Using Five Cards to Encode Each Integer in $\mathbb{Z}/6\mathbb{Z}$

Suthee Ruangwises$^1$

$^1$Department of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan

Abstract

Research in secure multi-party computation using a deck of playing cards, often called card-based cryptography, dates back to 1989 when Den Boer introduced the “five-card trick” to compute the logical AND function. Since then, many protocols to compute different functions have been developed. In this paper, we propose a new encoding scheme that uses five cards to encode each integer in $\mathbb{Z}/6\mathbb{Z}$. Using this encoding scheme, we develop protocols that can copy a commitment with 13 cards, add two integers with 10 cards, and multiply two integers with 14 cards. All of our protocols are the currently best known protocols in terms of the required number of cards. Our encoding scheme can be generalized to encode integers in $\mathbb{Z}/n\mathbb{Z}$ for other values of $n$ as well.

Keywords: card-based cryptography, secure multi-party computation, function, ring of integers modulo $n$

1 Introduction

Secure multi-party computation, one of the most actively studied areas in cryptography, involves situations where multiple parties want to compare their secret information without revealing it. Many researchers focus on developing secure multi-party computation protocols using physical objects such as a deck of playing cards, creating a research area often called card-based cryptography. The benefit of card-based protocols is that they provide simple solutions to real-world situations using only objects found in everyday life without requiring computers. Moreover, these intuitive protocols are easy to understand and verify the correctness and security, even for non-experts in cryptography, and thus can be used for educational purposes to teach the concept of secure multi-party computation.

1.1 Protocols of Boolean Functions

Research in card-based cryptography dates back to 1989 when Den Boer [3] proposed a protocol called the five-card trick to compute the logical AND function on two players’ bits $a$ and $b$.

The five-card trick protocol uses three identical $\spadesuit$s and two identical $\heartsuit$s, with all cards having indistinguishable back sides. We use a commitment $\spadesuit\heartsuit$ to encode 0, and a

---

$^1$ruangwises@gmail.com
commitment ♠ to encode 1. First, each player is given one ♠ and one ♥, and another ♠ is put face-down on a table. The first player places his/her commitment of a face-down to the left of the ♠ on the table, while the second player places his/her commitment of b face-down to the right of it. Then, we swap the two cards in the commitment of b, resulting in the following four possible sequences.

\[(a, b) = (0, 0) : \begin{array}{cccc}
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\end{array} \Rightarrow \begin{array}{cccc}
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\end{array}
\]

\[(a, b) = (0, 1) : \begin{array}{cccc}
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\end{array} \Rightarrow \begin{array}{cccc}
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\end{array}
\]

\[(a, b) = (1, 0) : \begin{array}{cccc}
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\end{array} \Rightarrow \begin{array}{cccc}
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\end{array}
\]

\[(a, b) = (1, 1) : \begin{array}{cccc}
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\end{array} \Rightarrow \begin{array}{cccc}
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit & \heartsuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\spadesuit & \spadesuit & \spadesuit & \spadesuit \\
\end{array}
\]

Among all cases, there are only two possible sequences in a cyclic rotation of the deck: ♠♠♠♠♠ and ♠♠♠♠♠, with the latter occurring if and only if a = b = 1. We can hide the initial position of the cards by applying a random cut to shift the sequence into a uniformly random cyclic shift, i.e. a permutation uniformly chosen at random from \{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 \& b = 1 without leaking any other information.

Since the introduction of the five-card trick, several other protocols to compute the AND function have been developed. These subsequent protocols \[1, 2, 5, 6, 7, 8, 11, 13, 14, 17, 21, 29\] either reduced the number of required cards or improved properties of the protocol involving output format, type of shuffles, running time, etc.

Apart from AND protocols, protocols to compute other Boolean functions have also been developed, such as logical XOR protocols \[5, 14, 15, 32\], copy protocols \[5, 9, 14\] (duplicating a commitment), majority function protocols \[19, 31\] (deciding whether there are more 1s than 0s in the inputs), equality function protocols \[25, 27\] (deciding whether all inputs are equal), and a voting protocol \[12\] (adding bits and storing the sum in binary representation).

Nishida et al. \[18\] 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 Protocols of Non-Boolean Functions

While almost all of the existing protocols were designed to compute Boolean functions, a few results also focused on computing functions in \(\mathbb{Z}/n\mathbb{Z}\) for \(n > 2\). Shinagawa et al. \[28\] used a regular n-gon card to encode each integer in \(\mathbb{Z}/n\mathbb{Z}\) and proposed a copy protocol and an addition protocol for integers in \(\mathbb{Z}/n\mathbb{Z}\). Their encoding scheme can be straightforwardly converted to the one using regular cards. In another result, Shinagawa and Mizuki \[26\] developed a protocol to multiply two integers in \(\mathbb{Z}/3\mathbb{Z}\) using triangle cards. Their idea can also be generalized to multiply integers in \(\mathbb{Z}/n\mathbb{Z}\) using regular cards.

Another straightforward method to compute functions on \(\mathbb{Z}/n\mathbb{Z}\) is to convert each integer in \(\mathbb{Z}/n\mathbb{Z}\) into its binary representation and encode each digit with two cards, resulting in the total of 2⌈lg n⌉ cards, and then apply the protocol of Nishida et al. \[18\] to compute these functions.
| Encoding Scheme            | Number of required cards |
|----------------------------|--------------------------|
|                            | Copy | Addition | Multiplication |
| Shinagawa et al. [28]      | 18   | 12       | 42             |
| Nishida et al. [18]        | 14   | 22       | 22             |
| Our scheme (§5)            | 13   | 10       | 14             |

Table 1: The number of required cards for copy, addition, and multiplication protocols in $\mathbb{Z}/6\mathbb{Z}$ using each encoding scheme

1.3 Our Contribution

In this paper, we propose a new encoding scheme that uses five cards to encode each integer in $\mathbb{Z}/6\mathbb{Z}$. The idea behind this scheme is to use the first two cards and the last three cards to represent its residues in modulo 2 and modulo 3, respectively, and then use the converted scheme of Shinagawa et al. [28] to encode each part. This simple trick significantly reduces the number of required cards for every basic protocol, which is the main objective of developing card-based protocols. Using this encoding scheme, we present protocols that can copy a commitment with 13 cards, add two integers with ten cards, and multiply two integers with 14 cards. These three protocols are the essential ones that enable us to compute any polynomial function $f : (\mathbb{Z}/6\mathbb{Z})^k \rightarrow \mathbb{Z}/6\mathbb{Z}$. All of these three protocols are the currently best known ones in terms of the required number of cards (see Table 1).

Our encoding scheme can be generalized to other rings of integers modulo $n$, including $\mathbb{Z}/12\mathbb{Z}$ where our protocols are the currently best known ones as well.

2 Preliminaries

2.1 Sequence of Cards

For $0 \leq a < n$, define $E_n(a)$ to be a sequence of consecutive $n$ cards, with all of them being $\heartsuit$ except the $(a + 1)$-th card from the left being $\clubsuit$, e.g. $E_4(2)$ is $\heartsuit\heartsuit\clubsuit\heartsuit$. Unless stated otherwise, the cards in $E_n(a)$ are arranged horizontally as defined above. In some situations, however, we may arrange the cards vertically, with the leftmost card becoming the topmost card and the rightmost card becoming the bottommost card.

Many existing protocols use the sequence $E_n(a)$ to encode an integer $a$ in $\mathbb{Z}/n\mathbb{Z}$, such as millionaire protocols [10, 16], a ranking protocol [30], and protocols of zero-knowledge proof for logic puzzles [4, 20, 22, 23, 24].

2.2 Matrix

In an $m \times n$ matrix of cards, let Row $i$ ($0 \leq i < m$) denote the $(i + 1)$-th topmost row of the matrix, and Column $j$ ($0 \leq j < n$) denote the $(j + 1)$-th leftmost column of the matrix.

2.3 Pile-Shifting Shuffle

In a pile-shifting shuffle on an $m \times n$ matrix, we shift the columns of the matrix by a random cyclic shift unknown to all parties, i.e. move each Column $\ell$ to Column $\ell + r$ for a uniformly
random \( r \in \mathbb{Z}/n\mathbb{Z} \) (where the indices are taken modulo \( n \)). See Fig. 1. This operation was introduced by Shinagawa et al. [28].

The pile-shifting shuffle can be implemented in real world by putting all cards in each column into an envelope, and then applying the random cut to the sequence of envelopes [33].

![Figure 1: An example of a pile-shifting shuffle on a 4 \times 6 matrix](image)

### 3 Encoding Scheme of Shinagawa et al.

Shinagawa et al. [28] proposed an encoding scheme that uses a regular \( n \)-gon card to encode each integer in \( \mathbb{Z}/n\mathbb{Z} \), which can be straightforwardly converted to the one using regular cards. In the converted scheme, an integer \( a \) in \( \mathbb{Z}/n\mathbb{Z} \) is encoded by a sequence \( E_n(a) \) introduced in the previous section (see Fig. 2 for the case \( n = 6 \)).

![Figure 2: Commitments of integers in \( \mathbb{Z}/6\mathbb{Z} \) in the encoding scheme of Shinagawa et al.](image)

We focus on three basic arithmetic protocols: the copy protocol, the addition protocol, and the multiplication protocol. These three protocols can be combined to compute any polynomial function \( f : (\mathbb{Z}/n\mathbb{Z})^k \to \mathbb{Z}/n\mathbb{Z} \).

#### 3.1 Copy Protocol

Given a sequence \( A \) of \( E_n(a) \), this protocol creates an additional copy of \( A \) without revealing \( a \). It was converted from a copy protocol of Shinagawa et al. [28] which uses regular \( n \)-gon cards, and is also a generalization of a Boolean copy protocol of Mizuki and Sone [14].

1. Reverse the \( n - 1 \) rightmost cards of \( A \), i.e. move each \((i + 1)\)-th leftmost card of \( A \) to become the \( i \)-th rightmost card for \( i = 1, 2, ..., n - 1 \). This modified sequence, called \( A' \), now encodes \(-a \mod n\).
2. Construct a $3 \times n$ matrix $M$ by placing the sequence $A'$ in Row 0 and a sequence $E_n(0)$ in Row 1 and Row 2.

3. Apply the pile-shifting shuffle to $M$. Note that Row 0 of $M$ now encodes $-a + r \mod n$, and Row 1 and Row 2 now encode $r \mod n$ for a uniformly random $r \in \mathbb{Z}/n\mathbb{Z}$.

4. Turn over all cards in Row 0 of $M$. Locate the position of a ♣. Suppose it is at Column $j$.

5. Shift the columns of $M$ to the left by $j$ columns, i.e. move every Column $\ell$ to Column $\ell - j$ (where the indices are taken modulo $n$). Turn over all face-up cards.

6. The sequences in Row 1 and Row 2 of $M$ now both encode $r - (-a + r) \equiv a \mod n$, so we now have two copies of $A$ as desired.

This protocol uses $n$ extra cards (one ♣ and $n - 1$ ♥s) in addition to the ones in $A$ and $A'$. Therefore, the total number of required cards is $3n$.

3.2 Addition Protocol

Given sequences $A$ and $B$ of $E_n(a)$ and $E_n(b)$, respectively, this protocol computes the sum $a + b \mod n$ without revealing $a$ or $b$. It was also converted from an addition protocol of Shinagawa et al. [28] which uses regular $n$-gon cards.

1. Reverse the $n - 1$ rightmost cards of $A$, i.e. move each $(i+1)$-th leftmost card of $A$ to become the $i$-th rightmost card for $i = 1, 2, ..., n - 1$. This modified sequence, called $A'$, now encodes $-a \mod n$.

2. Construct a $2 \times n$ matrix $M$ by placing the sequence $A'$ in Row 0 and the sequence $B$ in Row 1.

3. Apply the pile-shifting shuffle to $M$. Note that Row 0 and Row 1 of $M$ now encode $-a + r \mod n$ and $b + r \mod n$, respectively, for a uniformly random $r \in \mathbb{Z}/n\mathbb{Z}$.

4. Turn over all cards in Row 0 of $M$. Locate the position of a ♣. Suppose it is at Column $j$.

5. Shift the columns of $M$ to the left by $j$ columns, i.e. move every Column $\ell$ to Column $\ell - j$ (where the indices are taken modulo $n$). Turn over all face-up cards.

6. The sequence in Row 1 of $M$ now encodes $(b + r) - (-a + r) \equiv a + b \mod n$ as desired.

This protocol does not use any extra card other than the ones in $A$ and $B$. Therefore, the total number of required cards is $2n$. 
3.3 Multiplication Protocol

Given sequences $A$ and $B$ of $E_n(a)$ and $E_n(b)$, respectively, this protocol computes the product $a \cdot b \pmod{n}$ without revealing $a$ or $b$. It is a generalization of the protocol of Shinagawa and Mizuki [26] to multiply two integers in $\mathbb{Z}/3\mathbb{Z}$, and is also a generalization of the Boolean AND protocol of Mizuki and Sone [14].

The intuition of this protocol is that we will create sequences $A_0, A_1, ..., A_{n-1}$ encoding $0, a, 2a, ..., (n-1)a \pmod{n}$, respectively, and then select the sequence $A_b$ as an output.

1. Let $A_1 = A$. If $n \geq 4$, we perform the following procedures for $n - 3$ rounds. In each $i$-th round ($i = 1, 2, ..., n - 3$) when we already have sequences $A_1, A_2, ..., A_i$, apply the copy protocol to create a copy of $A_1$ and a copy of $A_i$. Then, apply the addition protocol to the copy of $A_1$ and the copy of $A_i$. The resulting sequence, called $A_{i+1}$, encodes $a + ia \equiv (i+1)a \pmod{n}$.

2. We now have sequences $A_1, A_2, ..., A_{n-2}$. If $n \geq 3$, apply the copy protocol to create a copy of $A_1$ again. Reverse the $n - 1$ rightmost cards of that copy, i.e. move each $(i+1)$-th leftmost card to become the $i$-th rightmost card for $i = 1, 2, ..., n - 1$. This modified sequence, called $A_{n-1}$, now encodes $-a \equiv (n-1)a \pmod{n}$.

3. Arrange $n$ extra cards (which can be taken from the cards left from the copy protocol in Step 2 for $n \geq 3$) as a sequence $E_n(0)$, called $A_0$. We now have sequences $A_0, A_1, ..., A_{n-1}$ as desired.

4. Construct an $(n + 1) \times n$ matrix $M$ by the following procedures (see Fig. 3).

(a) In Row 0, place the sequence $B$.

(b) In each column $\ell = 0, 1, ..., n - 1$, place the sequence $A_\ell$ arranged vertically from Row 1 to Row $n$.

5. Apply the pile-shifting shuffle to $M$.

6. Turn over all cards in Row 0. Locate the position of a $\spadesuit$. Suppose it is at Column $j$. 

![Figure 3: An $(n + 1) \times n$ matrix $M$ constructed in Step 4](image-url)
7. Select the sequence in Column \( j \) arranged vertically from Row 1 to Row \( n \). This is the sequence \( A_b \) encoding \( a \cdot b \mod n \) as desired.

In Step 1, in the \( i \)-th round we use \( n \) extra cards in the copy protocol besides the cards in \( A_1, A_2, \ldots, A_i, B \), and the copies of \( A_1 \) and \( A_i \), so the total number of cards is \((i + 3)n + n \leq n^2 + n\). In Step 2, we use \( n \) extra cards in the copy protocol besides the cards in \( A_1, A_2, \ldots, A_{n-2}, B \), and the copy of \( A_1 \), so the total number of cards is \( n^2 + n \). Therefore, the total number of required cards for this protocol is \( n^2 + n \). Note that the special case \( n = 3 \) works exactly like the multiplication protocol of Shinagawa and Mizuki [26], and the special case \( n = 2 \) works exactly like the six-card AND protocol of Mizuki and Sone [14].

In summary, using the encoding scheme of Shinagawa et al. for \( n = 6 \) requires 18, 12, and 42 cards for the copy, addition, and multiplication protocols, respectively.

4 Encoding Scheme of Nishida et al.

Nishida et al. [18] developed a protocol to compute any \( n \)-variable Boolean function using \( 2n + 6 \) cards, where each bit \( x \) in the inputs and output is encoded by \( E_2(x) \). Their protocol also retains commitments of the inputs for further use. Hence, this protocol requires \( 2n \) cards for the inputs and two cards for the output, and actually uses four extra cards besides the ones in the inputs and output for the computation: two \( \spadesuit \)s and two \( \heartsuit \)s.

We write each integer \( a \in \mathbb{Z}/6\mathbb{Z} \) in its binary representation \( a = (a_2, a_1, a_0) \), where \( a = 4a_2 + 2a_1 + a_0 \) and \( a_0, a_1, a_2 \in \{0, 1\} \). Each bit \( a_i \) is encoded by \( E_2(a_i) \), so we encode \( a \) by a commitment of six cards consisting of \( E_2(a_2), E_2(a_1), \) and \( E_2(a_0) \) arranged in this order from left to right (see Fig. 4).

\begin{align*}
\begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} \\
\begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} \\
\begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} \\
\begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} & \begin{array}{ccc}
\spadesuit & \heartsuit & \spadesuit \\
\heartsuit & \spadesuit & \heartsuit \\
\spadesuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\heartsuit & \heartsuit & \heartsuit \\
\end{array} \\
\end{align*}

Figure 4: Commitments of integers in \( \mathbb{Z}/6\mathbb{Z} \) in the encoding scheme of Nishida et al.

4.1 Copy Protocol

To copy a commitment of \( a = (a_2, a_1, a_0) \), we apply the protocol in Section 3.1 to copy the sequences \( E_2(a_0), E_2(a_1), \) and \( E_2(a_2) \) separately. Since the two extra cards used in that protocol can be reused in each computation, we use only two extra cards (one \( \spadesuit \) and one \( \heartsuit \)) besides the 12 cards encoding the inputs and outputs, resulting in the total of 14 cards.

4.2 Addition Protocol

Suppose we have integers \( a = (a_2, a_1, a_0) \) and \( b = (b_2, b_1, b_0) \).
Let $S = \{0, 1\}^3 - \{(1, 1, 0), (1, 1, 1)\}$. Consider the following function $f_+ : \{0, 1\}^6 \rightarrow \{0, 1\}^3$. Define

$$f_+(a_2, a_1, a_0, b_2, b_1, b_0) := (c_2, c_1, c_0),$$

where $(c_2, c_1, c_0)$ is the binary representation of $a + b \pmod{6}$ if $(a_2, a_1, a_0), (b_2, b_1, b_0) \in S$. We can define $f_+(a_2, a_1, a_0, b_2, b_1, b_0)$ to be any value if either $(a_2, a_1, a_0)$ or $(b_2, b_1, b_0)$ is not in $S$.

We apply the protocol of Nishida et al. [18] to compute $c_0, c_1, c_2$ separately. As explained at the beginning of Section 4, this protocol retains the commitments of the inputs, and uses four extra cards (which can be reused in each computation) besides the ones in the inputs and outputs. Therefore, we use only four extra cards (two ♣️s and two ♥️s) besides the 18 cards encoding the inputs and outputs, resulting in the total of 22 cards.

### 4.3 Multiplication Protocol

Similarly to the addition protocol, consider a function $f_\times : \{0, 1\}^6 \rightarrow \{0, 1\}^3$ with

$$f_\times(a_2, a_1, a_0, b_2, b_1, b_0) := (c_2, c_1, c_0),$$

where $(c_2, c_1, c_0)$ is the binary representation of $a \cdot b \pmod{6}$ if $(a_2, a_1, a_0), (b_2, b_1, b_0) \in S$, and with $f_\times(a_2, a_1, a_0, b_2, b_1, b_0)$ being any value if either $(a_2, a_1, a_0)$ or $(b_2, b_1, b_0)$ is not in $S$.

Like in the addition protocol, we apply the protocol of Nishida et al. [18] to compute $c_0, c_1, c_2$ separately, which requires 22 cards in total.

In summary, using the encoding scheme of Nishida et al. requires 14, 22, and 22 cards for the copy, addition, and multiplication protocols, respectively.

### 5 Our Encoding Scheme

In our encoding scheme, each integer $a \in \mathbb{Z}/6\mathbb{Z}$ is written as $(a_1, a_2)$, where $a_1$ and $a_2$ are remainders of $a$ when divided by 2 and 3, respectively. By Chinese remainder theorem, the value of $a$ is uniquely determined by $(a_1, a_2)$\(^1\) We encode $a$ by a commitment of five cards, the first two cards being $E_2(a_1)$ and the last three cards being $E_3(a_2)$ (see Fig. 5).

![Commitments of integers in \(\mathbb{Z}/6\mathbb{Z}\) in our encoding scheme](image)

\(^1\)This can also be viewed as ring isomorphism $\mathbb{Z}/6\mathbb{Z} \cong (\mathbb{Z}/2\mathbb{Z}) \times (\mathbb{Z}/3\mathbb{Z})$. 

8
5.1 Copy Protocol

To copy a commitment of \( a = (a_1, a_2) \), we apply the protocol in Section 3.1 to copy the sequences \( E_2(a_1) \) and \( E_3(a_2) \) separately. Since the extra cards used in that protocol can be reused in each computation, we use only three extra cards (one ♣ and two ♥️) besides the ten cards encoding the inputs and outputs, resulting in the total of 13 cards.

5.2 Addition Protocol

Given \( a = (a_1, a_2) \) and \( b = (b_1, b_2) \), we have \( a + b \mod 6 = (c_1, c_2) \), where \( c_1 = a_1 + b_1 \mod 2 \) and \( c_2 = a_2 + b_2 \mod 3 \). The values of \( c_1 \) and \( c_2 \) can be computed separately by applying the protocol in Section 3.2, which does not use any extra card. Therefore, the total number of required cards is ten.

5.3 Multiplication Protocol

Like the addition protocol, we have \( a \cdot b \mod 6 = (c_1, c_2) \), where \( c_1 = a_1 \cdot b_1 \mod 2 \) and \( c_2 = a_2 \cdot b_2 \mod 3 \). The values of \( c_1 \) and \( c_2 \) can be computed separately by applying the protocol in Section 3.3, which in total uses six extra cards (two ♣️s and four ♥️s), so the total number of required cards is 16.

5.3.1 Optimization

By reusing cards, we can do a little better for the multiplication protocol. First, we compute \( c_1 \) using two extra cards (one ♣️ and one ♥️). After the computation, we only use two cards (one ♣️ and one ♥️) to encode \( c_1 \), so we now have four free cards (two ♣️s and two ♥️s) that can be used in other computation. Since computing \( c_2 \) requires six extra cards (two ♣️s and four ♥️s), we actually need only two more ♥️s besides the four free cards we have. Therefore, in total we can use only four extra cards (one ♣️ and three ♥️s), which reduces the number of required cards by two to 14.

In summary, using our encoding scheme for \( n = 6 \) requires 13, 10, and 14 cards for the copy, addition, and multiplication protocols, respectively.

6 Encoding Integers in Other Congruent Classes

Our encoding scheme can be generalized to encode integers in \( \mathbb{Z}/n\mathbb{Z} \) for any \( n = p_1^{b_1} p_2^{b_2} \ldots p_k^{b_k} \) such that \( k > 1 \), where \( p_1, p_2, \ldots, p_k \) are different primes and \( b_1, b_2, \ldots, b_k \) are positive integers.

For each \( a \in \mathbb{Z}/n\mathbb{Z} \), let \( a = (a_1, a_2, \ldots, a_k) \), where each \( a_i \) is the remainder of \( a \) when divided by \( p_i^{b_i} \). By Chinese remainder theorem, the value of \( a \) is uniquely determined by \( (a_1, a_2, \ldots, a_k) \). We encode each \( a_i \) by \( E_{p_i^{b_i}}(a_i) \), so we use total of \( \sum_{i=1}^{k} p_i^{b_i} \) cards for each commitment. We apply the protocols in Sections 3.1, 3.2, and 3.3 on each \( E_{p_i^{b_i}}(a_i) \) separately to perform the copy, addition, and multiplication, respectively.

Let \( m = \max_{i=1}^{k} p_i^{b_i} \), our encoding scheme requires \( 2 \sum_{i=1}^{k} p_i^{b_i} + m \) cards for the copy protocol, \( 2 \sum_{i=1}^{k} p_i^{b_i} \) cards for the addition protocol, and \( 2(\sum_{i=1}^{k} p_i^{b_i}) + m^2 - m \) cards for the multiplication protocol (before the optimization). By using the optimization technique
in Section 5.3.1 (computing the smallest modulus first and reusing the free cards in larger modulii), we can slightly reduce the number of required cards for the multiplication protocol.

In comparison, the encoding scheme of Shinagawa et al. requires $3n$ cards for copy, $2n$ cards for addition, and $n^2 + n$ cards for multiplication, while the encoding scheme of Nishida et al. requires $4\lceil \lg n \rceil + 2$ cards for copy and $6\lceil \lg n \rceil + 4$ cards for addition and multiplication.

The numbers of required cards for the copy, addition, and multiplication protocols for each applicable $n$ up to 20 are shown in Table 2. Besides $\mathbb{Z}/6\mathbb{Z}$, our encoding scheme is also the currently best known schemes in $\mathbb{Z}/12\mathbb{Z}$ for the protocols of all three functions. For the addition protocol, our encoding scheme is the currently best known one for every such $n$.

### Table 2: The number of required cards for copy, addition, and multiplication protocols (after the optimization) in each $\mathbb{Z}/n\mathbb{Z}$ using each encoding scheme, with the lowest number among each type of protocol boldfaced

| $\mathbb{Z}/n\mathbb{Z}$ | Shinagawa et al. [28] | Nishida et al. [18] | Our scheme ($\S$5) |
|-------------------------|----------------------|--------------------|--------------------|
|                         | Copy | Add. | Mult. | Copy | Add. | Mult. | Copy | Add. | Mult. |
| $\mathbb{Z}/6\mathbb{Z}$ | 18   | 12   | 42    | 14   | 22   | 22    | 13   | 10   | 14    |
| $\mathbb{Z}/10\mathbb{Z}$ | 30   | 20   | 110   | 18   | 28   | 28    | 19   | 14   | 32    |
| $\mathbb{Z}/12\mathbb{Z}$ | 36   | 24   | 156   | 18   | 28   | 28    | 18   | 14   | 23    |
| $\mathbb{Z}/14\mathbb{Z}$ | 42   | 28   | 210   | 18   | 28   | 28    | 25   | 18   | 58    |
| $\mathbb{Z}/15\mathbb{Z}$ | 45   | 30   | 240   | 18   | 28   | 28    | 21   | 16   | 33    |
| $\mathbb{Z}/18\mathbb{Z}$ | 54   | 36   | 342   | 22   | 34   | 34    | 31   | 22   | 92    |
| $\mathbb{Z}/20\mathbb{Z}$ | 60   | 40   | 420   | 22   | 34   | 34    | 23   | 18   | 34    |

7 **Future Work**

We developed an encoding scheme for integers in $\mathbb{Z}/6\mathbb{Z}$ which allows us to perform the copy, addition, and multiplication using 13, 10, and 14 cards, respectively, which are the lowest numbers among the currently known protocols. We also generalized our encoding scheme to other rings of integers modulo $n$, including $\mathbb{Z}/12\mathbb{Z}$ where our protocols are the currently best known ones as well.

A challenging future work is to develop encoding schemes in $\mathbb{Z}/n\mathbb{Z}$ that requires fewer cards for other values of $n$, especially when $n$ is a prime, or prove the lower bound of the number of required cards for each $n$. For $\mathbb{Z}/6\mathbb{Z}$, we have to use at least four cards to encode each integer, no matter what the encoding scheme is (because three cards of two types can be rearranged in at most three ways). Hence, the trivial lower bound of the number of required cards for every protocol is eight.

Also, all results so far have been focused on using only two types of cards. An interesting question is that if we allow more than two types of cards, can we lower the number of required cards? (In particular, three different cards can be rearranged in six ways, so it might be possible to encode each integer in $\mathbb{Z}/6\mathbb{Z}$ with three cards.)
References

[1] Y. Abe, Y. Hayashi, T. Mizuki and H. Sone. Five-Card AND Computations in Committed Format Using Only Uniform Cyclic Shuffles. *New Generation Computing*, 39(1): 97–114 (2021).

[2] Y. Abe, T. Mizuki and H. Sone. Committed-format AND protocol using only random cuts. *Natural Computing*, 20(4): 639–645 (2021).

[3] B. den Boer. More Efficient Match-Making and Satisfiability: the Five Card Trick. In *Proceedings of the Workshop on the Theory and Application of of Cryptographic Techniques (EUROCRYPT ’89)*, pp. 208–217 (1990).

[4] X. Bultel, J. Dreier, J.-G. Dumas, P. Lafourcade, D. Miyahara, T. Mizuki, A. Nagao, T. Sasaki, K. Shinagawa and H. Sone. Physical Zero-Knowledge Proof for Makaro. In *Proceedings of the 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pp. 111–125 (2018).

[5] C. Crépeau and J. Kilian. Discreet Solitary Games. In *Proceedings of the 13th Annual International Cryptology Conference (CRYPTO ’93)*, pp. 319–330 (1994).

[6] R. Isuzugawa, K. Toyoda, Y. Sasaki, D. Miyahara and T. Mizuki. A Card-Minimal Three-Input AND Protocol Using Two Shuffles. In *Proceedings of the 27th International Computing and Combinatorics Conference (COCOON)*, pp. 668–679 (2021).

[7] A. Koch. The Landscape of Optimal Card-based Protocols. Cryptology ePrint Archive https://eprint.iacr.org/2018/951 (2018).

[8] A. Koch, S. Walzer and K. Hārtel. Card-Based Cryptographic Protocols Using a Minimal Number of Cards. In *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 783–807 (2015).

[9] H. Kayama, K. Toyoda, D. Miyahara and T. Mizuki. New Card-based Copy Protocols Using Only Random Cuts. In *Proceedings of the 8th ACM on ASIA Public-Key Cryptography Workshop (APKC)*, pp. 13–22 (2021).

[10] D. Miyahara, Y. Hayashi, T. Mizuki and H. Sone. Practical card-based implementations of Yao’s millionaire protocol. *Theoretical Computer Science*, 803: 207–221 (2020).

[11] T. Mizuki. Card-based protocols for securely computing the conjunction of multiple variables. *Theoretical Computer Science*, 622: 34–44 (2016).

[12] T. Mizuki, I.K. Asiedu and H. Sone. Voting with a Logarithmic Number of Cards. In *Proceedings of the 12th International Conference on Unconventional Computation and Natural Computation (UCNC)*, pp. 162–173 (2013).

[13] T. Mizuki, M. Kumamoto and H. Sone. The Five-Card Trick Can Be Done with Four Cards. In *Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pp. 598–606 (2012).
[14] T. Mizuki and H. Sone. Six-Card Secure AND and Four-Card Secure XOR. In Proceedings of the 3rd International Frontiers of Algorithmics Workshop (FAW), pp. 358–369 (2009).

[15] T. Mizuki, F. Uchiike and H. Sone. Securely computing XOR with 10 cards. The Australasian Journal of Combinatorics, 36: 279–293 (2006).

[16] T. Nakai, Y. Misawa, Y. Tokushige, M. Iwamoto and K. Ohta. How to Solve Millionaires’ Problem with Two Kinds of Cards. New Generation Computing, 39(1): 73–96 (2021).

[17] V. Niemi and A. Renvall. Secure multiparty computations without computers. Theoretical Computer Science, 191: 173–183 (1998).

[18] T. Nishida, Y. Hayashi, T. Mizuki and H. Sone. Card-Based Protocols for Any Boolean Function. In Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC), pp. 110–121 (2015).

[19] T. Nishida, T. Mizuki and H. Sone. Securely Computing the Three-Input Majority Function with Eight Cards. In Proceedings of the 2nd International Conference on the Theory and Practice of Natural Computing (TPNC), pp. 193–204 (2013).

[20] L. Robert, D. Miyahara, P. Lafourcade and T. Mizuki. Physical Zero-Knowledge Proof for Suguru Puzzle. In Proceedings of the 22nd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), pp. 235–247 (2020).

[21] S. Ruangwises and T. Itoh. AND Protocols Using Only Uniform Shuffles. In Proceedings of the 14th International Computer Science Symposium in Russia (CSR), pp. 349–358 (2019).

[22] S. Ruangwises and T. Itoh. Physical Zero-Knowledge Proof for Numberlink Puzzle and $k$ Vertex-Disjoint Paths Problem. New Generation Computing, 39(1): 3–17 (2021).

[23] S. Ruangwises and T. Itoh. Physical Zero-Knowledge Proof for Ripple Effect. Theoretical Computer Science, 895: 115–123 (2021).

[24] S. Ruangwises and T. Itoh. Physical ZKP for Connected Spanning Subgraph: Applications to Bridges Puzzle and Other Problems. In Proceedings of the 19th International Conference on Unconventional Computation and Natural Computation (UCNC), pp. 149–163 (2021).

[25] S. Ruangwises and T. Itoh. Securely Computing the $n$-Variable Equality Function with $2n$ Cards. Theoretical Computer Science, 887: 99–100 (2021).

[26] K. Shinagawa and T. Mizuki. Card-based Protocols Using Triangle Cards. In Proceedings of the 9th International Conference on Fun with Algorithms (FUN), pp. 31:1–31:13 (2018).

[27] K. Shinagawa and T. Mizuki. The Six-Card Trick: Secure Computation of Three-Input Equality. In Proceedings of the 21st Annual International Conference on Information Security and Cryptology (ICISC 2018), pp. 123–131 (2019).
[28] K. Shinagawa, T. Mizuki, J.C.N. Schuld, K. Nuida, N. Kanayama, T. Nishide, G. Hanaoka and E. Okamoto. Card-Based Protocols Using Regular Polygon Cards. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100.A(9): 1900–1909 (2017).

[29] A. Stiglic. Computations with a deck of cards. *Theoretical Computer Science*, 259: 671–678 (2001).

[30] K. Takashima, Y. Abe, T. Sasaki, D. Miyahara, K. Shinagawa, T. Mizuki and H. Sone. Card-based protocols for secure ranking computations. *Theoretical Computer Science*, 845: 122–135 (2020).

[31] K. Toyoda, D. Miyahara and T. Mizuki. Another Use of the Five-Card Trick: Card-Minimal Secure Three-Input Majority Function Evaluation. In *Proceedings of the 22nd International Conference on Cryptology in India (INDOCRYPT)*, pp. 536–555 (2021).

[32] K. Toyoda, D. Miyahara, T. Mizuki and H. Sone. Six-Card Finite-Runtime XOR Protocol with Only Random Cut. In *Proceedings of the 7th ACM Workshop on ASIA Public-Key Cryptography (APKC)*, pp. 2–8 (2020).

[33] I. Ueda, D. Miyahara, A. Nishimura, Y. Hayashi, T. Mizuki and H. Sone. Secure implementations of a random bisection cut. *International Journal of Information Security*, 19(4): 445–452 (2020).