Efficient Private PEZ Protocols for Symmetric Functions*

Yoshiki Abe, Mitsugu Iwamoto, and Kazuo Ohta

The University of Electro-Communications,
1–5–1 Chofugaoka, Chofushi, 182–8585 Tokyo, Japan
{yoshiki,mitsugu,kazuo.ohta}@uec.ac.jp

Abstract. A private PEZ protocol is a variant of secure multi-party computation performed using a (long) PEZ dispenser. The original paper by Balogh et al. presented a private PEZ protocol for computing an arbitrary function with n inputs. This result is interesting, but no follow-up work has been presented since then, to the best of our knowledge. We show herein that it is possible to shorten the *initial string* (the sequence of candies filled in a PEZ dispenser) and the number of *moves* (a player pops out a specified number of candies in each move) drastically if the function is *symmetric*. Concretely, it turns out that the length of the initial string is reduced from $\mathcal{O}(2^n!)$ for general functions in Balogh et al.'s results to $\mathcal{O}(n \cdot n!)$ for symmetric functions, and 2^n moves for general functions are reduced to n^2 moves for symmetric functions. Our main idea is to utilize the recursive structure of symmetric functions to construct the protocol recursively. This idea originates from a *new* initial string we found for a private PEZ protocol for the three-input majority function, which is different from the one with the same length given by Balogh et al. without describing how they derived it.

Keywords: Private PEZ protocol · Multi-party computation · Symmetric functions · Threshold functions.

1 Introduction

1.1 Background and Motivation

A private PEZ protocol is a type of implementation of secure multi-party computation (MPC, [5]) that employs a (long) PEZ dispenser.[1] The private PEZ protocol is interesting not only because MPC can be implemented by physical tools[2] such as a PEZ dispenser but also because the protocol does not require

 This work was supported by JSPS KAKENHI Grant Numbers JP17H01752, JP18K19780, JP18K11293, JP18H05289, and JP18H03238.

1 An ordinary PEZ dispenser can store 12 candies.

2 The other examples are card-based protocols [2, 3].

Table 1. Comparison of the lengths of initial strings

n	2	3	4	5	6	7
Balogh et al. [1]	7	72	6941	$6.3 \cdot 10^7$	$5.3 \cdot 10^{15}$	$3.8 \cdot 10^{31}$
Recursive construction (Sect. 4)	7	31	165	$1,031$	$7,243$	$60,621$
Efficient $\mathsf{maj}_n^{\lceil n/2 \rceil}$ (Sect. 5)	3	13	21	131	223	$1,821$

randomness for executing MPC.[3] The original paper by Balogh et al. [1] presented a model of the PEZ protocol for computing a function f_n with n inputs without privacy, and they extended it to a model of the private PEZ protocol. The paper also proposed a method for constructing private PEZ protocols for a general function f_n.

There are two major efficiency measures of a private PEZ protocol: the length of the *initial string* and the number of *moves*. An initial string is a sequence of candies filled in a PEZ dispenser at the beginning of the protocol. A move refers to the execution step in which each player reads the candies according to the protocol. The shorter the initial string and the smaller the number of moves, the better. Unfortunately, Balogh et al.'s protocol is very inefficient in terms of both measures although the proposed protocol can compute an *arbitrary* function f_n.

The length of an initial string for computing f_n presented in [1] is $\mathcal{O}(2^n!)$, which does not depend on f_n itself but depends only on n. The numbers of candies for specific $n \leq 7$ are provided in Table 1. For instance, for $n = 7$, almost 3.8×10^{31} candies are required for the initial string, which is far from practical. The other efficiency measure is also impractical because the number of moves in [1] is $2^n - 1$.

Although the initial strings are *very long* for computing the general function f_n, Balogh et al.'s paper also presented a private PEZ protocol with a *very short* initial string for the majority function with three inputs, which is denoted by maj_3^2 in this paper. Surprisingly, only 13 candies are shown to be sufficient for the initial string in this protocol, whereas 72 candies are required for an arbitrary f_3. Unfortunately, nothing was mentioned about why and how the authors obtained this protocol, and no follow-up work on private PEZ protocols has been presented after Balogh et al.'s original paper.

1.2 Our Contributions

Efficient protocols for symmetric functions (Section 4): Our motivation is to propose a more efficient private PEZ protocol. A shorter length for the initial string and a smaller number of moves are desirable, but it is not easy to realize these for a general function f_n. We instead succeeded in making the length of the initial strings shorter and the number of moves smaller by restricting the class of functions to be computed to *symmetric* functions.

The impact of the restriction is so great that the length of the initial string is reduced from $\mathcal{O}(2^n!)$ for general functions in [1] to $\mathcal{O}(n \cdot n!)$ for symmetric

[3] Several card-based protocols, e.g., [4], do not require any randomness, either.

functions. For instance, the case where $n = 7$ in Table 1 shows us that the length of the initial string is almost 3.8×10^{31} for a general function, but it is reduced to only $60,621$ for symmetric function. Furthermore, $2^n - 1$ moves in [1] for general functions are considerably reduced to n^2 moves for symmetric functions.

Why are 13 candies sufficient for maj_3^2? (Sections 3 and 5): Our main idea for constructing a private PEZ protocol for a symmetric function f_n is to utilize the recursive structure of f_n to construct the protocol recursively. This idea is suggested by observation of the *new* initial string with length 13 for computing maj_3^2, which is different from the initial string with the same length presented in [1]. We will explain how we obtained such a short initial string in Section 3 as a preliminary step before proposing a general protocol for symmetric functions.

As is explained in Section 4, observation of our new initial string suggests how private PEZ protocols with much shorter initial strings for arbitrary symmetric functions can be constructed. Furthermore, our new initial string also suggests that the initial string of majority functions can be further shortened, which will be explained in Section 5. The difference between the constructions of initial strings for symmetric functions in Section 4 and for majority functions in Section 5 is that instead of constructing the initial string completely recursively, we use the initial string for and/or functions in the middle of the recursions because these functions can be implemented by very short initial strings. As seen from Table 1, we can further shorten the initial strings compared to the case of recursive construction proposed in Section 4. For $n = 7$, only $1,821$ candies are sufficient: i.e., our new construction requires initial strings with a length of only $4.8 \times 10^{-27}\%$ of Balogh et al.'s result!

1.3 Organization of the Paper

The remaining part of this paper is organized as follows: In Section 2, the notations and models of PEZ protocols with/without privacy [1] are provided. Section 3 is devoted to finding initial strings of private PEZ protocols for computing maj_3^2 and maj_4^2, which suggests how private PEZ protocols can be constructed recursively. Then, we propose the recursive construction of a private PEZ protocol for computing arbitrary symmetric functions in Section 4. Section 5 revisits a private PEZ protocol for computing majority functions. We show how to further shorten the (short) initial strings presented in Section 4. Section 6 concludes this paper. Technical lemma and proofs are provided in Appendix A.

2 Preliminaries

2.1 Notations

- For integers a and b such that $a \leq b$, $[a : b] := \{a, a+1, \ldots, b\}$.
- For a bit $b \in \{0, 1\}$, define $\bar{b} := 1 - b$.

Table 2. A PEZ protocol for a function f_n

Move	Player to move	# of symbols to be read		
		$x_{i_j} = \sigma_1$	$\cdots$	$x_{i_j} = \sigma_k$
M_1	P_{i_1}	$\mu_1(\sigma_1)$	$\cdots$	$\mu_1(\sigma_k)$
M_2	P_{i_2}	$\mu_2(\sigma_1)$	$\cdots$	$\mu_2(\sigma_k)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
M_m	P_{i_m}	$\mu_m(\sigma_1)$	$\cdots$	$\mu_m(\sigma_k)$

- For two binary strings $a, b \in \{0,1\}^*$, $|a|$ denotes the length of a, and $a \prec b$ means that the string a is a prefix of the string b.
- λ is the empty string. Note that $|\lambda| = 0$.
- For two binary strings $a, b \in \{0,1\}^*$, $a \circ b$ is the concatenation of a and b. Concatenations of n identical strings of a are expressed as $[a]^n := \underbrace{a \circ \cdots \circ a}_{n \text{ times}}$.
- For a binary string a, $\mathsf{hw}(a)$ is the Hamming weight of a.
- For a binary string a of length n, $\mathsf{and}_n(a)$ and $\mathsf{or}_n(a)$ are the results of the conjunction and disjunction of all the elements of a, respectively.
- For two sets X and Y, the difference set is defined as $X \setminus Y := X \cap Y^c$, where Y^c is the complement set of Y.

2.2 PEZ Protocols

Suppose that there are $n(\geq 2)$ semi-honest players $P_1, P_2, \ldots, P_n$. Each player P_i has an input $x_i \in \Sigma$, and the players wish to compute a function $f_n : \Sigma^n \to \Gamma$ while hiding their inputs from each other. The PEZ protocol for computing f_n consists of the following three steps:

Initialization Prepare a public fixed string, called an *initial string*, $\alpha \in \Gamma^*$ depending on only f_n.

Execution Follow a sequence of *moves* $M_1, M_2, \ldots, M_m$: at each move M_j, player P_{i_j} reads the next $\mu_j(x_{i_j})$ symbols of α privately, where $\mu_j(\sigma)$ indicates the number of symbols read at the j-th move with input $\sigma \in \Sigma$. The sequence $i_1, i_2, \ldots, i_m$ is called the *move order*.

Output Read the first symbol of the unread string in α.

By defining in advance the one-to-one correspondence between the colors of candies and symbols in α, the PEZ protocol can be interpreted as follows: Initialization consists of filling a sequence of candies represented by α into a PEZ dispenser. Execution consists of popping out $\mu_j(x_{i_j})$ candies from the dispenser privately. Finally, we output the topmost candy left in the PEZ dispenser, which indicates the result of $f_n(x_1, x_2, \ldots, x_n)$.

We define a PEZ protocol with an initial string α that computes a function $f_n : \Sigma^n \to \Gamma$, where n is the number of players of this protocol. When we denote an initial sequence by $\alpha(f_n)$, it means an initial sequence of the PEZ protocol for computing the function f_n.

Definition 1 (PEZ protocol [1]) Let $\alpha \in \Gamma^*$ be an initial string and $f_n : \Sigma^n \to \Gamma$ be a function to be computed. A PEZ protocol Π_{α,f_n} is defined by an initial string α and a sequence of m moves $(M_1, M_2, \dots, M_m)$. Each move M_j consists of a pair (i_j, μ_j), where i_j is a player index specifying who moves (i.e., reads symbols from α) and $\mu_j : \Sigma \to \{0, 1, \dots, |\alpha| - 1\}$ maps each input of players to the number of symbols to be read.

Table 2 indicates how many candies are popped out by the i_j-th player at the j-th move.

2.3 Private PEZ Protocols

Definition 2 (Private PEZ protocol [1]) For an initial string α and a function f_n with n inputs, a PEZ protocol Π_{α,f_n} is called *private* if there exists a mapping $\nu : \{1, 2, \dots, m\} \times \Sigma \to \Gamma^*$ that satisfies the following two conditions[4]:

1. For all $j \in \{1, 2, \dots, m\}$, and for all $\sigma \in \Sigma$, the following holds:
$$|\nu(j, \sigma)| = \mu_j(\sigma).$$
2. For any $x := (x_1, x_2, \dots, x_n) \in \Sigma^n$, the following holds:
$$\nu(1, x_{i_1}) \circ \nu(2, x_{i_2}) \circ \cdots \circ \nu(m, x_{i_m}) \circ f_n(x) \prec \alpha.$$

Intuitively, Definition 2 can be explained as follows: Condition 1 means that the number of candies read by player P_{i_j} at the j-th move is specified by j and the input x_{i_j} of P_{i_j}. Condition 2 requires that if we read the candies with the number specified by Condition 1, the output becomes $f_n(x)$, which implies *correctness*.

Condition 2 simultaneously requires that if every player reads the candies by following Condition 1, player P_{i_j} with input x_{i_j} at the j-th move must eventually read the same sequence $\nu(j, x_{i_j})$, which guarantees *privacy*. In other words, the substrings to be read by P_i, i.e., the view of P_i, in each move depends on x_i only, so that the view contains no information about the other players' inputs.

Definition 3 (Round in a private PEZ protocol) In a private PEZ protocol for computing f_n, we call a series of n moves a *round* if the n moves satisfy the following conditions. The ℓ-th round is denoted by R_ℓ, $\ell \geq 0$.

1. Every player P_i moves only once in the n moves.[5]
2. During the same round, every P_i reads the same sequence if the input of P_i is the same.

[4] In [1], β was used instead of ν, but in this paper, we use β to express a string.
[5] If f_n is symmetric, the move order in a round does not affect the output.

Table 3. A private PEZ protocol $\Pi_{\alpha,\mathsf{and}_n}$ where $\alpha \coloneqq \alpha(\mathsf{and}_n) = [0]^n \circ 1$

Round	Player to move	# of bits to be read		Substring to be read	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	0	1	–	0

Table 4. A private PEZ protocol $\Pi_{\alpha,,\mathsf{or}_n}$ where $\alpha \coloneqq \alpha(\mathsf{or}_n) = [1]^n \circ 0$

Round	Player to move	# of bits to be read		Substring to be read	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	1	0	1	–

Example 1 (Private PEZ protocols for and_n and or_n [1]) We show private PEZ protocols for computing AND and OR of n binary inputs which are denoted by and_n and or_n, respectively. The private PEZ protocols for and_n and or_n are useful for constructing efficient private PEZ protocols discussed hereafter.

A private PEZ protocol for and_n uses the $(n+1)$-bit initial string $\alpha(\mathsf{and}_n) \coloneqq [0]^n \circ 1$ and has n moves: each player P_i reads one candy of "0" from $\alpha(\mathsf{and}_n)$ if the input value of P_i is 1, otherwise each P_i does not read any candy. After round R_0, i.e., n moves by P_1, P_2, and P_n, the first remaining candy of $\alpha(\mathsf{and}_n)$ becomes "1" only when all the input values of P_i are 1, otherwise it becomes "0". This protocol $\Pi_{\alpha(\mathsf{and}_n),\mathsf{and}_n}$ is summarized in Table 3. A private PEZ protocol for or_n can be specified analogously by letting $\alpha(\mathsf{or}_n) \coloneqq [1]^n \circ 0$ and following the moves in Table 4.

The privacy of and_n is easy to see because every player reads "0" during R_0 if P_i inputs 1, and hence, no information leaks until the output phase. The privacy of or_n is also easy to understand.

Example 2 (Private PEZ protocol for maj_3^2) Consider the case of a majority function with three inputs denoted by maj_3^2, which outputs 1 if two or more inputs are 1. The initial string for maj_3^2 is given by $\alpha(\mathsf{maj}_3^2) = 0010010010001$, and the protocol works as shown in Table 5.

In this example, correctness is easy to check. Privacy is easy to see as well because every player who inputs 1 reads "001" and "0" in rounds R_0 and R_1, respectively, and nothing is read by the player who inputs 0.

Remark. Note that a private PEZ protocol for maj_3^2 with an initial string with length 13 was presented in [1], which is different from the one in Example

Table 5. A private PEZ protocol $\Pi_{\alpha,\mathsf{maj}_3^2}$ where $\alpha \coloneqq \alpha(\mathsf{maj}_3^2) = 0010010010001$

Round	Players to move	# of bits to be read		Substring to be read	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^3$	0	3	–	001
R_1	$\{P_i\}_{i=1}^3$	0	1	–	0

2. Unfortunately, however, no description was given in [1] regarding why and how the protocol was derived. On the other hand, in this paper, we will explain how we derived the protocol in Example 2, which is insightful for constructing a private PEZ protocol for symmetric functions and its improvement for maj_n^t that are proposed in Sections 4 and 5, respectively.

2.4 Symmetric Functions

Definition 4 (Symmetric functions) A function $f_n : \{0,1\}^n \to \{0,1\}$ is called *symmetric* if

$$f_n(x_1, x_2, \ldots, x_n) = f_n(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}) \tag{1}$$

holds for all $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ and an arbitrary permutation $\sigma : [n] \to [n]$.

Since the symmetric function does not depend on the order of $x_1, x_2, \ldots, x_n$, we will sometimes regard f_n as a function taking a multiset as an input. For instance, a symmetric function $f_2(x_1, x_2) = f_2(x_2, x_1)$ is also written as $f_2^{\mathsf{m}}(\{x_1, x_2\}) := f_2(x_1, x_2) = f_2(x_2, x_1)$. Furthermore, if f_n takes n binary inputs, f_n depends only on the Hamming weight of the n binary inputs. Summarizing, we use the following equivalent expressions for symmetric functions.

Definition 5 (Equivalent expressions for symmetric functions) For a symmetric function $f_n : \{0,1\}^n \to \{0,1\}$, define $f_n^{\mathsf{m}} : \{\{x_1, x_2, \ldots, x_n\} \mid x_i \in \{0,1\}\} \to \{0,1\}$ and $f_n^{\mathsf{w}} : [0:n] \to \{0,1\}$ as

$$f_n^{\mathsf{m}}(\{x_1, x_2, \ldots, x_n\}) := f_n(x_1, x_2, \ldots, x_n), \tag{2}$$

$$f_n^{\mathsf{w}}(w) := f_n(x_1, x_2, \ldots, x_n), \tag{3}$$

where $\{x_1, x_2, \ldots, x_n\}$ is a multiset and $x_1, x_2, \ldots, x_n$ are the binary inputs satisfying $w = \mathsf{hw}(x_1, x_2, \ldots, x_n)$.

Hereafter, we choose an appropriate expression for a symmetric function from (1)–(3) depending on the context. The superscripts m and w will be omitted if they are clear from the context.

3 Warm-Up: Private PEZ Protocols for Majority Voting

Before presenting our construction of private PEZ protocols for general symmetric functions, we show private PEZ protocols for n-input majority voting ($n \geq 3$).

Definition 6 (Majority function with threshold) Let n and t be positive integers with $n \geq t$. For $x_1, x_2, \ldots, x_n \in \{0,1\}$, define a majority function with threshold t by

$$\mathsf{maj}_n^t(x_1, x_2, \ldots, x_n) := \begin{cases} 1, & \text{if } \sum_{i=1}^n x_i \geq t \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Table 6. Truth table of maj_3^2 classified by x_3

x_1	x_2	$x_3 = 0$	$x_3 = 1$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

For $t = \lceil n/2 \rceil$, maj_n^t reduces to the ordinary majority voting.

In this section, for intuitive understanding, we construct private PEZ protocols from the perspective of each player's view and do not explicitly prove that proposed protocols satisfy Definition 2. Later, we directly prove that the proposed private PEZ protocols for symmetric functions satisfies Definition 2.

Example 3 (A private PEZ protocol for three-input majority voting) Table 6 is the truth table of maj_3^2 classified by the value of $x_3 \in \{0,1\}$. Observing the input values of x_1 and x_2, and the output values for $x_3 = 0$ in Table 6, we can see that $\mathsf{maj}_3^2(\{x_1, x_2, 0\})$ can be regarded as $\mathsf{and}_2(\{x_1, x_2\})$: i.e.,

$$\mathsf{maj}_3^2(\{x_1, x_2, 0\}) = \mathsf{maj}_2^2(\{x_1, x_2\}) = \mathsf{and}_2(\{x_1, x_2\}). \tag{5}$$

We can also find

$$\mathsf{maj}_3^2(\{x_1, 0, x_3\}) = \mathsf{maj}_2^2(\{x_1, x_3\}) = \mathsf{and}_2(\{x_1, x_3\}), \tag{6}$$

$$\mathsf{maj}_3^2(\{0, x_2, x_3\}) = \mathsf{maj}_2^2(\{x_2, x_3\}) = \mathsf{and}_2(\{x_2, x_3\}). \tag{7}$$

(5)–(7) imply that if there exists at least one input with value 0, maj_3^2 can be computed by and_2 with two inputs obtained by eliminating 0 from three inputs of maj_3^2. Only when $x_1 = x_2 = x_3 = 1$, maj_3^2 is not representable by and_2. Therefore, maj_3^2 can be represented as follows:

$$\begin{aligned} &\mathsf{maj}_3^2(\{x_1, x_2, x_3\}) \\ &= \begin{cases} \mathsf{maj}_3^2(\{x_1, x_2, x_3\}), & \text{if } x_1 = x_2 = x_3 = 1, \\ \mathsf{maj}_2^2(\{x_1, x_2, x_3\} \setminus \{0\}) = \mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\}), & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

Equivalently, for $w \in [0:3]$,

$$\mathsf{maj}_3^2(w) = \begin{cases} \mathsf{maj}_3^2(w), & \text{if } w = 3, \\ \mathsf{maj}_2^2(w - 0) = \mathsf{and}_2(w - 0), & \text{otherwise.}^6 \end{cases} \tag{9}$$

We construct a private PEZ protocol for maj_3^2 based on (8). Let $\alpha(\mathsf{maj}_3^2)$ be an initial string for maj_3^2.

Let β_0 and β_1 be strings used for computing $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ and $\mathsf{maj}_3^2(\{x_1, x_2, x_3\})$, in (8), respectively. The actual sequence of β_0 and β_1 are

[6] "-0" means that the weight does not change, which is used to aid in understanding.

undetermined so far, but will be specified below. Note here that the cases in (8) are classified by the results of and_3. From Example 1, the initial string for computing and_3 is given by $\alpha(\mathsf{and}_3) = 0001$, and we replace 0 and 1 in $\alpha(\mathsf{and}_3)$ with β_0 and β_1, respectively. Then we obtain

$$\alpha(\mathsf{maj}_3^2) = \beta_0 \circ \beta_0 \circ \beta_0 \circ \beta_1 \tag{10}$$

as the initial string for computing maj_3^2. The reason of these replacements will be explained later more clearly.

Using the initial string $\alpha(\mathsf{maj}_3^2)$ in (10), the private PEZ protocol for computing maj_3^2 can be described as follows: In round R_0, each player P_i who inputs $x_i = 1$ reads $|\beta_0|$ bits: otherwise ($x_i = 0$), P_i does not read any bit. The following shows the remaining string α' after round R_0.

$$\alpha' \succ \begin{cases} \beta_1, & \text{if } x_1 = x_2 = x_3 = 1 \\ \beta_0, & \text{otherwise.} \end{cases} \tag{11}$$

After R_0, several moves are added to compute maj_3^2 using β_0 or β_1, which will specify β_0 and β_1.

The correctness ((11) holds) and privacy (no player obtains information about other players' inputs) in R_0 are guaranteed by the correctness and privacy of the private PEZ protocol for and_3, which is the reason why we replaced 0 and 1 in $\alpha(\mathsf{and}_3)$ with β_0 and β_1, respectively, to obtain (10). Therefore, to construct the private PEZ protocol for maj_3^2, the additional moves have to satisfy the following requirements:

Correctness of the additional moves

- **C-1** Compute $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ using $\beta_0 (\prec \alpha')$.
- **C-2** Compute $\mathsf{maj}_3^2(\{1, 1, 1\})$ using $\beta_1 (\prec \alpha')$.

Privacy of the additional moves

- **P-1** Computation of $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ is private.
- **P-2** Computation of $\mathsf{maj}_3^2(\{1, 1, 1\})$ is private.
- **P-3** Each string read by each player for $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ is the same as the one for $\mathsf{maj}_3^2(\{1, 1, 1\})$.

First, we discuss **C-1** and **P-1** to specify β_0 and the additional moves. Since $0 \in \{x_1, x_2, x_3\}$, $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ can compute in the same way as the private PEZ protocol for and_2. That is, using the string $\beta_0 := 001$, each player P_i, $1 \le i \le 3$, reads one bit represented by "0" if $x_i = 1$, otherwise P_i does not read any bit. Note that there exists a bit in β_0 to be output when three players execute the move in the private PEZ protocol for and_2 with $\alpha(\mathsf{and}_2) = 001$, because there exists at least one player P_j whose input $x_j = 0$ and does not read any bit. Therefore, $\mathsf{and}_2(\{x_1, x_2, x_3\} \setminus \{0\})$ can be computed using $\beta_0 = 001$. In addition, no player can obtain information about other players' inputs because one bit read by a player is always "0" regardless of other players' inputs. Hence, in summary, to satisfy **C-1** and **P-1**, we should use $\beta_0 = 001$ and add three

Table 7. Truth tables of maj_4^2 and maj_4^3 classified by x_4

(a) maj_4^2

x_1 x_2 x_3	$x_4 = 0$	$x_4 = 1$
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
1 0 0	0	1
1 1 0	1	1
1 0 1	1	1
1 0 1	1	1
1 1 1	1	1

(b) maj_4^3

x_1 x_2 x_3	$x_4 = 0$	$x_4 = 1$
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
1 0 0	0	1
1 1 0	0	1
1 0 1	0	1
1 0 1	0	1
1 1 1	1	1

Table 8. A private PEZ protocol $\Pi_{\alpha', \mathsf{maj}_4^2}$ where $\alpha' := \alpha(\mathsf{maj}_4^2) =$ 1110111011101110111110

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^4$	4	0	1110	–
R_1	$\{P_i\}_{i=1}^4$	1	0	1	–

moves as round R_1; each player P_i, $1 \le i \le 3$, reads one bit from β_0 if $x_i = 1$, which is always "0", otherwise the player does nothing.

Next, we discuss **C-2**, **P-2**, and **P-3**, to specify β_1. To satisfy **P-3**, we must follow the moves in round R_1 in the same manner as when $\{x_1, x_2, x_3\}$ contains at least one 0. To be specific, every player P_i, $1 \le i \le 3$, reads one bit "0" in round R_1, which determines the prefix of β_1 as 000, i.e., $000 \prec \beta_1$. These three moves also satisfy **P-2**. Since the remaining one bit is read as output after round R_1, the next bit of 000 in β_1 is determined as $1 (= \mathsf{maj}_3^2(\{1,1,1\}))$ to satisfy **C-2**. As a result, we obtain $\beta_1 = 0001$.

Summarizing the above discussion, a private PEZ protocol for maj_3^2 is shown in Table 5, which can be implemented with the 13-bit initial string $\alpha = \beta_0 \circ \beta_0 \circ \beta_0 \circ \beta_1 = 0010010010001$ and six moves.

Throughout Example 3, the input of a function is written as a multiset such as $\mathsf{maj}_3^2(\{x_1, x_2, x_3\})$. Hereafter, an input of a symmetric function is represented by the Hamming weight of the input such as $\mathsf{maj}_3^2(3)$ $(= \mathsf{maj}_3^2(\{1,1,1\}))$. This is because the Hamming weight of the input is sufficient information to compute the symmetric function.

Example 4 (Private PEZ protocols for four-input majority voting) In Table 7, (a) and (b) are the truth tables of maj_4^2 and maj_4^3, respectively. Note that the truth values in these tables are classified by the value of $x_4 \in \{0,1\}$. In the following, we mainly explain the construction of a private PEZ protocol for maj_4^2 in the same way as for maj_3^2 in the Example 3, but a protocol for maj_4^3 is obtained analogously.

Let $w \in [0:4]$ be the Hamming weight of the four inputs of maj_4^2.

When $w \neq 0$, i.e., when there exists at least one input whose value is 1, the outputs of $\mathsf{maj}_4^2(w)$ is equal to the outputs of or_3 with input $w-1$, i.e., three inputs obtained by eliminating 1 from four inputs of maj_4^2. Actuary, the right column of the outputs in Table 7(a) shows the case when $x_4 = 1$ is eliminated, and it can be regarded as $\mathsf{maj}_3^1(w-1) = \mathsf{or}_3(w-1)$.

Together with the case where $w = 0$, the following holds:

$$\mathsf{maj}_4^2(w) = \begin{cases} \mathsf{maj}_4^2(0), & \text{if } w = 0 \\ \mathsf{maj}_3^1(w-1) = \mathsf{or}_3(w-1), & \text{otherwise } (w \neq 0). \end{cases} \tag{12}$$

We can construct a private PEZ protocol for maj_4^2 based on (12) similar to how the private PEZ protocol for maj_3^2 was constructed in Example 3. Let $\tilde{\beta_0}$ and $\tilde{\beta_1}$ be the undetermined strings which are used for computing $\mathsf{maj}_4^2(0)$ and $\mathsf{or}_3(w-1)$, respectively.

Let $\alpha(\mathsf{maj}_4^2)$ be an initial string for computing maj_4^2. In the same way as Example 3, set $\alpha(\mathsf{maj}_4^2) := [\tilde{\beta_1}]^4 \circ \tilde{\beta_0}$, which is obtained by replacing "0" and "1" in $\alpha(\mathsf{or}_4) = 1110$ with $\tilde{\beta_0}$ and $\tilde{\beta_1}$, respectively. In round R_0, each player P_i, $1 \leq i \leq 4$, reads $|\tilde{\beta_1}|$ bits if the input $x_i = 0$, otherwise the player does not read any bit. Then, at the end of round R_0, the remaining string $\tilde{\alpha}'$ satisfies

$$\tilde{\alpha}' \succ \begin{cases} \tilde{\beta_0}, & \text{if } w = 0 \\ \tilde{\beta_1}, & \text{otherwise.} \end{cases} \tag{13}$$

In round R_1, we will add four moves for computing $\mathsf{maj}_4^2(0)$ and $\mathsf{or}_3(w-1)$ by using $\tilde{\beta_0}$ and $\tilde{\beta_1}$, respectively.

The function $\mathsf{or}_3(w-1)$ can be computed in the same way as the private PEZ protocol for or_3. That is, by using a string $\tilde{\beta_1} := 1110$ in round R_1, each player P_i, $1 \leq i \leq 4$, reads one bit represented by "1" if $x_i = 0$, otherwise the player does not read any bit. Then, every move is specified as shown in Table 8.

Now we are prepared to compute $\mathsf{maj}_4^2(0)$ privately; the first four bits of $\tilde{\beta_0}$ have to be 1111, which is the string read in round R_1. Since the next bit of 1111 becomes the output, $\tilde{\beta_0}$ is obtained by appending $0 (= \mathsf{maj}_4^2(0))$ to the rightmost part of 1111. Therefore, we obtain $\tilde{\beta_0} := 11110$. In summary, Table 8 shows a private PEZ protocol for maj_4^2, which uses the 21-bit initial string $\alpha \coloneqq \alpha(\mathsf{maj}_4^2) = 111011101110111011110$, and has eight moves.

The private PEZ protocol for computing maj_4^3 can be derived in the same manner starting from the truth table (b) in Table 7 and based on

$$\mathsf{maj}_4^3(w) = \begin{cases} \mathsf{maj}_4^3(4), & \text{if } w = 4 \\ \mathsf{maj}_3^3(w) = \mathsf{and}_3(w), & \text{otherwise } (w \neq 4). \end{cases} \tag{14}$$

and the private PEZ protocol for and_4. The protocol is shown in Table 9, with eight moves and the initial string $\alpha \coloneqq \alpha(\mathsf{maj}_4^3) = 000100010001000100001$.

Table 9. A private PEZ protocol $\Pi_{\alpha,\mathsf{maj}_4^3}$ where $\alpha := \alpha(\mathsf{maj}_4^3) =$ 0001000100010001000001

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^4$	0	4	–	0001
R_1	$\{P_i\}_{i=1}^4$	0	1	–	0

4 A Private PEZ Protocol for Symmetric Functions

4.1 Recursive Structure of Symmetric Functions

We generalize the discussion in Section 3 for a general symmetric function f_n with n inputs. First, we generalize the relations (12) and (14) as follows.

Theorem 1 Let $f_n : \{0,1\}^n \to \{0,1\}$ be an arbitrary symmetric function with n binary inputs. Then, we recursively define the symmetric functions $g_k : \{0,1\}^k \to \{0,1\}$ and $h_k : \{0,1\}^k \to \{0,1\}$, for $1 \le k \le n$, by $g_n := f_n, h_n := f_n$, and

$$g^{\mathsf{m}}_{k-1}(\{x_1, x_2, \ldots, x_{k-1}\}) := g^{\mathsf{m}}_k(\{x_1, x_2, \ldots, x_k\} \setminus \{0\}), \quad \text{if } \{x_1, x_2, \ldots, x_k\} \text{ contains at least one } 0, \tag{15}$$

$$h^{\mathsf{m}}_{k-1}(\{x_1, x_2, \ldots, x_{k-1}\}) := h^{\mathsf{m}}_k(\{x_1, x_2, \ldots, x_k\} \setminus \{1\}). \quad \text{if } \{x_1, x_2, \ldots, x_k\} \text{ contains at least one } 1. \tag{16}$$

Then, the following holds for $w \in [0:k]$:

$$g^{\mathsf{w}}_k(w) = \begin{cases} g^{\mathsf{w}}_k(k) = f^{\mathsf{w}}_n(k) & \text{if } w = k \\ g^{\mathsf{w}}_{k-1}(w) & \text{otherwise,} \end{cases} \tag{17}$$

$$h^{\mathsf{w}}_k(w) = \begin{cases} h^{\mathsf{w}}_k(0) = f^{\mathsf{w}}_n(n-k) & \text{if } w = 0 \\ h^{\mathsf{w}}_{k-1}(w-1) & \text{otherwise,} \end{cases} \tag{18}$$

where $g^{\mathsf{w}}_0(0) := f^{\mathsf{w}}_n(0)$ and $h^{\mathsf{w}}_0(0) := f^{\mathsf{w}}_n(n)$.

The proof is provided in Appendix A.2.

Example 5 We revisit the case of Example 4. Let $f_4 = h_4 = \mathsf{maj}_4^2$. Then, it is easy to see that $h_3 = \mathsf{or}_3$ from Table 7(a). On the other hand, for $f_4 = g_4 = \mathsf{maj}_4^3$, we can choose $g_3 = \mathsf{and}_3$ from Table 7(b). This is the reason why (8), (9), (12), and (14) hold.

4.2 Proposed Construction for General Symmetric Functions

We propose the construction of a private PEZ protocol for computing a symmetric function f_n. Let $\alpha(f_n)$ be an initial sequence of the private PEZ protocol for computing the function f_n. There are two ways of constructing $\alpha(f_n)$, as shown below.

Table 10. A private PEZ protocol for f_n using $\alpha(g_n)$ as the initial string

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(g_{n-1})\rvert$	–	$\alpha(g_{n-1})$
R_1	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(g_{n-2})\rvert$	–	$\alpha(g_{n-2})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
R_{n-1}	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(g_0)\rvert$	–	$\alpha(g_0)$

Construction 1 Assume that a symmetric function f_n is recursively decomposed into two cases by either (17) or (18). For $1 \le k \le n$, $\alpha(g_k)$ and $\alpha(h_k)$ are the initial strings for computing g_k and h_k, respectively. From (17) and (18), $\alpha(g_k)$ and $\alpha(h_k)$ can be recursively constructed as follows:

$$\alpha(g_k) := [\alpha(g_{k-1})]^k \circ [\alpha(g_{k-2})]^k \circ \cdots \circ [\alpha(g_0)]^k \circ g_k^{\mathsf{w}}(k), \tag{19}$$

$$\alpha(h_k) := [\alpha(h_{k-1})]^k \circ [\alpha(h_{k-2})]^k \circ \cdots \circ [\alpha(h_0)]^k \circ h_k^{\mathsf{w}}(0), \tag{20}$$

where $\alpha(g_0) := g_0 = f_n^{\mathsf{w}}(0)$ and $g_k^{\mathsf{w}}(k) = f_n^{\mathsf{w}}(k)$ in (19), and $\alpha(h_0) := h_0 = f_n^{\mathsf{w}}(n)$ and $h_k^{\mathsf{w}}(0) = f_n^{\mathsf{w}}(n-k)$ in (20). Finally, we obtain two types of the initial string $\alpha(g_n)$ and $\alpha(h_n)$ recursively from (19) and (20). Then, we have

$$\alpha(f_n) := \alpha(g_n), \tag{21}$$

$$\alpha(f_n) := \alpha(h_n). \tag{22}$$

Note that the sequences of $\alpha(f_n)$ obtained from (21) and (22) are not in general the same.

First, we describe the private PEZ protocol for f_n using $\alpha(f_n)$ obtained from (21) as the initial string. In this protocol, the sequence of n^2 moves $(M_1, M_2, \ldots, M_{n^2})$ for computing f_n is determined as follows: Each move M_j consists of $((j \bmod n) + 1, \mu_j)$ where $\mu_j : \{0,1\} \to \{0, |\alpha(g_0)|, |\alpha(g_1)|, \cdots, |\alpha(g_{n-1})|\}$ and $\mu_j(0) = 0, \mu_j(1) = |\alpha(g_{n-\lceil j/n \rceil})|$. These moves can be represented as n rounds $(R_0, R_1, \ldots, R_{n-1})$, and each player P_i reads $\mu_{rn+i}(x_i)$ bits in the r-th round. These n rounds are shown in Table 10.

Second, the private PEZ protocol for f_n using $\alpha(f_n)$ obtained from (22) as the initial string is similar to the protocol for $\alpha(f_n)$ obtained from (21) and is shown in Table 11.

Theorem 2 The private PEZ protocol obtained from Construction 1 satisfies Definition 2.

The proof is provided in Appendix A.3.

4.3 Evaluation of the Length of Initial Strings

For a symmetric function f_n, let $a_n := |\alpha(f_n)|$ for simplicity.

Table 11. A private PEZ protocol for f_n using $\alpha(h_n)$ as the initial string

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	$\lvert\alpha(h_{n-1})\rvert$	0	$\alpha(h_{n-1})$	–
R_1	$\{P_i\}_{i=1}^n$	$\lvert\alpha(h_{n-2})\rvert$	0	$\alpha(h_{n-2})$	–
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
R_{n-1}	$\{P_i\}_{i=1}^n$	$\lvert\alpha(h_0)\rvert$	0	$\alpha(h_0)$	–

Theorem 3 The length of the initial string of a private PEZ protocol for computing a symmetric function f_n is computed as

$$a_n = n \cdot n! \sum_{i=1}^{n} \frac{1}{i!} + 1, \tag{23}$$

from which we can conclude that $a_n = \mathcal{O}(n \times n!)$.

The proof is provided in Appendix A.4.

5 A More Efficient Private PEZ Protocol for the Majority Function maj_n^t

By restricting the functions to be computed to maj_n^t, the length of the initial string α becomes much shorter than that for computing symmetric functions.

Consider the case $t = 1$ and $t = n$ for maj_n^t. Since maj_n^1 and maj_n^n are equivalent to or_n and and_n, respectively, they can be computed with only $(n+1)$-bit strings $[1]^n \circ 0$ and $[0]^n \circ 1$ as shown in Example 1. Therefore, $|\alpha(\mathsf{and}_n)| = |\alpha(\mathsf{or}_n)| = \mathcal{O}(n)$ while $|\alpha(f_n)| = \mathcal{O}(n \times n!)$ as shown in Theorem 3 for a symmetric function f_n.

In Examples 3 and 4, $\alpha(\mathsf{maj}_3^2) = 13$ and $\alpha(\mathsf{maj}_4^2) = \alpha(\mathsf{maj}_4^3) = 21$, whereas $|\alpha(f_3)| = 31$ and $|\alpha(f_4)| = 165$ in Construction 1. The reason for this difference is the number of times decomposition is performed by either (15) or (16). In Construction 1, $\alpha(f_3)$, and $\alpha(f_4)$ are obtained by decomposing two times and three times, respectively, to the end. On the other hand, in Examples 3 and 4, $\alpha(\mathsf{maj}_3^2)$, $\alpha(\mathsf{maj}_4^2)$ and $\alpha(\mathsf{maj}_4^3)$ are obtained by decomposing only once. After the one-time decomposition by (15) or (16), the functions $\alpha(\mathsf{maj}_3^2)$, $\alpha(\mathsf{maj}_4^2)$ and $\alpha(\mathsf{maj}_4^3)$ can be computed using and_2, or_3, and and_3, respectively.

In general, we can construct private PEZ protocols for majority functions $\alpha(\mathsf{maj}_n^t)$ using the initial strings for $\{\mathsf{and}_i\}_{i=2}^n$ or $\{\mathsf{or}_n\}_{i=2}^n$. Therefore, the initial string becomes shorter by reducing the number of decompositions using a private PEZ protocol for and_n or or_n rather than by recursively decomposing to the end.

Let s be the number of times maj_n^t is decomposed by (15) where $0 \le s \le n-1$. From observation of (15), we can learn that if maj_n^t is decomposed by (15), maj_n^t becomes maj_{n-1}^t. Therefore, after s decompositions by (15), maj_n^t becomes

maj^t_{n-s}. In addition, if $n-s=t$, i.e., $s=n-t$, then $\mathsf{maj}^t_{n-s}=\mathsf{maj}^{n-s}_{n-s}$ is identical to and_{n-s}. Similarly, when (16) is used for s decompositions, maj^t_n becomes maj^{t-s}_{n-s} and if $t-s=1$, i.e., $s=t-1$, then $\mathsf{maj}^t_{n-s}=\mathsf{maj}^1_{n-s}$ is identical to or_{n-s}.

To reduce the number of decompositions, s should be as small as possible. Thus, if $n-t\leq t-1$, i.e., $t\geq(n+1)/2$, maj^t_n should be decomposed by (15). On the other hand, if $n-t\geq t-1$, i.e., $t\leq(n+1)/2$, maj^t_n should be decomposed by (16).

Construction 2 Assume that maj^t_n is decomposed s times by either (15) or (16), where

$$s=\begin{cases}n-t & \text{if } t\geq(n+1)/2,\\ t-1 & \text{if } t\leq(n+1)/2,\end{cases}$$

and we define $\alpha(\mathsf{maj}^t_k)$, where $n-s\leq k\leq n$, as follows:

$$\alpha(\mathsf{maj}^t_k):=\begin{cases}[\alpha(\mathsf{maj}^t_{k-2})]^k\circ[\alpha(\mathsf{maj}^t_{k-2})]^k\circ\cdots\circ[\alpha(\mathsf{maj}^t_{n-s})]^k\circ[\alpha(\mathsf{maj}^t_{n-s-1})]^k\circ\mathsf{maj}^t_k(k), & \text{if } t\geq(n+1)/2,\\ [\alpha(\mathsf{maj}^{t-1}_{k-1})]^k\circ[\alpha(\mathsf{maj}^{t-2}_{k-2})]^k\circ\cdots\circ[\alpha(\mathsf{maj}^1_{n-s})]^k\circ[\alpha(\mathsf{maj}^0_{n-s-1})]^k\circ\mathsf{maj}^t_k(0), & \text{if } t\leq(n+1)/2,\end{cases}\tag{24}$$

where $\alpha(\mathsf{maj}^t_{n-s-1}):=0$ and $\alpha(\mathsf{maj}^0_{n-s-1}):=1$. Then, we obtain the initial string $\alpha(\mathsf{maj}^t_n)$ by substituting n for k in (24). Note that if n is odd, we can use either equation.

If $t\leq(n+1)/2$, the sequence of $n(s+1)$ moves $(M_1,M_2,\ldots,M_{n(s+1)})$ for computing maj^t_n is determined as follows: each move M_j consists of $((j \bmod n)+1,\mu'_j)$, where $\mu'_j:\{0,1\}\to\{0,|\alpha(\mathsf{maj}^1_{n-s-1})|,|\alpha(\mathsf{maj}^2_{n-s})|,\ldots,|\alpha(\mathsf{maj}^{t-1}_{n-1})|\}$ and $\mu'_j(0)=|\alpha(\mathsf{maj}^{t-\lceil j/n\rceil}_{n-\lceil j/n\rceil})|,\mu'_j(1)=0$. These moves can be represented as $s+1$ rounds $(R_0,R_1,\ldots,R_s)$, and each player P_i reads $\mu'_{rn+i}(x_i)$ bits in the r-th round. These $s+1$ rounds are shown in Table 13. The sequences of moves for $t\geq(n+1)/2$ are similar to those for $t\leq(n+1)/2$ and are shown in Table 12.

Example 6 Consider the case of maj^2_4 in Example 4. In this case, since $t=2<5/2=(n+1)/2$, $s=t-1=1$. Therefore, we decompose maj^2_4 once by using (15). Then, we obtain $\alpha(\mathsf{maj}^2_4)=[\alpha(\mathsf{maj}^1_3)]^4\circ[1]^4\circ\mathsf{maj}^2_4(0)$ and $\alpha(\mathsf{maj}^1_3)=[1]^3\circ\mathsf{maj}^1_3(0)=1110$, which yields

$$\alpha(\mathsf{maj}^2_4)=[1110]^4\circ[1]^4\circ 0.\tag{25}$$

Therefore, this initial string $\alpha(\mathsf{maj}^2_4)$ coincides with the initial string obtained in Example 4. We can also see that the moves (rounds) of this protocol obtained by Construction 2 coincide with the rounds of the protocol for maj^2_4 in Table 8, which is obtained in Example 4.

Table 12. A private PEZ protocol for maj_n^t for $t \geq (n+1)/2$

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(\mathsf{maj}_{n-1}^t)\rvert$	–	$\alpha(\mathsf{maj}_{n-1}^t)$
R_1	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(\mathsf{maj}_{n-2}^t)\rvert$	–	$\alpha(\mathsf{maj}_{n-2}^t)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
R_s	$\{P_i\}_{i=1}^n$	0	$\lvert\alpha(\mathsf{maj}_{n-s-1}^t)\rvert$	–	$\alpha(\mathsf{maj}_{n-s-1}^t)$

Table 13. A private PEZ protocol for maj_n^t for $t \leq (n+1)/2$

Round	Players to move	# of bits to read		Substring of read bits	
		$x_i = 0$	$x_i = 1$	$x_i = 0$	$x_i = 1$
R_0	$\{P_i\}_{i=1}^n$	$\lvert\alpha(\mathsf{maj}_{n-1}^{t-1})\rvert$	0	$\alpha(\mathsf{maj}_{n-1}^{t-1})$	–
R_1	$\{P_i\}_{i=1}^n$	$\lvert\alpha(\mathsf{maj}_{n-2}^{t-2})\rvert$	0	$\alpha(\mathsf{maj}_{n-2}^{t-2})$	–
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
R_s	$\{P_i\}_{i=1}^n$	$\lvert\alpha(\mathsf{maj}_{n-s-1}^{1})\rvert$	0	$\alpha(\mathsf{maj}_{n-s-1}^{1})$	–

Theorem 4 The private PEZ protocol obtained from Construction 2 satisfies Definition 2.

The proof is omitted since it is similar to the proof of Theorem 2.

Finally, let $a_{n,t}$ be the length of the initial string of a private PEZ protocol for computing maj_n^t obtained from Construction 2. From (24), the following holds:

$$a_{n,t} = \begin{cases} na_{n-1,t} + na_{n-2,t} + s + na_{n-s,t} + a_{n-s-1,t} & \text{if } t \leq n/2 \\ na_{n-1,t-1} + na_{n-2,t-2} + s + na_{n-s,1} + a_{n-s-1,0} & \text{otherwise} \end{cases}$$
$$= \begin{cases} n\sum_{i=1}^{s} a_{n-i,t} + a_{n-s-1,t} & \text{if } t \leq n/2 \\ n\sum_{i=1}^{s} a_{n-i,t-i} + a_{n-s-1,0} & \text{otherwise} \end{cases} \tag{26}$$

where $a_{n-s-1,t} = 1$ and $a_{n-s-1,0} = 1$. Then, the theorem below immediately follows from Lemma 1 in Appendix A.1 and (24), and hence, the proof is omitted.

Theorem 5 The length of the initial string of a private PEZ protocol for computing maj_n^t is computed as

$$a_{n,t} = n \sum_{i=n-s}^{n} \frac{n!}{i!} + 1, \quad \text{where } s = \begin{cases} t-1 & \text{if } t \leq n/2 \\ n-t & \text{otherwise} \end{cases} \tag{27}$$

from which we can conclude that $a_n = \mathcal{O}(n \times n^s)$.

6 Conclusion

In the previous work [1], a general, but inefficient private PEZ protocol was presented. By restricting our attention to the symmetric functions, we achieved the exponential improvement on a private PEZ protocol for symmetric functions using the recursive structure of symmetric functions. Specifically, the double exponential length of initial string is reduced to exponential length, and the exponential number of moves is reduced to polynomial moves. Furthermore, in the case of threshold functions, the length of an initial string and the number of moves are further reduced compared with the ones for symmetric functions. These results resolve a part of open problems suggested in [1].

Finally, we mention the relationship between our construction for symmetric functions and the general construction [1]. A general function f_n with n inputs can be easily computed by applying our construction for symmetric function g_{2^n-1} with 2^n-1 inputs in the following manner.

- For $w \in [0 : 2^n-1]$, $g_{2^n-1}(w) := f_n(w)$
- For $i \in [0 : n-1]$, each player P_i behaves as if P_i were $P_j, j \in [2^i-1 : 2^{i+1}-2]$ in the protocol for g_{2^n-1}.

However, the private PEZ protocol obtained from this method is different from the one obtained from the general construction in [1] although the *order* of an initial string is the same: $\mathcal{O}((2^n-1) \times (2^n-1)!) = \mathcal{O}(2^n!)$. For instance, for $n = 3$, $|\alpha(g_{2^n-1})| = |\alpha(g_7)| = 60,621$ for the above protocol, whereas $|\alpha(f_n)| = |\alpha(f_3)| = 72$ for original protocol in [1], as you can see in Table 1. Therefore, it seems that the general protocol in [1] cannot be directly obtained from our construction.

Acknowledgements. The authors would like to thank the reviewers for their helpful comments and suggestions. They are also grateful to Mr. Shota Yamamoto for insightful discussions.

References

1. Balogh, J., Csirik, J.A., Ishai, Y., Kushilevitz, E.: Private computation using a PEZ dispenser. Theoretical Computer Science **306**(1-3), 69–84 (sep 2003). https://doi.org/10.1016/S0304-3975(03)00210-X, `http://linkinghub.elsevier.com/retrieve/pii/S030439750300210X`
2. den Boer, B.: More efficient match-making and satisfiability: The five card trick. In: Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings. Lecture Notes in Computer Science, vol. 434, pp. 208–217. Springer (1989). https://doi.org/10.1007/3-540-46885-4_23
3. Nishida, T., Hayashi, Y.i., Mizuki, T., Sone, H.: Card-based protocols for any boolean function. In: International Conference on Theory and Applications of Models of Computation. pp. 110–121. Springer (2015)
4. Watanabe, Y., Kuroki, Y., Suzuki, S., Koga, Y., Iwamoto, M., Ohta, K.: Card-based majority voting protocols with three inputs using three cards. In: 2018 International Symposium on Information Theory and Its Applications (ISITA). pp. 218–222. IEEE (2018)

5. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982). pp. 160–164. IEEE (November 1982). https://doi.org/10.1109/SFCS.1982.38, `http://ieeexplore.ieee.org/document/4568388/`

A Technical Lemma and Proofs

A.1 Technical Lemma

Lemma 1. *Let n and s ($n \geq s$) be nonnegative integers such that $n-s-1 \geq 0$. For $k \in [0:n]$, let*

$$a_{n-s-1} = 1 \quad \text{and} \quad a_k = k \sum_{i=n-s-1}^{k-1} a_i + 1, \tag{28}$$

be a recurrence relation with respect to $(a_i)_{i=n-s-1}^{k-1}$. Then, the following holds:

$$a_n = n \cdot n! \sum_{i=n-s}^{n} \frac{1}{i!} + 1. \tag{29}$$

Proof of Lemma 1: For fixed s and n, let

$$S_k \coloneqq \sum_{i=n-s-1}^{k} a_n. \tag{30}$$

Then, we have $S_{n-s-1} = a_{n-s-1} = 1$ and

$$a_k = kS_{k-1} + 1. \tag{31}$$

We also have

$$S_k - S_{k-1} = a_k = kS_{k-1} + 1, \tag{32}$$

where the first and the second equalities are due to (30) and (31), respectively.

Equation (32) can be rearranged as $S_k = (k+1)S_{k-1} + 1$. Dividing both sides of this equality by $(k+1)!$, we obtain

$$T_k = T_{k-1} + \frac{1}{(k+1)!}, \quad \text{and} \quad T_{n-s-1} = \frac{1}{(n-s)!}, \tag{33}$$

where $T_k := S_k/(k+1)!$. Equation (33) is easy to solve. That is,

$$\begin{aligned}
T_k &= T_{k-1} + \frac{1}{(k+1)!} \\
&= T_{k-2} + \frac{1}{(k+1)!} + \frac{1}{k!} \\
&\cdots \\
&= T_{n-s-1} + \frac{1}{(k+1)!} + \cdots \frac{1}{(n-s+1)!} \\
&= \sum_{i=n-s}^{k+1} \frac{1}{i!}.
\end{aligned} \tag{34}$$

Therefore, we have $S_k = (k+1)!\,T_k = (k+1)! \sum_{i=n-s}^{k+1} 1/i!$. Substituting this into (31), we obtain (29). □

A.2 Proof of Theorem 1

We prove (17). Equation (18) can be proved similarly.

For the weight $w \in [0:k]$, fix the input $(x_1, x_2, \ldots, x_k)$ arbitrarily such that $\mathsf{hw}(x_1, x_2, \ldots, x_k) = w$. Then, $g_k^{\mathsf{w}}(w) = g_k(x_1, x_2, \ldots, x_k)$ holds. If $w = k$, the following holds:

$$g_k^{\mathsf{w}}(w) = g_k^{\mathsf{w}}(k) = g_k^{\mathsf{m}}(\{\underbrace{1, 1, \ldots, 1}_{k}\}) \overset{\text{(a)}}{=} g_n^{\mathsf{m}}(\{\underbrace{1, 1, \ldots, 1}_{k}, \underbrace{0, 0, \ldots, 0}_{n-k}\}) \overset{\text{(b)}}{=} f_n^{\mathsf{w}}(k) \tag{35}$$

where the marked equalities are due to the following reasons:

(a): From (15), the value of g_k equals to the value of g_n if the Hamming weights of inputs are equal.

(b): Definition of g_n: $g_n := f_n$, given in Theorem 1.

If $w \neq k$, there exists an index $i \in [0:k-1]$ such that $x_i = 0$, and we have

$$\begin{aligned}
g_k^{\mathsf{w}}(w) &= g_k(x_1, x_2, \ldots, x_k) \\
&= x_k g_k(x_1, x_2, \ldots \overset{i}{\check{0}}, \ldots, x_{k-1}, 1) + \overline{x}_k g_k(x_1, x_2, \ldots, x_{k-1}, 0) \\
&\overset{\text{(c)}}{=} x_k g_k(x_1, x_2, \ldots, \overset{i}{\check{1}}, \ldots, x_{k-1}, 0) + \overline{x}_k f_k(x_1, x_2, \ldots, x_{k-1}, 0) \\
&\overset{\text{(d)}}{=} x_k g_{k-1}(x_1, x_2, \ldots, \overset{i}{\check{1}}, \ldots, x_{k-1}) + \overline{x}_k g_{k-1}(x_1, x_2, \ldots, x_{k-1}) \\
&\overset{\text{(e)}}{=} g_{k-1}^{\mathsf{w}}(w),
\end{aligned} \tag{36}$$

where the marked equalities are due to the following reasons:

(c): Symmetry of g_k.
(d): Definition of g_{k-1} given by (15).
(e): For $x_k = 1$, $\mathsf{hw}(x_1, x_2, \ldots, \overset{i}{\check{1}}, \ldots, x_{k-1}) = w$ holds, otherwise $\mathsf{hw}(x_1, x_2, \ldots, x_{k-1}) = w$ holds. □

A.3 Proof of Theorem 2

We show the proof for the private PEZ protocol constructed by using (19) in Construction 1. If (20) is used, the proof is similar to that for (19).

Let $\Sigma = \{0,1\}$ and $\Gamma = \{0,1\}$. Let $\nu : \{1, 2, \ldots, n^2\} \times \{0,1\} \to \{0,1\}^*$ be a mapping such that $\nu(j,0) = \lambda$, and $\nu(j,1) = \alpha(g_{n-\lceil j/n \rceil})$ for all $j \in \{1, 2, \ldots, n^2\}$. From the definition of ν and μ, we obtain for all $j \in \{1, 2, \cdots, n^2\}$,

$$|\nu(j,0)| = |\lambda| = 0 = \mu_j(0) \tag{37}$$

$$|\nu(j,1)| = |\alpha(g_{n-\lceil j/n \rceil})| = \mu_j(1) \tag{38}$$

Therefore, ν satisfies the first condition in Definition 2.

Next, we show that ν also satisfies the second condition in Definition 2. Let w be a Hamming weight of n inputs where $0 \leq w \leq n$, and $N(w)$ be the substring read by players throughout n rounds when the Hamming weight of n inputs is w. Since the substring read in the j-th round can be represented by $[\alpha(g_{n-j})]^w$, we have

$$N(w) = [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_0)]^w. \tag{39}$$

Note that f_n is symmetric, and therefore it is not necessary to care about the move order in each round, but it is necessary to care about the Hamming weight of n inputs. Using $N(w)$, the second condition in Definition 2 can be rewritten as follows: For all $w \in \{0, 1, \ldots, n\}$,

$$N(w) \circ f_n^{\mathsf{w}}(w) = [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_0)]^w \circ f_n^{\mathsf{w}}(w) \prec \alpha(g_n). \tag{40}$$

Noting that $g_w^{\mathsf{w}}(w) = f_n^{\mathsf{w}}(w)$ and (19), we have

$$\alpha(g_w) = [\alpha(g_{w-1})]^w \circ [\alpha(g_{w-2})]^w \circ \cdots \circ [\alpha(g_0)]^w \circ f_n^{\mathsf{w}}(w). \tag{41}$$

Hence, $N(w) \circ f_n^{\mathsf{w}}(w)$ is written as follows:

$$\begin{aligned}
&N(w) \circ f_n^{\mathsf{w}}(w) \\
&= [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_{w+1})]^w \circ [\alpha(g_w)]^w \circ \alpha(g_w) \\
&= [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_{w+1})]^w \circ [\alpha(g_w)]^{(w+1)} \\
&\prec [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_{w+1})]^w \circ [\alpha(g_w)]^{(w+1)} \\
&\quad \circ [\alpha(g_{w-1})]^{(w+1)} \circ \cdots \circ [\alpha(g_0)]^{(w+1)} \circ f_n^{\mathsf{w}}(w+1) \\
&= [\alpha(g_{n-1})]^w \circ [\alpha(g_{n-2})]^w \circ \cdots \circ [\alpha(g_{w+2})]^w \circ [\alpha(g_{w+1})]^w \circ \alpha(g_{w+1}) \\
&\cdots \\
&\prec [\alpha(g_{n-1})]^w \circ \alpha(g_{n-1}) \\
&= [\alpha(g_{n-1})]^{w+1},
\end{aligned} \tag{42}$$

where the first and the third equalities are due to (41). Therefore, for all $w \in [0 : n-1]$, $N(w) \circ f_n^{\mathsf{w}}(w) \prec [\alpha(g_{n-1})]^{w+1} \prec [\alpha(g_{n-1})]^n \prec \alpha(g_n)$ holds. In addition, for $w = n$, $N(w) \circ f_n^{\mathsf{w}}(w) = \alpha(g_n)$. Thus, for all $w \in [0 : n]$, $N(w) \circ f_n^{\mathsf{w}}(w) \prec \alpha(g_n)$.

Therefore, there exists a mapping ν for a PEZ protocol of Construction 1 using an initial string $\alpha(g_n)$ such that ν satisfies the two condition in Definition 2. □

A.4 Proof of Theorem 3

From (19) and (20), we obtain the length of the initial string $|\alpha(g_n)|$ and $|\alpha(h_n)|$ as follows:

$$\begin{aligned} |\alpha(g_n)| &= n|\alpha(g_{n-1})| + n|\alpha(g_{n-2})| + \cdots + n|\alpha(g_0)| + 1 \\ &= n \sum_{i=0}^{n-1} |\alpha(g_i)| + 1, \end{aligned} \tag{43}$$

$$\begin{aligned} |\alpha(h_n)| &= n|\alpha(h_{n-1})| + n|\alpha(h_{n-2})| + \cdots + n|\alpha(h_0)| + 1 \\ &= n \sum_{i=0}^{n-1} |\alpha(h_i)| + 1, \end{aligned} \tag{44}$$

where $|\alpha(g_0)| = |f_n^{\mathsf{w}}(0)| = 1$ and $|\alpha(h_0)| = |f_n^{\mathsf{w}}(n)| = 1$. Therefore, we obtain the same relation between $|\alpha(g_n)|$ and $|\alpha(h_n)|$. Summarizing the above, and noting that $a_n = |\alpha(f_n)| = |\alpha(g_n)| = |\alpha(h_n)|$, $\{a_i\}_{i=0}^n$ satisfies the following recurrence relation:

$$a_0 = 1, \quad a_n = n \sum_{i=0}^{n-1} a_i + 1, \tag{45}$$

which is a special case of (28) in Theorem 1 with $s = n - 1$. Thus, we obtain (23).

In addition, the following relations hold:

$$\sum_{i=1}^{n} \frac{1}{i!} < \sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - (1/2)^n < 2 \tag{46}$$

Therefore, $a_n < 2n \cdot n! + 1$, which yields $a_n = \mathcal{O}(n \times n!)$. □