Securely Computing the n-Variable Equality Function with $2n$ Cards

Suthee Ruangwises(✉) and Toshiya Itoh

Department of Mathematical and Computing Science,
Tokyo Institute of Technology, Tokyo, Japan
ruangwises.s.aa@m.titech.ac.jp, titoh@c.titech.ac.jp

Abstract. Research on the area of secure multi-party computation using a deck of playing cards, often called card-based cryptography, started from the introduction of the "five-card trick" to compute the logical AND function by den Boar in 1989. Since then, many protocols to compute various functions have been developed. In this paper, we propose a new card-based protocol that securely computes the n-variable *equality function* using $2n$ cards. We also show that the same technique can be applied to compute any *doubly symmetric function* $f : \{0,1\}^n \to \mathbb{Z}$ using $2n$ cards, and any symmetric function $f : \{0,1\}^n \to \mathbb{Z}$ using $2n + 2$ cards.

Keywords: Card-based cryptography · Secure multi-party computation · Equality function · Symmetric function · Doubly symmetric function

1 Introduction

During a two-candidate election, a group of n friends decides that they should discuss about the election only if everyone in the group supports the same candidate. However, each person does not know other people's preferences and wants to hide his/her own preference from the others unless they all support the same candidate in order to avoid awkwardness in the conversation. How can they know whether their preferences all coincide without leaking any other information?

In terms of secure multi-party computation, this situation can be viewed as a group of n players where the ith player has a bit a_i of either 0 or 1. Define the *equality function* $E(a_1, ..., a_n) = 1$ if $a_1 = ... = a_n$ and $E(a_1, ..., a_n) = 0$ otherwise. Our goal is to design a protocol that announces only the value of $E(a_1, ..., a_n)$ without leaking any other information, such as the preference of any player or the number of players who support each candidate (not even probabilistic information).

Secure multi-party computation is one of the most actively studied research areas in cryptography. It involves situations where multiple parties want to compare their private information without revealing it. In particular, this paper focuses on secure multi-party computation using a deck of playing cards, often

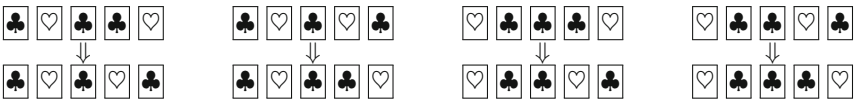
J. Chen et al. (Eds.): TAMC 2020, LNCS 12337, pp. 25–36, 2020.
https://doi.org/10.1007/978-3-030-59267-7_3

called card-based cryptography. The benefit of card-based protocols is that they provide solutions to real-world situations using only a small deck of cards, which is portable and can be found in everyday life, and do not require computers. Moreover, these straightforward protocols are easy to understand and verify the correctness and security, even for non-experts.

1.1 Related Work

The first research on card-based cryptography started in 1989 with the "five-card trick" introduced by den Boer [3] to compute the logical AND function on two players' bits a and b. This protocol uses three identical ♣ cards and two identical ♡ cards.

Throughout this paper, a bit 0 is encoded by a *commitment* ♣♡ and a bit 1 by a commitment ♡♣. We give each player one ♣ card and one ♡ card, and put another ♣ card face-down on a table. The first player then places his commitment of a face-down to the left of the ♣ card, while the second player places his commitment of b face-down to the right of it. Then, we swap the fourth and the fifth cards from the left, resulting in the following four possible sequences.



$(a,b)=(0,0)$ $(a,b)=(0,1)$ $(a,b)=(1,0)$ $(a,b)=(1,1)$

Observe that there are only two possible sequences in a cyclic rotation of the deck, ♡♣♡♣♣ and ♡♡♣♣♣, with the latter showing up if and only if $a=b=1$. We can obscure the initial position of the cards by making a *random cut* to shuffle the deck into a uniformly random cyclic permutation, i.e. a permutation uniformly chosen at random from $\{\mathrm{id},\pi,\pi^2,\pi^3,\pi^4\}$ where $\pi=(1\ 2\ 3\ 4\ 5)$, before turning all cards face-up. Hence, we can determine whether $a\wedge b=1$ from the cycle.

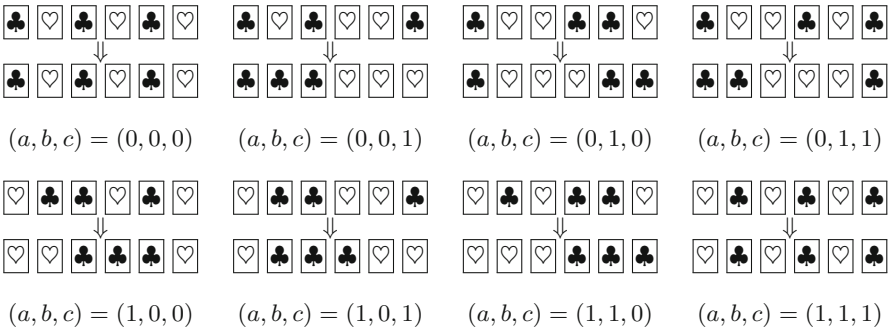
Since the introduction of the five-card trick, several other protocols to compute the AND function have been developed. These subsequent results [1,2,4,5,7,8,10,13,16] aimed to either reduce the number of required cards or improve properties of the protocol involving output format, running time, type of shuffles, etc.

Apart from the AND function protocol, various kinds of protocols have been developed as well, such as the XOR function protocol [2,8,9], the copy protocol [8] (creating multiple copies of the commitment), the *majority function* protocol [12] (deciding whether there are more 0s or 1s in the inputs), and the adder protocol [6] (adding bits and storing the sum in binary representation). Nishida et al. [11] proved that any n-variable Boolean function can be computed with $2n+6$ cards, and any such function that is symmetric can be computed with $2n+2$ cards.

1.2 The Six-Card Trick

For the equality function, the case $n = 2$ is a negation of the XOR function, which can be easily computed with four cards. For the case $n = 3$, Shinagawa and Mizuki [14] developed the following protocol called the "six-card trick" to compute the function $E(a, b, c)$ on three players' bits a, b, and c using six cards.

First, the players put the commitments of a, b, and c face-down on a table in this order from left to right. Then, we rearrange the cards into a (2 4 6) permutation, i.e. move the second leftmost card to the fourth leftmost position, the fourth card to the sixth position, and the sixth card to the second position, resulting in the following eight possible sequences.



Observe that there are only two possible sequences in a cyclic rotation of the deck, ♣♣♣♡♡♡ and ♣♡♣♡♣♡, with the latter showing up if and only if $a = b = c$, i.e. $E(a, b, c) = 1$. Again, we can obscure the initial position of the cards by making a random cut before turning all cards face-up, hence we can determine the value of $E(a, b, c)$ from the cycle.

The six-card trick has a benefit that it uses only one random cut. However, the technique used in this protocol heavily relies on the symmetric nature of the special case $n = 3$, suggesting that there might not be an equivalent protocol using $2n$ cards for a general n. In fact, in [14] they found by using a computer that in the case $n = 4$, an eight-card protocol that uses only one random cut does not exist.

1.3 Our Contribution

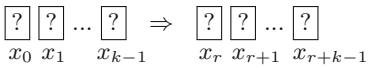
In this paper, we develop a card-based protocol that securely computes the n-variable equality function using $2n$ cards. We also show that the same technique can be applied to compute any *doubly symmetric function* (see the definition in Sect. 4.1) $f : \{0, 1\}^n \to \mathbb{Z}$ using $2n$ cards, and any symmetric function $f : \{0, 1\}^n \to \mathbb{Z}$ using $2n + 2$ cards.

2 Basic Operations

First, we will introduce basic operations on a deck of cards that will be used in our protocols.

2.1 Random Cut

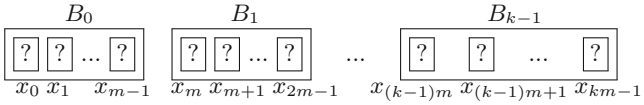
Suppose we have a sequence of cards $(x_0, x_1, ..., x_{k-1})$. A random cut is an operation to shuffle the deck into a uniformly random cyclic permutation, shifting the sequence into $(x_r, x_{r+1}, ..., x_{r+k-1})$, where r is a uniformly random integer from $\{0, 1, ..., k-1\}$ and the indices are taken in mod k.



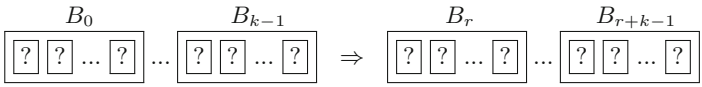
In real world, a random cut can be performed by applying a *Hindu cut*, which is a basic shuffling operation commonly used in card games [17].

2.2 Random k-Section Cut

A *random k-section cut* is a generalization of a *random bisection cut* introduced by Mizuki and Sone [8]. Suppose we have a sequence of km cards $(x_0, x_1, ..., x_{km-1})$. We divide the cards into k blocks $B_0, ..., B_{k-1}$, with each block B_i consisting of m consecutive cards $x_{im}, x_{im+1}, ..., x_{(i+1)m-1}$.



Then, we shuffle the blocks into a uniformly random cyclic permutation, shifting the order of them into $(B_r, B_{r+1}, ..., B_{r+k-1})$, where r is a uniformly random integer from $\{0, 1, ..., k-1\}$ and the indices are taken in mod k. This operation shifts the sequence of cards into $(x_{rm}, x_{rm+1}, ..., x_{(r+k)m-1})$, where the indices are taken in mod km.

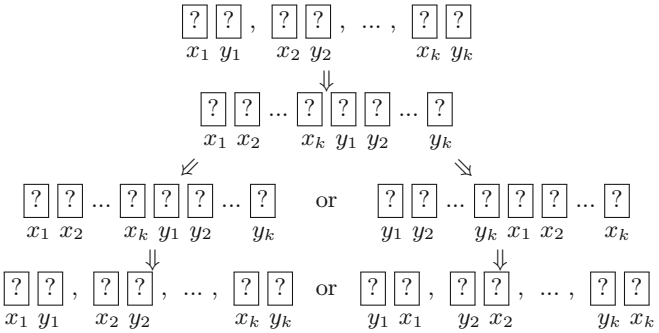


In real world, a random k-section cut can be performed by putting each block of cards into an envelope and applying a random cut on the pile of envelopes before taking the cards out.

2.3 XOR with a Random Bit

Recall that we encode 0 and 1 by commitments ♣♡ and ♡♣, respectively. Suppose we have a sequence of k bits $(a_1, a_2, ..., a_k)$ as an input, with each a_i encoded by a commitment (x_i, y_i). We want to securely perform the XOR operation with the same random bit on every input bit, i.e. output the sequence $(a_1 \oplus r, a_2 \oplus r, ..., a_k \oplus r)$ where $r \in \{0, 1\}$ is a uniformly random bit.

We can achieve this by applying a random 2-section cut in a way similar to the copy protocol of Mizuki and Sone [8]. First, arrange the cards as $X = (x_1, x_2, ..., x_k, y_1, y_2, ..., y_k)$ and apply a random 2-section cut on X. Then, for each $i = 1, 2, ..., k$, take the ith and the $(i+k)$-th cards from X in this order as the commitment of the ith output bit.



Observe that after applying the random 2-section cut, the sequence X will become either $(x_1, x_2, ..., x_k, y_1, y_2, ..., y_k)$ or $(y_1, y_2, ..., y_k, x_1, x_2, ..., x_k)$ with equal probability. In the former case, the commitment of every ith output bit will be (x_i, y_i), which is $a_i \oplus 0$; in the latter case, the commitment of every ith output bit will be (y_i, x_i), which is $a_i \oplus 1$. Therefore, the correctness of the operation is verified.

2.4 Adding Two Integers in $\mathbb{Z}/k\mathbb{Z}$

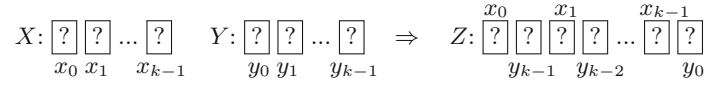
For $k \geq 3$, we first introduce two schemes of encoding integers in $\mathbb{Z}/k\mathbb{Z}$, the ♣-scheme and the ♡-scheme. The ♣-scheme uses one ♣ card and $k-1$ ♡ cards arranged in a row. An integer i corresponds to an arrangement where the ♣ card is the $(i+1)$-th card from the left, e.g.. ♡♣♡ encodes 1 in $\mathbb{Z}/3\mathbb{Z}$. Conversely, the ♡-scheme uses one♡ card and $k-1$ ♣ cards arranged in a row. An integer i corresponds to an arrangement where the ♡ card is the $(i+1)$-th card from the left, e.g. ♣♣♡♣ encodes 2 in $\mathbb{Z}/4\mathbb{Z}$.

Suppose we have integers a and b in $\mathbb{Z}/k\mathbb{Z}$, with a encoded in ♡-scheme by a sequence of face-down cards $X = (x_0, x_1, ..., x_{k-1})$, and b encoded in ♣-scheme by a sequence of face-down cards $Y = (y_0, y_1, ..., y_{k-1})$. We want to securely compute the sum $a+b \pmod k$ and have it encoded in ♡-scheme without using any additional card.

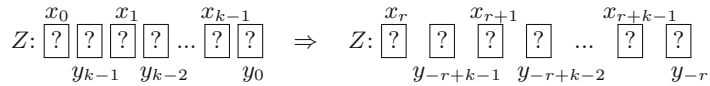
The intuition of this protocol is that we transform a and b into $a-r$ and $b+r$ for a random $r \in \mathbb{Z}/k\mathbb{Z}$, reveal $b+r$, and then shift the cards encoding $a-r$ to the right by $b+r$ positions to make them encode $(a-r)+(b+r) = a+b$. This technique was first used by Shinagawa et al. [15] in the context of using regular k-gon cards to encode integers in $\mathbb{Z}/k\mathbb{Z}$.

First, take the cards from X and Y in the following order and place them on a single row from left to right: the leftmost card of X, the rightmost card of Y,

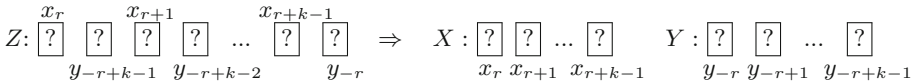
the second leftmost card of X, the second rightmost card of Y, and so on. The cards now form a new sequence $Z = (x_0, y_{k-1}, x_1, y_{k-2}, ..., x_{k-1}, y_0)$.



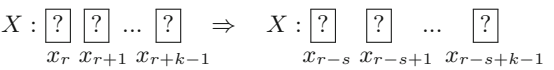
Apply a random k-section cut on Z, transforming the sequence into $(x_r, y_{-r+k-1}, x_{r+1}, y_{-r+k-2}, ..., x_{r+k-1}, y_{-r})$ for a uniformly random $r \in \mathbb{Z}/k\mathbb{Z}$, where the indices are taken in mod k.



Take the cards in Z from left to right and place them at these positions in X and Y in the following order: the leftmost position of X, the rightmost position of Y, the second leftmost position of X, the second rightmost position of Y, and so on. We now have sequences $X = (x_r, x_{r+1}, ..., x_{r+k-1})$ and $Y = (y_{-r}, y_{-r+1}, ..., y_{-r+k-1})$, which encode $a - r$ and $b + r$, respectively.



Turn all cards in Y face-up to reveal $s = b + r$. Note that this revelation does not leak any information of b because $b + r$ has an equal probability to be any integer in $\mathbb{Z}/k\mathbb{Z}$ no matter what b is. Then, we shift the cards in X to the right by s positions, transforming X into $(x_{r-s}, x_{r-s+1}, ..., x_{r-s+k-1})$.



Therefore, we now have a sequence X encoding $a - r + s = (a - r) + (b + r) = a + b$ in $\heartsuit$-scheme as desired.

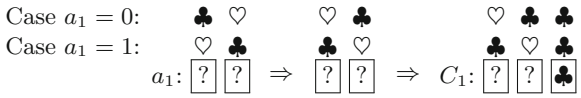
3 Our Main Protocol

We get back to our main problem. Observe that if we treat each input a_i as an integer, the value of $E(a_1, ..., a_n)$ depends only on the sum $s_n = \sum_{i=1}^{n} a_i$. Therefore, we will first develop a protocol to compute that sum. The intuition of this protocol is that for each $k = 2, 3, ..., n$, we inductively compute the sum $s_k = \sum_{i=1}^{k} a_i$ in $\mathbb{Z}/(k+1)\mathbb{Z}$. Note that since s_k is at most k, its value in $\mathbb{Z}/(k+1)\mathbb{Z}$ does not change from its actual value.

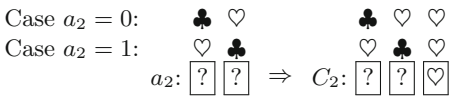
3.1 Summation of the First k Bits

We will show that if we have two additional cards, one ♣ and one ♡, we can compute the sum s_k for every $k = 2, 3, ..., n$ by the following procedure.

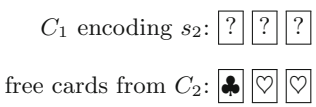
First, swap the two cards in the commitment of a_1 and place an additional ♣ card face-down to the right of them. The resulting sequence, called C_1, encodes a_1 in $\mathbb{Z}/3\mathbb{Z}$ in ♡-scheme.



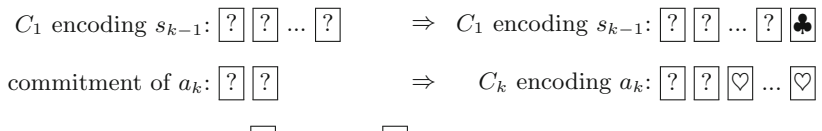
Then, put an additional ♡ card face-down to the right of the commitment of a_2. The resulting sequence, called C_2, encodes a_2 in $\mathbb{Z}/3\mathbb{Z}$ in ♣-scheme.



We then apply the addition protocol introduced in Sect. 2.4 to store the sum $s_2 = a_1 + a_2$ in $\mathbb{Z}/3\mathbb{Z}$ encoded in ♡-scheme in C_1. We also now have two ♡ cards and one ♣ card from C_2 after we turned them face-up. These cards are called *free cards* and are available to be used later in the protocol.



Inductively, for each $k \geq 3$, after we finish computing s_{k-1}, we now have a sequence C_1 of k face-down cards encoding s_{k-1} in $\mathbb{Z}/k\mathbb{Z}$ in ♡-scheme. We also have $k-1$ free ♡ cards and one free ♣ card from C_{k-1} after we turned them face-up. Append the free ♣ card face-down to the right of C_1, making the sequence now encode s_{k-1} in $\mathbb{Z}/(k+1)\mathbb{Z}$ in ♡-scheme. Also, place the $k-1$ free ♡ cards face-down to the right of the commitment of a_k. The resulting sequence, called C_k, encodes a_k in $\mathbb{Z}/(k+1)\mathbb{Z}$ in ♣-scheme.



Then, apply the addition protocol to compute the sum $s_{k-1} + a_k \pmod{k+1} = s_k \pmod{k+1} = s_k$ and have it encoded in ♡-scheme by C_1 as desired.

C_1 encoding s_k: $\boxed{?}\ \boxed{?}\ ...\ \boxed{?}$

free cards from C_k: $1\times\boxed{\clubsuit}$, $k\times\boxed{\heartsuit}$

Therefore, starting with one additional ♣ card and one additional ♡ card, we can compute the sum $s_k = \sum_{i=1}^{k} a_i$ for every $k = 2, 3, ..., n$.

3.2 Putting Together

The summation protocol introduced in Sect. 3.1 requires two additional cards to compute s_k. However, we can compute the equality function without using any additional card by the following procedure.

First, apply the random bit XOR protocol in Sect. 2.3 to transform the input into $(a_1 \oplus r, a_2 \oplus r, ..., a_n \oplus r)$ for a random bit $r \in \{0, 1\}$. Then, turn the two cards encoding the nth bit face-up to reveal $a_n \oplus r$. Note that this revelation does not leak any information of a_n because seeing ♣♡ and ♡♣ each has probability 1/2 no matter whether a_n is 0 or 1.

If the cards are ♣♡, i.e. $a_n \oplus r = 0$, the equality function outputs 1 if and only if $a_i \oplus r = 0$ for every $i = 1, ..., n-1$, which is equivalent to $\sum_{i=1}^{n-1}(a_i \oplus r) = 0$. Note that we now have one free ♣ card and one free ♡ card from the cards we just turned face-up. With these two additional cards, we can apply the summation protocol to compute $\sum_{i=1}^{n-1}(a_i \oplus r)$ as desired. On the other hand, if the two rightmost cards are ♡♣, i.e. $a_n \oplus r = 1$, the equality function outputs 1 if and only if $a_i \oplus r = 1$ for every $i = 1, ..., n-1$, which is equivalent to $\sum_{i=1}^{n-1}(a_i \oplus r \oplus 1) = 0$. Therefore, we can swap the two cards encoding every bit so that each ith bit becomes $a_i \oplus r \oplus 1$ and then apply the same protocol.

Note that the final sum is encoded in ♡-scheme by a row of n cards, where the equality function outputs 1 if and only if the sum is zero, i.e. the ♡ card is at the leftmost position. However, we do not want to reveal any information about the actual value of the sum except whether it is zero or not. Therefore, we apply a final random cut on the sequence of $n-1$ rightmost cards (all cards in the row except the leftmost one) to make all the cases where the sum is not zero indistinguishable. Finally, we turn all cards face-up and locate the position of the ♡ card. If it is the leftmost card in the row, then output 1; otherwise output 0.

We use one random 2-section cut in the random bit XOR operation, $n-2$ random k-section cuts for computing the sum of $n-1$ bits, and one random cut in the final shuffle. Therefore, the total number of shuffles used in the whole protocol is n.

4 Applications

4.1 Computing Other Symmetric Functions

A function $f : \{0, 1\}^n \to \mathbb{Z}$ is called symmetric if

$$f(a_1, ..., a_n) = f(a_{\sigma_1}, ..., a_{\sigma_n})$$

for any $a_1, ..., a_n$ and any permutation $(\sigma_1, ..., \sigma_n)$ of $(1, ..., n)$. A symmetric function f is called doubly symmetric if

$$f(a_1, ..., a_n) = f(1 - a_1, ..., 1 - a_n)$$

for any $a_1, ..., a_n$. For example, the equality function is doubly symmetric, while the majority function is symmetric but not doubly symmetric. Another example of a doubly symmetric function is $f(a_1, ..., a_n) = a_1 \oplus ... \oplus a_n$ for an even n.

Observe that for any symmetric function $f : \{0,1\}^n \to \mathbb{Z}$, the value of $f(a_1, ..., a_n)$ depends only on the sum $\sum_{i=1}^{n} a_i$, hence f can be written as

$$f(a_1, ..., a_n) = g\left(\sum_{i=1}^{n} a_i\right)$$

for some function $g : \{0, ..., n\} \to \mathbb{Z}$. Also, if f is doubly symmetric, we have $g(a) = g(n - a)$ for any $a \in \{0, ..., n\}$.

Our protocol can also be applied to compute any doubly symmetric function. Let $f : \{0,1\}^n \to \mathbb{Z}$ be any doubly symmetric function and let $g : \{0, ..., n\} \to \mathbb{Z}$ be a function such that

$$f(a_1, ..., a_n) = g\left(\sum_{i=1}^{n} a_i\right).$$

First, we apply the random bit XOR protocol with a random bit $r \in \{0,1\}$ to every input a_i and then reveal $a_n \oplus r$ (without leaking any information of a_n since $a_n \oplus r$ has an equal probability to be 0 and 1 no matter whether a_n is 0 or 1).

Since f is doubly symmetric, if $a_n \oplus r = 0$, we have

$$\begin{aligned} f(a_1, ..., a_n) &= f(a_1 \oplus r, ..., a_n \oplus r) \\ &= g\left(\sum_{i=1}^{n} (a_i \oplus r)\right) \\ &= g\left(\sum_{i=1}^{n-1} (a_i \oplus r)\right), \end{aligned}$$

so we can apply the summation protocol to compute $\sum_{i=1}^{n-1}(a_i \oplus r)$. On the other hand, if $a_n \oplus r = 1$, we have $a_n \oplus r \oplus 1 = 0$, so

$$\begin{aligned} f(a_1, ..., a_n) &= f(a_1 \oplus r \oplus 1, ..., a_n \oplus r \oplus 1) \\ &= g\left(\sum_{i=1}^{n} (a_i \oplus r \oplus 1)\right) \\ &= g\left(\sum_{i=1}^{n-1} (a_i \oplus r \oplus 1)\right), \end{aligned}$$

hence we can swap the two cards encoding every bit and apply the same protocol to compute $\sum_{i=1}^{n-1}(a_i \oplus r \oplus 1)$.

For each $b \in \operatorname{Im} f = \operatorname{Im} g$, let $P_b = \{a \in \{0, 1, ..., n\} | g(a) = b\}$. We now have a row of n cards encoding the sum in ♡-scheme. Recall that in ♡-scheme, an integer i corresponds to an arrangement where the $(i+1)$-th card from the left being ♡. Therefore, we can take from the row all the cards corresponding to integers in P_b, i.e. the $(i+1)$-th card from the left for every $i \in P_b$, apply a random cut on them, and put them back into the row at their original positions in order to make all the cases where the sum is in P_b indistinguishable. We need to separately apply such random cut for every $b \in \operatorname{Im} f$ such that $|P_b| > 1$. These random cuts ensure that turning the cards face-up does not reveal any information about the sum except the output value of g. Finally, we turn all cards face-up to reveal an integer s and output $g(s)$. The number of required cards is $2n$, and the total number of shuffles is at most $n - 1 + |\operatorname{Im} f|$.

For a function that is symmetric but not doubly symmetric, we can directly apply the summation protocol to compute the sum $s_n = \sum_{i=1}^{n} a_i$, apply the above random cut for every $b \in \operatorname{Im} f$ such that $|P_b| > 1$, and output $g(s_n)$, although it requires two additional cards at the beginning. Therefore, the number of required cards is $2n + 2$, and the total number of shuffles is at most $n - 1 + |\operatorname{Im} f|$.

4.2 Optimality

There is a protocol developed by Mizuki et al. [6] that can compute the sum of n input bits using only $O(\log n)$ cards, but their protocol restricts the order of submission of the inputs so that the cards can be reused. Any protocol that the inputs are submitted simultaneously requires at least $2n$ cards as we need two cards for a commitment of each person's bit, hence our protocol is the optimal one for computing any doubly symmetric function.

For computing symmetric functions that are not doubly symmetric, the protocol of Nishida et al. [11] also uses $2n + 2$ cards to compute any symmetric function $f : \{0, 1\}^n \to \{0, 1\}$. Their protocol has a benefit that the output is in *committed-format*, i.e. encoded in the same format as the input (♣♡ for 0 and ♡♣ for 1), so the output can be securely used as an input of another function. However, our protocol uses fewer number of shuffles and also has a benefit that the output is not restricted to be binary, hence supporting functions with more than two possible outputs (an example of such function is the majority function that supports the case of a tie for an even n, which has three possible outputs).

5 Future Work

For computing the equality function or any doubly symmetric function, our protocol is optimal in terms of number of cards as it matches the trivial lower bound of $2n$. However, there is still an open problem to find a committed-format protocol that uses $2n$ cards, or a non-committed-format one with the same number

of cards but uses a fewer number of shuffles. For symmetric functions that are not doubly symmetric, an open problem is to find a protocol that computes such functions with less than $2n + 2$ cards.

Another interesting future work is to prove the lower bound of the number of cards or the number of shuffles required to compute such functions, either for a committed-format protocol or for any protocol.

References

1. Abe, Y., Hayashi, Y., Mizuki, T., Sone, H.: Five-card AND protocol in committed format using only practical shuffles. In: Proceedings of the 5th ACM on ASIA Public-Key Cryptography Workshop (APKC 2018), pp. 3–8 (2018)
2. Crépeau, C., Kilian, J.: Discreet solitary games. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 319–330. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_27
3. Boer, B.: More efficient match-making and satisfiability *the five card trick*. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 208–217. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_23
4. Koch, A.: The Landscape of Optimal Card-based Protocols. Cryptology ePrint Archive https://eprint.iacr.org/2018/951/20181009:160322 (2018)
5. Koch, A., Walzer, S., Härtel, K.: Card-based cryptographic protocols using a minimal number of cards. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 783–807. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_32
6. Mizuki, T., Asiedu, I.K., Sone, H.: Voting with a logarithmic number of cards. In: Mauri, G., Dennunzio, A., Manzoni, L., Porreca, A.E. (eds.) UCNC 2013. LNCS, vol. 7956, pp. 162–173. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39074-6_16
7. Mizuki, T., Kumamoto, M., Sone, H.: The five-card trick can be done with four cards. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 598–606. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_36
8. Mizuki, T., Sone, H.: Six-card secure AND and four-card secure XOR. In: Deng, X., Hopcroft, J.E., Xue, J. (eds.) FAW 2009. LNCS, vol. 5598, pp. 358–369. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02270-8_36
9. Mizuki, T., Uchiike, F., Sone, H.: Securely computing XOR with 10 cards. Australas. J. Comb. **36**, 279–293 (2006)
10. Niemi, V., Renvall, A.: Secure multiparty computations without computers. Theor. Comput. Sci. **191**, 173–183 (1998)
11. Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Card-based protocols for any boolean function. In: Jain, R., Jain, S., Stephan, F. (eds.) TAMC 2015. LNCS, vol. 9076, pp. 110–121. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17142-5_11
12. Nishida, T., Mizuki, T., Sone, H.: Securely computing the three-input majority function with eight cards. In: Dediu, A.-H., Martín-Vide, C., Truthe, B., Vega-Rodríguez, M.A. (eds.) TPNC 2013. LNCS, vol. 8273, pp. 193–204. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45008-2_16
13. Ruangwises, S., Itoh, T.: AND protocols using only uniform shuffles. In: van Bevern, R., Kucherov, G. (eds.) CSR 2019. LNCS, vol. 11532, pp. 349–358. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19955-5_30

14. Shinagawa, K., Mizuki, T.: The six-card trick: secure computation of three-input equality. In: Lee, K. (ed.) ICISC 2018. LNCS, vol. 11396, pp. 123–131. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12146-4_8
15. Shinagawa, K., et al.: Multi-party computation with small shuffle complexity using regular polygon cards. In: Au, M.-H., Miyaji, A. (eds.) ProvSec 2015. LNCS, vol. 9451, pp. 127–146. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26059-4_7
16. Stiglic, A.: Computations with a deck of cards. Theor. Comput. Sci. **259**, 671–678 (2001)
17. Ueda, I., Nishimura, A., Hayashi, Y., Mizuki, T., Sone, H.: How to implement a random bisection cut. In: Martín-Vide, C., Mizuki, T., Vega-Rodríguez, M.A. (eds.) TPNC 2016. LNCS, vol. 10071, pp. 58–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49001-4_5