Card-Based Protocol Counting Connected Components of Graphs

Koji Nuida[1][2]
[1] Institute of Mathematics for Industry (IMI), Kyushu University
nuida@imi.kyushu-u.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST)

September 16, 2025

Abstract

Card-based cryptography is a research area for realizing cryptographic functionality, such as secure multiparty computation and zero-knowledge proofs, by using a deck of physical cards and/or other non-electrical tools. Motivated by zero-knowledge proofs for solutions in pencil puzzles, there is a direction of recent studies on card-based protocols to verify connectivity of a set of cells or edges on lattice-shaped boards. In this paper, we generalize the problem to counting connected components of a subset of the vertex set on any graph, and propose a card-based protocol for the problem.

1 Introduction

Card-based cryptography is a research area for realizing cryptographic functionality with concealed input data, such as secure multiparty computation and zero-knowledge proofs, by using a deck of physical cards and/or other "physical" (non-electrical) tools like envelopes and rubber bands. After the invention by den Boer [1], this area has history of about 35 years, and is actively studied especially in recent years. Topics of this area include construction of protocols for specific functions such as AND function [1, 7], optimization of protocols in terms of certain complexity measures such as the number of cards [8] and the number of shuffle operations [15], impossibility results [4], and relations to other conventional cryptography (performed on electrical computers) [16]. Among them, one of the most active research topics is zero-knowledge proofs for solutions of pencil puzzles such as Sudoku [3, 13, 14, 17, 18], where a prover can convince a verifier that the prover knows a solution of a puzzle instance without revealing information on the solution.

In particular, recently card-based zero-knowledge proofs are studied for puzzles where the condition for a solution includes connectivity of a set of cells or edges on lattice-shaped boards, such as Nurikabe [9] and Slitherlink [5]. Such protocols require methods for checking connectivity of a solution by using physical cards. Most of the existing methods like the aforementioned ones [5, 9] are *interactive*, that is, the prover's knowledge of a solution is required not only to generate input data but also to execute the protocol. On the other hand, in a recent work by Sasaki and Shinagawa [12] on card-based zero-knowledge proofs for Sukoro, they introduced a *non-interactive* method for checking whether the publicly known points in a solution belong to the same connected component of the solution. However, their method is dependent on properties of a specific puzzle (Sukoro, in their case). To the author's best knowledge, non-interactive card-based protocols that deal with connected components of a region in a more general manner are not known. Moreover, their protocol in [12] requires $O(k^4)$ shuffles for Sukoro instances on a $k \times k$ board, therefore it is not very efficient.

1.1 Our Contribution

In this paper, we propose a card-based protocol for the first time for the problem of counting the number of connected components of a given subset S of the vertex set of a predetermined (finite, simple, and undirected) graph G (more precisely, connected components of the full subgraph of G with vertex set S) without revealing

information on the set S. More precisely, our protocol securely and non-interactively computes a complete set of representatives $R_S \subseteq S$ for connected components of S (that is, R_S involves exactly one vertex from each connected component of S and no other vertices) and outputs a commitment of the bit γ_v, for every vertex v of G, indicating whether $v \in R_S$ or not. After executing our protocol, the number of connected components of S can be obtained via any protocol of securely adding the bits γ_v, such as the protocol proposed by Ruangwises and Itoh [11] (or, if a non-committed output is sufficient, by completely shuffling the commitments of γ_v's, opening them, and counting the number of bit 1's). On the other hand, for any fixed value c, whether S has exactly c connected components or not can also be determined by performing, after our protocol, any protocol determining whether there are exactly c bit 1's among γ_v's. By setting $c = 1$, this implies that connectivity of S can be determined as well.

In comparison to the aforementioned protocol in [12] using $O(k^4)$ shuffles, our protocol for the case of the $k \times k$ lattice-shaped graph requires $O(k^3 \log k)$ shuffles. Therefore, the efficiency in terms of the number of shuffles of our protocol outperforms the protocol in [12], at the cost of requiring a larger number $O(k^2 \log k)$ of cards instead of $O(k^2)$ for the case of [12].

We explain an overview of our proposed protocol. In each step of the main loop, starting from the empty set $V' := \emptyset$, a subset V' of the whole vertex set V of the graph G is enlarged by adding a vertex, say v, to V', and a complete set of representatives $R_{S \cap V'}$ for connected components of $S \cap V'$ obtained at the previous step is updated to a complete set of representatives $R_{S \cap (V' \cup \{v\})}$ for connected components of $S \cap (V' \cup \{v\})$. Now our protocol is designed in a way that the process of updating $R_{S \cap V'}$ to $R_{S \cap (V' \cup \{v\})}$ requires only local information on the graph G around the vertex v. How the updating process is complicated is in general dependent on the structure of the graph G, but for the case of the $k \times k$ lattice-shaped graph (with k^2 vertices), the updating process searches for only $O(k)$ vertices v' of the graph. The process focusing on each vertex v' is executed by using $O(\log k)$ shuffles, therefore the protocol requires $k^2 \cdot O(k) \cdot O(\log k) = O(k^3 \log k)$ shuffles in total.

1.2 Related Work

In a work by Ruangwises and Itoh [10], they also considered a problem of non-interactively checking connectivity of an input graph by using physical cards. However, the problem setting is significantly different from ours. In their protocol, the underlying graph $G = (V, E)$ is known in advance (similarly to our protocol), but a private input is a subset S of the *edge set* E (instead of the vertex set V as in our protocol). Then their goal is to securely verify that the pair (V, S) forms a connected graph. Now if we try to apply their protocol to our problem setting, the vertex set of the input subgraph should be publicly known, which contradicts our problem setting where the input is a private subset of the whole vertex set. Therefore, our result in the present paper is independent of the result in [10].

We show comparison of our result with the related results in Table 1. For comparison with results of [9, 12] where the settings are similar to ours, there is mainly the following trade-off. First, compared to [9], our protocol becomes non-interactive but requires more shuffles and cards than [9]. Secondly, compared to [12], our protocol requires less shuffles than [12] but more cards than [12].

1.3 Organization of the Paper

In Section 2, we summarize notations and terminology used in this paper and explain some basic information on card-based protocols. In Section 3, we describe the subprotocols used as building blocks in our proposed card-based protocol. In Section 4, we first construct a (not card-based) algorithm for the problem in the case of non-secret input (Section 4.1), and then convert the algorithm into our proposed card-based protocol (Section 4.2). We evaluate the numbers of cards and shuffles used in our proposed protocol in Sections 4.3 and 4.4, respectively, and give a comparison with the existing method in Section 4.5. Section 5 concludes this paper.

Table 1: Comparison of our protocol with related protocols, where $G = (V, E)$ denotes a known underlying graph: Here the efficiency of [5, 10] are not shown as their settings are too different from ours; (*) more precisely, their protocol checks if all the "hint cells" belong to the same connected component; (†) though they only considered lattice-shaped graphs, their protocol can be extended to other graphs by the same idea

Protocol	Input S	What to see	Non-interactive?	Efficiency (on $k \times k$ lattice-shaped graph)		Condition on G
				# of Shuffles	# of Cards	
[5]	$S \subseteq E$	being a single loop	no	—		lattice-shaped
[10]	$S \subseteq E$	connectivity	**yes**	—		any
[9]	$S \subseteq V$	connectivity	no	$O(k^2)$	$O(k^2)$	lattice-shaped
[12]	$S \subseteq V$	connectivity (*)	**yes**	$O(k^4)$	$O(k^2)$	any (†)
Ours	$S \subseteq V$	# of connected components	**yes**	$O(k^3 \log k)$	$O(k^2 \log k)$	any

2 Preliminaries

In this paper, we write an integer interval $\{m, m+1, \ldots, n\}$ as $[m, n]$. A *graph* means a pair $G = (V, E)$ of a finite vertex set V and an edge set E where each edge is an unordered pair of two different vertices in V; namely, we consider finite, simple, and undirected graphs only. We say that two vertices $v_1, v_2 \in V$ of G are *adjacent* if E involves the edge $\{v_1, v_2\}$. We define a binary relation $\sim$ on a given subset S of the vertex set V in a way that we have $v_1 \sim v_2$ if and only if there is a path in G from v_1 to v_2 that consists of edges whose two vertices both belong to S. This is an equivalence relation on S. Each equivalence class of the relation $\sim$ is called a *connected component* of S in G. We say that S is *connected* if S has exactly one connected component.

Any card-based protocol in this paper uses a deck of *two-colored cards*; that is, the front sides have two kinds of symbols ♡ and ♣ where cards with the same symbol are supposed to be indistinguishable, and the back sides have the same symbol '?' and are supposed to be indistinguishable. As a secret encoding of a bit $b \in \{0, 1\}$, we use an ordered pair of two cards $\boxed{\clubsuit}\boxed{\heartsuit}$ for $b = 0$ and $\boxed{\heartsuit}\boxed{\clubsuit}$ for $b = 1$, which are faced down as $\boxed{?}\boxed{?}$. We call the pair $\boxed{?}\boxed{?}$ of these face-down cards a *commitment* of bit b, denoted by $\mathsf{Com}(b)$. Moreover, for the k-bit binary expression $x = (x_{k-1} \cdots x_1 x_0)_{(2)}$ of a non-negative integer x (where $x_i \in \{0, 1\}$, and the most significant bits may be zero), the value x is encoded as the following sequence of k commitments:

$$\mathsf{Com}(x)^{(k)} := (\mathsf{Com}(x_0), \mathsf{Com}(x_1), \ldots, \mathsf{Com}(x_{k-1})) \ .$$

We call the sequence a *commitment* of integer x.

Any card-based protocol in this paper is a *committed* protocol, namely the input and the output consist of a number of commitments of bits. The security of a committed card-based protocol is defined to satisfy that the probability distribution of the symbols of cards that are opened during a protocol execution is independent of the input of the protocol (see [6] for a formal description of the security). For our proposed protocol, the security of the whole protocol is immediately implied by the security of the subprotocols. By the reason, we do not discuss the security of our proposed protocol from now on.

In this paper, we deal with the following two kinds of shuffles for a set of face-down cards. First, a *random bisection cut* (RBC) [7] is an operation for two piles of cards α_0 and α_1 consisting of the same number of face-down cards that results in a pair (α_0, α_1) or (α_1, α_0) with probability 1/2 each. We assume that which of the two results is actually obtained is not known by any party in a protocol. Secondly, a *complete shuffle* (CS) is an operation for a set of face-down cards that results in a sequence of those cards

permuted uniformly at random. We assume that which permutation is actually performed is not known by any party in a protocol.

In this paper, we focus on the following quantities as complexity measures of card-based protocols:

Number of Shuffles: The number of shuffles performed in a protocol (in the worst case). This is regarded as the primary complexity measure of card-based protocols in this paper.

Number of Input Cards: The number of cards for input encoding in a protocol. Note that in this paper, the numbers of $\boxed{\clubsuit}$'s and $\boxed{\heartsuit}$'s in an input for a card-based protocol are always equal.

Number of Additional Cards: The number of all cards used in a protocol except for input encoding (in particular, including cards for output encoding). Note that in this paper, the numbers of $\boxed{\clubsuit}$'s and $\boxed{\heartsuit}$'s in the additional cards for a card-based protocol are always equal.

Number of Free Cards: We say that a card used in a card-based protocol is a *free card* if it is opened at the end of a protocol execution. The free cards can be securely reused in a subsequent protocol. Note that in this paper, the numbers of $\boxed{\clubsuit}$'s and $\boxed{\heartsuit}$'s in the free cards for a card-based protocol are always equal.

Number of Discarded Cards: We say that a card used in a card-based protocol is a *discarded card* if it is neither opened nor used in output encoding at the end of a protocol execution. Note that in this paper, the numbers of $\boxed{\clubsuit}$'s and $\boxed{\heartsuit}$'s in the discarded cards for a card-based protocol are always equal. Note also that directly opening a discarded card may compromise security of a protocol; on the other hand, discarded cards may be securely reused after performing a CS for the discarded cards.

3 Subprotocols for Our Protocol

In this section, we explain the following subprotocols (see also Table 2) used as building blocks in our proposed card-based protocol.

Duplication $\mathsf{Dup}^{\ell}(\alpha)$: When the input α is a single commitment $\mathsf{Com}(b)$, the original $\mathsf{Com}(b)$ is preserved and the subprotocol outputs ℓ additional commitments $\mathsf{Com}(b)$ (that is, $\ell+1$ commitments $\mathsf{Com}(b)$ in total). (We write Dup^1 simply as Dup.) This is simply realized by Mizuki–Sone's COPY protocol [7]. On the other hand, when α is a sequence of multiple commitments, we duplicate individual commitments in α one by one (where the free cards in the previous duplication are reused).

NOT $\neg$, XOR $\oplus$, AND $\wedge$: In order to obtain $\mathsf{Com}(\neg b)$ from input commitment $\mathsf{Com}(b)$, we simply swap the two cards of $\mathsf{Com}(b)$. For XOR and AND operations, we use Mizuki–Sone's XOR and AND protocols [7]. Note that these subprotocols do not preserve input commitments.

Input-Preserving XOR $\oplus_{\mathsf{Dup}}$: For inputs $\mathsf{Com}(b_1)$ and $\mathsf{Com}(b_2)$, this subprotocol outputs $\mathsf{Com}(b_1 \oplus b_2)$ while preserving the two inputs. Naively, this is realized by using the aforementioned Dup protocol twice and XOR. But in fact, the numbers of shuffles and additional cards can be reduced by the following procedure:

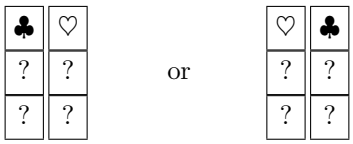
1. Execute $\mathsf{Com}(b_1') \leftarrow \mathsf{Dup}(\mathsf{Com}(b_1))$ (and reuse two free cards in the subsequent steps).
2. Arrange cards as follows:

$$\begin{aligned} \mathsf{Com}(b_2) &= \boxed{?}\,\boxed{?} \\ \mathsf{Com}(b_1') &= \boxed{?}\,\boxed{?} \\ \mathsf{Com}(0) &= \boxed{?}\,\boxed{?} \end{aligned}$$

Table 2: Our subprotocols: (*) input consists of k commitments; (§) only the first commitment is counted (not including the remaining card sequences); (†) the first input component is a k-bit commitment, and the cost for internal op is excluded.

		# of Cards			
Protocols	Shuffles	Input	Additional	Free	Discarded
Dup^{ℓ} (*)	k RBC	$2k$	$2\ell k + 2$	2	0
$\neg$	0	2	0	0	0
$\oplus$	1 RBC	4	0	2	0
$\wedge$	1 RBC	4	2	2	2
$\oplus_{\mathsf{Dup}}$	2 RBC	4	4	2	0
OExc OSel OSub	1 RBC	2 (§)	0	2	0
$\mathsf{OExc}_{\mathsf{Dup}}$ $\mathsf{OSel}_{\mathsf{Dup}}$ $\mathsf{OSub}_{\mathsf{Dup}}$	1 RBC	2 (§)	2	2	0
$\wedge^{k}_{\mathsf{Dup}}$	k RBC	$2k$	$2k + 2$	2	$2k - 2$
OApp (†)	$2k$ RBC	$2k$ (§)	2	$2k + 2$	0
$\mathsf{Com}(x)^{(k)} -_{\mathsf{Dup}} y$ $(k \geq 2)$	$6k - 8$ RBC $+ 1$ CS	$2k$	$4k$	$2k$	0

3. Perform an RBC that swaps the left and the right columns, and open the cards in the first row.



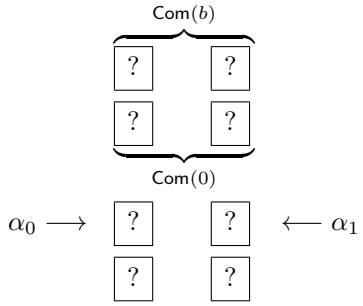
4. If the left opened card is $\boxed{\heartsuit}$, swap the two columns; otherwise, do nothing. Then the second row gives the output $\mathsf{Com}(b_1 \oplus b_2)$ of the protocol, while the third row gives duplication of $\mathsf{Com}(b_2)$.

Oblivious Exchange OExc: Let α_0 and α_1 be piles of the same number of face-down cards. An *oblivious exchange* protocol $\mathsf{OExc}(\mathsf{Com}(b); \alpha_0, \alpha_1)$ for inputs α_0 and α_1 outputs a pair (α_0, α_1) if $b = 0$, and outputs a pair (α_1, α_0) if $b = 1$. That is, the subprotocol swaps the inputs α_0 and α_1 only when $b = 1$, without revealing the control bit b. As explained below, this subprotocol is realized by essentially the same idea as Mizuki–Sone's AND protocol [7].

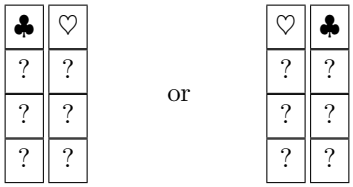
We also let $\mathsf{OExc}_{\mathsf{Dup}}(\mathsf{Com}(b); \alpha_0, \alpha_1)$ denote a variant of the subprotocol where the commitment $\mathsf{Com}(b)$ is preserved. This is realized by the following procedure. We note that a description of $\mathsf{OExc}(\mathsf{Com}(b); \alpha_0, \alpha_1)$ is obtained by omitting the row of $\mathsf{Com}(0)$ in the procedure.

1. Place a commitment $\mathsf{Com}(0)$ at the row next to $\mathsf{Com}(b)$, and place the cards α_0 at the left column and the cards α_1 at the right column next to the row of $\mathsf{Com}(0)$. The following figure shows the case that

each of α_0 and α_1 consists of two cards.



2. Perform an RBC that swaps the left and the right columns, and open the first row.



3. If the left opened card is $\boxed{\heartsuit}$, swap the two columns; otherwise, do nothing. Then the second row gives duplication of $\mathsf{Com}(b)$; and the left column α' and the right column α'' of the remaining rows give the output (α', α'').

Oblivious Selection OSel, Oblivious Substitution OSub: For oblivious exchange $\mathsf{OExc}(\mathsf{Com}(b); \alpha_0, \alpha_1)$, when we change the protocol in a way that it outputs the left part α' only instead of an output pair (α', α''), the resulting operation is to output α_0 if the bit b is 0 and output α_1 if the bit b is 1, hence it outputs α_b. We call this protocol an *oblivious selection* protocol, denoted by $\mathsf{OSel}(\mathsf{Com}(b); \alpha_0, \alpha_1)$. Moreover, we consider its variant where the commitment $\mathsf{Com}(b)$ is preserved, denoted by $\mathsf{OSel}_{\mathsf{Dup}}(\mathsf{Com}(b); \alpha_0, \alpha_1)$.

On the other hand, we can interpret the functionality of oblivious selection $\mathsf{OSel}(\mathsf{Com}(b); \alpha, \alpha')$ in the following manner: Given a pile of cards α and a commitment $\mathsf{Com}(b)$, the protocol replaces α with α' if $b = 1$, and does nothing if $b = 0$. We call the protocol with the latter interpretation an *oblivious substitution* protocol, denoted by $\mathsf{OSub}(\mathsf{Com}(b); \alpha \,/\, \alpha')$. Moreover, we consider its variant where the commitment $\mathsf{Com}(b)$ is preserved, denoted by $\mathsf{OSub}_{\mathsf{Dup}}(\mathsf{Com}(b); \alpha \,/\, \alpha')$.

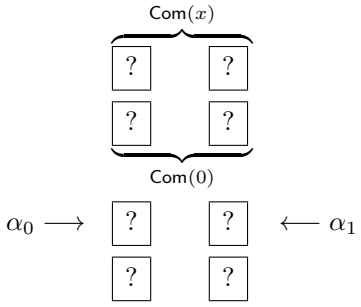
Input-Preserving Multi-Input AND $\wedge^k_{\mathsf{Dup}}$: Note that for oblivious selection $\mathsf{OSel}(\mathsf{Com}(b); \alpha_0, \alpha_1)$, the output for inputs $\alpha_0 := \mathsf{Com}(0)$ and $\alpha_1 := \mathsf{Com}(b')$ becomes $\mathsf{Com}(b')$ if $b = 1$ and $\mathsf{Com}(0)$ if $b = 0$; i.e., the result of AND operation $\mathsf{Com}(b \wedge b')$. In fact, this protocol $\mathsf{OSel}(\mathsf{Com}(b); \mathsf{Com}(0), \mathsf{Com}(b'))$ is the same as Mizuki–Sone's AND protocol [7], except that our protocol does not arrange the cards as a straight sequence, while Mizuki–Sone's protocol arranges the cards strictly as a straight sequence.

Similarly, by using an input-preserving oblivious selection protocol $\mathsf{OSel}_{\mathsf{Dup}}(\mathsf{Com}(b_2); \mathsf{Com}(0), \mathsf{Com}(b_1))$, we can compute $\mathsf{Com}(b_1 \wedge b_2)$ and preserve the commitment $\mathsf{Com}(b_2)$. Note that it requires two additional cards to generate a commitment $\mathsf{Com}(0)$ in comparison to the non-input-preserving case. Moreover, by executing the procedure recursively, and by duplicating $\mathsf{Com}(b_1)$ by Dup at the beginning, we can preserve k input commitments $\mathsf{Com}(b_1), \ldots, \mathsf{Com}(b_k)$ in computation of $\mathsf{Com}(b_1 \wedge \cdots \wedge b_k)$. We write the resulting protocol as $\wedge^k_{\mathsf{Dup}}$, or simply $\wedge_{\mathsf{Dup}}$ when $k = 2$. Note that in comparison with the protocol of Nishida et al. [8] for the same functionality, our protocol requires a smaller number of shuffles at the cost of using a larger number of cards.

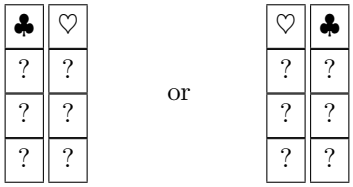
Oblivious Application OApp**:** We consider the following situation: A k-bit commitment $\mathsf{Com}(x)^{(k)}$ of a non-negative integer x and a sequence $\alpha_0, \alpha_1, \ldots, \alpha_{2^k-1}$ of piles of the same number of face-down cards are given and we apply a certain operation op to the x-th pile α_x without revealing the value of x. We call such a protocol an *oblivious application* protocol for operation op, denoted by $\mathsf{OApp}(\mathsf{Com}(x)^{(k)}; \alpha_0, \ldots, \alpha_{2^k-1} \mid \mathsf{op})$.

Except that we use a different encoding of integer x, the functionality of oblivious application is similar to Card Choosing Protocol [2] and Chosen Pile Protocol [9]. Now when $k = 1$, i.e., when x is a bit, the encoding of x in this paper coincides with that in the existing methods above, therefore we can use those existing methods. Precisely, the procedure is as follows.

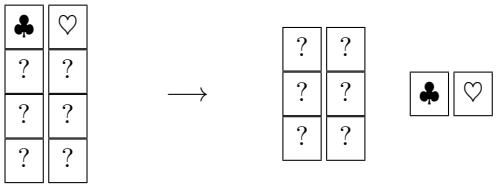
1. Place a commitment $\mathsf{Com}(0)$ at the row next to $\mathsf{Com}(x) = \mathsf{Com}(x_0)$, and place the pile α_0 at the left column and the pile α_1 at the right column next to the row of $\mathsf{Com}(0)$. The following figure shows the case that each of α_0 and α_1 consists of two cards.



2. Apply an RBC swapping the left and the right columns, and open the first row.



3. If the left opened card is $\boxed{\heartsuit}$, swap the two columns; otherwise, do nothing. Then remove the cards in the first row as free cards.



4. Apply the operation op to the second to the last rows of the left column.

5. Apply an RBC swapping the two columns, and open the first row. Moreover, if the left opened card is $\boxed{\heartsuit}$, swap the two columns; otherwise, do nothing.

6. Output the second to the last rows of the left column as the new α_0, and the second to the last rows of the right column as the new α_1.

When $k \geq 2$, let op' denote the operation to apply $\mathsf{OApp}((\mathsf{Com}(x_i))_{i=0}^{k-2}; \alpha'_0, \ldots, \alpha'_{2^{k-1}-1} \mid \mathsf{op})$ to a sequence of piles of cards $\alpha'_0, \ldots, \alpha'_{2^{k-1}-1}$. Then we have

$$\begin{aligned}&\mathsf{OApp}(\mathsf{Com}(x)^{(k)}; \alpha_0, \ldots, \alpha_{2^k-1} \mid \mathsf{op})\\ &= \mathsf{OApp}(\mathsf{Com}(x_{k-1}); (\alpha_i)_{i=0}^{2^{k-1}-1}, (\alpha_{2^{k-1}+i})_{i=0}^{2^{k-1}-1} \mid \mathsf{op}') \ .\end{aligned}$$

Protocol 1 Input-Preserving Subtraction by Public Value $\mathsf{Com}(x)^{(k)} -_{\mathsf{Dup}} y$

Input A commitment $\mathsf{Com}(x)^{(k)}$ of an integer x, and an integer y, where $0 \leq y \leq x \leq 2^k - 1$
Output A commitment $\mathsf{Com}(z)^{(k)}$ of $z := x - y$; $\mathsf{Com}(x)^{(k)}$ is also preserved

if $y_0 = 1$ **then**
  $\big(\mathsf{Com}(x'_0), \mathsf{Com}(x''_0)\big) \leftarrow \mathsf{Dup}^2\big(\mathsf{Com}(x_0)\big)$
  $\mathsf{Com}(z_0) \leftarrow \neg\mathsf{Com}(x'_0)$, $\mathsf{Com}(c_1) \leftarrow \neg\mathsf{Com}(x''_0)$
else
  $\mathsf{Com}(z_0) \leftarrow \mathsf{Dup}\big(\mathsf{Com}(x_0)\big)$, $\mathsf{Com}(c_1) \leftarrow \mathsf{Com}(0)$
end if
for $i = 1, 2, \ldots, k-2$ **do**
  if $y_i = 1$ **then**
    $\mathsf{Com}(c_{i+1}) \leftarrow \mathsf{Dup}\big(\mathsf{Com}(c_i)\big)$, $\mathsf{Com}(y'_i) \leftarrow \neg\mathsf{Com}(c_i)$
  else
    $\mathsf{Com}(c_{i+1}) \leftarrow \mathsf{Com}(0)$, $\mathsf{Com}(y'_i) \leftarrow \mathsf{Com}(c_i)$
  end if
  $\mathsf{Com}(z_i) \leftarrow \mathsf{Com}(x_i) \oplus_{\mathsf{Dup}} \mathsf{Com}(y'_i)$
  $\mathsf{Com}(c_{i+1}) \leftarrow \mathsf{Com}(c_{i+1}) \oplus \Big(\big(\neg\mathsf{Dup}(x_i)\big) \wedge \mathsf{Com}(y'_i)\Big)$
end for
if $y_{k-1} = 1$ **then**
  $\mathsf{Com}(z_{k-1}) \leftarrow \mathsf{Com}(0)$
else
  $\mathsf{Com}(z_{k-1}) \leftarrow \mathsf{Dup}\big(\mathsf{Com}(x_{k-1})\big) \oplus \mathsf{Com}(c_{k-1})$
end if
Output $\mathsf{Com}(z)^{(k)} = \big(\mathsf{Com}(z_0), \ldots, \mathsf{Com}(z_{k-1})\big)$

Therefore, the desired operation can be performed for the case $k \geq 2$ as well by recursively executing the aforementioned procedure for the case $k = 1$.

Input-Preserving Subtraction by Public Value $\mathsf{Com}(x)^{(k)} -_{\mathsf{Dup}} y$: We consider a protocol described in Protocol 1 that, for given commitment $\mathsf{Com}(x)^{(k)}$ of a non-negative integer $x = (x_{k-1} \cdots x_1 x_0)_{(2)}$ and a publicly known non-negative integer $y = (y_{k-1} \cdots y_1 y_0)_{(2)} \leq x$, outputs a commitment of $x - y$ and preserves the commitment $\mathsf{Com}(x)^{(k)}$. Note that in the protocol, c_i means the carry bit to be considered in the computation at the i-th bit. See Table 2 for the complexity of the protocol, where the numbers of shuffles and additional cards are counted for the worst case because these values depend on the value of y. Moreover, the numbers of free cards and discarded cards also depend on the value of y; for the sake of simplicity, Table 2 shows the values by assuming that all the discarded cards are made free by applying a CS and then opening them.

4 Our Proposed Protocol

In this section, we describe our proposed protocol. First, in Section 4.1, we consider a situation where the input is not secret, and give an algorithm that is suitable to be converted into a card-based protocol. Then in Section 4.2, we implement our algorithm as a card-based protocol, which is our main proposed protocol, by using the subprotocols in Section 3.

Here we clarify the problem description. Let $G = (V, E)$ be a graph, and enumerate the vertices as $V = \{v_1, v_2, \ldots, v_{|V|}\}$. We suppose that a partition $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_L$ into non-empty disjoint subsets V_i of V is given in a way that if $i, j \in [1, L]$ and $|i - j| \geq 2$, then any vertex in V_i is not adjacent to a vertex in V_j. We set $V_0 := \emptyset$. We suppose that $0 = \ell(0) < \ell(1) < \cdots < \ell(L) = |V|$ is a sequence of integers satisfying

that for any $i \in [1, L]$, we have

$$V_i = \{v_h : h \in [\ell(i-1)+1, \ell(i)]\} \ .$$

We set $\ell(-1) := 0$. Moreover, for each $i \in [1, L]$ and $h \in [0, |V|]$, we define

$$V_i^{\leq h} := \{v_{h'} \in V_i : h' \leq h\} \ .$$

For each $i \in [1, L]$ and $v_h \in V_i$, we define

$$N(v_h) := \{v_{h'} \in V_{i-1} \cup V_i^{\leq h-1} : v_{h'} \text{ and } v_h \text{ are adjacent}\} \ .$$

These data are supposed to be given in advance and be publicly known. On the other hand, let S be a subset of V. For each $h \in [0, |V|]$, we define

$$S_{\leq h} := \{v_{h'} \in S : h' \leq h\}$$

(in particular, $S_{\leq 0} = \emptyset$ and $S_{\leq |V|} = S$). In our proposed protocol, we count the number of connected components of S (more precisely, connected components of the full subgraph of G with vertex set S) without revealing the input S. Here the input $S \subseteq V$ is given as a sequence of (commitments of) bits $b_1, \ldots, b_{|V|}$ defined by

$$b_h := \begin{cases} 1 & (\text{if } v_h \in S) \\ 0 & (\text{if } v_h \notin S) \ . \end{cases} \tag{1}$$

Remark 1. Here we assumed that an appropriate partition $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_L$ of the vertex set V of the underlying graph G is given in advance. For a choice of such a partition, we mention that from the viewpoint of efficiency of our proposed protocol, it is desirable that every part V_i is fairly small compared to the whole set V. For lattice-shaped graphs G, which are the main target of the proposed protocol, a "good" partition is easily obtained as shown in Figure 3 below. However, in general, not every graph G admits a "good" partition of V. For example, when G is a complete bipartite graph $K_{n,n}$ with $2n$ vertices, it can be shown that every partition of V satisfying our condition involves a part V_i with at least n vertices.

In a general graph G, a possible algorithm to find a partition of V satisfying our condition is as follows: We start with a part $V_1 = \{v\}$ consisting of any single vertex v. Then we recursively construct a next part V_{i+1} by collecting all the neighbors of V_i that have not been included in the previous parts $V_1, \ldots, V_i$. For the construction, whether the resulting partition is "good" or not depends on the structure of G and the choice of the initial vertex v. An analysis of this algorithm and a search for a better algorithm to find a "good" partition are future research topics.

4.1 Our Algorithm for Non-Secret Input

Here we consider the problem above for the input S being not secret, and give an algorithm as in Algorithm 2. We convert the algorithm into a card-based protocol later in Section 4.2. Note that some parts of Algorithm 2 are made redundant in order to make it easier to convert into a card-based protocol.

In Algorithm 2, we use non-negative integer-valued variables $\beta_h \in [0, |V|]$ and bit-valued variables $\gamma_h \in \{0, 1\}$ ($h \in [1, |V|]$). Intuitively, the variables β_h maintain information on which vertex in V belongs to S and which pair of two vertices belongs to the same connected components. On the other hand, the variables γ_h maintain information on representative points satisfying that a single point is chosen from each connected component of S. As shown in Theorem 1 later, the number of connected components of S is obtained by adding the output bits $\gamma_1, \ldots, \gamma_{|V|}$ of Algorithm 2.

Example 1. In an execution of Algorithm 2 for the graph $G = (V, E)$ and the subset $S \subseteq V$ as in Figure 1, the transition of states of the variables β_h and γ_h is as in Figure 2. In the final result, the number of vertices indicated by squares (that is, vertices v_h with $\gamma_h = 1$) is indeed equal to two, which is the number of connected components of S.

Algorithm 2 Algorithm Counting Connected Components of a Subset

Input Bits $b_1, \ldots, b_{|V|}$ satisfying Eq. (1)
Output Bits $\gamma_1, \ldots, \gamma_{|V|} \in \{0,1\}$ as in the statement of Theorem 1 (given later)

1: **for** $i = 1, 2, \ldots, L$ **do**
2: **for** $j = 1, 2, \ldots, \ell(i) - \ell(i-1)$ **do**
3: $h_0 \leftarrow \ell(i-1) + j$
4: **for each** $v_{h_1} \in N(v_{h_0})$ **do**
5: **for each** $v_{h_2} \in (V_{i-1} \cup V_i^{\leq h_0 - 1}) \setminus \{v_{h_1}\}$ **do**
6: **if** $b_{h_0} = 1$ AND $b_{h_1} = 1$ AND $\beta_{h_2} = \beta_{h_1}$ **then**
7: $\beta_{h_2} \leftarrow h_0$
8: **else**
9: $\beta_{h_2} \leftarrow \beta_{h_2}$ `// Do nothing`
10: **end if**
11: **end for**
12: **if** $b_{h_0} = 1$ AND $b_{h_1} = 1$ **then**
13: $(\gamma_{\beta_{h_1}}, \beta_{h_1}) \leftarrow (0, h_0)$
14: **else**
15: $(\gamma_{\beta_{h_1}}, \beta_{h_1}) \leftarrow (\gamma_{\beta_{h_1}}, \beta_{h_1})$ `// Do nothing`
16: **end if**
17: **end for**
18: **if** $b_{h_0} = 1$ **then**
19: $(\gamma_{h_0}, \beta_{h_0}) \leftarrow (1, h_0)$
20: **else**
21: $(\gamma_{h_0}, \beta_{h_0}) \leftarrow (0, 0)$
22: **end if**
23: **end for**
24: **end for**
25: Output γ_h for all $h \in [1, |V|]$

Example 2. In an execution of Algorithm 2 for the graph $G = (V, E)$ and the subset $S \subseteq V$ as in Figure 3, the transition of states of the variables β_h and γ_h is as in Figure 4. In the final result, the number of vertices indicated by squares (that is, vertices v_h with $\gamma_h = 1$) is indeed equal to one, which is the number of connected components of S.

We give the following lemma to recursively analyze the behavior of variables in our algorithm.

Lemma 1. *Let $i \in [1, L]$ and $j \in [0, \ell(i) - \ell(i-1)]$, and put $h_0 := \ell(i-1) + j$. Among the procedures in Algorithm 2, when $j = 0$, we focus on the moment of moving from Line 1 to Line 2 in the primary loop with this value of i (namely, the beginning of the secondary loop); and when $j \geq 1$, we focus on the moment of arriving at Line 23 in the secondary loop with this value of j. Then for the values of variables β_h and γ_h, we have the following properties.*

1. *We have $\beta_h \leq h_0$ for any $h \in [1, h_0]$.*
2. *Let $h'_1, \ldots, h'_m$ denote the indices $h \in [1, h_0]$ satisfying the condition $\gamma_h = 1$. Then the set $R := \{v_{h'_1}, \ldots, v_{h'_m}\}$ is a complete set of representatives for connected components of the subset $S_{\leq h_0} \subseteq V$. That is, R involves exactly one vertex from each connected component of $S_{\leq h_0}$ and no other vertices.*
3. *Let C be any connected component of $S_{\leq h_0}$. If $C \cap (V_{i-1} \cup V_i^{\leq h_0}) \neq \emptyset$, then the set R in the previous item involves a vertex, say v_h, in the set $C \cap (V_{i-1} \cup V_i^{\leq h_0})$, which also satisfies that $\beta_h = h$. If moreover $C \cap V_i^{\leq h_0} \neq \emptyset$, then the set R involves a vertex, say v_h, in the set $C \cap V_i^{\leq h_0}$, which also satisfies that $\beta_h = h$.*

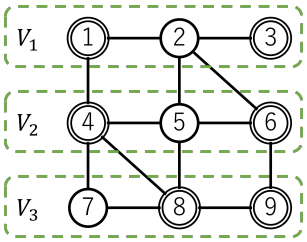


Figure 1: Graph G and subset $S \subseteq V$ for Example 1 (double circles indicate vertices in S and the value inside each vertex indicates the index h for vertex v_h)

4. *Suppose that* $v_h \in V_{i-1} \cup V_i^{\leq h_0}$. *Then we have* $\beta_h \neq 0$ *if and only if* $v_h \in S$.

5. *Suppose that* $v_{h_1}, v_{h_2} \in V_{i-1} \cup V_i^{\leq h_0}$ *satisfy that* $\beta_{h_1} \neq 0$ *and* $\beta_{h_2} \neq 0$. *Then we have* $\beta_{h_1} = \beta_{h_2}$ *if and only if* v_{h_1} *and* v_{h_2} *belong to the same connected component of* $S_{\leq h_0}$.

Proof. We proceed the proof recursively with respect to i, and for each fixed i, recursively with respect to j. First we consider the case $j = 0$. When $i = 1$, the conditions in the statement are verified as follows.

- Condition 1: It is trivially satisfied because the range of h is an empty interval $[1, 0]$.
- Condition 2: We have $R = \emptyset$ similarly to the previous item, while we have $S_{\leq \ell(i-1)+j} = S_{\leq 0} = \emptyset$. Therefore, the condition is satisfied.
- Condition 3: Similarly to the previous item, the condition is trivially satisfied because no such connected component C exists.
- Condition 4: It is trivially satisfied because the range of v_h is $V_0 \cup \emptyset = \emptyset$.
- Condition 5: It is trivially satisfied similarly to the previous item.

On the other hand, when $i \geq 2$, the algorithm arrived at Line 23 with values of i and j given by $i' := i - 1$ and $j' := \ell(i-1) - \ell(i-2)$ (at the moment, the conditions in the statement are satisfied by the assumption of recursion) and proceeded to Line 24, Line 1 (at the moment, the value of i was increased from i' to i, and the value of j was reset to 0), and Line 2; during these steps, the values of variables β_h and γ_h were not changed. Now since

$$\ell(i'-1) + j' = \ell(i-2) + (\ell(i-1) + \ell(i-2)) = \ell(i-1) = \ell(i-1) + j \ ,$$

for Conditions 1 and 2, those conditions at Line 2 with the values i and j are the same as those conditions at Line 23 with the values i' and j'. As mentioned above, the latter conditions were satisfied, therefore the former conditions are satisfied as well. On the other hand, since

$$V_{i'}^{\leq \ell(i'-1)+j'} = V_{i-1}^{\leq \ell(i-1)} = V_{i-1} \ ,$$

$$V_{i'-1} \cup V_{i'}^{\leq \ell(i'-1)+j'} = V_{i-2} \cup V_{i-1} \ ,$$

$$V_i^{\leq \ell(i-1)+j} = \emptyset \ ,$$

$$V_{i-1} \cup V_i^{\leq \ell(i-1)+j} = V_{i-1} \cup \emptyset = V_{i-1} \ ,$$

for Conditions 3, 4, and 5, those conditions at Line 2 with the values i and j are weaker than or equal to those conditions at Line 23 with the values i' and j'. As mentioned above, the latter conditions were

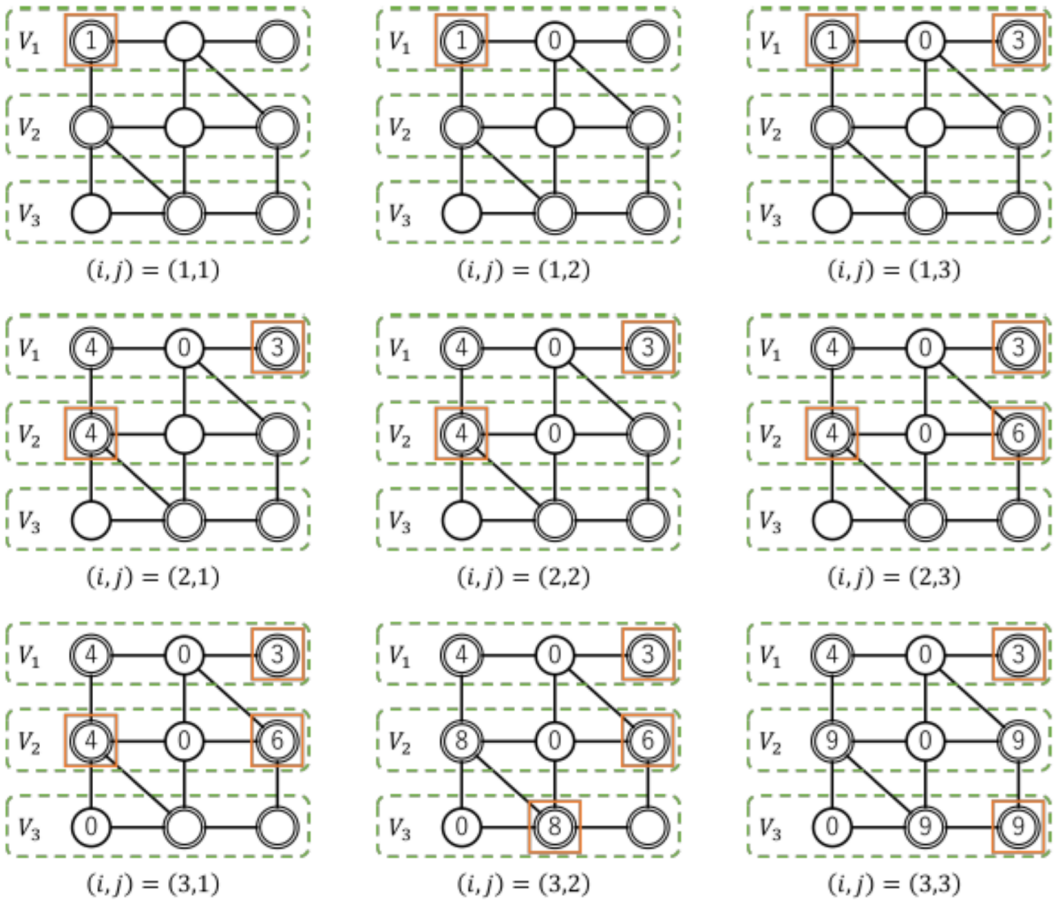


Figure 2: Intermediate states of Algorithm 2 for Example 1 (the value of β_h is written in each vertex, and vertices v_h with $\gamma_h = 1$ are indicated by squares)

satisfied, therefore the former conditions are satisfied as well. Hence the statement is proved for the case $j = 0$.

From now, we consider the other case $j \geq 1$. The algorithm arrived at Line 2 with the values i and j where the conditions in the statement with the same value of i and the value of j being $j' := j - 1$ were satisfied (by the assumption of recursion), and eventually proceeded to Line 23 with the same values i and j. Now the conditions mentioned above are as follows (where $h_0 := \ell(i-1) + j$):

1' We have $\beta_h \leq h_0 - 1$ for any $h \in [1, h_0 - 1]$.

2' Let $h'_1, \ldots, h'_m$ denote the indices $h \in [1, h_0 - 1]$ satisfying the condition $\gamma_h = 1$. Then the set $R := \{v_{h'_1}, \ldots, v_{h'_m}\}$ is a complete set of representatives for connected components of the subset $S_{\leq h_0 - 1} = S_{\leq h_0} \setminus \{v_{h_0}\}$.

3' Let C be any connected component of $S_{\leq h_0} \setminus \{v_{h_0}\}$. If $C \cap (V_{i-1} \cup V_i^{\leq h_0 - 1}) \neq \emptyset$, then the set R in the previous item involves a vertex, say v_h, in the set $C \cap (V_{i-1} \cup V_i^{\leq h_0 - 1})$, which also satisfies that $\beta_h = h$. If moreover $C \cap V_i^{\leq h_0 - 1} \neq \emptyset$, then the set R involves a vertex, say v_h, in the set $C \cap V_i^{\leq h_0 - 1}$, which also satisfies that $\beta_h = h$.

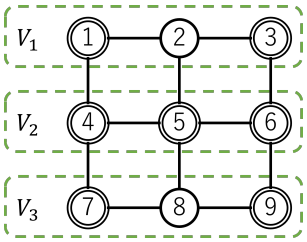


Figure 3: Graph G and subset $S \subseteq V$ for Example 2 (double circles indicate vertices in S and the value inside each vertex indicates the index h for vertex v_h)

4' Suppose that $v_h \in V_{i-1} \cup V_i^{\leq h_0-1}$. Then we have $\beta_h \neq 0$ if and only if $v_h \in S$.

5' Suppose that $v_{h_1}, v_{h_2} \in V_{i-1} \cup V_i^{\leq h_0-1}$ satisfy that $\beta_{h_1} \neq 0$ and $\beta_{h_2} \neq 0$. Then we have $\beta_{h_1} = \beta_{h_2}$ if and only if v_{h_1} and v_{h_2} belong to the same connected component of $S_{\leq h_0-1} = S_{\leq h_0} \setminus \{v_{h_0}\}$.

Now note that during the procedure from Line 2 to Line 23, the value of each variable β_h satisfies one of the following: it is changed to h_0; it is changed to 0; or it is not changed. Therefore, Condition 1 with the value j is implied by Condition 1'. From now, we consider the remaining conditions.

In the case $v_{h_0} \notin S$, during the procedure from Line 2 to Line 23, the values of variables β_h and γ_h for $h \leq h_0 - 1$ are not changed. Moreover, since $v_{h_0} \notin S$, we have $S_{\leq h_0} = S_{\leq h_0} \setminus \{v_{h_0}\}$ and $\beta_{h_0} = \gamma_{h_0} = 0$. Therefore, Conditions 2–5 are implied by Conditions 2'–5', as desired. From now, we consider the other case $v_{h_0} \in S$. Then we have $S_{\leq h_0} = S_{\leq h_0-1} \cup \{v_{h_0}\}$.

First, any connected component, say C, of $S_{\leq h_0} \setminus \{v_{h_0}\}$ that is not adjacent to v_{h_0} is still a connected component of $S_{\leq h_0}$. Moreover, C does not involve any vertex in $N(v_{h_0})$ (since otherwise C must be adjacent to v_{h_0}). Accordingly, let C_0 denote the union of all such connected components C, and let $C_1, \ldots, C_n$ denote the connected components of $S_{\leq h_0} \setminus \{v_{h_0}\}$ that are adjacent to v_{h_0}. Then by the property of the partition $V_1 \sqcup \cdots \sqcup V_L$ of V, each of $C_1, \ldots, C_n$ involves a vertex in $N(v_{h_0})$. In particular, they have non-empty intersection with $V_{i-1} \cup V_i^{\leq h_0-1}$. Moreover, the connected components of $S_{\leq h_0}$ are either the original connected components included in C_0 or $C_1 \cup \cdots \cup C_n \cup \{v_{h_0}\}$.

For Condition 2, only the changes of variables γ_h during the procedure from Line 2 to Line 23 are either that γ_{h_0} is changed from 0 to 1, or that for some $v_{h_1} \in N(v_{h_0})$ with $b_{h_1} = 1$, the value of $\gamma_{\beta_{h_1}}$ is changed to 0. For the latter change, by Condition 4' and the fact $v_{h_1} \in S$, we have $\beta_{h_1} \neq 0$. Since v_{h_1} is adjacent to v_{h_0}, the connected component of $S_{\leq h_0-1}$ involving v_{h_1} is one of $C_1, \ldots, C_n$, say C_ν. Since $v_{h_1} \in C_\nu$, by Conditions 2' and 3', the complete set of representatives R in Conditions 2' involves some vertex in $C_\nu \cap (V_{i-1} \cup V_i^{\leq h_0-1})$, say $v_{h'}$, which also satisfies that $\beta_{h'} = h'$. Now Conditions 4' and 5' imply that $\beta_{h'} = \beta_{h_1}$, therefore $\beta_{h_1} = h'$, hence we have $\gamma_{\beta_{h_1}} = \gamma_{h'} = 1$. This means that the change of $\gamma_{\beta_{h_1}}$ currently considered corresponds to removing the representative vertex of the connected component C_ν from the set R. Moreover, such a change of removing a representative vertex occurs for every connected component $C_1, \ldots, C_n$, since any of those components involves a vertex in $N(v_{h_0})$. Furthermore, at Line 19 the value of γ_{h_0} is set to 1, that is, the vertex v_{h_0} is added to the set R as a representative vertex of the new connected component $C_1 \cup \cdots \cup C_n \cup \{v_{h_0}\}$ of $S_{\leq h_0}$. Hence Condition 2 is satisfied.

For Condition 3, by the argument in the previous paragraph, if the connected component C is one of the original connected components included in C_0, then both of the component C and the corresponding values of γ_h are not changed, and we have $C \cap (V_{i-1} \cup V_i^{\leq h_0}) = C \cap (V_{i-1} \cup V_i^{\leq h_0-1})$ and $C \cap V_i^{\leq h_0} = C \cap V_i^{\leq h_0-1}$ since $v_{h_0} \notin C$. Therefore, Condition 3 is implied by Condition 3'. On the other hand, if C is the new connected component $C_1 \cup \cdots \cup C_n \cup \{v_{h_0}\}$, then the vertex v_{h_0} is added to R as a new representative vertex, and a change $\beta_{h_0} \leftarrow h_0$ occurs at Line 19, therefore Condition 3 is satisfied. Hence Condition 3 is satisfied in any case.

For Condition 4, for each vertex v_h ($h \leq h_0 - 1$), whether or not the value of β_h is 0 is not changed during

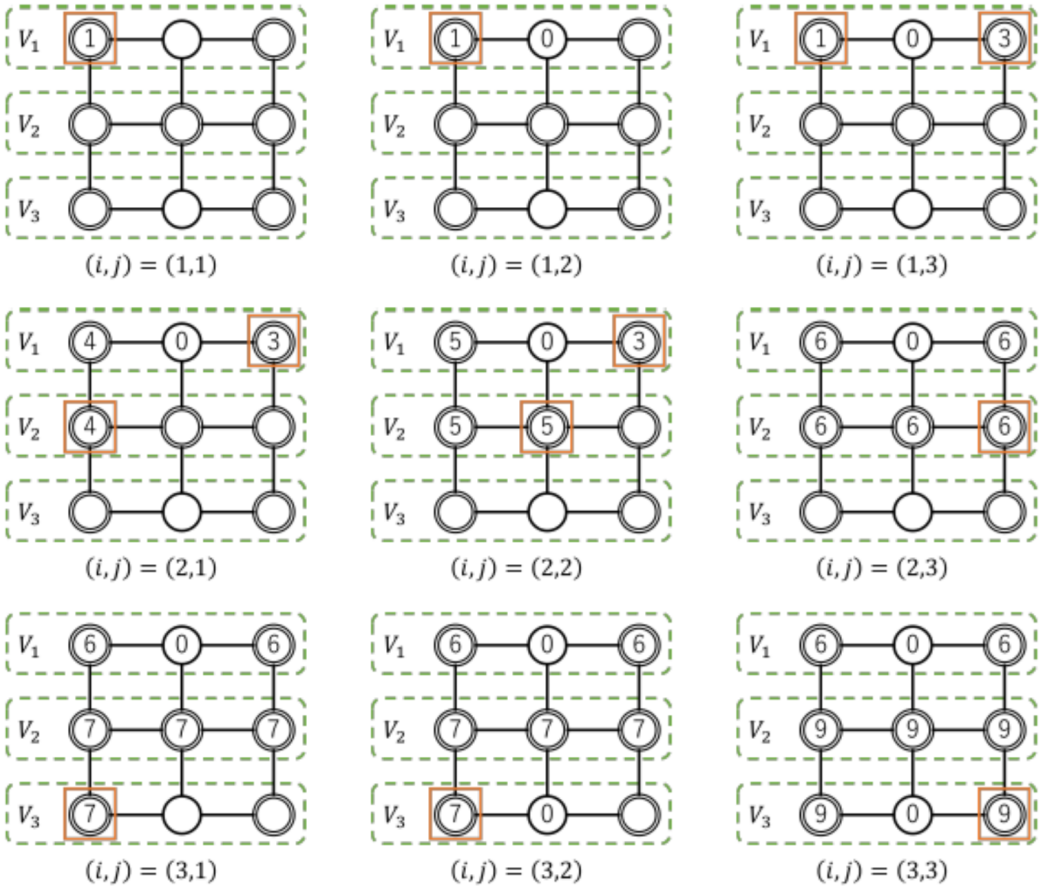


Figure 4: Intermediate states of Algorithm 2 for Example 2 (the value of β_h is written in each vertex, and vertices v_h with $\gamma_h = 1$ are indicated by squares)

the procedure. Indeed, at Line 7 the value of $\beta_{h_2} = \beta_{h_1}$ was originally non-zero by Condition 4', and at Line 13 the value of β_{h_1} was originally non-zero by Condition 4'. On the other hand, for the vertex $v_{h_0} \in S$, the value of β_{h_0} is changed to the non-zero value h_0 at Line 19. Hence Condition 4 is implied by Condition 4'.

For Condition 5, according to Condition 5' and the argument for the case of Condition 2, only the changes for variables β_h are that for each vertex $v_h \in V_{i-1} \cup V_i^{\leq h_0 - 1}$ belonging to $C_1 \cup \cdots \cup C_n$, the value of β_h is changed to the same value of β_{h_0}, i.e., h_0. Since $C_1 \cup \cdots \cup C_n \cup \{v_{h_0}\}$ forms a connected component after the vertex v_{h_0} is added, this means that Condition 5 is implied by Condition 5'.

Hence all of Conditions 1–5 have been verified. This completes the proof. □

In particular, for the case of the final parameter for Lemma 1, i.e., $i = L$ and $j = \ell(i) - \ell(i-1)$, the algorithm proceeds immediately to the output phase after Line 23 with these values of i and j, therefore Condition 2 of Lemma 1 is still satisfied for the output $\gamma_1, \ldots, \gamma_{|V|}$ of the algorithm. Hence we obtain the following result.

Theorem 1. *For the output $\gamma_1, \ldots, \gamma_{|V|}$ of Algorithm 2, let $h'_1, \ldots, h'_m$ denote the indices $h \in [1, |V|]$ satisfying that $\gamma_h = 1$. Then the set $\{v_{h'_1}, \ldots, v_{h'_m}\}$ is a complete set of representatives for connected components of the subset $S \subseteq V$. In particular, the number of connected components of S is equal to the number of bit 1's involved in the output of the algorithm.*

Protocol 3 Card-Based Protocol Counting Connected Components of a Subset

Input Commitments $\mathsf{Com}(b_1), \ldots, \mathsf{Com}(b_{|V|})$ for bits satisfying Eq. (1)

Output Commitments $\mathsf{Com}(\gamma_1), \ldots, \mathsf{Com}(\gamma_{|V|})$ for bits as in the statement of Theorem 1: in particular, $\sum_{h=1}^{|V|} \gamma_h$ equals the number of connected components of S

1: **for** $i = 1, 2, \ldots, L$ **do**
2: **for** $j = 1, 2, \ldots, \ell(i) - \ell(i-1)$ **do**
3: $h_0 \leftarrow \ell(i-1) + j$, $h_{\min} \leftarrow \ell(i-2) + 1$, $\delta_0 \leftarrow \lfloor \log_2 h_0 \rfloor + 1$, $\delta \leftarrow \lfloor \log_2 (h_0 - h_{\min}) \rfloor + 1$
4: **for each** $v_{h_1} \in N(v_{h_0})$ **do**
5: **for each** $v_{h_2} \in (V_{i-1} \cup V_i^{\leq h_0 - 1}) \setminus \{v_{h_1}\}$ **do**
6: $\mathsf{Com}(\mathrm{flg}) \leftarrow \wedge_{\mathsf{Dup}}^{\delta_0+2}\Big(\mathsf{Com}(b_{h_0}), \mathsf{Com}(b_{h_1}), \Big(\neg\big(\mathsf{Com}(\beta_{h_2,\mu}) \oplus_{\mathsf{Dup}} \mathsf{Com}(\beta_{h_1,\mu})\big)\Big)_{\mu=0}^{\delta_0-1}\Big)$
7: Execute $\mathsf{OSub}\big(\mathsf{Com}(\mathrm{flg}); \mathsf{Com}(\beta_{h_2})^{(\delta_0)} \,/\, \mathsf{Com}(h_0)^{(\delta_0)}\big)$
8: **end for**
9: $\mathsf{Com}(\mathrm{flg}) \leftarrow \mathsf{Com}(b_{h_0}) \wedge_{\mathsf{Dup}} \mathsf{Com}(b_{h_1})$
10: $\mathsf{Com}(\beta'_{h_1})^{(\delta)} \leftarrow \mathsf{Com}(\beta_{h_1})^{(\delta_0)} -_{\mathsf{Dup}} h_{\min}$
11: Execute $\mathsf{OApp}\big(\mathsf{Com}(\beta'_{h_1})^{(\delta)}; \mathsf{Com}(\gamma_{h_{\min}}), \ldots, \mathsf{Com}(\gamma_{h_{\min}+2^\delta-1}) \mid \mathsf{op}\big)$
where $\mathsf{op} = \mathsf{OSub}\Big(\mathsf{Com}(\mathrm{flg}); \big(\mathsf{Com}(\gamma_{\beta_{h_1}}), \mathsf{Com}(\beta_{h_1})^{(\delta_0)}\big) \,/\, \big(\mathsf{Com}(0), \mathsf{Com}(h_0)^{(\delta_0)}\big)\Big)$
12: **end for**
13: $\big(\mathsf{Com}(\beta_{h_0})^{(\delta_0)}, \mathsf{Com}(\gamma_{h_0})\big) \leftarrow \mathsf{OSel}_{\mathsf{Dup}}\Big(\mathsf{Com}(b_{h_0}); \big(\mathsf{Com}(0)^{(\delta_0)}, \mathsf{Com}(0)\big), \big(\mathsf{Com}(h_0)^{(\delta_0)}, \mathsf{Com}(1)\big)\Big)$
14: **end for**
15: **end for**
16: Output $\mathsf{Com}(\gamma_h)$ for all $h \in [1, |V|]$

4.2 Our Proposed Card-Based Protocol

In this section, we convert Algorithm 2 in Section 4.1 into a card-based protocol; the result is given as Protocol 3. We give some explanations below.

- Throughout the protocol, we utilize the fact, from Statement 1 of Lemma 1, that the values of variables β_h satisfy that $\beta_h \leq h_0$. In particular, the binary expression of such a value of β_h requires only $\delta_0 := \lfloor \log_2 h_0 \rfloor + 1$ (or fewer) bits; see Line 3. Similarly, for the value δ at Line 3, any non-negative integer up to $h_0 - h_{\min}$ can be expressed by using δ bits.
- For Line 6: We utilize the fact that for the μ-th bits $\beta_{h_1,\mu}$ and $\beta_{h_2,\mu}$ of variables β_{h_1} and β_{h_2}, respectively, the equality check $\beta_{h_2,\mu} \stackrel{?}{=} \beta_{h_1,\mu}$ (where equality is represented by 1 and inequality is represented by 0) can be executed by computing $\neg(\beta_{h_2,\mu} \oplus \beta_{h_1,\mu})$.
- For Lines 10 and 11: Since the values of variables β_h satisfy Lemma 2 below, it suffices to search for the target $\mathsf{Com}(\gamma_{\beta_{h_1}})$ of the oblivious application at Line 11 from the range of indices from $h_{\min}$ to h_0. Accordingly, at Line 10, we normalize the range of indices by shifting by $h_{\min}$ to the lower.
  Note that the explanation above implicitly assumed the case where $\beta_{h_1} \neq 0$ (i.e., $v_{h_1} \in S$); but in fact, if $\beta_{h_1} = 0$ (i.e., $v_{h_1} \notin S$), then the oblivious application at Line 11 has no effect regardless of the computation result at Line 10.

Lemma 2. *Let $i \in [1, L]$ and $j \in [1, \ell(i) - \ell(i-1)]$, and put $h_0 := \ell(i-1) + j$. During the loop in Algorithm 2 with these values of i and j, each $v_h \in V_{i-1} \cup V_i^{\leq h_0}$ satisfies either $\beta_h = 0$ or $\ell(i-2) + 1 \leq \beta_h \leq h_0$.*

Proof. For Algorithm 2, when the value of variable β_h is non-zero, the value was firstly changed to $\beta_h \leftarrow h$ (in the loop with indices satisfying that $\ell(i-1) + j = h$), and after that, the value was only increased and was never decreased. This and the fact that $h \geq \ell(i-2) + 1$ holds for any $v_h \in V_{i-1} \cup V_i^{\leq h_0}$ (note that this

also holds for $i = 1, 2$) imply that $\beta_h \geq \ell(i-2)+1$, while we have $\beta_h \leq h_0$ by Statement 1 of Lemma 1. This completes the proof. □

By these observations, it follows that the output of Protocol 3 is equal to the output of Algorithm 2 (except that the former is concealed as commitments while the latter is not secret).

In order to obtain a commitment of the number $\sum_{h=1}^{|V|} \gamma_h$ of the connected components from the output $\mathsf{Com}(\gamma_1), \ldots, \mathsf{Com}(\gamma_{|V|})$ of Protocol 3, we can use (for example) the addition protocol by Ruangwises and Itoh [11]. But we note that the encoding rule of the result becomes a so-called "♡-scheme" instead of the binary encoding used in our paper. We note also that if a non-committed output is sufficient, we can completely shuffle the commitments $\mathsf{Com}(\gamma_h)$, open them, and count the number of bits γ_h being 1.

On the other hand, as for the connectivity check for S, namely determining whether the number of connected components is exactly 1 or not, we can naively apply computation of any circuit representing "exactly one of the input bits is 1". As another method, we can firstly interpret the sum σ of the γ_h's as "adding the bits γ_h to 0", encode the first '0' in a way that $\mathsf{Com}(0)$ and $\mathsf{Com}(1)$ are used instead of $\boxed{\clubsuit}$ and $\boxed{\heartsuit}$, and then apply the aforementioned addition protocol by Ruangwises and Itoh. The result becomes a sequence of commitments where the σ-th component (counted from the 0-th) is $\mathsf{Com}(1)$ and the other components are $\mathsf{Com}(0)$; therefore we can obtain the desired result by extracting the 1-th component of the sequence. This method can be applied also to the case determining whether the number of connected components is exactly equal to some fixed value different from 1.

4.3 Evaluation of the Number of Cards

In this and the next sections, we evaluate the numbers of cards and shuffles required in our proposed card-based protocol in Section 4.2. Here, in order to make the evaluation of the number of cards simpler, we suppose that after execution of each of Line 6, Line 9, and Line 10, we always perform a CS for the discarded cards and convert them into free cards (to be reused in subsequent steps).

For the number of cards, we evaluate it in a slightly non-tight manner for the sake of simplicity. First, the variables used in Protocol 3 are as follows:

- Input bits $b_1, \ldots, b_{|V|}$; there are $|V|$ commitments in total, using $2|V|$ cards.
- Output bits $\gamma_1, \ldots, \gamma_{|V|}$; there are $|V|$ commitments in total, using $2|V|$ cards.
- Variables β_h for $h \in [1, |V|]$; by Statement 1 of Lemma 1, its value is $\beta_h \leq h_0 \leq |V|$ in the worst case, therefore $\lfloor \log_2 |V| \rfloor + 1$ bits are sufficient to express each β_h. Hence there are at most $|V|(\lfloor \log_2 |V| \rfloor + 1)$ commitments in total, using at most $2|V|(\lfloor \log_2 |V| \rfloor + 1)$ cards.
- Temporary variable flg; there is one commitment, using two cards.
- Temporary variable β'_{h_1}; similarly to the third item, its value is at most $|V|$, therefore there are at most $\lfloor \log_2 |V| \rfloor + 1$ commitments in total, using at most $2\lfloor \log_2 |V| \rfloor + 2$ cards.

Summarizing, the number of cards required in encoding of those variables is at most

$$2|V| + 2|V| + 2|V|(\lfloor \log_2 |V| \rfloor + 1) + 2 + 2\lfloor \log_2 |V| \rfloor + 2 = (2|V| + 2)\lfloor \log_2 |V| \rfloor + 6|V| + 4 \ .$$

Secondly, the number of additional cards used in each line of Protocol 3 is evaluated as follows:

- Line 6: First, $\oplus_{\mathsf{Dup}}$ is executed δ_0 times. Each of them uses four additional cards and leaves two free cards. By reusing the free cards, δ_0 times executions of $\oplus_{\mathsf{Dup}}$ require $4 + (\delta_0 - 1) \cdot 2 = 2\delta_0 + 2$ additional cards and finally leave two free cards.
  After that, $\wedge_{\mathsf{Dup}}^{\delta_0+2}$ is executed. It requires $2(\delta_0 + 2) + 2 = 2\delta_0 + 6$ additional cards including two cards for the output $\mathsf{Com}(\mathrm{flg})$; since the two cards for $\mathsf{Com}(\mathrm{flg})$ are already counted, and there are two free cards left above, the actual number of additional cards is $(2\delta_0 + 6) - 2 - 2 = 2\delta_0 + 2$.
  In total, this procedure requires $(2\delta_0 + 2) + (2\delta_0 + 2) = 4\delta_0 + 4$ additional cards.

- Line 7: An execution of OSub itself requires no additional cards, but we require $2\delta_0$ cards to prepare the commitment $\mathsf{Com}(h_0)^{(\delta_0)}$ to be substituted. Hence this procedure requires $2\delta_0$ additional cards.
- Line 9: An execution of $\wedge_{\mathsf{Dup}} = \wedge^2_{\mathsf{Dup}}$ requires $2 \cdot 2 + 2 = 6$ additional cards, including two cards for the output $\mathsf{Com}(\mathrm{flg})$ which are already counted. Hence, the actual number of additional cards required in this procedure is $6 - 2 = 4$.
- Line 10: An execution of $-_{\mathsf{Dup}}$ for δ_0-bit input requires $4\delta_0$ additional cards, including $2\delta_0$ cards for the output commitment of β'_{h_1} which are already counted. Hence, the actual number of additional cards required in this procedure is $4\delta_0 - 2\delta_0 = 2\delta_0$.
- Line 11: An execution of OApp itself (excluding the internal op) requires two additional cards. On the other hand, for the internal op, an execution of OSub itself requires no additional cards, but we require $2\delta_0 + 2$ cards to prepare the commitments $\mathsf{Com}(h_0)^{(\delta_0)}$ and $\mathsf{Com}(0)$ to be substituted. In total, this procedure requires $2 + (2\delta_0 + 2) = 2\delta_0 + 4$ additional cards.
- Line 13: An execution of $\mathsf{OSel}_{\mathsf{Dup}}$ itself requires two additional cards. On the other hand, we require $2 \cdot (2\delta_0 + 2) = 4\delta_0 + 4$ cards to prepare the input. Among them, $2\delta_0 + 2$ cards are used as the output, which are already counted. Hence in total, this procedure requires $2 + (4\delta_0 + 4) - (2\delta_0 + 2) = 2\delta_0 + 4$ additional cards.

Since all the free or discarded cards are reused as mentioned above, we only need to consider the maximum number of additional cards in these arguments, which is $4\delta_0 + 4$ (for Line 6). Since $\delta_0 = \lfloor \log_2 h_0 \rfloor + 1 \leq \lfloor \log_2 |V| \rfloor + 1$, the number of additional cards required except for encoding of variables is at most

$$4(\lfloor \log_2 |V| \rfloor + 1) + 4 = 4\lfloor \log_2 |V| \rfloor + 8 \ .$$

Summarizing, the number of cards used in our proposed protocol is at most

$$((2|V| + 2)\lfloor \log_2 |V| \rfloor + 6|V| + 4) + (4\lfloor \log_2 |V| \rfloor + 8) = (2|V| + 6)\lfloor \log_2 |V| \rfloor + 6|V| + 12 \ . \tag{2}$$

4.4 Evaluation of the Number of Shuffles

In this section, we evaluate the number of shuffles required in our proposed protocol, in an asymptotic manner for the sake of simplicity. First, the number of shuffles used in each line of Protocol 3 is evaluated as follows:

- Line 6: First, $\oplus_{\mathsf{Dup}}$ is executed δ_0 times. Each of them uses two RBCs, therefore it uses $2\delta_0$ RBCs in total.
  After that, $\wedge^{\delta_0+2}_{\mathsf{Dup}}$ is executed. It requires $\delta_0 + 2$ RBCs, and after the execution, a CS is performed as mentioned at the beginning of Section 4.3.
  In total, this procedure requires $2\delta_0 + (\delta_0 + 2) = 3\delta_0 + 2$ RBCs and 1 CS.
- Line 7: It requires 1 RBC.
- Line 9: An execution of $\wedge_{\mathsf{Dup}} = \wedge^2_{\mathsf{Dup}}$ requires two RBCs, and after the execution, a CS is performed as mentioned at the beginning of Section 4.3. Hence this procedure requires 2 RBCs and 1 CS.
- Line 10: An execution of $-_{\mathsf{Dup}}$ for δ_0-bit input requires $6\delta_0 - 8$ RBCs, and after the execution, a CS is performed as mentioned at the beginning of Section 4.3. Hence this procedure requires $6\delta_0 - 8$ RBCs and 1 CS.
- Line 11: An execution of OApp itself (excluding the internal op) requires 2δ RBCs. On the other hand, for the internal op, an execution of OSub requires one RBC. In total, this procedure requires $2\delta + 1$ RBCs.
- Line 13: It requires 1 RBC.

Secondly, the number of shuffles required in each loop of Protocol 3 is evaluated as follows, where we assume (for the sake of simplicity) upper bounds

$$|N(v_{h_0})| \le N_{\max} \ , \ \ell(i) - \ell(i-1) \le \ell_{\max} \le |V|$$

for the parameters (hence in particular $\ell(1) \le \ell_{\max}$):

- For the loop with vertex v_{h_2}: Lines 6 and 7 require $(3\delta_0 + 2) + 1 = 3\delta_0 + 3$ RBCs and 1 CS. Now we have $\delta_0 = \lfloor \log_2 h_0 \rfloor + 1 \le \log_2 h_0 + 2$ and $h_0 = \ell(i-1) + j \le |V|$, therefore $3\delta_0 + 3 \le 3\log_2 |V| + O(1)$. Since there are $|V_{i-1}| + (j-1) - 1 \le \ell(i-1) - \ell(i-2) + j \le \ell_{\max} + j$ choices of v_{h_2}, this loop requires at most

$$(\ell_{\max} + j)(3\log_2 |V| + O(1)) \tag{3}$$

  RBCs and at most

$$\ell_{\max} + j \tag{4}$$

  CSs.

- For the loop with vertex v_{h_1}: Lines 9, 10, and 11 require $2 + (6\delta_0 - 8) + (2\delta + 1) \le 6\delta_0 + 2\delta$ RBCs and 2 CSs. Now we have $\delta_0 \le \log_2 |V| + O(1)$ as above and

$$\begin{aligned}\delta = \lfloor \log_2(h_0 - h_{\min}) \rfloor + 1 &\le \log_2(\ell(i) - h_{\min}) + 2 \\ &\le \log_2(\ell(i) - \ell(i-2)) + O(1) \le \log_2(2\ell_{\max}) + O(1) = \log_2 \ell_{\max} + O(1) \ ,\end{aligned}$$

  therefore

$$6\delta_0 + 2\delta \le 6\log_2 |V| + 2\log_2 \ell_{\max} + O(1) \ .$$

  By adding the right-hand side to the right-hand side of Eq. (3), the number of RBCs for each v_{h_1} is at most

$$\begin{aligned}&(\ell_{\max} + j)(3\log_2 |V| + O(1)) + (6\log_2 |V| + 2\log_2 \ell_{\max} + O(1)) \\ &= (3\ell_{\max} + O(1))\log_2 |V| + (3\log_2 |V| + O(1))j + O(\ell_{\max}) \ .\end{aligned}$$

  On the other hand, by adding 2 to the right-hand side of Eq. (4), the number of CSs for each v_{h_1} is at most $\ell_{\max} + j + O(1)$. Since there are at most $N_{\max}$ choices of v_{h_1}, this loop requires at most

$$N_{\max}\big((3\ell_{\max} + O(1))\log_2 |V| + (3\log_2 |V| + O(1))j + O(\ell_{\max})\big) \tag{5}$$

  RBCs and at most

$$N_{\max}(\ell_{\max} + j + O(1)) \tag{6}$$

  CSs.

- For the loop with index j: Line 13 requires 1 RBC. By using the value in Eq. (5), the number of RBCs for each j is at most

$$N_{\max}\big((3\ell_{\max} + O(1))\log_2 |V| + (3\log_2 |V| + O(1))j + O(\ell_{\max})\big) \ .$$

  By adding it for $j = 1, \ldots, \ell(i) - \ell(i-1)$, and by using $\ell(i) - \ell(i-1) \le \ell_{\max}$ and $\sum_{j=1}^{\ell(i)-\ell(i-1)} j = {\ell_{\max}}^2/2 + O(\ell_{\max})$, the number of RBCs required by this loop is at most

$$N_{\max}\left(\left(\frac{9}{2}{\ell_{\max}}^2 + O(\ell_{\max})\right)\log_2 |V| + O({\ell_{\max}}^2)\right) \ . \tag{7}$$

  Similarly, by using the value in Eq. (6), the number of CSs required by this loop is at most

$$N_{\max}\left(\frac{3}{2}{\ell_{\max}}^2 + O(\ell_{\max})\right) \ . \tag{8}$$

- For the loop with index i: By adding the values in Eq. (7) for $i = 1, \ldots, L$, the number of RBCs required by this loop is at most

$$LN_{\max}\left(\left(\frac{9}{2}{\ell_{\max}}^2 + O(\ell_{\max})\right)\log_2|V| + O({\ell_{\max}}^2)\right)$$
$$= \frac{9}{2}LN_{\max}{\ell_{\max}}^2\log_2|V| + O(LN_{\max}\ell_{\max}\log_2|V|) + O(LN_{\max}{\ell_{\max}}^2) \ .$$

  On the other hand, by adding the values in Eq. (8) for $i = 1, \ldots, L$, the number of CSs required by this loop is at most

$$LN_{\max}\left(\frac{3}{2}{\ell_{\max}}^2 + O(\ell_{\max})\right) = \frac{3}{2}LN_{\max}{\ell_{\max}}^2 + O(LN_{\max}\ell_{\max}) \ .$$

Summarizing, the total number of shuffles required in our proposed card-based protocol is at most

$$\frac{9}{2}LN_{\max}{\ell_{\max}}^2\log_2|V| + O(LN_{\max}\ell_{\max}\log_2|V|) + O(LN_{\max}{\ell_{\max}}^2) \ . \tag{9}$$

4.5 Comparison with the Existing Method

In this section, we compare the efficiency of our proposed protocol with the existing method of Sasaki and Shinagawa [12] mentioned in Section 1 (where the setting is slightly different from ours). Their method is considered for the case of a $k \times k$ board, which can be modeled as a lattice-shaped graph G with k rows and k columns, hence with $|V| = k^2$ vertices in total. Here we enumerate the vertices of G in a way that the vertices in the i-th row (for $1 \leq i \leq k$) are numbered as $v_{(i-1)k+1}, v_{(i-1)k+2}, \ldots, v_{ik}$ from the leftmost column to the rightmost column. We partition the vertex set into subsets $V_1, \ldots, V_k$ in a way that V_i consists of the vertices in the i-th row. See Example 2 for the case $k = 3$. Therefore, the parameters satisfy the following:

$$L = k \ , \ \ell(i) = ik \ , \ |N(v)| \leq 2 \ .$$

Hence we can choose parameters in Section 4.4 as $N_{\max} := 2$ and $\ell_{\max} := k$. By Eq. (2), the number of cards used in our proposed protocol for this case is at most

$$(2k^2 + 6)\lfloor\log_2 k^2\rfloor + 6k^2 + 12 = (2k^2 + 6)\lfloor 2\log_2 k\rfloor + 6k^2 + 12 \ .$$

By Eq. (9), the number of shuffles used in our proposed protocol for this case is at most

$$9k^3\log_2 k^2 + O(k^2\log_2 k^2) + O(k^3) = 18k^3\log_2 k + O(k^3) \ .$$

On the other hand, for the method in [12], the number of cards is $O(k^2)$, and it is evaluated in [12, Section 3.1.4] that their method requires at most $\frac{22}{3}k^4 + O(k^3)$ shuffles. Hence our proposed protocol improves the order of the number of shuffles from $O(k^4)$ to $O(k^3\log k)$ at the cost of larger order $O(k^2\log k)$ (instead of $O(k^2)$) for the number of cards.

5 Conclusion

In this paper, we proposed for the first time a non-interactive card-based protocol for counting the number of connected components for a subset of the vertex set of a given graph. Our protocol is also applicable to determining whether the input subset is connected in the graph or not. We first constructed a (not card-based) algorithm with non-secret input, and then converted it into a card-based protocol. In comparison to the existing method by [12] (with a slightly different setting), there is a trade-off that the order of the number of shuffles in our protocol is improved from $O(k^4)$ to $O(k^3\log k)$ while the number of cards in our protocol is $O(k^2\log k)$ instead of $O(k^2)$. Future research directions include further improvements of efficiency and applications to card-based zero-knowledge proofs for specific puzzles.

Acknowledgements This work was supported by JSPS KAKENHI Grant Number JP25K14994, Japan, and by Institute of Mathematics for Industry, Joint Usage/Research Center in Kyushu University: FY2023 Short-term Joint Research "Organizing open problems in card-based cryptography through industry-academia collaboration" (2023a020) and FY2024 Short-term Joint Research "Deepening and new frontiers in card-based cryptography through industry-academia collaboration and cooperation in the fields of mathematics and cryptography" (2024a035).

References

[1] den Boer, B.: More Efficient Match-Making and Satisfiability: The Five Card Trick. In: Proceedings of EUROCRYPT 1989, 208–217 (1990)

[2] Dumas, J.-G., Lafourcade, P., Miyahara, D., Mizuki, T., Sasaki, T., Sone, H.: Interactive Physical Zero-Knowledge Proof for Norinori. In: Proceedings of COCOON 2019, 166–177 (2019)

[3] Gradwohl, R., Naor, M., Pinkas, B., Rothblum, G.N.: Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles. In: Proceedings of FUN 2007, 166–182 (2007)

[4] Koch, A., Schrempp, M., Kirsten, M.: Card-Based Cryptography Meets Formal Verification. In: Proceedings of ASIACRYPT 2019 (Part I), 488–517 (2019)

[5] Lafourcade, P., Miyahara, D., Mizuki, T., Robert, L., Sasaki, T., Sone, H.: How to Construct Physical Zero-Knowledge Proofs for Puzzles with a "Single Loop" Condition. Theor. Comput. Sci. 888, 41–55 (2021)

[6] Mizuki, T., Shizuya, H.: A Formalization of Card-Based Cryptographic Protocols via Abstract Machine. Int. J. Inf. Secur. 13(1), 15–23 (2014)

[7] Mizuki, T., Sone, H.: Six-Card Secure AND and Four-Card Secure XOR. In: Proceedings of FAW 2009, 358–369 (2009)

[8] Nishida, T., Hayashi, Y., Mizuki, T., Sone, H.: Card-Based Protocols for Any Boolean Function. In: Proceedings of TAMC 2015, 110–121 (2015)

[9] Robert, L., Miyahara, D., Lafourcade, P., Mizuki, T.: Card-Based ZKP for Connectivity: Applications to Nurikabe, Hitori, and Heyawake. New Gener. Comput. 40(1), 149–171 (2022)

[10] Ruangwises, S., Itoh, T.: Physical ZKP for Connected Spanning Subgraph: Applications to Bridges Puzzle and Other Problems. In: Proceedings of UCNC 2021, 149–163 (2021)

[11] Ruangwises, S., Itoh, T.: Securely Computing the n-Variable Equality Function with $2n$ Cards. Theor. Comput. Sci. 887, 99–110 (2021)

[12] Sasaki, S., Shinagawa, K.: Physical Zero-Knowledge Proof for Sukoro. New Gener. Comput. 42(3), 381–398 (2024)

[13] Sasaki, T., Miyahara, D., Mizuki, T., Sone, H.: Efficient Card-Based Zero-Knowledge Proof for Sudoku. Theor. Comput. Sci. 839, 135–142 (2020)

[14] Sasaki, T., Mizuki, T., Sone, H.: Card-Based Zero-Knowledge Proof for Sudoku. In: Proceedings of FUN 2018, 29:1–29:10 (2018)

[15] Shinagawa, K., Nuida, K.: A Single Shuffle Is Enough for Secure Card-Based Computation of Any Boolean Circuit. Discrete Appl. Math. 289, 248–261 (2021)

[16] Shinagawa, K., Nuida, K.: Card-Based Protocols Imply PSM Protocols. In: Proceedings of STACS 2025, 72:1–72:18 (2025)

[17] Tanaka, K., Mizuki, T.: Two UNO Decks Efficiently Perform Zero-Knowledge Proof for Sudoku. In: Proceedings of FCT 2023, 406–420 (2023)

[18] Tanaka, K., Sasaki, S., Shinagawa, K., Mizuki, T.: Only Two Shuffles Perform Card-Based Zero-Knowledge Proof for Sudoku of Any Size. In: Proceedings of SOSA 2025, 94–107 (2025)