Optimally Efficient Multi-Party Fair Exchange and Fair Secure Multi-Party Computation

Handan Kılınç*[1] and Alptekin Küpçü†[2]

[1]EPFL, Koç University
[2]Koç University

Abstract

Multi-party fair exchange (MFE) and fair secure multi-party computation (fair SMPC) are under-studied fields of research, with practical importance. We examine MFE scenarios where every participant has some item, and at the end of the protocol, either every participant receives every other participant's item, or no participant receives anything. This is a particularly hard scenario, even though it is directly applicable to protocols such as fair SMPC or multi-party contract signing. We further generalize our protocol to work for any exchange topology. We analyze the case where a trusted third party (TTP) is optimistically available, although we emphasize that the trust put on the TTP is only regarding the *fairness*, and our protocols preserve the *privacy* of the exchanged items even against a malicious TTP.

We construct an *asymptotically optimal* (for the complete topology) multi-party fair exchange protocol that requires a constant number of rounds, in comparison to linear, and $O(n^2)$ messages, in comparison to cubic, where n is the number of participating parties. We enable the parties to efficiently exchange any item that can be efficiently put into a verifiable escrow (e.g., signatures on a contract). We show how to apply this protocol on top of *any* SMPC protocol to achieve a fairness guarantee with very little overhead, especially if the SMPC protocol works with arithmetic circuits. Our protocol guarantees fairness in its strongest sense: even if all $n-1$ other participants are malicious and colluding, fairness will hold.

Keywords: multi-party fair exchange, fair computation, optimistic model, secure multi-party computation, electronic payments

1 Introduction

An exchange protocol allows two or more parties to exchange items. It is *fair* when the exchange guarantees that either all parties receive their desired items or none of them receives any item. Examples of such exchanges include signing electronic contracts, certified e-mail delivery, and fair purchase of electronic goods over the Internet. In addition, a fair exchange protocol can be adopted by secure two- or multi-party computation protocols [12, 29, 19, 9, 32, 49, 38] to achieve fairness [33].

Even in two-party fair exchange scenarios, preventing unfairness completely and efficiently without a trusted third party (TTP) is shown to be impossible [23, 45]. The main reason is that one of the parties will be sending the last message of the protocol, regardless of how the protocol looks like, and may choose not to send that message, potentially

*handan.kilinc@epfl.ch
†akupcu.ku.edu.tr

causing unfairness. In an *optimistic* protocol, the TTP is involved in the protocol *only* when there is a malicious behavior [3, 4]. However, it is important not to give a lot of work to the TTP, since this can cause a bottleneck. Furthermore, the TTP is required *only* for *fairness*, and should not learn more about the exchange than is required to provide fairness. In particular, **in our protocols, we show that the TTP does *not* learn the *items*** that are exchanged.

Fair exchange with two parties have been extensively studied and efficient solutions [4, 11, 35, 34, 36] have been proposed, but the multi-party case does not have efficient and general solutions. Multi-party fair exchange (MFE) can be described based on *exchange topologies*. For example, a *ring topology* describes an MFE scenario where each party receives an item from the previous party in the ring [7, 40, 30, 40]. A common scenario with the ring topology is a customer who wants to buy an item offered by a provider: the provider gives the item to the customer, the customer sends a payment authorization to her bank, the customer's bank sends the payment to the provider's bank, and finally the provider's bank credits the provider's account.

Ring topology cannot be used in scenarios like contract-signing and secure multi-party computation (SMPC), since in such scenarios the parties want items from all other parties. In particular, in such settings, **we want that either every participant receives every other participant's item, or no participant receives anything**. This corresponds to the contract being signed only if everyone agrees, or the SMPC output to be revealed only when every participant receives it. We call this kind of topology a *complete topology*. We can think of the parties as nodes in a complete graph and the edges between parties show the exchange links. The complete topology was researched mostly in the contract-signing setting [28, 10, 27], with one exception [3]. Unfortunately, all these protocols are inefficient compared to ours (see Table 1). Since there was no an efficient MFE protocol that achieves the complete topology, the fairness problem in SMPC protocols still could not be completely solved. Existing fair SMPC solutions either work with inefficient gradual release [26], or require the use of bitcoins [13, 1].

Our Contributions: We suggest a new optimistic multi-party fair exchange protocol that guarantees fairness in every topology, including the complete topology, efficiently.

- Our MFE requires only $\mathbf{O}(\mathbf{n^2})$ messages and **constant** number of rounds for n parties, being much more efficient than the previous works (see Table 1). These are asymptotically **optimal** for a complete topology, since each party should send his item to all the other parties, even in an unfair exchange. Furthermore, our MFE does *not* necessitate a *broadcast*.
- Our MFE **optimally** guarantees fairness (for honest parties) even when $n-1$ out of n parties are malicious and colluding.
- Our MFE has an easy setup phase, which is employed only **once** for exchanging **multiple** sets of items, thus improving efficiency even further for *repeated exchanges* among the same set of participants.
- The TTP for fairness in our MFE is in the *optimistic* model [4]. The TTP has a very low workload, since the parties only employ efficient discrete-logarithm-based sigma proofs to show their honesty. More importantly, the TTP does *not* learn any exchanged item, so **privacy against the TTP** is preserved.
- We show how to employ our MFE protocol for **any exchange topology**, with the performance improving as the topology gets sparser.
- We formulate MFE as a secure multi-party computation protocol. We then **prove** security and fairness **via ideal-real world simulation** [33]. To the best of our knowledge, no multi-party fair exchange protocol was proven as an SMPC protocol

	Solution for	Topology	Round Complexity	Number of Messages	Broadcast
[28]	MPCS	Complete	$O(n^2)$	$O(n^3)$	Yes
[10]	MPCS	Complete	$O(tn)$	$O(tn^2)$	Yes
[44]	MPCS	Complete	$O(n)$	$O(n^3)$	Yes
[42]	MPCS	Complete	$O(n)$	$O(n^2)$✓	Yes
[3]	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^3)$	Yes
Ours	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^2)$ ✓	No ✓

Table 1: Efficiency comparison with previous works. n is the total number of parties, t is number of dishonest parties, and MPCS means multi-party contract signing protocol.

before.

- Based on the definition in [33], we provide an ideal world definition for *fair SMPC*, and prove via simulation that our MFE can be employed **on top of any SMPC protocol** to obtain a *fair* SMPC protocol, with the fairness extension leaking nothing about the inputs, and without necessitating a payment system.

2 Related Works

Two-party Fair Exchange: Most of the previous work in the fair exchange setting was done on the two-party case. The interesting case is the optimistic case, where there exists a trusted third party (TTP), but the TTP is not involved if both participants are honest [4, 3, 7, 6, 43, 5, 35].

Multi-party Fair Exchange: Franklin and Tsudik [25] classified multi-party fair exchange based on the number of items that a participant can exchange and the dispositions of the participants. Asokan et al. [3] described a generic optimistic fair exchange with a general topology. They define a description matrix to represent the topology, and proposed a fair protocol. The parties are restricted to exchange *exchangeable items*, requiring the TTP to be able to replace or revoke the items, greatly decreasing the applicability of their protocol. In addition, broadcast is used to send the items, rendering their protocol inefficient.

	Number Messages	All or None	TTP-Party Dependency	TTP Privacy
[8]	$O(n)$	No	Yes	Not Private
[30]	$O(n^2)$	No	Yes	Not Private
[40]	$O(n)$	No	Yes	Not Private
Ours	$O(n^2)$	Yes ✓	No ✓	Private ✓

Table 2: Efficiency comparison with previous works in the ring topology. n is number of parties. 'All or None' represents our fairness definition, where either the whole topology is satisfied, or no exchange occurs.

Ring Topologies: Bao et al. [8] proposed an optimistic multi-party fair exchange protocol based on the ring topology. In their protocol, one of the participants is the initiator, who starts the first and second phases of the protocol. The initiator is required to contact the TTP to acknowledge the completion of the first phase of the protocol. Thus, firstly, this is not a strictly optimistic protocol, secondly, there is a necessity of trusting the initiator, and thirdly, there is a passive conspiracy problem [25], which means that a dishonest party may conspire with an honest party without the latter's consent.

Later, Gonzales-Deleito and Markowitch [30] solved the malicious initiator problem of Bao et al. [8]. But, the problem in their protocol is in the recovery protocol: when one of the participants contacts the TTP, the TTP has to contact the previous participant in the ring. This is not preferable because it is not guaranteed that previous participant will be available. The protocol in [40] solves the passive conspiracy problem of Bao et al. [8], however the problem in the recovery protocol still remains.

Markowitch and Kremer [41] proposed a non-repudiation protocol, where their fairness definition is such that one of the parties sends some information to the other parties, and neither the sender nor others can deny that they participated. However, it does not solve the fairness problem in general.

Understanding Fairness: There is an important difference between our understanding of fairness, and existing ring-topology protocols' [8, 30, 40]. According to their definition, in the end of the protocol there will be no party such that he does not receive his desired item from the previous party but sends his item to the next party. It means that *there can be some parties who received their desired items and some other parties who did not receive or send anything.* Whereas, **according to our definition, either the whole topology is satisfied (all the necessary exchanges are complete), or no exchange takes place**. We believe this is a very important distinction, and is the right way of framing multi-party fair exchange. We further observe that this all-or-none type of fairness also requires a quadratic number of messages, which we achieve optimally. Table 2 summarizes comparison for the ring topology.

Complete Topologies: Multi-party contract signing indeed corresponds to a complete topology. Garay and Mackenzie [27] proposed the first optimistic multi-party contract signing protocol that requires $O(n^2)$ rounds and $O(n^3)$ messages. Because of its inefficiency, Baum-Waidner and Waidner [10] suggested a more efficient protocol, whose complexity depends on the number of dishonest parties, and if the number of dishonest parties is $n-1$, its efficiency is the same as [27]. Mukhamedov and Ryan [44] decreased the round complexity to $O(n)$. Lastly, Mauw et al. [42] gave the lower bound of $O(n^2)$ for the number of messages to achieve fairness. Their protocol requires $O(n^2)$ messages, but the round complexity is not constant. **We achieve both lower bounds ($\mathbf{O(n^2)}$ messages and constant round) for the first time.**

Fair Secure Multi-party Computation: Secure multi-party computation had an important position in the last decades, but its fairness property did not receive a lot of attention. One SMPC protocol that achieves fairness is designed by Garay et al. [31]. It uses gradual release, which is the drawback of this protocol, because each party broadcasts its output gradually in each round. At each round the number of messages is $O(n^3)$ and there are a lot of rounds due to gradual release.

Another approach is using bitcoin to achieve fairness using a TTP in the optimistic model [13, 1]. When one of the parties does not receive the output of the computation, he receives a bitcoin instead. This fairness approach was used by Lindell [37] for the two-party computation case, and by Küpçü and Lysyanskaya [35] and Belenkiy et al. [11] for peer-to-peer systems. However, this approach is not appropriate for multi-party computation since **we do not necessarily know how valuable the output will be before evaluation**. Finally, reputation-based fairness solutions [2] talk about fairness probabilities, rather than complete fairness.

3 Definitions and Preliminaries

3.1 Preliminaries

Threshold Public Key Encryption: In such schemes, encryption is done with a single public key, generated jointly by n decrypters, but decryption requires at least k decrypters to cooperate. It consists of the probabilistic polynomial time (PPT) protocols *Key Gen-*

eration, Verification, Decryption and a PPT algorithm for *Encryption* [48]. We describe these via the *ElGamal* $(n, k = n)$ threshold encryption scheme we will employ, as follows:

- *Key Generation:* It generates a list of private keys $SK = \{x_1, ..., x_n\}$, where $x_i \in \mathbb{Z}_p$, public key $PK = (g, h)$, where g is a generator of a large prime p-order subgroup of $\mathbb{Z}_q^*$ with q prime, together with $h = g^{\sum x_i}$, and public verification key $VK = \{vk_1, ..., vk_n\} = \{g^{x_1}, ..., g^{x_n}\}$, where $n \geq 1$. Note that this can be done in a distributed manner [47].
- *Encryption:* It computes the ciphertext for plaintext m as $E = (a, b) = (g^r, mh^r)$ where $r \in \mathbb{Z}_p$.
- *Verification:* It is between a verifier and a prover. Verifier, using VK, E, and the given decryption share of the prover $d_i = g^{rx_i}$, outputs *valid* if prover shows that $\log_g vk_i$ is equal to $\log_a d_i$. Otherwise, it outputs *invalid*.
- *Decryption:* It takes as input n decryption shares $\{d_1, ..., d_n\}$, where $d_i = g^{rx_i}$, VK, and E. Then, it outputs a message m with the following computation (in $\mathbb{Z}_q^*$),

$$\frac{b}{\prod d_i} = \frac{mh^r}{g^{r \sum x_i}} = \frac{mh^r}{h^r} = m$$

or $\perp$ if the decryption shares are invalid.

Verifiable Encryption: It is an encryption that enables the recipient to verify, using a public key, that the plaintext satisfies some relation, without performing any decryption [17, 16]. A public non-malleable label can be attached to a verifiable encryption [48].

Verifiable Escrow: An escrow is a ciphertext under the public key of the TTP. A *verifiable* escrow [4, 17] is a *verifiable* encryption under the public key of the TTP. We employ *ElGamal* verifiable encryption scheme [22, 15].

Notation. The n parties in the protocol are represented by P_i, where $i \in \{1, ..., n\}$. P_h is to show the honest parties, and P_c is to show the corrupted parties controlled by the adversary $\mathcal{A}$.

VE_i and VS_i is used to show the verifiable encryption and escrow prepared by P_i, respectively. The descriptive notation for verifiable encryption and escrow is $V(E, pk; l)\{(v, \xi) \in R\}$. It denotes the verifiable encryption and escrow for the ciphertext E whereby ξ –whose relation R with the public value v can be verified– is encrypted under the public key pk, and labeled by l. For escrows, pk is the TTP's public key.

$PK(v)\{(v, \xi) \in R\}$ denotes the zero-knowledge proof of knowledge of ξ that has a relation R with the public value v. All relations R in our protocols have an honest-verifier zero-knowledge three-move proof of knowledge [20], so can be implemented very efficiently. The notation ⓩ shows the number z in the Figure 1.

3.2 Definitions

Optimistic Fair Secure Multi-Party Computation: A group of parties with their private inputs w_i desire to compute a function ϕ [10, 29]. This computation is *secure* when the parties do not learn anything beyond what is revealed by the output of the computation. It is *fair* if either all of the parties learn the output in the end of the computation, or none of them learns the output. For an *optimistic* protocol, the TTP is involved *only* when there is a dispute about fairness between parties. This is formalized by ideal-real world simulations, defined below.

Ideal World: It consists of an adversary $\mathcal{A}$ that corrupts the set $\mathcal{P}_c$ of m parties where $m \in \{1, ..., n-1\}$, the set of remaining honest party(s) $\mathcal{P}_h$, and the universal trusted party U (*not the TTP*). The ideal protocol is as follows:

1. U receives inputs $\{w_i\}_{\{i \in \mathcal{P}_c\}}$ or the message ABORT from $\mathcal{A}$, and $\{w_j\}_{\{j \in \mathcal{P}_h\}}$ from the honest party(s). If the inputs are invalid or $\mathcal{A}$ sends the message ABORT, then U sends $\perp$ to all of the parties and halts.
2. Otherwise U computes $\phi(w_1, ..., w_n) = (\phi_1(w_1, ..., w_n), \phi_2(w_1, ..., w_n), ..., \phi_n(w_1, ..., w_n))$. Let $\phi_i = \phi_i(w_1, ..., w_n)$ be the i^{th} output. Then he sends $\{\phi_i\}_{\{i \in \mathcal{P}_c\}}$ to $\mathcal{A}$ and $\{\phi_j\}_{\{j \in \mathcal{P}_h\}}$ to the corresponding honest party(s).

The outputs of the parties in an ideal execution between the honest party(s) and an adversary $\mathcal{A}$ controlling the corrupted parties where U computes ϕ is denoted $\mathsf{IDEAL}_{\phi,\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)$, where $\{w_i\}_{1 \leq i \leq n}$ are the respective private inputs of the parties, aux is an auxiliary input of $\mathcal{A}$, and λ is the security parameter.

Real World: There is no universally trusted party U for a real protocol π to compute the functionality ϕ. There is an adversary $\mathcal{A}$ that controls the set $\mathcal{P}_c$ of corrupted parties and there is a TTP who is involved in the protocol when there is unfair behavior. The pair of outputs of the honest party(s) P_h and the adversary $\mathcal{A}$ in the real execution of the protocol π, possibly employing the TTP, is denoted $\mathsf{REAL}_{\pi,\mathsf{TTP},\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)$, where $w_1, w_2, ...w_n, aux$, and λ are like above.

Note that U and TTP are not related to each other. TTP is the part of the real protocol to solve the fairness problem when it is necessary, but U is not real (just an ideal entity).

Definition 1 (**Fair Secure Multi-Party Computation**). *Let π be a probabilistic polynomial time (PPT) protocol and let ϕ be a PPT multi-party functionality. We say that π computes ϕ **fairly and securely** if for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world simulator S so that for every $w_1, w_2, ..., w_n$, the ideal and real world outputs are computationally indistinguishable:*

$$\{\mathsf{IDEAL}_{\phi,S(aux)}(w_1, w_2, ..., w_n, \lambda)\}_{\lambda \in \mathbb{N}} \equiv_c \{\mathsf{REAL}_{\pi,\mathsf{TTP},\mathcal{A}(aux)}(w_1, w_2, ..., w_n, \lambda)\}_{\lambda \in \mathbb{N}}$$

The standard secure multi-party ideal world definition [39] lets the adversary $\mathcal{A}$ to abort *after* learning his output but *before* the honest party(s) learns her output. Thus, proving protocols secure using the old definition would not meet the fairness requirements. Therefore, we prove our protocols' security and fairness under the modified definition above. Canetti [18] gives general definitions for security for multi-party protocols with the same intuition as the security and fairness definition above. Further realize that since the TTP T does not exist in the ideal world, the simulator should also simulate its behavior.

Optimistic Multi-Party Fair Exchange: The participants are $P_1, P_2, ..., P_n$. Each participant P_i has an item f_i to exchange, and wants to exchange his own item f_i with the other parties' items $\{f_j\}_{j \neq i}$, , where $i, j \in \{1, ..., n\}$. Thus, at the end, every participant should obtain $\{f_i\}_{1 \leq i \leq n}$ in a complete topology, or some subset of it defined by some other exchange topology.

Multi-Party fair exchange is also a multi-party computation where the functionality ϕ is defined via its parts ϕ_i as below (we exemplify using a complete topology):

$$\phi_i(f_1, ..., f_n) = (f_1, f_2, ..., f_{i-1}, f_{i+1}, ..., f_n)$$

The actual ϕ_i would depend on the topology. For example, for the ring topology, it would be defined as $\phi_i(f_1, ..., f_n) = f_{i-1 \mod n}$ if $i \neq 1$, $\phi_i(f_1, ..., f_n) = f_n$ if $i = 1$. Therefore we can use Definition 1 as the security definition of the multi-party fair exchange, using the ϕ_i representing the desired topology.

Adversarial Model: When there is dispute between the parties, the TTP resolves the conflict *atomically*. We assume that the adversary cannot prevent the honest party(s) from reaching the TTP before the specified time interval. Secure channels are used to exchange the decryption shares and when contacting the TTP. The adversary may control up to $n-1$ out of n parties in the exchange, and is probabilistic polynomial time (PPT).

4 Description of the Protocol

Remember that our aim is to create efficient multi-party fair exchange protocols for every topology. The most important challenges of these kind of protocols are the following:

- Even if there are $n-1$ colluding parties, the protocol has to guarantee the fairness. Consider a simple protocol for the complete topology: each party first sends the verifiable escrow of the his/her item to the other parties, and after all the verifiable escrows are received, each of them sends the (plaintext) items to each other. If one of the parties comes to the TTP for resolution, the TTP decrypts the verifiable escrow(s) and stores the contacting party's item for the other parties.
  Assume now that P_i and P_j are colluding, and P_i receives verifiable escrow of the honest party P_h. Then P_i contacts the TTP, receives f_h via the decryption of the verifiable escrow of P_h, and gives his item f_i to the TTP. At this moment, if P_i and P_j leave the protocol before P_j sends his verifiable escrow to P_h, then fairness is violated because P_h never gets the item of P_j, whereas, by colluding with P_i, P_j also received f_h.
  Thus, it is important *not* to let a party learn some item *before all the parties are guaranteed* that they will get all the items. We used this intuition while designing our protocols. Therefore, we oblige **parties to depend on some input from every party in every phase of the protocol**. Hence, even if there is only one honest party, the dishonest ones have to contact and provide their correct values to the honest party so that they can continue with the protocol.
- It is desirable and more applicable to use a semi-honest TTP. Therefore, privacy against the TTP needs to be satisfied. For the example protocol above, privacy against the TTP is violated as well since the TTP learns the items of the parties.
- The parties do not receive or send any item to some of the other parties in some topologies (e.g., in the ring topology, P_2 receives an item only from P_1 and sends an item to P_3 only). Yet, a multi-party fair exchange protocol must ensure that either the whole topology is satisfied, or no party obtains any item. Previous protocols fail in this regard, and allow, for example P_2 to receive the item of P_1 as long as she sends her item to P_3, while it may be the case that P_4 did not receive the item of P_3. The main issue here is that, if a multi-party fair exchange protocol lets the topology to be partially satisfied, we might as well replace that protocol with multiple executions of two-party fair exchange protocols. The main goal of MFE is to ensure that either the whole topology is satisfied, or no exchange happens.

We succeed in overcoming the challenges above with our MFE protocol. We first describe the protocol for the complete topology for the sake of simplicity. Then, we show how we can use our MFE protocol for other topologies in Section 5. All zero-knowledge proof of knowledge protocols are executed non-interactively in the random oracle model [14].

4.1 Multi-Party Fair Exchange Protocol (MFE)

There is a trusted third party (TTP) that is involved in the protocol when a dispute happens between the participants about fairness. His public key pk is known to every participant.

Overview: The protocol has three phases. In the first phase, parties jointly generate a public key for the threshold encryption scheme using their private shares. This phase needs to be done only once among the same set of participants. In the second phase, they send to each other the verifiable encryptions of the items that they want to exchange. If anything goes wrong up till here, the protocol is aborted. In the final phase, they exchange

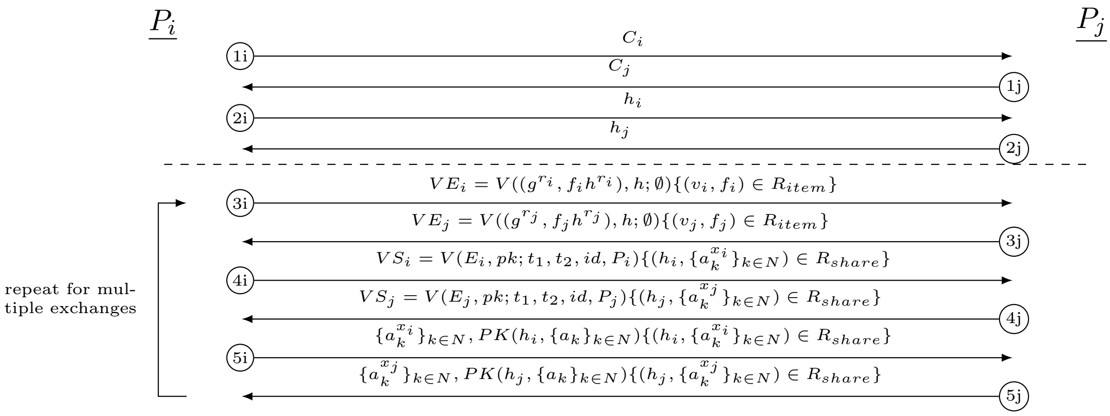

Figure 1: Our MFE Protocol. Each (i, j) message pair can be performed in any order or in parallel within a step.

decryption shares for each item. If something goes wrong during the final phase, resolutions with the TTP are performed. The details are below (see also Figure 1).

Phase 1 (① and ② in Figure 1): All participants agree on the prime p-order subgroup of $\mathbb{Z}_q^*$, where q is a large prime, and a generator g of this subgroup. Then each P_i does the following [47]:

- P_i randomly selects his share x_i from $\mathbb{Z}_p$ and computes the verification key $h_i = g^{x_i}$. Then he commits to h_i and sends the commitment C_i to other parties [47].
- After receiving all commitments from the other parties, P_i opens C_i and obtains all other parties' h_j.
  Note that this must be done after exchanging all the commitments, since otherwise we cannot claim independence of the shares, and then the threshold encryption scheme's security argument would fail. But with the two steps above, the security proof for threshold encryption holds here.
- After receiving all h_i values successfully, P_i computes the threshold encryption's public key

$$h = \prod_i h_i = \prod_i g^{x_i} = g^{\sum_i x_i} = g^x.$$

Phase 1 is executed only once. Afterward, the same set of parties can exchange as many items as they want by performing only Phase 2 and Phase 3.

Phase 2 (③ in Figure 1): Firstly, parties agree on two time parameters t_1 and t_2, and identification id of the protocol. (Time parameters can also be agreed in Phase 1.) Each participant P_i does the following:

- P_i sends a verifiable encryption of his item f_i as

$$VE_i = V((g^{r_i}, f_i h^{r_i}), h; \emptyset)\{(v, f_i) \in R_{item}\}$$

  where r_i is randomly selected from $\mathbb{Z}_p$. For the notation simplicity, we denote $(a_i, b_i) = (g^{r_i}, f_i h^{r_i})$. VE_i includes the encryption of the item f_i with public key h and it can be verified that the encrypted item f_i and the public value v_i has the relation R_{item}. Shortly, P_i proves he encrypts desired item. (e.g., if f_i is a signature on a contract, then v_i contains the signature verification key of P_i together with the contract, and R_{item} is the relation that f_i is a valid signature with respect to v_i.)
  Note that without knowing n decryption shares, no party can decrypt any VE_j and learn the items. Thus, if anything goes wrong up to this point, the parties can locally

abort the protocol. After this point, they need a fair exchange protocol to obtain all the decryption shares. This is done in the following phase.

Phase 3 (④ and ⑤ in Figure 1): No party begins this phase without completing Phase 2 and receiving all verifiable encryptions VE_j correctly.

- P_i sends to other parties a verifiable escrow VS_i that includes the decryption shares for each verifiable encryption VE_j. VS_i is computed as

$$VS_i = V(E_i, pk; t_1, t_2, id, P_i)\{(h_i, \{a_k^{x_i}\}_{1 \leq k \leq n}) \in R_{share}\}$$

where E_i is the encryption of $a_1^{x_i}, a_2^{x_i}, ..., a_n^{x_i}$ with the TTP's public key pk. The relation R_{share} is:

$$\log_g h_i = \log_{a_k} a_k^{x_i} \text{ for each } k. \quad (1)$$

Simply, the verifiable escrow VS_i includes the encryption of the decryption shares of P_i that will be used to decrypt the encrypted items of all parties. It can be verified that it has the correct decryption shares. In addition, only the TTP can open it. The label t_1, t_2, id, P_i contains the public parameters of the protocol, and P_i is just a name that the participant chooses. Here, we assume that each party knows the other parties' names.

Remark: The name P_i is necessary to show the VS_i belongs him. It is not beneficial to put a wrong name in a verifiable escrow's label, since a corrupted party can convince TTP to decrypt VS_i by showing P_i is dishonest. The other labels id, t_1, t_2 are to show the protocol parameters to the TTP. Exchange identifier id is necessary to prevent corrupted parties to induce TTP to decrypt VS_j for another exchange. Consider that some exchange protocol ended unsuccessfully, which means nobody received any item. The corrupted party can go to the TTP as if VS_j is the verifiable escrow of the next protocol, and have it decrypted, if we were not using exchange identifiers. We will see in our resolution protocols that **cheating in the labels do not provide any advantage to an adversary**. Furthermore, the party names can be random and distinct in each exchange, as long as the parties know each others' names, and hence it does not violate the privacy of the parties.

- P_i waits for VS_j from each P_j. If anything is wrong with some VS_j (e.g., verification fails or the label is not as expected), or P_i does not receive the verifiable escrow from at least one participant, he executes **Resolve 1** before t_1. Otherwise, P_i continues with the next step.
- P_i sends his decryption shares $(a_1^{x_i}, a_2^{x_i}, ..., a_n^{x_i})$ to each P_j. In addition, he executes the zero-knowledge proof of knowledge showing that these are the correct decryption shares

$$PK(h_i, \{a_k\}_{k \in N})\{(h_i, \{a_k^{x_i}\}_{1 \leq k \leq n}) \in R_{share}\}. \quad (2)$$

- P_i waits for $(a_1^{x_j}, a_2^{x_j}, ..., a_n^{x_j})$ from each P_j, together with the same proof that he does. If one of the values that he receives is not as expected or if he does not receive them from some P_j, he performs the **Resolve 2** protocol with the TTP, before t_2 and after t_1. Otherwise, P_i continues with the next step.
- After receiving all the necessary values, P_i can decrypt each VE_i and get all the items. The decryption for item f_j is as below:

$$b_j / \prod_k a_j^{x_k} = f_j h^{r_j} / g^{r_j \sum_k x_k} = f_j h^{r_j} / (g^{\sum_k x_k})^{r_j} = f_j h^{r_j} / h^{r_j} = f_j$$

4.1.1 Resolve 1

The goal of Resolve 1 is to *record* the corrupted parties that did *not* send their verifiable escrow in ④. Resolve 1 needs to be done **before $\mathbf{t_1}$**. Parties do *not* learn any decryption shares here. They can just complain about other parties to the TTP. The TTP creates a fresh *complaintList* for the protocol with parameters id, t_1, t_2. The *complaintList* contains the names of pairs of parties that have a dispute because of the missing verifiable escrow. The ***complainant*** is the party that complains, whose name is saved as the first of the pair, and the ***complainee*** is saved as the second of the pair. In addition, the TTP saves *complainee's verification key* given by the complainant; in the case that the complainee contacts the TTP, he will be able to prove that he is the complainee. See Algorithm 1.

Algorithm 1 Resolve 1

P_i sends id, t_1, t_2, P_j, h_j to the TTP where P_j is the party that did not send VS_j to P_i. The TTP does the following:
if *currenttime* $> t_1$ **then**
    send *msg* "Abort Resolve 1"
else
    complaintList = GetComplaintList(id, t_1, t_2)
    if *complaintList* == NULL **then**
        complaintList = CreateEmptyList(id, t_1, t_2) // initialize empty list
        solvedList = CreateEmptyList(id, t_1, t_2) // will be used in Resolve 2
    end if
    complaintList.add($P_i, (P_j, h_j)$)
    send *msg* "Come after t_1 for Resolve 2"
end if

4.1.2 Resolve 2

Resolve 2 is the resolution protocol where the parties come to the TTP to ask him to decrypt verifiable escrows and the TTP solves the complaint problems recorded in Resolve 1. The TTP does *not* decrypt any verifiable escrow until the *complaintList* is *empty*.

The party P_i, who comes for Resolve 2 **between $\mathbf{t_1}$ and $\mathbf{t_2}$**, gives all verifiable escrows that he has already received from the other parties and his own verifiable escrow to the TTP. The TTP uses these verifiable escrows to save the decryption shares required to solve the complaints in the *complaintList*. If the *complaintList* is not empty in the end, P_i comes after t_2 for Resolve 3. Otherwise, P_i can perform Resolve 3 immediately and get all the decryption shares that he requests.

Algorithm 2 Resolve 2

P_i gives $\mathcal{M}$, which is the set of verifiable escrows that P_i has. The TTP does the following:
complaintList = GetComplaintList (id, t_1, t_2)
for all VS_j in $\mathcal{M}$ **do**
    if $(*, (P_j, h_j)) \in$ *complaintList* AND CheckCorrectness(VS_j, h_j) is true **then**
        $shares_j$ = Decrypt(sk, VS_j)
        solvedList.Save($P_j, shares_j$)
        complaintList.remove($(*, (P_j, h_j))$)
    end if
end for
if *complaintList* is empty **then**
    send msg "Do Resolve 3"
else
    send msg "Come after t_2 for Resolve 3"
end if

CheckCorrectness(VS_j, h_j) returns *true* if the TTP can verify the relation in equation (1) using verifiable escrow VS_j and h_j. Otherwise it returns *false*.

4.1.3 Resolve 3

If the *complaintList* still has parties that have a conflict, even after t_2, the TTP answers each resolving party saying that the protocol is **aborted**, which means nobody is able to learn any item. If the *complaintList* is *empty*, the TTP decrypts any verifiable escrow that is given to him. Besides, if the complainants in the *solvedList* come, he gives the stored decryption shares. See Algorithm 3.

Algorithm 3 Resolve 3

P_i gives $\mathcal{C}$, which is the set of parties that did not perform step ④ or ⑤ with P_i, and $\mathcal{V}$, which is the set of verifiable escrows that belongs to parties in $\mathcal{C}$ who performed step ④. The TTP does the following:

- $complaintList$ = GetComplaintList (id, t_1, t_2)
- **if** $complaintList$.isEmpty() **then**
  - **for all** P_j in $\mathcal{C}$ **do**
    - **if** $VS_j \in \mathcal{V}$ **then**
      - **send** Decrypt(sk, VS_j)
    - **else**
      - **send** $solvedList$.GetShares(P_j)
    - **end if**
  - **end for**
- **else if** $currenttime > t_2$ **then**
  - **send** msg "Protocol is aborted"
- **else**
  - **send** msg "Try after t_2"
- **end if**

4.2 Security

Theorem 1. *The MFE protocol above is fair according to Definition 1, assuming that ElGamal threshold encryption scheme is a secure threshold encryption scheme, the associated verifiable escrow scheme is secure, all commitments are hiding and binding, and the discrete logarithm problem is hard (so that the proofs are sound and zero-knowledge).*

Proof. Assume the worst-case that adversary $\mathcal{A}$ corrupts $n-1$ parties. The simulator S simulates the honest party in the real world and the corrupted parties in the ideal world. S also acts as the TTP in the protocol if any resolution protocol occurs, so S publishes a public key pk as the TTP, and knows the corresponding secret key. Without loss of generality assume that the parties $\{P_i\}_{2\leq i\leq n}$ are the corrupted parties and S simulates the behavior of the honest party P_1. S does the following:

Phase 1: S behaves the same as in the real protocol.

Phase 2:

- Phase 2 is almost the same as in the protocol. Since S does not know the item f_1, he encrypts a random item $\tilde{f_1}$ and sends verifiable encryption $\tilde{VE_1}$ to the other parties.
- S waits for the verifiable encryptions of the corrupted parties $\{VE_i\}_{2\leq i\leq n}$. S does not continue to the next step until he receives all VEs. When a party sends his verifiable encryption, S behaves as the verifiable encryption extractor and learns the party's item.

In the end of this step, S learns all the items $\{f_i\}_{2\leq i\leq n}$ of the corrupted parties. Note that U will immediately let the ideal honest party obtain her output when S sends the corrupted parties' inputs. Since it is not guaranteed at this point that the real honest party (that S is simulating) will receive the desired items, S does not contact U immediately, even though he has enough knowledge to do so.

Phase 3: S behaves as in Phase 3. Additionally, he learns decryption shares $\{a_k^{x_i}\}_{1\leq k\leq n}$ of a party P_i when P_i sends VS_i to S. One of the following situations must have happened:

(i) All corrupted parties have already sent their VS to S.

(ii) Some of the corrupted parties did not send their VS to S (or sent incorrect VS).

We analyze these two cases.

(i) means it is guaranteed that the real honest party would obtain her desired items, because S in the real world is now able to learn all the decryption shares from the corrupted parties via resolutions. Therefore, S sends all items $\{f_i\}_{2\leq i\leq n}$ he learned in Phase 2 to U. Then, U sends f_1 to S. At this point, S should send his decryption shares to the other parties. However, S has sent encryption of a random item $\tilde{f_1}$ and so he does not have the real decryption share d_1. Therefore, he calculates Equation 3 to find the appropriate decryption share d_1 such that the other parties can get the item f_1 from a_1, b_1 using d_1. The other decryption shares $\{d_i\}_{2 \le i \le n}$ are calculated as $a_i^{x_1}$ as in the real protocol.

$$d_1 = \frac{b_1}{f_1 a_1^{x_2} ... a_1^{x_n}} \tag{3}$$

Here, d_1 value represents $a_1^{x_1}$ in the real protocol. He can do this calculation because he learned f_1 from U, and $\{a_1^{x_j}\}_{2 \le j \le n}$ while the corrupted parties are sending their own verifiable escrows.

S sends the d_i values to all of the parties and simulates the proof of knowledge showing d_i is correct. S also waits decryption shares from the corrupted parties. If all of them send their own decryption shares with a valid proof of knowledge showing that they sent the correct decryption shares, then the simulation ends.

If some parties does not send their decryption shares to S before t_2, S does Resolve 2 as in the real protocol and clears the *complaintList* because he has all the verifiable escrows of the corrupted parties.

(ii) requires S to behave as the TTP and add the corrupted parties who did not send their verifiable escrows to the *complaintList*, because in reality the honest party would have complained about them before t_1 in Resolve 1. In addition, if a corrupted party does Resolve 1, S behaves like the TTP and adds him and his complainee to the *complaintList*.

Moreover, S does not send any of his decryption shares, as in the real protocol. If some of the corrupted parties comes for Resolve 2, S behaves exactly as the TTP and clears the parties from the *complaintList* according to the given verifiable escrows. Each time he is clearing the *complaintList*, he learns the decryption shares of the complainee. He himself can perform Resolve 2 and clear some parts of the *complaintList* where he already received the corresponding verifiable escrows. In the end, if the *complaintList* is empty, it means that he learned all the decryption shares of the corrupted parties. If so, S sends all the items $\{f_i\}_{2 \le i \le n}$ of the corrupted parties to U. Then U sends f_1 to S and S calculates his shares d_1 as in Equation (3). Therefore, when parties come for Resolve 3, S can give every share that they want. The simulation ends.

If *complaintList* is not empty at time t_2, S sends message ABORT to U and all Resolve 3 attempts will return an abort message.

Claim 1. *The view of adversary $\mathcal{A}$ in his interaction with the simulator S is indistinguishable from the view in his interaction with a real honest party.*

Proof: S behaves different from the real protocol while sending $\tilde{VE}_1$ and $\tilde{VS}_1$. $\tilde{VE}_1$ is indistinguishable from the real one because of the security of the ElGamal encryption scheme [22]. Similarly, $\tilde{VS}_1$ is indistinguishable from the real one because of the security of the verifiable escrow [17, 4]. Besides, S acts as the TTP without any difference, so the interaction with the TTP is indistinguishable too. The outputs of the parties at the end of the protocol are identical to the real protocol. □

5 All Topologies for MFE

In this section, we adapt our MFE protocol to every topology. Our fairness definition remains the same: either the whole topology is satisfied, or no party learns any item. As an example, consider the ring topology as in Figure 3. Parties want an item from only the previous party. For example, P_2 only wants P_1's item f_1. However, P_2 should contact all other parties because of our all-or-none fairness condition. Besides, we are not limited with a topology that follows a specific pattern such as the number of parties and items being necessarily equal. For example, it is possible to provide fairness in the topology in Figure 5 even though P_2, P_3, and P_4 do not have exchange item with each other.

	f_1	f_2	f_3	f_4
P_1		$\odot$		
P_2			$\odot$	
P_3				$\odot$
P_4	$\odot$			

Figure 2: Desired items by each parties in matrix form in the ring topology.

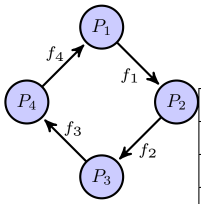

Figure 3: Graph representation of the ring topology.

	f_1	f_2	f_3	f_4	f_5
P_1		$\odot$	$\odot$	$\odot$	$\odot$
P_2	$\odot$				
P_3	$\odot$				
P_4	$\odot$				

Figure 4: Matrix representation of a topology.

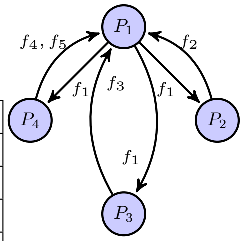

Figure 5: Graph representation of a topology in Figure 4.

Consider some arbitrary topology described by the matrix in Figure 6. If a party desires an item from another party, he should have all the shares of the item as shown in Figure 7. In general, we can say that if a party P_i wants the item f_t he should receive $\{a_t^{x_j}\}_{\{1 \leq j \leq n\}}$ from all the parties $\{P_j\}_{\{1 \leq j \leq n\}}$. Therefore, our MFE can be applied to any topology with the same fairness condition, which is **all parties will receive all their desired items or none of them receives anything** in the end of the protocol.

	f_1	f_2	f_3	f_4	f_5
P_1	$\odot$		$\odot$		
P_2	$\odot$			$\odot$	$\odot$
P_3	$\odot$		$\odot$		
P_4	$\odot$	$\odot$	$\odot$		$\odot$

Figure 6: Desired items by each parties in matrix form. Each party wants the marked items that corresponds to his/her row. P_i has f_i, except P_4 has both f_4 and f_5.

	f_1	f_2	f_3	f_4	f_5
P_1		$\{a_2^{x_i}\}$	$\{a_3^{x_i}\}$		$\{a_5^{x_i}\}$
P_2	$\{a_1^{x_i}\}$			$\{a_4^{x_i}\}$	$\{a_5^{x_i}\}$
P_3	$\{a_1^{x_i}\}$		$\{a_3^{x_i}\}$		
P_4	$\{a_1^{x_i}\}$	$\{a_2^{x_i}\}$	$\{a_3^{x_i}\}$		$\{a_5^{x_i}\}$

Figure 7: Necessary shares for each party to get the desired items that are shown in Figure 6. Sets are over $i \in \{1, 2, ..., 5\}$

Our strong fairness condition requires that all parties have to depend each other. Even though P_i does not want an item f_j from P_j, getting his desired item has to also depend on P_j. Therefore we cannot decrease number of messages even in a simpler (e.g., ring) topology.

On the other hand, the size of the verifiable escrow, meaning that the number of shares in the verifiable escrow, decreases in topologies other than the complete one. If we represent the topology in a matrix form as in Figure 6, each party P_i has to add the number of $\odot$ many shares corresponding to the row of the party P_j to the verifiable escrow that is sent to P_j. We can conclude that the total size of the verifiable escrows that a party sends is $O(\#\odot)$ where $\odot$ is as in Figure 6.

6 Efficient Fair Secure Multi-Party Computation

In this section, we show how to adapt the MFE protocol to **any** secure multi-party computation (SMPC) protocol [12, 29, 19, 9, 50] to achieve fairness.

Assume n participants want to compute a function $\phi(w_1, ..., w_n) = (\phi_1(w_1, ..., w_n), ..., \phi_n(w_1, ..., w_n))$, where w_i is the input and $\phi_i = \phi_i(w_1, ..., w_n)$ is the output of party P_i.

- P_i randomly chooses a share $x_i \in \mathbb{Z}_p$. Then P_i gives his share and w_i to an SMPC protocol that outputs the computation of the functionality ψ where $\psi_i(z_1, z_2, ..., z_n) =$

$(E_i(\phi_i(w_1, ..., w_n)), \{g^{x_j}\}_{1 \leq j \leq n})$ is the output to, and $z_i = (w_i, x_i)$ is the input of P_i. This corresponds to a circuit encrypting the outputs of the original function ϕ using the shares provided as input, and also outputting the verification shares of all parties to everyone. Encryption E_i is done with the key $h = g^{\sum_{j=1}^{n} x_j}$ as follows:

$$E_i(\phi_i(w_1, ..., w_n)) = (g^{r_i}, \phi_i h^{r_i})$$

where $r_i \in \mathbb{Z}_p$ are random numbers chosen by the circuit (or they can also be inputs to the circuit), similar to the original MFE protocol.
It is expected that everyone learns the output of ψ before a fair exchange occurs. If some party did not receive his output at the end of the SMPC protocol, then they do not proceed with the fair exchange, and hence no party will be able to decrypt and learn their output.

- If everyone received their output from the SMPC protocol, then they execute the Phase 3 of the MFE protocol above, using g^{x_i} values obtained from the output of ψ as verification shares, and x_i values as their secret shares. Furthermore, the a_i, b_i values are obtained from E_i.
Note that each function output is encrypted with all the shares. But, for party P_i, she need not provide her decryption share for f_i to any other party. Furthermore, instead of providing n decryption shares to each other party as in a complete topology, she needs to provide only one decryption share, $a_j^{x_i}$, to each P_j. Therefore, the Phase 3 of MFE here is a more efficient version. Indeed, the verifiable escrows, the decryption shares, and their proofs each need to be only on a *single* value instead of n values.

Phases 1 and 2 of the fair exchange protocol have already been done during the modified SMPC protocol, since the parties get the encryption of the output that is encrypted by their shares. Since the SMPC protocol is secure, it is guaranteed to output the correct ciphertexts, and we do not need further verification. We also do not need to first commit to x_i values, since the SMPC protocol ensures independence of inputs as well. So, the parties only need to perform Phase 3.

In the end of the exchange, each party can decrypt only their own output, because they do not give away their own output's decryption share to anyone else. Indeed, if a symmetric functionality is desired for SMPC, $\psi(z_1, z_2, ..., z_n) = (\{E_i(\phi_i(w_1, ..., w_n)), g^{x_i}\}_{1 \leq i \leq n})$ may be computed, and since P_i does not give the decryption share of f_i to anyone else, each party will still only be able to decrypt their own output. Therefore, ***a symmetric functionality SMPC protocol may be employed to compute an asymmetric functionality fairly using our modification***. Note also that we view the SMPC protocol as *black box*.

Our overhead over performing *unfair* SMPC is minimal. Even though the input and output sizes are extended additionally by $O(n)$ values and the circuit is extended to perform encryptions, these are minimal requirements, especially if the underlying SMPC protocol works over *arithmetic circuits* (e.g., [9, 50]). In such a case, performing ElGamal and creating verification values g^{x_i} are very easy. Afterward, we only add two rounds of interaction for the sake of fairness (i.e., Phase 3 of MFE, with smaller messages). Moreover, all the benefits of our MFE protocol apply here as well.

Theorem 2. *The SMPC protocol above is fair and secure according to Definition 1 for the functionality ϕ, assuming that ElGamal threshold encryption scheme is a secure, the discrete logarithm assumption holds, and the underlying SMPC protocol that computes functionality ψ is secure.*

Proof. Assume that adversary $\mathcal{A}$ corrupts $n-1$ parties, which is the worst possible case. The simulator S simulates the honest party in the real world and the corrupted parties

Solutions	Technique	TTP	Number of Rounds	Proof Technique
[26]	Gradual Release	No	$O(\lambda)$	NFS
[13]	Bitcoin	Yes	Constant ✓	NFS
[1]	Bitcoin	Yes	Constant ✓	NFS
Ours	MFE	Yes	Constant ✓	FS ✓

Table 3: Comparison of our fair SMPC solution with previous works. NFS indicates simulation proof given but not for fairness, FS indicates full simulation proof including fairness, and λ is the security parameter.

in the ideal world. Without loss of generality, assume that the parties $\{P_i\}_{2 \leq i \leq n}$ are the corrupted parties and S simulates the honest party P_1. S does the following:

- Simulator S chooses a random input $\tilde{w}_1$ and share x_1. Then S acts as the simulator of the underlying SMPC protocol; call that simulator S'. He learns the inputs of the corrupted parties while he is acting as S', and evaluates the circuit together with the corrupted parties, using $\tilde{w}_1, x_1$ as its input.
  We view S' almost as a black box. The only different behavior we request from S' is that, instead of providing the inputs of the corrupted parties he learned to the universal trusted party U and learning their outputs immediately, we need S' to finish its simulation without contacting U and instead using its random input $\tilde{w}_1, x_1$. The reason we call this change almost black box is that it will be indistinguishable to the adversary due to the security of the ElGamal threshold encryption scheme. Even though the simulator computed different outputs for the original functionality ϕ, since the computed functionality ψ outputs encrypted values, these will be indistinguishable (otherwise a simple reduction breaks the security of ElGamal encryption). During MFE, the simulator S will learn the real outputs, and fake its share such that the decryptions of ψ_i outputs would result in actual ϕ_i outputs obtained from U. The output verification shares are distributed identically, and there is no problem there as well.
  Remark: We need S' not to contact U due to the fairness simulation requirement of Definition 1. Note that the simulator S is allowed to learn the outputs from U only once it is guaranteed that all parties will learn their outputs. Otherwise, the real outputs would be distinguishable from the ideal.
- Once S' is done, all parties learned some encrypted output together with the verification shares. At this point, these encryptions contain random values. Now S continues simulating the Phase 3 of MFE. If during that simulation, it is guaranteed that all parties will learn their outputs, then he contacts U, providing the inputs of the corrupted parties S' extracted, and obtains the outputs ϕ_i from U. Accordingly, S calculates decryption shares d_i for each corrupted party using the Equation (4). Then he sends (differently from the MFE proof) only d_i to P_i. Note that, S cannot calculates $\{d_i\}_{2 \leq i \leq n}$ as in the simulation of MFE protocol because the encryptions ψ_is do not include correct output values ϕ_is since S uses a random input $\tilde{w}_1$ as an input of SMPC protocol.

$$d_i = \frac{b_i}{\phi_i a_i^{x_2} ... a_i^{x_n}} \tag{4}$$

The behavior of S is indistinguishable since he acts as a simulator of SMPC and MFE. This completes the proof. □

7 Performance and Privacy Analysis of the Protocols

MFE: Each party P_i in MFE prepares one verifiable encryption and one verifiable escrow, and sends them to $n-1$ parties. The verification of them are efficient because the relation they show can be proven using discrete-logarithm-based honest-verifier zero-knowledge three-move proofs of knowledge [20]. In the end, P_i sends a message including decryption shares to $n-1$ parties, again with an efficient proof of knowledge. So, for each party P_i, the number of messages that he sends is $O(n)$. Since there are n parties, the total message complexity is $O(n^2)$. Note that there is *no* requirement to have these messages broadcast; just ensuring all previous step's messages are received before moving further is enough for security. Table 1 shows the comparison to the previous works, MFE is much more efficient, obtaining **optimal asymptotic efficiency**.

When there is a malicious party or a party suffering from network failure, MFE protocol ends at the latest, immediately after t_2. In the worst case, n parties contact the TTP, so it is important to reduce his workload. TTP's duties include checking some list from his records, verifying efficient zero-knowledge proofs of knowledge from some number of parties (depending on the size of the *complaintList*), and decrypting verifiable escrows. These actions are all efficient.

Moreover, the **privacy against the TTP is preserved**. He just learns some decryption shares, but he cannot decrypt the encryption of exchanged items, since he never gets the encrypted items.

We used ElGamal threshold encryption for presentation simplicity. Instead, any threshold encryption scheme such as the Pailler cryptosystem [46], Franklin and Haber's cryptosystem [24], or Damgard-Jurik cryptosystem [21] can be employed.

Finally, our MFE protocol achieves the intuitive fairness definition of 'either the whole topology is satisfied, or no item is exchanged' for any topology. Such a strong fairness definition necessitates that the exchanges depend on all parties, necessitating quadratic number of messages.

Fair MPC: The overhead of our fairness solution on top of an existing unfair SMPC protocol is increased input/output size, and additional computation of encryptions and verification shares. If an arithmetic circuit is used in the underlying SMPC protocol [9, 50, 19], then there are only $O(n)$ additional exponentiations required, which does not extend circuit size a lot. If boolean circuits are used, the size of the circuit increases more than an arithmetic circuit would have, but it is still tolerable, especially considering in comparison to the related work.

As seen in Table 3, [26] uses gradual release for fairness. However, this brings many extra rounds and messages to the protocol. Each round each party releases his item by broadcasting it. Recent, bitcoin-based approaches [13, 1] also require broadcasting in the bitcoin network, which increases message complexity. Our only overhead is a constant number of rounds, and $O(n^2)$ messages. Remember again that these are asymptotically optimal, since fair SMPC necessitates a complete topology.

Acknowledgements

The authors acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 111E019, as well as European Union COST Action IC1306.

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Symposium on Security and Privacy*, 2014.

[2] G. Asharov, Y. Lindell, and H. Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *ASIACRYPT*, 2013.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for multi-party fair exchange, 1996.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, pages 591–610, 2000.

[5] G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.

[6] G. Ateniese and C. Nita-rotaru. Stateless-recipient certified e-mail system based on verifiable encryption. In *In: CT-RSA'02, LNCS 2271*, 2002.

[7] F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy*, 1998.

[8] F. Bao, R. H. Deng, K. Q. Nguyen, and V. Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *DEXA Workshop*, 1999.

[9] C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Report 2014/075.

[10] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *ICALP*, 2000.

[11] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *WPES*, 2008.

[12] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM STOC*, 1988.

[13] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*.

[14] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *ACM STOC*, 1988.

[15] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO*, 2000.

[16] J. Camenisch and I. Damgård. Verifiable encryption and applications to group signatures and signature sharing. Technical report, 1999.

[17] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.

[18] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, pages 143–202, 2000.

[19] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, 2001.

[20] I. Damgård. On sigma protocols. http://www.daimi.au.dk/ ivan/Sigma.pdf.

[21] I. B. Damgård and M. J. Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP*, 2003.

[22] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1985.

[23] S. Even and Y. Yacobi. Relations among public key signature scheme. Technical

report, Department of Computer Science, TechUnion, 1980.

[24] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. In *CRYPTO*, 1994.

[25] M. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *FC*, 1998.

[26] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *Theory of Cryptography*. 2006.

[27] J. A. Garay, M. Jakobsson, and P. D. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO*, 1999.

[28] J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In *DISC*, 1999.

[29] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[30] N. González-Deleito and O. Markowitch. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *ICISC*, 2002.

[31] S. D. Gordon. Complete fairness in multi-party computation without an honest majority. In *TCC*, 2009.

[32] M. Hirt and D. Tschudi. Efficient general-adversary multi-party computation. In *ASIACRYPT*, 2013.

[33] H. Kılınç and A. Küpçü. Efficiently making secure two-party computation fair. Cryptology ePrint Archive, Report 2014/896.

[34] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, 2010.

[35] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 2012.

[36] A. Küpçü and A. Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *Computer Security–ESORICS 2010*. Springer, 2010.

[37] A. Y. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, 2008.

[38] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[39] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 2009.

[40] Y. Liu and H. Hu. An improved protocol for optimistic multi-party fair exchange. In *EMEIT*, 2011.

[41] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In *ICISC*, 2000.

[42] S. Mauw, S. Radomirovic, and M. T. Dashti. Minimal message complexity of asynchronous multi-party contract signing. In *CSF*, 2009.

[43] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, 2003.

[44] A. Mukhamedov and M. D. Ryan. Fair multi-party contract signing using private contract signatures. *Inf. Comput.*, pages 272–290, 2008.

[45] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, 1999.

[46] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[47] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, 1991.

[48] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, pages 75–96, 2002.

[49] A. C. Yao. Protocols for secure computations. In *FOCS*, 1982.

[50] M. Zamani, M. Movahedi, and J. Saia. Millions of millionaires: Multiparty computation in large networks.