?}}\underset{\overline{x_1}}{\boxed{?}}\underset{x_2}{\boxed{?}}\underset{\overline{x_2}}{\boxed{?}}\underset{\overline{x_3}}{\boxed{?}}\underset{x_3}{\boxed{?}}.$$

2. Apply a shuffle $(\mathsf{shuffle}, \Pi, \mathcal{F})$, where $\Pi = \langle(1\ 2)(3\ 6)\rangle$ and $\mathcal{F}$ is a uniform distribution over Π, as follows:

$$\overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}} \rightarrow \overset{1}{\boxed{?}}\overset{2}{\boxed{?}}\overset{3}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{6}{\boxed{?}} \text{ or } \overset{2}{\boxed{?}}\overset{1}{\boxed{?}}\overset{6}{\boxed{?}}\overset{4}{\boxed{?}}\overset{5}{\boxed{?}}\overset{3}{\boxed{?}}.$$

3. Apply a random cut to the card sequence except the first card as follows:

$$\boxed{?}\left\langle \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?} \right\rangle \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}\boxed{?}.$$

4. Turn over all cards.

   (a) Suppose that the first card is $\boxed{\clubsuit}$. If the remaining five cards are $\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\clubsuit}$ up to cyclic shifts, then the output value is 1. If they are $\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}$ up to cyclic shifts, then the output value is 0.

   (b) Suppose that the first card is $\boxed{\heartsuit}$. If the remaining five cards are $\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\heartsuit}$ up to cyclic shifts, then the output value is 0. If they are $\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}\boxed{\heartsuit}\boxed{\clubsuit}$ up to cyclic shifts, then the output value is 1.

Four-Card Trick

The four-card trick [6] is a four-card two-party card-based protocol for the two-bit AND function. It proceeds as follows.

1. Arrange commitments to x_1, x_2 as follows:

$$\underset{x_1}{\boxed{?}}\underset{\overline{x_1}}{\boxed{?}}\underset{x_2}{\boxed{?}}\underset{\overline{x_2}}{\boxed{?}}.$$

2. Apply a random bisection cut to the four cards as follows:

$$\left[\, \boxed{?}\boxed{?} \,\middle|\, \boxed{?}\boxed{?} \,\right] \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}.$$

3. Apply a random cut to the second and third cards as follows:

$$\boxed{?}\left\langle \boxed{?}\boxed{?} \right\rangle\boxed{?} \rightarrow \boxed{?}\boxed{?}\boxed{?}\boxed{?}.$$

4. Open the second card.

   (a) If it is $\boxed{\clubsuit}$, then open the fourth card. If we have $\boxed{?}\boxed{\clubsuit}\boxed{?}\boxed{\clubsuit}$, then the output value is 1. If we have $\boxed{?}\boxed{\clubsuit}\boxed{?}\boxed{\heartsuit}$, then the output value is 0.

   (b) If it is $\boxed{\heartsuit}$, then open the first card. If we have $\boxed{\clubsuit}\boxed{\heartsuit}\boxed{?}\boxed{?}$, then the output value is 0. If we have $\boxed{\heartsuit}\boxed{\heartsuit}\boxed{?}\boxed{?}$, then the output value is 1.