Optimal Fair Computation

Rachid Guerraoui[*,†] Jingjing Wang[*,†]

Abstract

A computation scheme among n parties is *fair* if no party obtains the computation result unless all other $n-1$ parties obtain the same result. A fair computation scheme is *optimistic* if n honest parties can obtain the computation result without resorting to a trusted third party.

We prove, for the first time, a tight lower bound on the message complexity of optimistic fair computation for n parties among which $n-1$ can be malicious in an asynchronous network. We do so by relating the optimal message complexity of optimistic fair computation to the length of the shortest permutation sequence in combinatorics.

1 Introduction

In *fair* computation [1, 2], n parties possess n pieces of information and need to output a function of these n pieces of information (the inputs) *atomically*. Namely, a party obtains the output of the function if and only if the other $n-1$ parties obtain the same output. A prominent example is auctions: after n parties offer a price for some item, they wish to determine the highest price and the winner without ambiguity, e.g., when more than one party claims to win the item. A solution is the fair computation of the n bids (prices).

The difficulty of fair computation stems from the fact that a party might be *malicious* (dishonest) and try to obtain other parties' inputs, twist other parties' outputs, or arbitrarily delay other parties from obtaining an output. Still, honest parties should eventually obtain an output in a fair manner: they should all obtain the function of the n inputs, or all obtain a specific value $\perp$ (denoted *abort* in [1]). In fact, (deterministic) fair computation is in general impossible without a trusted third party [3]. Yet, this third party is not needed in every execution of a (deterministic) fair computation protocol.

Optimistic (deterministic) fair computation stipulates that the third party does not need to be invoked if all n parties are honest [1, 2], [4]. An execution where n honest parties output without invoking the third party is called an *optimistic* execution [1], [4]. Given that cheating is seldom and the third party is considered a bottleneck, optimism is practically appealing. To claim true practicality, however, optimistic executions should be efficient. To be specific, the number of messages exchanged among n honest parties (which compute the function without resorting to the third party) should not be prohibitive. Until the present paper, the optimal number of messages was unknown.

We prove in this paper that $\ell + 2n - 3$ is the optimal number of messages that an optimistic execution of optimistic fair computation may achieve in the presence of $n-1$ potentially malicious parties in an asynchronous network, where ℓ is the length of the shortest sequence that contains all permutations of n symbols as subsequences [5]. Given recent results in combinatorics [6, 7, 8, 9],

[*] École Polytechnique Fédérale de Lausanne, IC, Station 14, CH-1015, Lausanne, Switzerland
[†] firstname.lastname@epfl.ch

the optimal number of messages for optimistic fair computation is 4 for $n = 2$, n^2+1 for $3 \leq n \leq 7$, and asymptotically $\Theta(n^2)$ for $n \geq 8$.[1]

The main idea behind our proof of the $\ell + 2n - 3$ lower-bound is the identification of a *decision propagation* pattern according to which the n parties reach an agreement when any of the parties decides to *stop* the computation. Such ability of a party to stop at any time without jeopardizing *fairness* has been called *timely termination* [1]. It prevents an honest party from waiting forever and is crucial in an asynchronous context. The *decision propagation* pattern is between at least two parties P and Q. To get an intuition, consider an optimistic execution E, let event $E_P =$"P does not receive message m_P" and let event $E_Q =$"Q does not receive message m_Q". An honest party P's stop is a result of E_P. However, a malicious P's stop can impose an honest Q's stop: if when P and Q complete E, $\bar{E}_P$ (the complement of E_P) occurs before $\bar{E}_Q$ and Q does not receive any message between $\bar{E}_P$ and $\bar{E}_Q$, then without m_Q, Q is unable to distinguish whether E_P really occurs or not. An immediate result is that malicious P's decision may *propagate* to Q. To prevent *fairness* from being jeopardized by malicious propagation, in the context of possibly $n-1$ malicious parties, every party should participate in this propagation so that none has a chance to pretend being honest in front of the trusted third party T.

This yields a subsequence of n events E_P (one for each party P) and n messages (whose destinations are the n parties) in E. Clearly, the order of the parties does not matter and therefore, any *permutation* of the n events must occur as a subsequence in E. Hence the relation between the least number of messages of an optimistic execution and ℓ, the length of the shortest sequence that contains all permutations of n symbols as subsequences.

Our lower-bound on the number of messages is tight in the following sense. We present an $(\ell + 2n - 3)$-message optimistic fair computation scheme of some function f given a shortest permutation sequence $\underline{s}$. Our protocol, where the n parties are honest and compute without the third party, consists of three phases: (a) the n parties send *verifiable encryption* [12] of their n inputs respectively, in order to recover those inputs (if needed) in a *non-optimistic* execution, which defines the first n messages; (b) the n parties exchange $\ell - 2$ messages defined by $\underline{s}$; and (c) the n parties exchange the concatenation of the n inputs, which defines the last $n - 1$ messages. The $\ell - 2$ messages $m_1 m_2 \ldots m_{\ell-2}$ in phase (b) have their sources and destinations defined by the sequence $\underline{s} = s_1 s_2 \ldots s_\ell$ as follows. The party represented by symbol s_j is the source of m_{j-1} for $j = 2, \ldots, \ell-1$, and the destination of m_{j-2} for $j = 3, 4, \ldots, \ell$. (s_1 is the source of the last message m_0 of phase (a) and s_2 is the destination of m_0.) When a party resorts to T in a non-optimistic execution, T uses the decision propagation pattern to decide an output. The pattern is the same as in our proof of the lower-bound so that the number of messages in every optimistic execution is minimal.

As we will explain in Section 5, many results have been published on problems related to fair computation [13, 14, 15, 16, 17, 18]. None implies our lower-bound. On the other hand, our $(\ell + 2n - 3)$-message optimistic fair computation scheme can be used to implement fair exchange of certain digital signatures (including Schnorr signatures [19], DSS signatures [20], Fiat-Shamir signatures [21], Ong-Schnorr signatures [22], GQ signatures [23]). Thus, our scheme is also a message-optimal optimistic fair exchange scheme [1]. Moreover, combined with our proof of the lower-bound, this optimistic fair exchange scheme of digital signatures also implies that $\ell + 2n - 3$ is the optimal number of messages for optimistic fair contract signing [16]. Finally, our optimal message complexity may be considered as a first step to the optimal (round) complexity. For

[1]Newey [6] (and then many others [7, 8, 9, 10, 11]) studied the length ℓ of the shortest permutation sequence. Although Newey [6] showed that $\ell = 3$ for $n = 2$, and $\ell = n^2 - 2n + 4$ for $3 \leq n \leq 7$, the exact ℓ for $n \geq 8$ is still considered as an open problem [7, 8]. Up until now, the best upper-bound is $\lceil n^2 - \frac{7}{3}n + \frac{19}{3} \rceil$ for $n \geq 7$ [8], while a lower-bound of ℓ is of the form $n^2 - cn^{7/4} + \epsilon$ for some constant c and some $\epsilon > 0$ [9].

example, the decision propagation pattern is applicable for any optimistic execution, no matter whether the protocol is in a similar form as our optimal protocol or not.

The rest of this paper is organized as follows. Section 2 presents our general model and defines optimistic fair computation. Section 3 presents our lower-bound on the number of messages. Section 4 presents our $(\ell + 2n - 3)$-message optimistic fair computation scheme. Section 5 discusses related work. We defer the details of the proof of our lower-bound to Appendix A and the details of the correctness proof of our message-optimal scheme to Appendix B.

2 Model and Definitions

2.1 The parties

We consider a set Ω of n parties $P_1, P_2, \ldots, P_n$ (sometimes also denoted by P, Q). These parties are all *interactive* in the sense that they can communicate with each other by exchanging messages. All parties are *computationally-bounded* [24] in the sense that they run in time polynomial in some security parameter s.[2]

In addition to the n parties, we also assume a computationally-bounded trusted third party T. T follows the protocol assigned to it. The communication with T is such that when T is communicating with P, Q needs to wait for Q's turn to communicate with T for any two parties $P, Q \in \Omega$.

At most $n-1$ parties can be *malicious*. A malicious party could deviate arbitrarily from the protocol assigned to it. A malicious party could interact arbitrarily with the others as well as T. For example, a malicious party may drop certain messages. A party that *crashes* at some point in time is considered as a malicious party that drops all the messages from that point. Malicious parties may also collude (e.g., to obtain an output for themselves and to prevent an output to an honest party, i.e., to break *fairness*, which is defined later).

Communication channels do not modify, inject, duplicate or lose messages. Every message sent eventually reaches its destination. Any modified, injected, duplicate, or lost message is considered to be due to malicious parties. The delay on message transmission is finite but unbounded. Messages could be reordered. Communication channels are authenticated and secure such as Transport Layer Security [25]. No party can be masqueraded and no message can be eavesdropped.

2.2 Fair computation

We consider the problem of optimistic fair computation in the classical sense of [2, 1]. The problem involves a deterministic function f to be computed by the n parties. Function f is agreed upon by the n parties in advance. We assume that f takes n strings $x_1 \in \{0,1\}^{\ell_1}, x_2 \in \{0,1\}^{\ell_2}, \ldots, x_n \in \{0,1\}^{\ell_n}$ as inputs and returns $z \in \{0,1\}^{\ell_z}$ as its output.

Definition 1 (Computation)**.** A *computation* scheme for f is a collection $(P_1, P_2, \ldots, P_n)$ of n algorithms. The algorithms can carry out two protocols:[3]

[2]Hereafter, when we say that a probability is negligible, we mean that the probability is a *negligible* function $g(s)$ of the security parameter s; i.e., $\forall c \in \mathbb{N}$, $\exists C \in \mathbb{N}$ such that $\forall s > C$, $g(s) < \frac{1}{s^c}$.

[3]We consider deterministic protocols here (for Compute and Stop). In this paper, deterministic protocols consists of two classes of protocols: D1 and D2. In any protocol of D1, each party runs a deterministic algorithm and sends deterministic messages; and we define D2 based on D1: for any protocol π_1 in class D1, we can create a protocol π_2 in class D2 such that π_1 and π_2 are the same except for the message contents of π_2 which can be randomized.

- *Compute*: Each party $P_i, i \in \{1, 2, \ldots, n\}$ is initialized with a local input x_i. If P_i finishes this protocol, P_i returns a local output which can take a value in $\{0,1\}^{\ell_z} \cup \{\perp\}$. If Compute is interrupted by Stop (which we introduce below), Compute returns the same output as Stop.

- *Stop*: P_i invokes Stop when P_i wants to stop the computation. P_i can invoke this protocol at any point in time. P_i obtains P_i's *status* of Compute so far (i.e., the sequence of messages that have arrived at P_i so far) as a local input to Stop. P_i makes a local output which can take a value in $\{0,1\}^{\ell_z} \cup \{\perp\}$.

In the classical definition of fair computation [2], the problem is defined in the *simulatability paradigm* [26], which basically expresses a solution to fair computation in terms of a simulation of the *ideal process*. In what follows, we recall the notion of the ideal process in Definition 2, and then fair computation in Definition 3.

Definition 2 (Ideal process [2])**.** The *ideal* process for *fair* computation of f is a collection $(\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n,\ U)$ of $n+1$ algorithms. Each $\bar{P}_i, i \in \{1, 2, \ldots, n\}$ is initialized with a local input x_i. U is parameterized by f. $\bar{P}_i$ sends message $a_i = x_i$ to U. Messages are delivered instantly. U returns a message m_i to P_i according to Equation (1) as soon as $a_1, a_2, \ldots, a_n$ have arrived at U or one message of $\perp$ has arrived at U. $\bar{P}_i$ outputs whatever U returns to it.

$$\forall i \in \{1, 2, \ldots, n\}, m_i = \begin{cases} f(a_1, a_2, \ldots, a_n) & \text{if } a_1 \neq \perp, a_2 \neq \perp, \ldots, a_n \neq \perp \\ \perp & \text{if } \perp \in \{a_1, a_2, \ldots, a_n\} \end{cases} \tag{1}$$

The process is *ideal* in the sense that among $n+1$ parties, the information of a private input is only exposed to the *universally trusted* U, which we explain in Definition 3.

Definition 3 (Fair computation[4])**.** A computation scheme α solves *fair* computation for f [2] if it satisfies the following properties:

- *Fairness*: for any $e \in \mathbb{N}, 1 \leq e \leq n-1$ and any e malicious parties $P_{d_1}, P_{d_2}, \ldots, P_{d_e}$, for any computationally-bounded algorithm $\mathcal{A}$ that controls the e malicious parties[5], there exists a computationally-bounded algorithm $\mathcal{S}$ that controls $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$[6] such that for any $x_1, x_2, \ldots, x_n$, $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1,\ x_2, \ldots, x_n)$ and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable [27, 28];

- *Termination*: If an honest party P_i invokes Stop, then P_i eventually outputs.

- *Completeness*: $\forall x_1, x_2, \ldots, x_n$, if $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop, then all parties output $z = f(x_1, x_2, \ldots, x_n)$; if $P_1, P_2, \ldots, P_n$ are honest and some invokes Stop, then either all parties output $z = f(x_1, x_2, \ldots,\ x_n)$, or all parties output $\perp$.

- *Non-triviality*: There is at least one execution in which $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop.

[4]The original definition in [2] is ambiguous when all parties are honest: (1) if an algorithm $\mathcal{A}$ delays every message, then to ensure termination, every honest party should output $\perp$ at some point in time. However, for every computationally-bounded algorithm $\mathcal{S}$, the first n elements of the joint output $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are the same: $z = f(x_1, x_2, \ldots, x_n)$. Then by the original definition, all honest parties $P_1, P_2, \ldots, P_n$ output z, except with negligible probability, which yields a contradiction; and (2) if in a protocol, all parties send no message and only outputs $\perp$, then this protocol also matches the ideal process, which however is a trivial protocol.

[5]$\mathcal{A}$ also plays the role of the asynchronous network as defined in Section 2.1.

[6]In the ideal process, $\mathcal{S}$ sees $x_{d_1}, x_{d_2}, \ldots, x_{d_e}$, may change $a_{d_1}, a_{d_2}, \ldots, a_{d_e}$ and also sees $m_{d_1}, m_{d_2}, \ldots, m_{d_e}$ but $\mathcal{S}$ cannot see other messages from or to U, or U's internal state (which makes U *universally trusted*.

Assumptions and notations:

- W.l.o.g., $P_{d_1}, P_{d_2}, \ldots, P_{d_e}$ output nothing but $\mathcal{A}$ may output arbitrarily, [7] and similarly, $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$ output nothing but $\mathcal{S}$ may output arbitrarily; and
- $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ denotes the joint output of $P_1, P_2, \ldots, P_n, \mathcal{A}$ when running α for inputs $x_1, x_2, \ldots, x_n$, and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ denotes the joint output of $\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n, \mathcal{S}$ when running the ideal process for inputs $x_1, x_2, \ldots, x_n$.

Definition 4 (Optimistic fair computation)**.** A fair computation scheme is *optimistic* [1] if it satisfies the following property.

- *Optimism*: $\forall x_1, x_2, \ldots, x_n$, if $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop, then all parties output $z = f(x_1, x_2, \ldots, x_n)$ without interacting with T.

When $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop, $P_1, P_2, \ldots, P_n$ carry out Compute only. Thus, an optimistic execution is an execution of Compute, where every party finishes all communication steps of Compute and outputs.

We focus on the class $\mathcal{C}$ of function f such that for any $x_1 \in \{0,1\}^{\ell_1}, x_2 \in \{0,1\}^{\ell_2}, \ldots, x_n \in \{0,1\}^{\ell_n}$, no computationally-bounded algorithm is able to output $f(x_1, x_2, \ldots, x_n)$ using only $n-1$ out of the n strings, except with negligible probability.[8] For a function f in the complement of $\mathcal{C}$, a protocol that solves optimistic fair computation can still be vulnerable to the following attack: a subset of parties colludes, leaves with the evaluation of f immediately but an honest party outputs $\perp$. In the literature [29, 30], fair protocols for the complement of $\mathcal{C}$ are considered, but they ensure fairness different from Definition 2 and Definition 3, and are not the focus here. We also assume that T does not have prior knowledge of $x_1, x_2, \ldots, x_n$, and therefore no computationally-bounded algorithm, even with the help of T, is able to compute z from any $n-1$ out of the n inputs of $P_1, P_2, \ldots, P_n$. We call this assumption *no prior knowledge of T*.

3 Lower Bound

In this section, we prove our lower-bound on the number of messages exchanged during an optimistic execution of optimistic fair computation. Recall that we consider those functions that cannot be evaluated by only a subset of n parties, e.g., we do not consider constant functions. In addition, a scheme (or the Compute protocol of a scheme) which sends no message, invokes Stop and outputs $\perp$ only is excluded by the *non-triviality* property (Definition 3). Thus the lower-bound is non-zero.

In Theorem 1, we express our lower bound in terms of n and ℓ, the length of the shortest sequence that contains all permutations of n symbols as subsequences.

Theorem 1 (Message complexity)**.** *For any function $f \in \mathcal{C}$, for any optimistic fair computation scheme for f (for n parties, among which $n-1$ can be malicious), the n parties exchange at least $\ell + 2n - 3$ messages in every optimistic execution.*

Proof sketch. (The full proof is in Appendix A.)

To prove Theorem 1, we view every optimistic execution E as a sequence of messages ordered according to when they reach their destinations respectively. We first pinpoint two necessary

[7] The assumption that a malicious party outputs nothing is for definition only. In practice, a malicious party may output arbitrarily.

[8] For example, $f = x_1 \cdot x_2 \cdots x_n$ is not in $\mathcal{C}$ (since if one of the values is 0, the output is 0 with probability 1) while $f = x_1 + x_2 + \cdots + x_n$ is.

messages in E, and then we show that between these two messages, there must exist certain patterns of messages.

Intuitively, when starting E, no party knows anything about other parties' inputs; there is a border-line message m_1^* such that, after m_1^* reaches its destination, one and only one party knows something about all the other parties' inputs. If any honest party $P_i \in \Omega$ stops before m_1^* arrives at its destination, then P_i is unable to output $z = f(x_1, x_2, \ldots, x_n)$ with non-negligible probability by *no prior knowledge of* T.

By the end of E, every party receives sufficient messages to compute z (by the *optimism* property); there is another border-line message m_2^* such that, after m_2^* reaches its destination, one and only one party has sufficient messages to compute z. If any honest party P_i stops after m_2^* arrives at its destination, P_i outputs z by the *completeness* property (e.g., with the help of T, which might not be the only way and is not necessarily so for the proof). Figure 1a illustrates the two messages.

(a) P_i outputs $\perp$ if P_i stops before m_1^*; and z if P_i stops after m_2^*

(b) P_i decides the same value v after P_j if P_i stops between m_1^* and m_2^*.

Figure 1: The output of P_i if P_i stops at some point in execution E

What P_i should output if it stops between m_1^* and m_2^* requires a closer look. Suppose that when P_i wants to stop, P_i has not received some message m_i. (We clarify some terminology here. When we say that P_i has not received or does not receive some message m_i, we mean that P_i has not received m_i but received every message with destination P_i before m_i in E. The same for other parties.) When P_i wants to stop, either no other party has decided an output (and then P_i can easily decide), or some party $P_j \in \Omega, j \neq i$ has decided. If P_j claims that it has not received message m_j and m_i is the first message with destination P_i after m_j in E, , then P_i must adopt P_j's decision, or in other words, P_j's decision *propagates* to P_i. Because $n-1$ parties can be malicious, P_i is unable to distinguish whether P_j's claim is honest or not and then P_i has to decide the same output as P_j (except with negligible probability) by the *fairness* property. Figure 1b illustrates this agreement.

This agreement between two parties induces a *decision propagation* pattern, which gives rise to a certain pattern of messages in E: after a message m_j with destination P_j, there must exist a message m_i with destination P_i so that P_j could enforce P_i on the same output if (a) P_j does not receive m_j, (b) P_j invokes Stop and outputs $\perp$, and (c) P_i does not receive m_i and invokes Stop.

We use this decision propagation pattern to build the following scenario. Suppose one party P_1 stops before m_1^* arrives at its destination and then the other $n-1$ parties stop following the decision propagation pattern above: for $k = 1$, we denote by m_1 the message which P_1 has not received when P_1 stops; then for $k = 2, 3, \ldots, n$, if there is a message m_k in E that is the first message with destination P_k between m_{k-1} and m_2^* , then P_k stops when P_k has not received m_k, and if not, P_k stops after m_2^* arrives at its destination.

Clearly, if the pattern of the n messages whose destinations are $P_1, P_2, \ldots, P_n$ does not exist between m_1^* and m_2^* in E, then P_n would output z by the property of m_2^*. However, P_1, as well as other parties $P_2, P_3, \ldots, P_k$ for which messages $m_2, m_3, \ldots, m_k$ exist, would output $\perp$ by the property of m_1^* and decision propagation. This would violate the *fairness* property. Therefore, the pattern of the n messages whose destinations are n parties, or in fact any permutation of the n parties must exist as a subsequence of E between between m_1^* and m_2^*.

Thus, the number of messages between m_1^* and m_2^* (inclusive) of E is at least ℓ. In the meantime, in E, before m_1^*, there are at least $n-1$ messages to meet the definition of m_1^* and after m_2^*, there are at least $n-2$ messages to meet the definition of m_2^*. We add together the minimum numbers of messages before m_1^*, after m_2^* and between m_1^* and m_2^*, and then have $\ell + 2n - 3$ as the final minimum number of messages during every optimistic execution.

4 An Optimal Protocol

To prove that $\ell + 2n - 3$ is a tight lower bound, we describe in this section an $(\ell + 2n - 3)$-message optimistic fair computation scheme for the function that implements fair exchange of some items. This shows that the optimal message complexity can be achieved for some optimistic fair computation scheme.

Our optimal protocol relies on a publicly verifiable *transcript*. I.e., each destination can verify in an execution whether previous messages have arrived at their destinations correctly. This is realized by adding digital signatures [24, 31]. To help T recover the n inputs (if necessary) when some party invokes Stop, the n parties exchange *verifiable encryption* [12] of the n inputs in the protocol that computes without the third party. Section 4.1 recalls the basics of digital signatures and verifiable encryption, before describing our optimal protocol.

4.1 Preliminaries

We denote a digital signature on message m by $\sigma = Sig_{sk}(m)$, and the verification algorithm by $Ver_{pk}(\sigma, m)$, where pk is a public key and sk is the corresponding secret key. Sometimes we denote the signature of party $P_i, i \in \{1, 2, \ldots, n\}$ simply by $Sig_i(m)$.

A digital signature scheme is secure if no adversary is able to forge a signature even after seeing polynomially many valid signatures. See [24, 31] for a discussion on digital signature schemes and their levels of security.

A verifiable encryption scheme is a recovery algorithm D and a two-party protocol between prover P and verifier V [12]. To run the two-party protocol, P and V's common inputs are public key vk, public value x, condition κ and binary relation R; P takes witness w as an extra input. At the end of the protocol, if $(x, w) \notin R$, V rejects and outputs $\perp$; if V accepts, then V obtains string α such that $D(sk, \kappa, \alpha) = w$ and $(x, w) \in R$.

We denote an instance of verifiable encryption by $VE(vk, \kappa, w, x, R)$. Roughly speaking, a verifiable encryption scheme is secure if no malicious verifier is able to learn w without sk and no malicious prover is able to make V accept $\hat{\alpha}$ which gives $\hat{w}$ by D but $(x, \hat{w}) \notin R$, except with negligible probability. See [12] for a formal definition of security for verifiable encryption schemes. A prominent example of verifiable encryption is Asokan et al.'s non-interactive constructions of verifiable encryption for a list of digital signature schemes, which includes Schnorr signatures, DSS signatures, Fiat-Shamir signatures, Ong-Schnorr signatures and GQ signatures [1].

4.2 Protocol description

In this section, we prove that the lower-bound of $\ell + 2n - 3$ messages is tight (Theorem 2) and we show the tightness in a constructive way.

Theorem 2. *There exists an optimistic fair computation scheme for some function f where n honest parties can evaluate f after they exchange exactly $\ell + 2n - 3$ messages without resorting to T (i.e., in every optimistic execution).*

Algorithm 1 Compute π

Require: a sequence $\underline{i}$ of length l that contains all the permutations of $\{1, 2, \ldots, n\}$

Ensure: $(l + 2n - 3)$-message Compute π

1: Build sequence $\underline{j}$:

$$j_1, j_2, \ldots, j_{n-2}, \underline{i}, j_{n+l-1}, j_{n+l}, \ldots, j_{l+2n-3}$$

where (a) $j_1, j_2, \ldots, j_{n-2}, i_1$ are $n-1$ different symbols; and (b) $i_l, j_{n+l-1}, j_{n+l}, \ldots, j_{l+2n-3}$ are n different symbols.

2: Set $j_0 = \{1, 2, \ldots, n\} \backslash \{i_1, j_1, j_2, \ldots, j_{n-2}\}$.

3: In π, $P_{j_{k-1}}$ sends a message m_{k-1} to P_{j_k} upon receiving m_{k-2} for $k = 1, 2, \ldots, l+2n-3$ (except P_{j_0} who sends $m_0 = VE_{j_0}$ upon initialization) where

$$m_{k-1} = \begin{cases} m_{k-2}||VE_{j_{k-1}}||Sig_{j_{k-1}}(m_{k-2}||VE_{j_{k-1}}) & 2 \leq k \leq n \\ m_{k-2}||Sig_{j_{k-1}}(m_{k-2}) & n+1 \leq k \leq end(j_{k-1}) \\ m_{k-2}||x_{j_{k-1}}||Sig_{j_{k-1}}(m_{k-2}||x_{j_{k-1}}) & end(j_{k-1})+1 \leq k \leq l+n-2 \\ (x_1, x_2, \ldots, x_n) & l+n-1 \leq k \leq l+2n-3 \end{cases} \tag{2}$$

and

$$VE_{j_{k-1}} = VE(vk_T, \kappa, x_{j_{k-1}}, a_{j_{k-1}}, R_{j_{k-1}});$$

$$\kappa = (a_1, R_1), (a_2, R_2), \ldots, (a_n, R_n), \text{ which identifies the intended } x_1, x_2, \ldots, x_n;$$

$$end(j_{k-1}) = \max_{K \in \{1,2,\ldots,l\}} \{K | i_K = j_{k-1}\} + n - 2$$

4: $P_1, P_2, \ldots, P_n$ output $z = (x_1, x_2, \ldots, x_n)$.

We build our protocol with Compute π (Algorithm 1) and Stop μ (Algorithm 2) given *any* sequence that contains all the permutations of $\{1, 2, \ldots, n\}$. Let l be the length of the sequence. We then show in Theorem 3 that our protocol is an $(l+2n-3)$-message optimistic fair computation scheme for the following function:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} (x_1, x_2, \ldots, x_n) & (a_i, x_i) \in R_i \text{ for } i = 1, 2, \ldots, n \\ \perp & \text{otherwise} \end{cases} \tag{3}$$

where $R_1, R_2, \ldots, R_n$ are n relations that allow non-interactive construction of verifiable encryption and $a_1, a_2, \ldots, a_n$ are n public values.[9] $R_1, R_2, \ldots, R_n, a_1, a_2, \ldots, a_n$ are included in the public description of f.

The one-time setup of the protocol is not included in Algorithm 1 and Algorithm 2. Before π and μ are carried out, a one-time setup (a) distributes necessary keys: T's public key vk_T and secret key sk_T, n parties' public and secret keys correctly; (b) distributes the public description of f correctly; and (c) executes the one-time setup of the verifiable encryption. (If implemented, a trusted party Certificate Authority [32] can do this one-time setup.)

Some remarks on μ are in order: (a) as each part of the request message is publicly verifiable, T is able to efficiently verify whether a party P's request and P's claim are consistent by following Equation (2); and (b) P may invoke Stop at any point in time[10], e.g., when a message received by

[9] We also assume that for $i \in \{1, 2, \ldots, n\}$, given a_i, any computationally-bounded algorithm outputs x_i with negligible probability, and given (a_i, x_i) such that $(a_i, x_i) \in R_i$, any computationally-bounded algorithm outputs $y_i, y_i \neq x_i$ such that $(a_i, y_i) \in R_i$ with negligible probability.

[10] If messages are delivered instantly, P does not invoke Stop.

Algorithm 2 Stop μ

Require: sequence $\underline{j}$ of length $l + 2n - 3$ built for π

Ensure: Stop μ that accompanies π

1. For any $k \in \{0, 1 \ldots, l+2n-3\}$, P_{j_k} invokes μ when P_{j_k} wants to stop in π; otherwise, if π has not started, the n parties output $\perp$, or if π has finished, the n parties output $(x_1, x_2, \ldots, x_n)$.
2. For $k = 0$, when invoking μ, if P_{j_k} has not sent m_k, P_{j_k} quietly leaves π and μ and outputs $\perp$.
3. For $1 \leq k \leq n-1$, when invoking μ, if P_{j_k} has not received m_{k-1} correctly, P_{j_k} quietly leaves π and μ and outputs $\perp$.
4. For $n \leq k \leq l+2n-3$, let $I_k = \{index | j_{index} = j_k, index \in \{1, 2, \ldots, k-1\}\}$, let $last_k = \max I_k$ when $I_k \neq \emptyset$ and let $last_k = 0$ when $I_k = \emptyset$, and define m_{-1} as an empty string. Then, for $n \leq k \leq l + 2n - 3$, when invoking μ, if P_{j_k} has not received m_{k-1} correctly and has received $m_{last_k - 1}$, then P_{j_k} sends to T message $req_k = m_{last_k}$. By sending req_k, P_{j_k} claims that P_{j_k} does not receive m_{k-1}.
5. T verifies that req_k is consistent with P_{j_k}'s claim; and T calculates response

$$resp = \begin{cases} \text{``aborted''} & \text{if } req_k \text{ and } P_{j_k}\text{'s claim are not consistent} \\ & \text{or } P_{j_k} \text{ has sent a request before} \\ z = (x_1, x_2, \ldots, x_n) & \text{else if variable } z \text{ (which is initialized to } \perp\text{) is not } \perp \\ \text{``aborted''} & \text{else if } req_k \text{ does not contain } VE_1, VE_2, \ldots, VE_n \\ z \leftarrow (x_1, x_2, \ldots, x_n) & \text{else if } k > \min_{index \in \{progress+1, \ldots, l+2n-3\}} \{index | j_{index} = j_k\} \\ & \text{and } x_i \leftarrow D(sk_T, \kappa, VE_i) \text{ for } i = 1, 2, \ldots, n \\ z \leftarrow (x_1, x_2, \ldots, x_n) & \text{else if } k \geq l + n - 1 \\ & \text{and } x_i \leftarrow D(sk_T, \kappa, VE_i) \text{ for } i = 1, 2, \ldots, n \\ \text{``aborted''} & \text{otherwise} \end{cases}$$

   T updates $progress$ (which is initialized to 0) to k if $k > progress$, req_k and P_{j_k}'s claim are consistent and P_{j_k} has not sent a request before. T then sends $resp$ to P_{j_k}.
6. P_{j_k} outputs $\perp$ if $resp =$"aborted"; and P_{j_k} outputs z if $resp = z$.

P in π is incorrect, or when P is impatient while waiting for some message; our protocol allows every party to define their own strategy of invoking Stop, independent of the other $n-1$ parties.

We prove that this protocol (consisting of π and μ), given a shortest permutation sequence, is an $(\ell + 2n - 3)$-message optimistic fair computation scheme (of which the proof is deferred to Appendix B). This implies Theorem 2. Combined with Theorem 1, $\ell + 2n - 3$ is thus a tight lower-bound on the number of messages for optimistic fair computation.

Theorem 3. *Given a sequence $\underline{i}$ of length l that contains all the permutations of $\{1, 2, \ldots, n\}$, the protocol consisting of π and μ is an $(l + 2n - 3)$-message optimistic fair computation scheme for function f in Equation (3) in an asynchronous network with $n - 1$ potentially malicious parties.*

In fact, function f implements fair exchange among n parties for items $x_1, x_2, \ldots, x_n$ that satisfy relations $R_1, R_2, \ldots, R_n$. Then Algorithm 1 and Algorithm 2 form a compiler that can transform a shortest permutation sequence into an $(\ell + 2n - 3)$-message optimistic fair exchange scheme. An application is a message-optimal optimistic fair exchange scheme of digital signatures [1].[11]

[11] In the application of fair exchange of digital signatures, R_i is an homomorphism θ depending on the given digital

5 Related Work

5.1 Optimistic fair computation

Cachin and Camenisch [2] formalized optimistic fair computation for two parties and a third party T (that can also be malicious). Asokan et al. [1] formalized optimistic fair exchange of digital signatures between two parties and T (where T is honest). In this paper, we assume that T is honest. We briefly compare here the two definitions above. Cachin and Camenisch [2] formalized fair computation using the *simulatability paradigm* [26], while Asokan et al. [1] formalized fair exchange through games [24]. As the former can provide stronger security guarantee, we follow the definition of fair computation in [2]. Both formalizations consider the *termination* property in an asynchronous setting. We model this property using Stop, which is equivalent to the signal of termination in [1]. Asokan et al. [1] also defined the *completeness* property regarding the case where all parties are honest, while there is an ambiguity regarding this case in [2]. We adapt the definition of the *completeness* property from [1]. The *optimism* property was defined differently in [1] and [2]. In [2], the trusted party does not communicate with the n parties when n parties are honest and messages are delivered instantly, whereas in [1], the trusted party does not communicate with the n parties, when n parties are honest but the asynchronous network is allowed to deliver messages arbitrarily. We adopt the *optimism* property from [1], as it provides a stronger guarantee. Following Asokan et al.'s work [1], Küpçü and Lysyanskaya [33] defined *optimism* similarly in games.

In addition, we include the *non-triviality* property to rule out trivial protocols that send no message and abort all the time. (Our proof of the lower-bound is based on the existence of at least one optimistic execution guaranteed by *non-triviality* and *optimism*, but our fair computation scheme, on the other hand, allows arbitrarily many optimistic executions.)

5.2 Optimistic fair exchange

For two parties, Asokan et al. [1] proposed a 4-message optimistic fair exchange scheme that ensures termination. Since $\ell = 3$ for two parties, our Theorem 1 shows that the 4-message fair exchange scheme is optimal for two parties. This also implies that a 3-message fair exchange scheme does not meet all of the required properties. For example, the optimistic fair exchange scheme proposed in [4] was criticized by Asokan et al. [1] as not ensuring *termination*. Another example is Ateniese's 3-message optimistic fair exchange scheme [34], which also does not ensure termination as noted by the author himself [34]. A recent follow-up work [35] has the same drawback.

To the best of our knowledge, up to this paper (and our our fair computation scheme), no message-optimal optimistic fair exchange or optimistic fair computation scheme among n parties for an arbitrary n (with $n-1$ potentially malicious parties) has been proposed.

5.3 Optimal optimistic schemes

We explain here the relation between the optimal efficiency of optimistic schemes of related problems and our optimal message efficiency. Pfitzmann, Schunter, and Waidner (PSW) [16] determined the optimal efficiency of fair two-party contract signing, Schunter [17] determined the optimal efficiency of fair two-party certified email, whereas Dashti [18] determined the optimal efficiency of two-party fair exchange in the crash-recovery model with no amnesia [36]. None of these results implies our Theorem 1, even only for $n = 2$. For PSW's result as well as Schunter's result, this is because there

signature scheme [1], and each of the first n messages of π is appended with an image of θ such that its pre-image produces a correct signature.

is no reduction of the problem of fair computation to the problem of fair contract signing[12] or fair certified email; for Dashti's result, this is because our model can be considered as the Byzantine failure model [36], and is thus stronger than the model considered by Dashti. Our proof of the lower-bound, together with our message-optimal scheme, can be applied to prove that $\ell + 2n - 3$ is the optimal message efficiency of fair n-party contract signing in the model of PSW. The special case where $n = 2$ can be used to prove PSW's result, while PSW's proof was, unfortunately, flawed.

Draper-Gil et al. [37] determined the minimal message complexity of contract signing schemes with *weak fairness* on four topologies. Weak fairness implies that the honest parties might have different outputs as long as they can prove their honest behavior. On the contrary, our optimal message efficiency $\ell + 2n - 3$ applies to any topology, and employs a stronger fairness definition than [37]. Thus their result does not imply our Theorem 1 and vice versa.

5.4 The shortest permutation sequence

Mauw, Radomirović and Dashti (MRD) [13] proved that the optimal number of messages of *totally-ordered* fair contract signing schemes[13] falls between $\ell + n - 1$ and $\ell + 2n - 3$. Later, Mauw and Radomirović (MR) [15] generalized the result of MRD to *DAG-ordered* fair contract signing schemes[14]. Both [13] and [15] considered fair contract signing as fair exchange of digital signatures. They use a model different from PSW, and fall within the coverage of our Theorem 1. Neither MRD's result nor MR's result implies our Theorem 1. Neither allows arbitrarily interleaved messages as our Theorem 1; instead, they assume that communication steps are either totally ordered or ordered following a directed acyclic graph (DAG). In addition, both results [13, 15] propose a range of the optimal efficiency for fair exchange, instead of a concrete lower-bound for fair computation in general (as does our Theorem 1).

It is important to note that our Theorem 1 is not a generalization of MRD's result nor of MR's result. What MRD or MR count are the messages sent from some signer. This makes the proof difficult to extend: after a message m leaves its source s, due to the asynchronous network, m does not help s's knowledge about other parties' possible states. Thus m should not help s reach an agreement if s wants to stop after sending m, unless the messages after m are defined and ordered in advance. On the contrary, what we count throughout our proof are the messages received (or not) at a destination d, which affects d's stop event. This is the key in our case for not requiring any ordering.

Another crucial concept used by MRD is the idea of an *idealized* protocol. An *idealized* protocol is informally defined as a totally-ordered fair exchange protocol of which the number of messages in an optimistic execution is optimal [13]. (Here a protocol is equivalent as a Compute protocol in our Definition 1. The communication with a third party T is not considered as part of the protocol.) At the *end* phase of the *idealized* protocol, each of the n signers is supposed to send exactly one message [13]. It is not clear yet whether the assumption can be justified or not: the main theorem in [13] relates the end of an *idealized* protocol with part of the shortest permutation sequence; however, (the form of the end of) the shortest permutation sequence is still open for a large n [5]. This also leads to a non-optimal fair exchange protocol in [13] and a non-optimal protocol compiler in [14] which generates a protocol specification of an optimistic fair contract signing scheme given a shortest permutation sequence.[15] Compared with MRD's *idealized* protocol, our proof of Theorem

[12] The main difference is that contract signing outputs a proof which binds a contract agreed in advance while computation usually does not require such binding.

[13] In a *totally-ordered* contract signing scheme, signers execute totally-ordered communication steps; i.e., at any point in time, only one signer has sufficient messages to calculate and send the next message.

[14] In a *DAG-ordered* contract signing scheme, communication steps can be ordered in a directed acyclic graph.

[15] Although [14] proved that the resulting protocol needs at least $\ell + 2n - 3$ messages in an optimistic execution,

1 shows that, at the end of an optimal protocol, each of the n parties may receive exactly one message, and moreover, the end of an optimal protocol is *not* related to the shortest permutation sequence. We believe that this has further implications on the design of correct and efficient fair computation protocols.

Acknowledgements We are very grateful to the second author of [16] for the time devoted to understanding our argument and for his fairplay in recognizing the mistake. This work has been supported in part by the European ERC Grant 339539 - AOC.

References

[1] N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 4, pp. 593–610, 2000.

[2] C. Cachin and J. Camenisch, "Optimistic fair secure computation," in *CRYPTO 2000*, pp. 93–111.

[3] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *STOC '86*, pp. 364–369.

[4] S. Micali, "Simple and fast optimistic protocols for fair electronic exchange," in *PODC 2003*.

[5] D. E. Knuth, "Open problems with a computational flavor, mimeographed notes for a seminar on combinatorics," 1971.

[6] M. C. Newey, "Notes on a problem involving permutations as subsequences." Tech. Rep., 1973.

[7] E. Zălinescu, "Shorter strings containing all k-element permutations," *Information Processing Letters*, vol. 111, no. 12, pp. 605 – 608, 2011.

[8] S. Radomirovic, "A construction of short sequences containing all permutations of a set as subsequences," *The Electronic Journal of Combinatorics*, vol. 19, no. 4, 2012.

[9] D. Kleitman and D. Kwiatkowski, "A lower bound on the length of a sequence containing all permutations as subsequences," *Journal of Combinatorial Theory, Series A*, vol. 21, no. 2, pp. 129 – 136, 1976.

[10] L. Adleman, "Short permutation strings," *Discrete Mathematics*, vol. 10, no. 2, pp. 197 – 200, 1974.

[11] P. Koutas and T. Hu, "Shortest string containing all permutations," *Discrete Mathematics*, vol. 11, no. 2, pp. 125 – 132, 1975.

[12] J. Camenisch and I. Damgärd, "Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes," in *ASIACRYPT 2000*, pp. 331–345.

[13] S. Mauw, S. Radomirovic, and M. Dashti, "Minimal message complexity of asynchronous multi-party contract signing," in *CSF 2009*.

the number of messages exchanged during every optimistic execution is actually strictly larger than $\ell + 2n - 3$ for $n \geq 3$, and is thus not optimal.

[14] B. Kordy and S. Radomirovic, "Constructing optimistic multi-party contract signing protocols," in *CSF 2012*.

[15] S. Mauw and S. Radomirovic, "Generalizing multi-party contract signing," in *POST 2015*.

[16] B. Pfitzmann, M. Schunter, and M. Waidner, "Optimal efficiency of optimistic contract signing," in *PODC '98*, pp. 113–122.

[17] M. Schunter, "Optimistic fair exchange," Ph.D. dissertation, Universität des Saarlandes, 2000, http://scidok.sulb.uni-saarland.de/volltexte/2004/233.

[18] M. T. Dashti, "Efficiency of optimistic fair exchange using trusted devices," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 1, pp. 3:1–3:18, May 2012.

[19] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[20] D. Kravitz, "Digital signature algorithm," 1993, uS Patent 5,231,668.

[21] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO '86*, pp. 186–194.

[22] H. Ong and C. Schnorr, "Fast signature generation with a fiat shamir-like scheme," in *EUROCRYPT '90*, pp. 432–440.

[23] L. Guillou and J.-J. Quisquater, "A "paradoxical" indentity-based signature scheme resulting from zero-knowledge," in *CRYPTO '88*, pp. 216–231.

[24] G. Oded, *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2009.

[25] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[26] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[27] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270 – 299, 1984.

[28] A. C. Yao, "Theory and application of trapdoor functions," in *SFCS '82*, pp. 80–91.

[29] S. D. Gordon and J. Katz, *TCC 2009*, ch. Complete Fairness in Multi-party Computation without an Honest Majority, pp. 19–35.

[30] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell, "Complete fairness in secure two-party computation," *J. ACM.*

[31] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography.* CRC Press, Inc., 1996.

[32] I. 9594-8, "Information technology - open systems interconnection - the directory: Authentication framework," 1995, (equivalent to ITU-T Recommendation X.509, 1993).

[33] A. Küpçü and A. Lysyanskaya, "Usable optimistic fair exchange," *Computer Networks*, vol. 56, no. 1, pp. 50 – 63, 2012.

[34] G. Ateniese, "Efficient verifiable encryption (and fair exchange) of digital signatures," in *CCS '99*, pp. 138–146.

[35] A. M. Alaraj, "Simple and efficient contract signing protocol," *CoRR*, vol. abs/1204.1646, 2012. [Online]. Available: http://arxiv.org/abs/1204.1646

[36] R. Guerraoui and L. Rodrigues, *Introduction to Reliable Distributed Programming.* Springer-Verlag New York, Inc., 2006.

[37] G. Draper-Gil, J.-L. A. Ferrer-Gomila, M. Hinarejos, and J. Zhou, "On the efficiency of multi-party contract signing protocols," in *ISC 2015*, pp. 221–232.

[38] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *CRYPTO '87*, pp. 186–194.

[39] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS '93*, pp. 62–73.

A Appendix. Proof of Theorem 1

We give a detailed proof of Theorem 1. First, we give the *(weak) fairness* property that we use repeatedly in the proof.

Lemma 1. *For any $e \in \mathbb{N}$, any $1 \le e \le n-1$, any e malicious parties and any computationally-bounded algorithm $\mathcal{A}$ that controls the e malicious parties, $\forall x_1, x_2, \ldots, x_n$, any two honest parties $P_i, P_j, i, j, \in \{1, 2, \ldots, n\}$ output the same except with negligible probability.*

We show that this property is implied the *fairness* property in Definition 3. Before proving the property, we give formal definitions and terminologies used in Definition 3 such as computational indistinguishability [27, 28] and negligible function.

Definition 5 (Computationally indistinguishability)**.** If function g is a negligible function of variable s, then $\forall c \in \mathbb{N}$, $\exists C \in \mathbb{N}$ such that $\forall s > C$, $g(s) < \frac{1}{s^c}$.

Let $A = \{A(1^s, a)\}$ be a distribution ensemble, i.e., random variables indexed by 1^s and a. Let $B(1^s, a) = \{B(1^s, a)\}$ be also a distribution ensemble. Then A and B are computationally indistinguishable, if for any computationally-bounded algorithm $\mathcal{D}(1^s, a, w, D)$ that takes some $q = q(s)$ independently identically distributed random variables following the distribution D,

$$|Pr[\mathcal{D}(1^s, a, w, A(1^s, a)) = 1] - Pr[\mathcal{D}(1^s, a, w, B(1^s, a)) = 1]| = negl(s), \forall a, \forall w$$

where $negl(s)$ is a negligible function of s, $q(s)$ is a polynomial of s and the probabilities are taken over the random choices of D as well as q random variables.

In the context of Definition 3, s is the security parameter of the fair computation scheme. Recall that in Definition 3, we say that the joint outputs $O = O_{P_1, P_2, \ldots, P_n, \mathcal{A}}(x_1, x_2, \ldots, x_n)$ and $\bar{O} = O_{\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n, \mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable. This means that for any computationally-bounded algorithm $\mathcal{D}(1^s, a, w, D)$,

$$|Pr[\mathcal{D}(1^s, a, w, O) = 1] - Pr[\mathcal{D}(1^s, a, w, \bar{O}) = 1]| = negl(s), \forall a, \forall w$$

where $a = x_1 || x_2 || \cdots || x_n$, w may be arbitrary auxiliary information and both O and $\bar{O}$ are indexed by 1^s and a.

Proof of Lemma 1. Consider a computationally-bounded algorithm $\mathcal{A}$ that does not control P_i or P_j. I.e., both P_i and P_j are honest as in the statement of Lemma 1. Note that although $\mathcal{A}$ controls the rest of the parties, $\mathcal{A}$ can be an arbitrary algorithm, including running the rest of the parties honestly (and thus the proof applies to any execution as in the statement of Lemma 1.)

Let o_i, o_j be the random variables that represent P_i and P_j's outputs in the joint output O respectively. Suppose that $\mathcal{A}$ controls $e, 1 \le e \le n-1$ malicious parties $P_{d_1}, P_{d_2}, \ldots, P_{d_e}$.

By Definition 3, there exists a computationally-bounded algorithm $\mathcal{S}$ that controls $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots,$ $\bar{P}_{d_e}$ such that O and $\bar{O}$ are computationally indistinguishable. Let $\bar{o}_i$ and $\bar{o}_j$ be the random variables that represent $\bar{P}_i$ and $\bar{P}_j$'s outputs in the joint output $\bar{O}$ respectively. Since $\mathcal{S}$ does not control $\bar{P}_i$ or $\bar{P}_j$, $\bar{o}_i = \bar{o}_j$ with probability 1.

Consider a computationally-bounded algorithm $\mathcal{D}$ that tries to distinguish O and $\bar{O}$ as follows. $\mathcal{D}$ takes one sample from the given distribution D. If in the sample, the ith element and the jth element are the same, then $\mathcal{D}$ outputs 1; if not, $\mathcal{D}$ outputs 0. Then there exists a negligible function $negl(s)$ such that

$$|Pr[\mathcal{D}(1^s, a, w, O) = 1] - Pr[\mathcal{D}(1^s, a, w, \bar{O}) = 1]| = negl(s), \forall a$$

where $a = x_1||x_2||\cdots||x_n$ and w is an empty string.

Since $\bar{o}_i = \bar{o}_j$ with probability 1, $Pr[\mathcal{D}(1^s, a, w, \bar{O}) = 1] = 1$. Let ρ be the probability such that $o_i = o_j$. Then $Pr[\mathcal{D}(1^s, a, w, O) = 1] = \rho$. Thus $\rho = 1 - negl(s)$. I.e., for any algorithm $\mathcal{A}$, any two honest parties $P_i, P_j, i, j, \in \{1, 2, \ldots, n\}$ output the same except with negligible probability.. □

Then, we discuss some essential properties/convention of Stop, which we use later in the proof.

Preliminaries. If P invokes Stop several times, Stop returns the same value as the first time.

P may communicate with T in Stop, but P does not communicate with T in Compute. This is consistent to the *optimism* property.

When P invokes Stop, either P does not send messages to any other party including T and simply terminates, or P communicates with T and then terminates. If P communicates with T, P sends only one stop request. T does not ask any party (including P) for additional messages when computing an output for P. This is due to the atomicity of the communication with T and the *termination* property.

When P communicates with T and then terminates, T sends a response only to P. In the asynchronous network, even if T sends messages to parties other than P, they might receive the messages after they complete Compute or Stop in the worst case. Thus we consider that T does not send messages to other parties.

We say that an optimistic execution E is initialized with $x_1, x_2, \ldots, x_n$, if n parties in E are initialized with $x_1, x_2, \ldots, x_n$. When we discuss any optimistic execution E, E must have been initialized with some n strings. Thus the term E initialized with (some) $x_1, x_2, \ldots, x_n$ does not lose generality.

Sometimes we denote a party by O, P, Q, R, with an abuse of notations on O and R (as their meaning is clear in the context).

Recall the intuition (Section 3) that there are two necessary messages (of every optimistic execution). Here we precisely define the two messages and show their basic properties.

Lemma 2. *For any optimistic execution E, for any two parties P and Q, we say that P contacts Q in E if one of the two properties below holds: (a) P sends m to Q in E; or (b) there exists a party O such that P contacts O and subsequently O contacts Q.*

Then for any optimistic execution E and any $P \in \Omega$, there exists a message m such that before m arrives at its destination, $\exists Q \in \Omega \backslash \{P\}$ such that Q has not contacted P yet and after m arrives at its destination, $\forall Q \in \Omega \backslash \{P\}$, Q has contacted the destination P.

Let t be any status of P before P receives m in E. Then if P invokes Stop with t and no other party has invoked Stop, then Stop returns $\perp$ to P.

Proof. The lemma contains two parts. We first prove the existence of message m.

By contradiction. Suppose that for some optimistic execution E initialized with $x_1, x_2, \ldots, x_n$ and some $P \in \Omega$, after E finishes, $\exists Q \in \Omega \backslash \{P\}$ has not contacted P yet. Then by the *optimism* property, P performs a computationally-bounded algorithm that computes $f(x_1, x_2, \ldots, x_n)$ given $\Omega \backslash \{Q\}$'s inputs. A contradiction.

Second, we prove that if P invokes Stop with t and no other party has invoked Stop, then Stop returns $\perp$ to P. Since no other party has invoked Stop, Stop is only able to return to P a value based on t, P's input and T's input. Let E be initialized with $x_1, x_2, \ldots, x_n$. Since t is P's status in E before P receives m, $\exists Q \in \Omega \backslash \{P\}$ has not contacted P yet and thus t can be constructed given $\Omega \backslash \{Q\}$'s inputs. Since E is an optimistic execution, then if Stop returns a non-$\perp$ value, Stop returns $z = f(x_1, x_2, \ldots, x_n)$. Suppose that Stop returns z to P. Then there is a computationally-bounded algorithm that computes z given $\Omega \backslash \{Q\}$'s inputs and T's inputs, which gives a contradiction. □

Corollary 1. *For any optimistic execution E, there exists message m_1^* such that (a) before m_1^* arrives at its destination, $\forall P \in \Omega$, $\exists Q \in \Omega \backslash \{P\}$ such that Q has not contacted P yet and (b) after m_1^* arrives at its destination, there exists the destination R of m_1^* such that $\forall Q \in \Omega \backslash \{R\}$, Q has contacted R.*

Proof. The correctness follows from Lemma 2. □

Lemma 3. *For any optimistic execution E initialized with $x_1, x_2, \ldots, x_n$, there exists message m_2^* such that (a) before m_2^* arrives at its destination R, no P computes $z = f(x_1, x_2, \ldots, x_n)$ from P's status and P's input (according to the protocol underlying E) and (b) after m_2^* arrives at R, R computes z from R's status and R's input (according to the protocol underlying E).*

In E, before R receives m_2^*, $\forall P \in \Omega \backslash \{R\}$, P has been contacted by Q, $\forall Q \in \Omega \backslash \{P\}$.

Proof. The lemma contains two parts. The existence of message m_2^* follows from the *optimism* property.

We prove the second part by contradiction. Suppose that in E, $\exists O \in \Omega \backslash \{R\}, Q \in \Omega \backslash \{O\}$ such that when R receives m_2^*, O has not been contacted by Q. Consider an execution F that is the same as E for the prefix that ends at the event of m_2^* arriving at its destination (inclusive); in F, after R receives m_2^*, O invokes Stop, and Stop returns before any other party invokes Stop. In F, O is honest. By Lemma 2, O outputs $\perp$. However, an honest party R outputs z, which violates the *completeness* property. A contradiction. □

Corollary 2. *For any optimistic execution E, let m_1^* be defined as in Corollary 1 and let m_2^* be defined as in Lemma 3; then the event of m_1^* arriving at its destination precedes the event of m_2^* arriving at its destination.*

Proof. The correctness follows from Lemma 3 and $n \geq 2$. □

Corollary 3. *For any optimistic execution E, let m_2^* be defined as in Lemma 3 and let R be the destination of m_2^*; then in E, before R receives m_2^*, $\forall P \in \Omega \backslash \{R\}$, P has received at least one message.*

Proof. The correctness follows from Lemma 3 and $n \geq 2$. □

Hereafter, we present more properties about the two messages: m_1^* and m_2^*. They are always defined for any certain optimistic execution E. If it is clear in the context, we omit the re-definition in the statements in the following lemmas.

Lemma 4. *For any optimistic execution E initialized with $x_1, x_2, \ldots, x_n$, let R be the destination of m_2^*; for any $P \in \Omega \backslash \{R\}$, let m be the last message received by P before message m_2^* arrives its destination in E. By Corollary 3, m exists.*

Let t be the status of P in E right after P receives m. Then for any execution $E(P)$ such that $E(P)$ is the same as E for P until P invokes Stop, and P invokes Stop with t after P receives m (and before P's next receipt of some message), Stop returns $z = f(x_1, x_2, \ldots, x_n)$ to P.

Proof. For any $E(P)$, P's behavior is the same as an honest P to the parties in $\Omega \backslash \{P\}$ and T, w.l.o.g., we say that in $E(P)$, P is honest.

Let $\mathcal{M}_P$ be the set of messages sent by P before m_2^* arrives its destination in E. Then the event of P receiving m is the last non-local event in E that might trigger P to send some message in $\mathcal{M}_P$. Due to the arbitrary delay of communication channels and the arbitrary time instant of invoking Stop, there exists such an execution $E(P)$ that P has sent all the messages in $\mathcal{M}_P$ before

P's next receipt of some message and before P invokes Stop with t. For any such execution $E(P)$, the parties in $\Omega\backslash\{P\}$ may continue E without noticing P's invocation of Stop, and then an honest party R outputs z. Therefore, Stop should return z to P; otherwise, as all parties are honest here, this violates the *completeness* property.

Now due to the arbitrary time instant of invoking Stop, it is indistinguishable for T whether P, invoking Stop with t, has sent all the messages in $\mathcal{M}_P$ or not. Therefore, for any $E(P)$, Stop has to return z to P. □

Lemma 5. *For any optimistic execution E and any $k, 2 \leq k \leq n$, w.l.o.g., let $m_1, m_2, \ldots, m_k$ be k messages in E such that (a) the destination of $m_i, 1 \leq i \leq k$ is P_i; (b) $m_{i+1}, 1 \leq i \leq k-1$ is the first message received by P_{i+1} after P_i receives m_i in E. Let $t_i, 1 \leq i \leq k$ be the status of P_i in E right before P_i receives m_i.*

For $1 \leq i \leq k$, define execution $E(P_i)$ such that $E(P_i)$ is the same as E for P_i until P_i invokes Stop; in $E(P_i)$, P_i invokes Stop with t_i right before message m_i arrives at P_i.

Assume that for any $E(P_1)$, if no other party invokes Stop before P_1, then Stop returns $\perp$ to P_1. Then

- *for $k = 1$, for any $E(P_k)$, when P_k invokes Stop, if no other party has invoked Stop, then Stop returns $\perp$ to P_k.*
- *for $k = 2$, for any $E(P_k)$, when P_k invokes Stop, if P_{k-1} has invoked Stop with t_{k-1}, Stop has returned $\perp$ to P_{k-1} and no other party has invoked Stop, then Stop returns $\perp$ to P_k except with negligible probability.*
- *for $3 \leq k \leq n$, for any $E(P_k)$, when P_k invokes Stop if $P_1, P_2, \ldots, P_{k-1}$ have invoked Stop with $t_1, t_2, \ldots, t_{k-1}$ respectively and for $2 \leq i \leq k-1$, P_i invokes Stop after Stop returns to P_{i-1}, and Stop has returned $\perp$ to $P_1, \ldots, P_{k-1}$, and no other party has invoked Stop, then Stop returns $\perp$ to P_k except with negligible probability.*

Proof. Let E be initialized with $x_1, x_2, \ldots, x_n$. We prove the lemma by induction. The base case, for which $k = 1$, is trivial.

Suppose the statement is true for $k-1, 2 \leq k \leq n$. Assume any $E(P_k)$ as an execution such that when P_k invokes Stop, $P_1, \ldots, P_{k-1}$ have invoked Stop with $t_1, \ldots, t_{k-1}$ respectively according to the statement, Stop has returned $P_1, \ldots, P_{k-1}$ $\perp$ and no other party has invoked Stop, and Stop returns r to P_k, where r is a random variable.

For any $E(P_k)$, let $E^*(P_k)$ be an execution that is the same as $E(P_k)$ for $P_1, P_2, \ldots, P_n$ until P_k invokes Stop right before message m_{k-1} arrives at P_{k-1}. If $P_j, \ldots, P_{k-1}$ for some $j, 1 \leq j \leq k-1$ do not invoke Stop before message m_{k-1} arrives at P_{k-1} in $E(P_k)$, let $P_j, \ldots, P_{k-1}$ invoke Stop right before message m_{k-1} arrives at P_{k-1} in the same order with the same status as in $E^*(P_k)$. Also, let P_k invoke Stop after Stop has returned P_{k-1} $\perp$.

Due to the arbitrary delay of communication channels, in both $E(P_k)$ and $E^*(P_k)$, P_k's behavior is the same as an honest P_k to $\Omega\backslash P_k$ and T. Hereafter we say that P_k is honest. Again due to the arbitrary delay of communication channels,, to P_k and T, any $E^*(P_k)$ is indistinguishable from any $E(P_k)$ at the point when P_k invokes Stop. Furthermore, since m_k is the first message received by P_k after P_{k-1} receives m_{k-1} in E, the status of P_k in $E^*(P_k)$ is also t_k. Thus in any $E * (P_k)$, Stop returns r to P_k (where the distribution of r remains the same).

For any $E(P_{k-1})$ and for any $E^*(P_k)$, we define an execution F such that (a) F is the same as $E(P_{k-1})$ for P_{k-1} until P_{k-1} invokes Stop with t_{k-1} right before m_{k-1} arrives at P_{k-1}; (b) F is the same as $E^*(P_k)$ for P_k until P_k invokes Stop with t_k right before m_{k-1} arrives at P_{k-1}; (c) when P_k invokes Stop, $P_1, \ldots, P_{k-1}$ have invoked Stop with $t_1, \ldots, t_{k-1}$ respectively and $P_i, 2 \leq i \leq k-1$

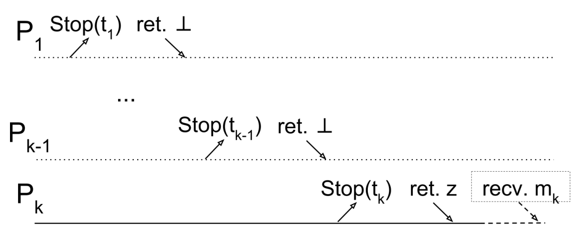
invokes Stop after Stop returns to P_{i-1}, Stop has returned $\perp$ to $P_1, \ldots, P_{k-2}$ and no other party has invoked Stop.

In F, P_{k-1}'s behavior is the same as an honest P_{k-1} to $\Omega \backslash \{P_{k-1}\}$ and T. Hereafter, we say that P_{k-1} is honest in F. If $n = 2$, then $k = 2$ and all parties are honest. Since the statement is true for $k-1$, Stop returns $\perp$ to P_{k-1} in F. Then by the *completeness* property, $r = \perp$ with probability 1. If $n > 2$, since the statement is true for $k-1$, then Stop returns $\perp$ to P_{k-1} except with negligible probability. When Stop returns $\perp$ to P_{k-1}, $E^*(P_k)$ and F are indistinguishable to T and P_k due to the arbitrary delay of communication channels. As a result, Stop returns r to P_k (where the distribution of r remains the same).

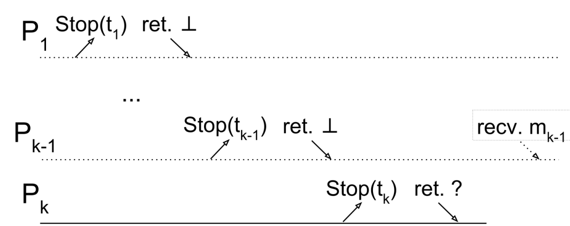
Then by the *(weak) fairness* property, $r = \perp$ except with negligible probability. We can show this by contradiction. Suppose that $r \neq \perp$ with non-negligible probability. We build an algorithm $\mathcal{A}$ such that (1) $\mathcal{A}$ controls all parties except for P_{k-1} and P_k, and (2) $\mathcal{A}$ plays the asynchronous network and the roles of the malicious parties such that every execution among $P_1, P_2, \ldots, P_n$ satisfies F. $\mathcal{A}$ is a computationally-bounded algorithm such that two honest parties P_{k-1} and P_k output differently with non-negligible probability. This violates the *(weak) fairness* property. A contradiction.

As a result, if the statement is true for $k-1, 2 \leq k \leq n$, the statement is true for k. Therefore, the lemma is true for any $k, 2 \leq k \leq n$. □

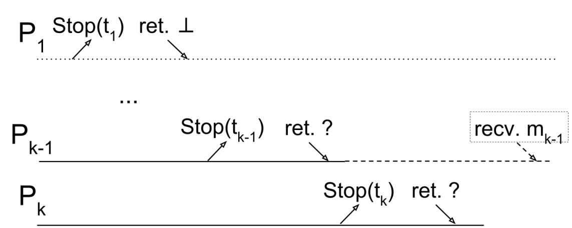
Figure 2 illustrates the three key executions in the proof of lemma 5: $E(P_k)$, $E^*(P_k)$, $E(P_{k-1})$.



(a) Execution $E(P_k)$



(b) Execution $E^*(P_k)$



(c) Execution F

Figure 2: The three key executions in the proof of Lemma 5. A dot line means that any event might occur. A dashed line means that an event does not occur. A solid line means that the same event as in E occurs.

Lemma 6. *For any optimistic execution E, let $\underline{R}$, a sequence of Ω, be the sequence of destinations of the messages received between the two events: the event of m_1^* arriving at its destination and the event of m_2^* arriving at its destination, inclusive.*

Then $\underline{R}$ contains all the permutations of Ω as subsequences.

Proof. Let E be initialized with $x_1, x_2, \ldots, x_n$. We prove by contradiction. Suppose that, w.l.o.g., $\underline{R}$ does not include $P_1, P_2, \ldots, P_n$ as a subsequence.

By Corollary 2, $\underline{R}$ starts at the destination of m_1^* and ends at the destination of m_2^*; and $\underline{R}$ includes P_1 as a subsequence, which is also true for $P_2, \ldots, P_n$. Then there exists some $k, 2 \leq k \leq n-1$ such that $\underline{R}$ includes $P_1, P_2, \ldots, P_k$ as a subsequence and does not include $P_1, P_2, \ldots, P_{k+1}$ as a subsequence.

As a result, there exists a sequence $m_1, m_2, \ldots, m_k$ of k messages in E such that (a) the destination of $m_i, 1 \leq i \leq k$ is P_i; (b) $m_{i+1}, 1 \leq i \leq k-1$ is the first message received by P_{i+1} after P_i receives m_i and (c) $m_1 = m_1^*$, or m_1 is the first message received by P_1 after m_1^* arrives at its destination; and (d) the event of m_k arriving at P_k precedes the event of m_2^* arriving at its destination. (The event of m_k may also be the event of m_2^*.)

Let t_1 be the status of P_1 right before P_1 receives m_1 in E. Define execution $E(P_1)$ such that $E(P_1)$ is the same as E for P_1 until P_1 invokes Stop with t_1 right before m_1 arrives at P_1. By Lemma 2, for any $E(P_1)$, if no other party invokes Stop before P_1, then Stop returns $\perp$ to P_1.

Let $t_i, 2 \leq i \leq k$ be the status of P_i right before P_i receives m_i in E. Define execution $E(P_k)$ such that (a) $E(P_k)$ is the same as E for P_k until P_k invokes Stop with t_k right before message m_k arrives at P_k; (b) $P_1, P_2, \ldots, P_{k-1}$ invoke Stop with $t_1, t_2, \ldots, t_{k-1}$ respectively; (c) for $2 \leq i \leq k$, P_i invokes Stop after Stop returns to P_{i-1}; (d) Stop returns $\perp$ to $P_1, P_2, \ldots, P_{k-1}$; and (5) no other party has invoked Stop. By Lemma 5, Stop returns $\perp$ to P_k in $E(P_k)$ except with negligible probability.

Let m be the last message received by P_{k+1} before message m_2^* arrives at its destination in E. By Corollary 3, m exists if P_{k+1} is not the destination of m_2^*. Therefore, if P_{k+1} is not the destination of m_2^*, then the event of m arriving at its destination precedes the event of m_k arriving at P_k in E; otherwise, we have a subsequence $P_1, P_2, \ldots, P_{k+1}$, which gives a contradiction. Moreover, P_{k+1} is not the destination of m_2^*; otherwise, we again have a subsequence $P_1, P_2, \ldots, P_{k+1}$, which gives a contradiction. (If the event of m_k is the event of m_2^*, then the event of m arriving at its destination obviously precedes the event of m_k arriving at P_k in E.)

Let t_{k+1} be the status of P_{k+1} right after P_{k+1} receives m in E. Consider an execution $E(P_k, P_{k+1})$ that is the same as $E(P_k)$ for all the parties in $\Omega \backslash \{P_{k+1}\}$ and is the same as E for P_{k+1} until P_{k+1} invokes Stop with t_{k+1} after Stop has returned to P_k. Since the event of m arriving at its destination precedes the event of message m_k arriving at P_k, in $E(P_k, P_{k+1})$, we let P_{k+1} invoke Stop with t_{k+1} also after P_{k+1} receives m.

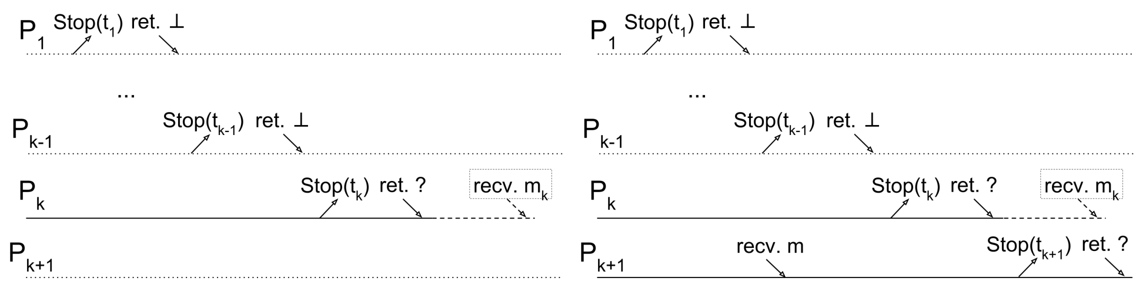
In $E(P_k, P_{k+1})$, P_k's behavior is the same as an honest P_k to $\Omega \backslash \{P_k\}$ and T; P_{k+1}'s behavior is the same as an honest P_{k+1} to $\Omega \backslash \{P_{k+1}\}$ and T. Hereafter, we say that P_k and P_{k+1} are honest in $E(P_k, P_{k+1})$. Moreover, until Stop returns to P_k, $E(P_k, P_{k+1})$ and $E(P_k)$ are indistinguishable to P_k and T and therefore Stop returns $\perp$ to P_k except with negligible probability also in $E(P_k, P_{k+1})$. However, by Lemma 4, Stop returns $z = f(x_1, x_2, \ldots, x_n)$ to P_{k+1}.

Now we build an algorithm $\mathcal{A}$ such that (1) $\mathcal{A}$ controls all parties except for P_{k-1} and P_k, and (2) $\mathcal{A}$ plays the asynchronous network and the roles of the malicious parties such that every execution among $P_1, P_2, \ldots, P_n$ satisfies $E(P_k.P_{k+1})$. $\mathcal{A}$ is a computationally-bounded algorithm such that two honest parties P_k and P_{k+1} output differently with non-negligible probability. This violates the *(weak) fairness* property. A contradiction. □

Figure 3 illustrates two key executions in the proof of Lemma 6: $E(P_k)$, $E(P_k, P_{k+1})$.

Now that we have all the necessary properties of any optimistic execution, we are ready to prove Theorem 1.

Proof of Theorem 1. Let $\underline{R}$ be defined as in Lemma 6. Then by Lemma 6, ℓ lower-bounds the length of $\underline{R}$. By the definition of m_1^*, there are at least $n-2$ messages that precede m_1^* in E; otherwise, at least one party has not yet contacted the destination of m_1^*. By the definition of



(a) Execution $E(P_k)$ (b) Execution $E(P_k, P_{k+1})$

Figure 3: Two key executions in the proof of Lemma 6. A dot line means that any event might occur. A dashed line means that an event does not occur. A solid line means that the same event as in E occurs.

m_2^*, there are at least $n-1$ messages that follow m_2^* in E; otherwise, at least one party P cannot compute z from P's input and P's status.

Therefore, during any optimistic execution E, the number of messages sent is at least $\ell + 2n - 3$. □

Remark 1 (Honest behavior in an execution)**.** Usually without a protocol specification, we cannot define any honest behavior. In the proof of Theorem 1, the honest behavior is relative to an optimistic execution.

B Appendix. Proof of Theorem 3

We give here a detailed proof of Theorem 3. We note that this is a proof of a stand-alone execution. This is consistent with Definition 3, which defines an isolated $(n+1)$-party case of optimistic fair computation.

Before we present the proof, we recall the formal definition and security guarantee of verifiable encryption from [12].

Definition 6 (Verifiable encryption [12])**.** Let (G, E, D) be the key generation, encryption and decryption algorithms of a semantically secure public-key encryption scheme. Let (vk, sk) be one key pair generated by G where vk is the public key and sk is the secret key. Let R be a relation and let $L_R = \{x | \exists w \text{ such that } (x, w) \in R\}$. A *verifiable encryption* scheme for a relation R consists of a two-party protocol (P, V) and a recovery algorithm D. P and V take as common inputs: vk, x, R (and some condition κ to open string α). P takes witness w such that $(x, w) \in R$ as an extra input. V rejects (i.e., outputs $\perp$), or accepts and obtains string α. D takes as inputs: sk, α (and κ). D outputs a witness $\hat{w}$ if κ is verified.

A verifiable encryption scheme is *secure* if it satisfies the following properties:

- *Completeness*: If P and V are honest, then V accepts in the two-party protocol for all (vk, sk) and for all $x \in L_R$.

- *Validity*: For any computationally-bounded algorithm $\hat{P}$, for all (vk, sk), if V accepts and obtains string α in the two-party protocol with $\hat{P}$, then given α and sk, D outputs a witness $\hat{w}$ such that $(x, \hat{w}) \notin R$ with negligible probability.

- *Computational zero-knowledge*: For every algorithm $\hat{V}$, there exists an expected polynomial-time simulator S given vk and x, and with black-box access to $\hat{V}$ such that for all $x \in L_R$, the output of $\hat{V}$ after the two-party protocol with an honest P is computationally indistinguishable from the output of S.

Furthermore, for the simplicity of the proof, we consider the particular verifiable encryption scheme proposed in [12]. As [12] pointed out, their construction of verifiable encryption can be made non-interactive via Fiat-Shamir heuristic [38]; the resulting non-interactive variant is secure in the random oracle model [39]. For the non-interactive variant, it is easy to see that the algorithm V in the scheme is *deterministic*; i.e., given the one message sent by $\hat{P}$, either V rejects (with probability 1) or V accepts (with probability 1); and the recovery algorithm D in the scheme is also deterministic; i.e., given sk, κ and α, either D rejects (with probability 1) or D outputs a witness (with probability 1).

Proof of Theorem 3. As shown in Algorithm 1, the number of messages is equal to the length of sequence $\underline{j}$ which is $l + 2n - 3$. Thus the n parties exchange exactly $l + 2n - 3$ messages in π. In what follows, we verify that our protocol satisfies Definition 3 and Definition 4.

Optimism. If $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop, then all parties follow π in which all parties output $z = f(x_1, x_2, \ldots, x_n)$ without interacting with T.

Non-triviality. As shown in Algorithm 1, if messages are delivered instantly, then $P_1, P_2, \ldots, P_n$ do not invoke Stop; therefore, we find one execution of π that $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop.

Completeness. If $P_1, P_2, \ldots, P_n$ are honest and none invokes Stop, then all parties follow π and output $z = f(x_1, x_2, \ldots, x_n)$. Next, we show by contradiction that if all parties are honest and some invokes Stop, then either all parties output $\perp$ or all parties output $z = f(x_1, x_2, \ldots, x_n)$. Suppose that an honest party P outputs $\perp$ and an honest party Q outputs z. Since P outputs $\perp$, then either (1) π has not started, or (2) $P = P_{j_k}$ and $0 \le k \le n-1$, or (3) $P = P_{j_k}$ and $n \le k \le l + 2n - 3$ For cases (1) and (2), since by Equation (2),

$$m_k = \begin{cases} m_{k-1}||VE_{j_k}||Sig_{j_k}(m_{k-1}||VE_{j_k}) & 1 \le k \le n-1 \\ VE_{j_0} & k = 0, \end{cases}$$

P has not sent VE_{j_k}. Again by Equation (2), $m_{end(j_k)} = m_{end(j_k)-1}||x_{j_k}||Sig_{j_k}(m_{end(j_k)-1}||x_{j_k})$. Since $end(j_k) > n-1$, P has not sent x_{j_k}. Since all parties are honest, Q does not output z from running π or μ in cases (1) and (2).

In case (3), since all parties are honest, by the *completeness* property of verifiable encryption, and the definition of digital signatures, T accepts that req_k and $P = P_{j_k}$'s claim are consistent. As P is honest, P has not sent a request before. In case (3), we consider two disjoint cases: (a) $\exists i \in \{1, 2, \ldots, n\}$, VE_i is not in req_k, and (b) $\forall i \in \{1, 2, \ldots, n\}$, VE_i is in req_k.

Consider case (3.a). By Equation (2), $\forall ix \ge n-1$, m_{ix} contains $VE_1, VE_2, \ldots, VE_n$. Then $0 \le last_k \le n-2$. Since $j_0, j_1, \ldots, j_{n-1}$ are different from each other, $j_k \ne j_{n-1}$. Moreover, $k = \min_{ix \in \{n, n+1, \ldots, l+2n-3\}}\{ix | j_{ix} = j_k\} \le end(j_k)$. Therefore, P has not sent x_{j_k}, and Q cannot output x_{j_k} following π.

Clearly, if Q does not interact with T, then Q outputs $\perp$, and furthermore, if Q interacts with T before P interacts with T, then Q also outputs $\perp$. If Q interacts with T after P interacts with T, then we assume that Q sends a request req_q to T. Since Q is honest, T accepts that req_q

is consistent with Q's claim that Q has not received m_{q-1} (but has received m_{last_q-1}). By the definition of $\underline{i}$, $q \le endj_q$. (Otherwise, we do not have j_k, j_q as a subsequence of $\underline{i}$). In addition, since Q is honest, $q \le \min_{ix\in\{k+1,k+2,\ldots,l+2n-3\}}\{ix|j_{ix} = j_q\}$. Therefore, T sends "aborted" to Q.

In case (3.b), w.l.o.g., assume that P is the earliest process that sends to T a request and receives "aborted". Then variable *progress* is 0 at T when P sends req_k. Then we have

$$k \le \min_{ix\in\{1,2,\ldots,l+2n-3\}}\{ix|j_{ix} = j_k\} \triangleq first(j_k).$$

If $j_k \ne j_0$, $k \le n-1$, which gives a contradiction. If $j_k = j_0$, then since $k \le first(j_k)$, $last_k = 0$; thus $req_k = m_{last_k} = VE_{j_0}$, which also gives a contradiction for $n \ge 2$.

Termination. As shown in Algorithm 1 and Algorithm 2, an honest party either follows π and outputs, or wants to stop, follows μ and outputs. Since any message between an honest party and T eventually reaches its destination, an honest party eventually outputs.

Fairness. We prove that for any $e \in \mathbb{N}$, $1 \le e \le n-1$, any e malicious parties $P_{d_1}, P_{d_2}, \ldots, P_{d_e}$, and any computationally-bounded algorithm $\mathcal{A}$, there exists a computationally-bounded algorithm $\mathcal{S}$ such that the joint outputs $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable for any $x_1, x_2, \ldots, x_n$.

We construct $\mathcal{S}$ that runs $\mathcal{A}$ as a black-box as follows.

1. $\mathcal{S}$ generates $n+1$ key pairs $(pk_1, sk_1), (pk_2, sk_2), \ldots, (pk_n, sk_n), (vk_T, sk_T)$; and then $\mathcal{S}$ invokes $\mathcal{A}$ and initializes $\mathcal{A}$ with inputs $x_{d_1}, x_{d_2}, \ldots, x_{d_e}$, $n+1$ parties' public keys $pk_1, pk_2, \ldots, pk_n, pk_T$ and malicious parties' private keys $sk_{d_1}, sk_{d_2}, \ldots, sk_{d_e}$.[16]

2. $\mathcal{S}$ plays the role of the $n-k$ honest parties $P_{h_1}, P_{h_2}, \ldots, P_{h_{n-e}}$ and T, and executes our protocol honestly with $\mathcal{A}$ except that:

   - If by Algorithm 1, $\mathcal{S}$ has to send the $(k-1)$th message for $1 \le k \le n$ on behalf of an honest party, then by the construction of Fiat-Shamir paradigm [38] and the *computational zero-knowledge* property of verifiable encryption, $\mathcal{S}$ can simulate the random oracle [39] and invoke the simulator (defined in the *computational zero-knowledge* property) to compute message $\hat{m}_{k-1}$ (that is computationally indistinguishable from the $(k-1)$th message except with negligible probability).
   - If by Algorithm 1, $\mathcal{S}$ has to send the $(k-1)$th message for $end(j_{k-1})+1 \le k \le l+n-2$ on behalf of an honest party P_{src}, then $\mathcal{S}$ sends $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ on behalf of $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$ respectively to U. $\mathcal{S}$ obtains a response from U, which contains $x_{h_1}, x_{h_2}, \ldots, x_{h_{n-e}}$. Then $\mathcal{S}$ uses x_{src} to compute message $\hat{m}_{k-1}$. (How to obtain $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ is explained later.)
   - If by Algorithm 2, $\mathcal{S}$ has to send a response including the $P_{h_1}, P_{h_2}, \ldots, P_{h_{n-e}}$'s inputs on behalf of T, then $\mathcal{S}$ sends $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ on behalf of $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$ respectively to U. $\mathcal{S}$ obtains a response from U, which contains $x_{h_1}, x_{h_2}, \ldots, x_{h_{n-e}}$. $\mathcal{S}$ uses $x_{h_1}, x_{h_2}, \ldots, x_{h_{n-e}}$ to compute a response. (How to obtain $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ is explained later.)

[16]Both $\mathcal{A}$ and $\mathcal{S}$ are also initialized with public information, including the relations $\kappa = (a_1, R_1)||(a_2, R_2)||\cdots||(a_n, R_n)$, the algorithms of our protocol and in particular, the deterministic strategy of when to invoke Stop for every honest party.

- ($\mathcal{S}$ sends $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ on behalf of $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$ respectively only once to U.)[17]

3. In addition, $\mathcal{S}$ executes the following.

   - If according to an honest party P's strategy of invoking Stop and μ, at some point in the execution with $\mathcal{A}$, P invokes Stop and outputs $\perp$, then $\mathcal{S}$ sends $\perp$ on behalf of an arbitrary party in $\bar{P}_{d_1}, \bar{P}_{d_2}, \ldots, \bar{P}_{d_e}$ to U. If $\mathcal{S}$ ever sends $\perp$, $\mathcal{S}$ sends $\perp$ only once.
   - $\mathcal{S}$ saves $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ by decrypting $VE_{d_1}, VE_{d_2}, \ldots, VE_{d_e}$ from the messages exchanged with $\mathcal{A}$ (in π or μ).

4. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

We verify that $\mathcal{S}$ has saved $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ before when $\mathcal{S}$ has to send a message that contains at least one honest party's input. If by Algorithm 1, $\mathcal{S}$ has to send the $(k-1)$th message for $end(j_{k-1}) + 1 \leq k \leq l + n - 2$, then $\mathcal{S}$ has received and verified the $(k-2)$th message. By the definition of sequence $\underline{i}$, if $j_{k-1} = j_{n-1}$, then $end(j_{k-1}) \geq n+1$; if $j_{k-1} \neq j_{n-1}$, then the first symbol of $\underline{i}$ is not j_{k-1} and thus $end(j_{k-1}) \geq n$. In either case, $k \geq n+1$, and therefore the $(k-2)$th message includes $VE_{d_1}, VE_{d_2}, \ldots, VE_{d_e}$. If by Algorithm 2, $\mathcal{S}$ has to send a response on behalf of T, then $\mathcal{S}$ has verified the corresponding request, which also includes verified $VE_{d_1}, VE_{d_2}, \ldots, VE_{d_e}$. Thus, by the *validity* property of verifiable encryption, $\mathcal{S}$ successfully decrypts $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ such that $\{a_{d_i}, \hat{x}_{d_i}\} \in R_{d_i}, \forall i \in \{1, 2, \ldots, e\}$ except negligible probability.

We also verify that $\mathcal{S}$ does not send $\perp$ and $\hat{x}_{d_1}, \hat{x}_{d_2}, \ldots, \hat{x}_{d_e}$ to U in the same execution, except with negligible probability in a separate lemma (Lemma 7, which is given and proved later).

To show that the joint outputs $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable, we first consider the transcript between $\mathcal{S}$ and $\mathcal{A}$, and the transcript among $P_1, P_2, \ldots, P_n$ and $\mathcal{A}$. By the *computational zero-knowledge* property of verifiable encryption and the definition of $\mathcal{S}$, any computationally-bounded algorithm $\mathcal{A}$ cannot distinguish the two transcripts except with negligible probability. Let F be any execution between $\mathcal{A}$ and $\mathcal{S}$ in the game above when $\mathcal{S}$ is well-defined.[18] W.l.o.g., in F, honest parties played by $\mathcal{S}$ output according to Algorithm 1. Denote by O_F the joint output of $P_1, P_2, \ldots, P_n$ and $\mathcal{A}$ in F. Then O_F and $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable.

We next consider the execution G among $\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n, \mathcal{S}$ and $\mathcal{U}$ when $\mathcal{S}$ runs F. We compare the joint output O_F with the joint output O_G of $\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_n, \mathcal{S}$ in G as follows. $\mathcal{S}$'s output is the same as $\mathcal{A}$'s output. For any honest party $P_{h_i}, i \in \{1, 2, \ldots, n-e\}$, we show that $\bar{P}_{h_i}$ outputs the same. There are three possibilities for P_{h_i}: P_{h_i} either (1) invokes Stop and outputs $\perp$, or (2) invokes Stop and outputs a non-$\perp$ value, or (3) does not invoke Stop but outputs a non-$\perp$ value. In case (1), $\mathcal{S}$ sends $\perp$ to U and thus in G, $\bar{P}_{h_i}$ also outputs $\perp$. In case (2), (a) if P_{h_i} interacts with T, then $\mathcal{S}$ uses U's response as T's response to P_{h_i}; (b) if not, then to P_{h_i}, π finishes and $\mathcal{S}$ must have obtained U's response to query inputs for honest parties including P_{h_i}. Thus whether P_{h_i} interacts with T or not, $\bar{P}_{h_i}$ also outputs the same. Case (3) is the same as case (2.b). Then O_F and O_G have the same distribution.

As a result, O_G and $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable. Denote by *event* the event that $\mathcal{S}$ is not well-defined. Since *event* occur with negligible probability, O_G

[17] We note that on behalf of T, when $\mathcal{S}$ has to verify the ciphertexts of verifiable encryption in a request, $\mathcal{S}$ only verifies those ciphertexts $VE_{d_1}, VE_{d_2}, \ldots, VE_{d_e}$ (as $\mathcal{S}$ creates the others).

[18] With negligible probability, $\mathcal{S}$ is not well-defined. I.e., $\mathcal{S}$ cannot simulate the game above with $\mathcal{A}$, for example, when the simulator defined in the *computational zero-knowledge* property of verifiable encryption exceeds polynomial time, when $\mathcal{S}$ decrypts $\hat{x}_{d_i}$ such that $(a_{d_i}, \hat{x}_{d_i}) \notin R_{d_i}$ for some $i \in \{1, 2, \ldots, e\}$, and when some honest party outputs $\perp$ but $\mathcal{S}$ still has to send a response that includes honest parties' inputs.

and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable. Then $\mathcal{S}$ a computationally-bounded algorithm such that for any $x_1, x_2, \ldots, x_n$ such that $O_{P_1,P_2,\ldots,P_n,\mathcal{A}}(x_1, x_2, \ldots, x_n)$ and $O_{\bar{P}_1,\bar{P}_2,\ldots,\bar{P}_n,\mathcal{S}}(x_1, x_2, \ldots, x_n)$ are computationally indistinguishable. □

Next, we give the necessary lemma, which we use to verify that $\mathcal{S}$ as defined in the proof of Theorem 3 does not send conflicting messages to U except with negligible probability. However, instead of discussing $\mathcal{S}$, we state the lemma in a more general but equivalent way.

Lemma 7 (Simulation of $\mathcal{S}$)**.** *By Algorithm 1 and Algorithm 2, for any $e \in \mathbb{N}, 1 \le e \le n-1$ and any e malicious parties $P_{d_1}, P_{d_2}, \ldots, P_{d_e}$, for any computationally-bounded algorithm $\mathcal{A}$ that controls the e malicious parties, $\forall x_1, x_2, \ldots, x_n$, for any honest party P,*

- *either P outputs $\perp$ with negligible probability,*
- *or P outputs $\perp$ with non-negligible probability and given that an honest party P outputs $\perp$, any other party Q outputs the honest parties' inputs except with negligible probability.*

By Lemma 7, for $\mathcal{S}$, as defined in the proof of Theorem 3, when $\mathcal{S}$ has to send $\perp$ to U with non-negligible probability, the probability that $\mathcal{S}$ has to send non-$\perp$ inputs to U is negligible. Lemma 7 also implies the inverse: if a party outputs the honest parties' inputs with non-negligible probability, then an honest party P outputs $\perp$ with negligible probability. In other words, when $\mathcal{S}$ has to send non-$\perp$ inputs to U with non-negligible probability, the probability that $\mathcal{S}$ has to send $\perp$ to U is negligible.

Proof of Lemma 7. We need only to prove the case where P outputs $\perp$ with non-negligible probability.

Since P is honest, then either (1) π has not started, or (2) $P = P_{j_k}$ for $0 \le k \le n-1$, or (3) $P = P_{j_k}$ for $n \le k \le l + 2n - 3$. (Some intermediary results are already deduced for the *completeness* property in the proof of Theorem 3 and is thus not repeated here.)

In cases (1) and (2), P has not sent x_{j_k} or VE_{j_k} and thus by the property of (a_{j_k}, R_{j_k}), any computationally-bounded algorithm outputs x with negligible probability. In case (3), since P is honest, then by the determinism of the verification algorithm V of verifiable encryption, T accepts that req_k is consistent with P's claim, and in addition, P has not sent a request before. When P interacts with T, at least one of the two holds: (a) $\exists i \in \{1, 2, \ldots, n\}$, VE_i is not in req_k, or (b) $\forall i \in \{1, 2, \ldots, n\}$, VE_i is in req_k.

In case (3.a), P has not sent x_{j_k}. If Q interacts with T before P interacts with T, then T sends "aborted" to Q. If Q interacts with T after P interacts with T, then we assume that Q sends a request req_q. We show that the following two events occur at the same time with negligible probability: event A is $q > \min_{ix\in\{k+1,k+2,\ldots,l+2n-3\}}\{ix | j_{ix} = j_q\} \triangleq next_k(q)$ and event B is that Q passes the consistency verification of req_q at T. We show this by contradiction. Suppose that the two events occur at the same time with non-negligible probability. Since $q > next_k(q)$, then $last_q \ge next_k(q) > k$; therefore, req_q includes message m_k which includes P's signature on message m_{k-1}. Then Q is a computationally algorithm which forges P's signature on m_{k-1} (which P has not signed before) with non-negligible probability, a contradiction to the unforgeability of digital signatures. Therefore, A and B occur at the same time with negligible probability. Let $\bar{A}$ be the complement of A and let $\bar{B}$ be the complement of B. Then $\bar{A} \cup \bar{B}$ occurs except with negligible probability. Since $next_k(q) \le end(j_q) \le l + n - 2$, T sends "aborted" to Q except with negligible probability. If Q does not interact with T, then Q only obtains VE_{j_k} from π. Thus by the property of (a_{j_k}, R_{j_k}) and by the *computational zero-knowledge* property of verifiable encryption, any computationally-bounded algorithm outputs x with negligible probability.

In case (3.b), since $k \leq l+n-2$, then by the definition of *end*, $k \leq end(j_k)$. By Equation (2), P has not sent x. Similar to case (3.b), If Q does not interact with T, then Q only obtains VE_{j_k} from π. Clearly, if Q interacts with T but T sends "aborted" to Q except with negligible probability, then by the property of (a_{j_k}, R_{j_k}) and the *computational zero-knowledge* property of verifiable encryption, the probability that Q outputs x_{j_k} is negligible.

We show by contradiction that Q interacts with T but T sends "aborted" to Q except with negligible probability. Suppose that Q interacts with T but T sends a non-"aborted" value to Q with non-negligible probability. Assume that Q sends a request req_q to T. Let pg be the value of the variable *progress* at T when Q starts to interact with T. Let event A be $q > next_pg(q)$ and let event B be the event that Q passes the consistency verification of req_q at T. Then similar to case (3.a), $\bar{A} \cup \bar{B}$ occurs except with negligible probability. If $\bar{B}$ occurs, then T sends "aborted" to Q. Since $\bar{A} \cup \bar{B}$ occurs except with negligible probability, then $\bar{A} \cap B$ occurs with non-negligible probability. Clearly, if the recovery of the inputs from their ciphertexts of verifiable encryption is not successful, then the condition κ is not satisfied and T sends "aborted" to Q. However, by the *validity* property of verifiable encryption, given that B occurs, the unsuccessful recovery occurs with negligible probability. In what follows, we consider the case where $\bar{A} \cap B$ occurs and the recovery for Q is successful.

W.l.o.g., Q is the first process that receives a non-"aborted" value from T. Then since Q is the first process that receives a non-"aborted" value, by the *validity* property of verifiable encryption, $pg \geq l+n-2$ except with negligible probability.

When Q invokes μ with request req_q, the variable z at T is $\perp$. By Algorithm 2, thus T updates *progress* in a specific way: *progress* is first updated with request req_{I_1} where I_1 is the first index of j_{I_1} in the suffix $\underline{j}[n-1:]$ of sequence $\underline{j}$, and then each update is with such a request req_{I_2} that I_2 is the first index of j_{I_2} in the suffix $\underline{j}[progress+1:]$. Let α be the sequence of parties who invoke μ and trigger T to update *progress* before P_{j_q} invokes μ for req_q. Let σ be the sequence of the subscripts of those parties.

Since $pg \geq l+n-2$ except with negligible probability, σ is a subsequence of $\underline{i} = \underline{j}[n-1 : l+n-2]$ and, moreover, must be the prefix of some permutation of $\{1, 2, \ldots, n\}$ in $\underline{i}$ except with negligible probability.

Clearly, if T returns a non-"aborted" to Q, then $q \geq l+n-1$. Since $\bar{A}$ occurs, $next_pg(q) \geq l+n-1$. When $pg \geq l+n-2$, since σ ends at j_{pg} (inclusive) and there is no j_q between j_{pg} and j_{next-1}, j_q must occur in σ. (Otherwise, as $next \geq l+n-1$, there is no hope for σ to include j_q in the permutation before j_{l+n-2} (inclusive), contradictory to the definition of $\underline{i}$.) In other words, P_{j_q} must have invoked μ before, except with non-negligible probability. Then T returns "aborted" to Q for req_q except with negligible probability. A contradiction.

Thus, we conclude that when P outputs $\perp$ with non-negligible probability, then given that an honest party P outputs $\perp$, any other party Q outputs the honest parties' inputs except with negligible probability. □