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

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

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

Abstract

Multi-party fair exchange (MFE) and fair secure multi-party computation (fair SMPC) are is under-studied field of research, with practical importance. In particular, we consider MFE scenarios where at the end of the protocol, either every participant receives every other participant's item, or no participant receives anything. 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 against the TTP. In the fair SMPC case, we prove that a malicious TTP can only harm fairness, but not *security*.

We construct two *asymptotically optimal* multi-party fair exchange protocols that require 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. In one protocol, we enable the parties to efficiently exchange any item that can be efficiently put into a verifiable encryption (e.g., signatures on a contract). We show how to apply this protocol on top of *any* SMPC protocol to achieve fairness with very little overhead (independent of the circuit size), especially if the SMPC protocol works with arithmetic circuits. In our other protocol, we let the parties exchange any verifiable item, without the constraint that it must be efficiently put into a verifiable encryption (e.g., a file cannot be efficiently verifiably encrypted, but if its hash is known, once obtained, the file can be verified). We achieve this via the use of electronic payments, where if an item is not obtained, the payment of its owner will be obtained in return of the item that is sent. We then generalize our protocols to efficiently handle any exchange topology (participants exchange items with arbitrary other participants). Our protocols guarantee fairness in its strongest sense: even if all $n-1$ other participants are malicious and colluding with each other, the fairness is still guaranteed.

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

*A preliminary version of this work appeared on CT-RSA 2015 [59].

[†]Work mostly done at Koç University. Email: handan.kilinc@epfl.ch

[‡]Email: akupcu@ku.edu.tr

Contents

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-party computation protocols to achieve fairness [66, 58, 64].

Even in two-party fair exchange scenarios, having fairness completely and efficiently without a trusted third party (TTP) is shown to be impossible in general [38, 76]. The main intuition of this impossibility 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 causing fairness problems. In an *optimistic* protocol, the TTP is involved in the protocol *only* when a violation of fairness occurs [7, 8]. However, it is important not to give a lot of work to the TTP, since this can cause a performance bottleneck. Furthermore, the TTP is required *only* for *fairness*, and should not learn more about the exchange than what is necessary 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 [8, 16, 61] have been proposed and even applied to secure two-party computation [21, 58, 64], but there do not exist efficient and general solutions for the multi-party case. Multi-party fair exchange (MFE) can be described based on *exchange topologies* (nothing to do with network topologies) that denote the directed graph of item exchanges. For example, a *ring topology* describes an MFE scenario where each party receives an item from the previous party in a ring [13, 67, 49, 67]. A common scenario with the ring topology is that a customer 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. Our fairness understanding dictates that either the whole ring completes, finishing the transaction, or no participant receives anything.

However, ring topology cannot be directly 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 output of SMPC being revealed only when every participant is able to receive it [59]. This is called a *complete topology*. We can think of the parties as nodes in a complete graph and the edges between parties show the exchange links (*not* communication links). The complete topology was researched mostly in the contract-signing setting [44, 15, 43], with one exception [7]. Unfortunately, all these protocols are inefficient compared to ours (see Table 2). Because of the lack of 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[42], or require the use of bitcoin-like payment systems [18, 2, 57]. For the two-party case, Gordon et al. [51] showed that essentially employing a fair exchange protocol on top of an unfair secure computation protocol would yield a fair and secure computation protocol. While it is possible to extend their results to the multi-party setting, we provide an efficient concrete solution with several advantages with our MFE.

Our Contribution: We propose two new optimistic multi-party fair exchange protocols that efficiently guarantee fairness in *any* exchange topology (including complete topology). Our protocols are constructed on top of generic cryptographic primitives and they have efficient instantiations. The first one is called MFE, and the other one is called CMFE (for Coined-MFE). The difference between MFE and CMFE is the fairness definition. Fairness in MFE means that either all parties receive all their desired items (according to the topology), or none of them receives any item at the end of the protocol. On the other hand, the fairness in CMFE is more relaxed, in the sense that if a party sends his item but in return he does not receive an item, he obtains monetary compensation (such relaxed fairness definitions exist in the two-party literature [16, 65, 61]).We then show how to apply our MFE protocol on top of any SMPC protocol to obtain a fair SMPC solution efficiently and securely, where the TTP cannot harm security. In addition, we have the following results:

- MFE and CMFE both require only $\mathbf{O(n^2)}$ messages and **constant** number of rounds for n parties, being much more efficient than the previous works (see Table 2). These are **asymptotically optimal** in a complete topology[1], since each party should send his item to all other parties, even in an unfair exchange protocol. Furthermore, *none* of our protocols necessitate a *broadcast*.
- Our protocols **optimally** (still having the complexity $O(n^2)$) guarantee fairness (against honest parties) even when $n-1$ out of n parties are malicious and colluding. To be able to prove this, we formulate MFE and CMFE as fair and secure multi-party computation protocols. We then **prove** their security and fairness **via ideal-real world simulation**. To the best of our knowledge, no multi-party fair exchange protocol was proven as a secure and fair SMPC protocol with the ideal-real simulation before.
- As an additional real-world usage, we show how to adapt our MFE protocol on top of a secure multi-party computation protocol with abort and obtain a fair SMPC protocol. For this, we employ the methodology of Gordon et al. [51]. In addition, we prove via simulation that even if the TTP employed in the adapted fair SMPC protocol acts maliciously and colludes with the participants, **only fairness** can be violated but **not** the privacy and the correctness (security) of the underlying computation.
- The TTP for fairness in MFE and CMFE is in the *optimistic* model [8]. The TTP has a very low workload; verifying efficient zero-knowledge proofs and signatures, and performing simple decryptions. More importantly, the TTP does *not* learn any exchanged item, so the **privacy against the TTP** is preserved.
- We show how to employ MFE and CMFE protocols for **any exchange topology**, with the performance improving as the topology gets sparser (fewer computations are done). Moreover, unlike some earlier works [12, 49, 67], we argue that fair exchange must mean that **either the whole topology is satisfied, or no participant receives anything**.
- Our protocols have easy setup phases, which are employed only **once** for exchanging **multiple** sets of items, thus improving efficiency even further for *repeated exchanges* among the same set of participants. The topology can change between repetitions, without the need to redo the setup.
- CMFE is the **first** protocol that enables multiple parties to exchange verifiable items that cannot be efficiently put in a verifiable encryption. For example, files may be

[1]We believe this optimality holds for every topology, but we have not proven it yet, so we do not claim optimality for other topologies in this paper.

verifiable via their hashes, but still there is no efficient way of putting a large file into a public-key verifiable encryption. Our CMFE protocol enables very efficient exchange of any size of files, by relaxing the fairness definition and employing electronic payments.

- MFE can be constructed with a verifiable deniable public-key threshold encryption scheme and a verifiable public-key encryption scheme. CMFE can be constructed with the same cryptographic primitives as MFE as well as a symmetric encryption scheme and a signature scheme. We also discuss efficient instantiations in the random oracle model and show performance measurements.
- As an independent contribution, we define and use a verifiable deniable (k,n)-threshold encryption scheme and show in the Appendix B that ElGamal threshold scheme leads to an efficient instantiation.

Overview of Techniques: Our MFE protocol starts with a setup phase where the parties jointly generate a reusable public-key for a threshold encryption. Then, they exchange encrypted items. Until this point, parties can locally abort if anything goes wrong. Next, they exchange decryption shares for the threshold encryption. During this phase, they need each others' help, and all parties must seek help from all other parties. They may also employ the TTP's help to resolve disputes. While an honest TTP guarantees fairness, a malicious TTP may break fairness by selectively helping some participants.

Our fair SMPC solution essentially performs the first two steps (setup and encrypted item exchange) within the underlying unfair SMPC protocol execution. The underlying SMPC protocol returns encrypted items (where items correspond to the functionality outputs here) and verification shares such that the parties can verify that the obtained decryption shares will be valid. Then, they continue from the last phase of our MFE protocol where they fairly exchange the decryption shares, potentially employing the TTP's help. Since we achieve privacy against the TTP and ensure that each party depends on each other party at every phase of our protocols, even a malicious TTP that may collude with other parties cannot enable the adversary to break privacy of the underlying unfair SMPC protocol. A malicious TTP can break fairness, but not privacy and correctness. Thus, even when the TTP colludes with some participants, our protocol provides security equivalent to the standard security with abort, thereby not having any disadvantage.

Our CMFE protocol extends the setup phase employing an e-cash (or similar payment) system, and employs digital signatures to link messages of the fair exchange protocol with the monetary compensation. The public-keys of the signatures can be put into verifiable escrows, thereby removing the need for a public-key infrastructure. If a party cheats, the affected parties may receive help from the TTP to obtain the cheating party's payment as compensation. A corrupted TTP may break fairness.

In our dispute-resolution parts with the TTP, we employ timeout-based mechanisms. Since these timeouts do not affect an honest run of the protocol, but only affect the dispute resolution, the timeout values can be large (e.g., on the order of hours), removing the need for tight synchronization of parties' clock. Lastly, in Section 7, we provide concrete efficiency numbers for our protocols employing efficient underlying primitives, showing the practicality of our solutions.

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 relevant case is the optimistic case, where a trusted third party (TTP) exists, but the TTP is not involved if both participants are honest [8, 13, 9, 10, 73, 35, 61].

Multi-party Fair Exchange: Franklin and Tsudik [40] 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. [7] described an optimistic fair exchange protocol for all topologies. Their work is the only previous multi-party fair exchange protocol that works in all topologies. They defined a description matrix to represent the topology, and proposed a fair protocol. The parties are restricted only to "exchangeable items", requiring the TTP to be able to replace or revoke the items, greatly decreasing the applicability of the protocol. In addition, the protocol needs broadcast to send the items, rendering the protocol inefficient.

Table 1: Efficiency comparison with previous works in the ring topology. 'All or None' represents our fairness definition, where either the whole topology is satisfied, or no exchange occurs, and n is the total number of parties. TTP-party dependency exists if the TTP has to contact with a prespecified party to resolve the fairness problem with another party. TTP privacy exists if the TTP does not learn any information about the identity of the parties and exchanged items.

	Num. Messages	**All or None**	**TTP-Party Dependency**	**TTP Privacy**
Bao et al. [12]	$O(n)$	No	Yes	Not Private
González-Deleito and Markowitch [49]	$O(n^2)$	No	Yes	Not Private
Liu and Hu [67]	$O(n)$	No	Yes	Not Private
Ours	$O(n^2)$	Yes ✓	No ✓	Private ✓

Ring Topology: All existing multi-party fair exchange protocols, except Asokan et al. [7] protocol, work in the ring topology. Bao et al. [12] 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 [40], which means that a dishonest party may conspire with an honest party without the latter's consent.

Later, Gonzales-Deleito and Markowitch [49] solved the malicious initiator problem of Bao et al. [12]. 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 the previous participant will be available. The protocol in [67] solves the passive conspiracy problem of Bao et al. [12], however the problem in the recovery protocol still remains.

Markowitch and Kremer [68] proposed a non-repudiation protocol, where their fairness definition is that whenever one of the parties sends some information to the other parties, neither the sender nor the others can deny that they participated. However, it does not solve the fairness problem in general.

Table 2: Efficiency comparison with previous works in complete topology. MPCS denotes multi party contract signing, n is the total number of parties, and t is the number of dishonest parties. A checkmark (✓) indicates the best solution in that column for easy comparison.

	Solution for	**Topology**	**Num. Rounds**	**Num. Messages**	**Broadcast**
Garay and MacKenzie [44]	MPCS	Complete	$O(n^2)$	$O(n^3)$	Yes
Baum-Waidner and Waidner [15]	MPCS	Complete	$O(tn)$	$O(tn^2)$	Yes
Mukhamedov and Ryan [74]	MPCS	Complete	$O(n)$	$O(n^3)$	Yes
Mauw et al. [70]	MPCS	Complete	$O(n)$	$O(n^2)$✓	Yes
Asokan et al. [7]	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^3)$	Yes
Ours	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^2)$ ✓	No ✓

In all these ring-topology fair-exchange protocols, no formal fairness proofs exist. Instead, case-by-case analyses are done to show that fairness is satisfied.

Understanding Fairness: There is an important difference between our understanding of fairness, and existing ring-topology solutions [12, 49, 67]. According to their definition, in the end of the protocol, there will be no honest 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. For instance, consider a ring topology with parties P_0, P_1, P_2, P_3 and P_4, and assume that P_1 and P_3 are malicious and colluding. In this case, the following scenario can happen using the previous understanding: P_0 receives an item from P_4 and sends an item to P_1, and similarly P_4 receives an item from P_3 and sends an item to P_0. However, P_2 did not send or receive any item. Whereas, according to our definition, we need that **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. Otherwise, multi-party fair exchange protocols can be achieved via multiple executions of two-party fair exchange protocols. We further observe that this **all-or-none** type of fairness also requires a quadratic number of messages, at least in the complete topology, which we achieve optimally. Table 1 summarizes comparison for the ring topology.

Complete Topology: Since a multi-party fair exchange protocol in complete topology does not exist, except the protocol by Asokan et al. [7], we also look at multi-party contract signing protocols (MPCS), which can be considered as fair exchange protocols where the exchanged items are signatures on a common message called the contract. They indeed correspond to a complete topology. Garay and Mackenzie [43] proposed the first optimistic multi-party

contract signing protocol that requires $O(n^2)$ rounds and $O(n^3)$ messages. Baum-Waidner and Waidner [15] suggested a more efficient protocol, whose complexity depends on the number of dishonest parties, and if the number of dishonest parties is $n-1$ (or $O(n)$), its efficiency is the same as [43]. Mukhamedov and Ryan [74] decreased the round complexity to $O(n)$. Lastly, Mauw et al. [70] 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.**

All these MPCS protocols are based on some multiple-level promise system, which can be considered as a commitment of the signed contract. The promise can be converted into the signature of the contract by the TTP. The complicated conditions for resolutions and aborting make their fairness analysis harder. Indeed, the first optimistic MPCS protocol by Garay and Mackenzie [44] is later found not to be fair when the number of parties is more than five [74]. Table 2 already summarized this result. Mauw and Radomirović [69] uses different approach for MPCS which uses skeletal graph. Even though MPCS is considered mostly in complete topologies, Draper et al. [36] also analyze the other topologies for contract signing and informally finds an optimal bound for every topology which is $O(n^2)$, supporting our optimality claim.

Overall, our MFE and CMFE protocols are the first protocols that benefit from the nice properties of a verifiable deniable threshold encryption scheme to achieve fairness. Thanks to the verifiable deniable (n,n)-threshold encryption scheme:

- We make all parties depend on each other easily (i.e., they need decryption shares of the other parties to obtain the exchanged items). For example, existing MPCS protocols [15, 70, 74, 15, 44] use the multiple levels of promises technique to have this dependency, but this decreases efficiency (number of rounds) and complicates the resolution and aborting protocols with the TTP. Existing multi-party fair exchange protocols [12, 49, 67] in the ring topology do not need this dependency because their understanding of fairness is different from our fairness criteria, as discussed above.
- We achieve privacy against the TTP (i.e, the TTP only learns decryption shares). The MPCS protocols do not have any privacy against the TTP because they are based on the TTP simply converting a promise into a valid signature of the contract. As seen in Table 2, none of the multi-party fair exchange protocols in the ring topology have privacy against the TTP.
- We succeed to have fairness in flexible topologies (i.e., a party P_i sends to party P_j only the decryption shares of encrypted items which should be obtained by the party P_j according to the exchange topology. See Section 6).

Besides, none of the existing multi-party fair exchange protocols in any topology have any formal fairness proof. The multi-party contract signing protocols [15, 70, 74, 15] also do not have any formal fairness proof. Garay and Mackenzie [43] proved their MPCS protocol is fair in the random oracle model, but it has been shown that it is not fair when the number of parties is greater than five [74]. In a nutshell, our protocols are the **first** multi-party fair exchange protocols that provide fairness **in all types of topologies** by protecting **privacy** against the TTP and have a **formal proof** on the fairness property in the ideal-real simulation paradigm.

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 (resource) fairness is designed by Garay et al. [42]. 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 many rounds due to gradual release. Another approach is using bitcoin-like payment systems to achieve fairness using a TTP in the optimistic model [18, 2, 57]. When one of the parties does not receive the output of the computation, he receives a bitcoin instead. A similar fairness approach was used by Lindell [65] for the two-party computation case, and by Belenkiy et al. [16] and Küpçü and Lysyanskaya [61] for peer-to-peer systems. However, this approach may not be appropriate for multi-party computation since we do not necessarily know how valuable the output will be before evaluation.

Other Works in Fair MPC: Reputation-based fairness solutions by Ashorov et al.[6]talk about fairness probabilities, rather than complete fairness. Because of the impossibility of general complete fairness [6], relaxed versions of fairness definition such as partial fairness [53, 54], resource fairness [42], game-theoretic techniques [5, 41], and rational fairness [55] are considered. Gordon et al. [52, 50] showed that fairness (without TTP) can be achieved for specific functionalities. Following that, [3, 4] succeeded to have fairness for specific functionalities in two-party computation. General fair two-party computation protocols (with TTP or gradual release) also exist [58, 64, 21, 78, 80, 60], but they do not have direct and efficient generalizations for the multi-party case.

3 Definitions and Preliminaries

3.1 Definitions

As we represent our protocols as SMPC protocols and provide ideal-real simulation proofs, we first define the related ideal worlds for security and fairness.

Secure Multi-Party Computation (SMPC): A set of parties with their private inputs w_i desire to compute a functionality ϕ [46]. This computation is *secure* when the parties do not learn anything beyond what is revealed by the output of the computation. This is formalized by ideal-real world simulations, defined below [45].

Ideal World: The ideal-world execution consists of honest party(s) $\mathcal{P}_h$, an adversary $\mathcal{A}$ that corrupts the parties in set $\mathcal{P}_c$, and the ideal functionality U_s^ϕ (*not the TTP*). The ideal protocol is as follows:

U_s^ϕ for security with abort

- U_s^ϕ receives inputs $\{w_i\}_{P_i \in \mathcal{P}_c}$ of corrupted parties or the message ABORT from $\mathcal{A}$, and inputs $\{w_i\}_{P_i \in \mathcal{P}_h}$ of the honest party(s).
- If the inputs are invalid or $\mathcal{A}$ sends the message ABORT, then U^ϕ sends $\perp$ to all of the parties and halts.
- Otherwise, U_s^ϕ 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 output of i^{th} party. Then, U_s^ϕ sends $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ to $\mathcal{A}$.
  - If $\mathcal{A}$ sends a message CONTINUE, then U_s^ϕ sends each one of $\{\phi_i\}_{P_i \in \mathcal{P}_h}$ to the corresponding honest party.
  - If $\mathcal{A}$ sends a message ABORT, U_s^ϕ sends $\perp$ to the honest party(s).

The outputs of the honest parties and of the adversary in an ideal execution between the honest party(s) and an adversary $\mathcal{A}$ controlling the corrupted parties where U_s^ϕ computes ϕ is denoted $IDEAL^s_{\phi,\mathcal{A}(aux),\mathcal{P}_c}(w_1, w_2, ...w_n, \ell)$, 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 ℓ is the security parameter.[2]

Real World: No ideal functionality U_s^ϕ exists in the real protocol π to compute the functionality ϕ. A PPT adversary $\mathcal{A}$ controls the set $\mathcal{P}_c$ of corrupted parties where $|\mathcal{P}_c| \leq n-1$. The outputs of the honest party(s) P_h and the adversary $\mathcal{A}$ in the real execution of the protocol π is denoted $REAL^s_{\pi,\mathcal{A}(aux),\mathcal{P}_c}(w_1, w_2, ...w_n, \ell)$, where $w_1, w_2, ...w_n, aux$, and ℓ are like above.

Optimistic Fair Secure Multi-Party Computation: The ideal world definition above does not guarantee fairness. Multi-party computation is *fair* if either all of the parties learn the output at the end of the computation, or none of them learns the output. We extend the two-party fair and secure computation definition by Cachin and Camenisch [21] to the multi-party case in the spirit of the standard secure computation definition. The advantage of this definition is that it allows the trusted party to be corrupted. In Appendix A, we show that when the TTP is honest, this definition is equivalent to a simpler and easier to understand definition.

Ideal World: The ideal-world execution consists of honest party(s) $\mathcal{P}_h$, an adversary $\mathcal{A}$ that corrupts the parties in set $\mathcal{P}_c$, the TTP, and the ideal functionality U_{fs}^ϕ. The ideal protocol is as follows ($\mathcal{Y}_i$ denotes the domain of ϕ_i):

U_{fs}^ϕ for security and fairness

- U_{fs}^ϕ receives inputs $\{w_i\}_{P_i\in\mathcal{P}_c}$ or the message ABORT from $\mathcal{A}$, and $\{w_i\}_{P_i\in\mathcal{P}_h}$ from the honest party(s). If the inputs are invalid or $\mathcal{A}$ sends the message ABORT, then U^ϕ sends $\perp$ to all of the parties and halts.
- Otherwise U_{fs}^ϕ 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 $\{\phi_i\}_{P_i\in\mathcal{P}_c}$ to $\mathcal{A}$ and expects back a response from the TTP.
- The TTP sends $\{b_i \in \{\mathcal{Y}_i \cup \perp \cup \text{CONTINUE}\}\}_{P_i\in\mathcal{P}_h}$ to U_{fs}^ϕ. Honest TTP always responds as $\{b_i = \text{CONTINUE}\}_{P_i\in\mathcal{P}_h}$.
- For each $P_i \in \mathcal{P}_h$, U_{fs}^ϕ does the following:
  - If $b_i = \text{CONTINUE}$, sends ϕ_i to P_i.
  - Else, sends b_i to P_i.

Observe that an honest TTP always sends CONTINUE for each honest party, meaning that once the adversary obtains its output, fairness is guaranteed and hence each honest party will also obtain its output. The other options for the TTP represent the case where the TTP is controlled by the adversary: $b_i = \perp$ represents the TTP not responding to an honest party P_i, breaking fairness, and any other b_i denotes that the TTP resolves with the party P_i such that P_i learns an incorrect output.

It is worth mentioning here that while we will employ this general definition in our proofs, for our protocol, even a malicious TTP cannot send a b_i value that breaks correctness. In our proof in Section 4.3.2 for our adapted fair SMPC with the adversarial TTP, only $\perp$ or CONTINUE messages are employed, which only affect fairness of the protocol.

The outputs of the honest parties and of the adversary in an ideal execution between the honest party(s) and an adversary $\mathcal{A}$ controlling the corrupted parties (and possibly the TTP)

[2]Superscript 's' in $REAL^s$ and $IDEAL^s$ represents that they are defined for **s**ecurity only, not fairness.

where U_{fs}^{ϕ} computes ϕ is denoted $IDEAL_{\phi,\mathsf{TTP},\mathcal{A}(aux),\mathcal{P}_c}^{fs}(w_1, w_2, ...w_n, \ell)$, 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 ℓ is the security parameter.[3]

Real World: No ideal functionality U_{fs}^{ϕ} exists in the real protocol π to compute the functionality ϕ. A PPT adversary $\mathcal{A}$ controls the set $\mathcal{P}_c$ of corrupted parties where $|\mathcal{P}_c| \leq n-1$. There is a TTP who is involved in the protocol when a dispute arises. The 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 $REAL_{\pi,\mathsf{TTP},\mathcal{A}(aux),\mathcal{P}_c}^{fs}(w_1, w_2, ...w_n, \ell)$, where $\{w_i\}_{1\leq i\leq n}, aux$, and ℓ are like above.

Adversarial Model: When there is a dispute between the parties, the TTP resolves the conflict *atomically*, dealing with one party at a time until that resolution protocol is finished. The communication channel is defined similar to previous work [11, 63, 61, 62], such that there are loosely-synchronized clocks (e.g., we employ large timeouts on the order of potentially hours or days since they only affect dispute resolution, not the execution of the protocol), and while the communication channel is under adversary's control, it is assumed that the adversary cannot prevent the honest parties from reaching the TTP before the specified time interval ends. Essentially, we are in a bounded-delay model where the delay the adversary can enforce on the communication between the honest parties and the TTP is bounded by some α, which is assumed to be less than the timeouts stated in the protocols. Note that there is no bound on the delay on the messages between the parties; only those between a party and the TTP has a bounded delay. Thus, message passing between parties and local aborts can be done in an asynchronous manner, but communication with the TTP must be done via loosely synchronized clocks with bounded adversarial delay. One may simplify this and think of synchronous network connections everywhere for the sake of the proofs. Secure channels are used to exchange the decryption shares and endorsement of e-coin. A secure and server-authenticated channel is employed 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).

Observe that we aim to provide fairness (where if all parties abort without learning any useful information about result, it is also considered fair), not guaranteed output delivery. Cohen and Lindell [31] show fairness with broadcast implies guaranteed output delivery. We believe that using a broadcast channel in our solution would satisfy their requirement, but here we focus on efficiently obtaining fairness.

Definition 1 (Secure Multi-Party Computation)**.** *Let π^{ϕ} be a PPT protocol and let ϕ be a PPT multi-party functionality. If for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world simulator S for all $w_1, w_2, ..., w_n$ such that*

$$\{IDEAL_{\phi,S(aux),\mathcal{P}_c}^{s}(w_1, w_2, ..., w_n, \ell)\}_{\ell\in\mathbb{N}} \equiv_c \{REAL_{\pi,\mathcal{A}(aux),\mathcal{P}_c}^{s}(w_1, w_2, ..., w_n, \ell)\}_{\ell\in\mathbb{N}}$$

then, we say that π computes ϕ ***securely****.*

Definition 2 (Fair and Secure Multi-Party Computation)**.** *Let π^{ϕ} be a PPT protocol and let ϕ be a PPT multi-party functionality. If for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world simulator S for all $w_1, w_2, ..., w_n$ such that*

$$\{IDEAL_{\phi,S(aux),\mathcal{P}_c}^{fs}(w_1, w_2, ..., w_n, \ell)\}_{\ell\in\mathbb{N}} \equiv_c \{REAL_{\pi,\mathsf{TTP},\mathcal{A}(aux),\mathcal{P}_c}^{fs}(w_1, w_2, ..., w_n, \ell)\}_{\ell\in\mathbb{N}}$$

then, we say that π computes ϕ ***fairly and securely****.*

[3]Superscript 'fs' in $REAL^{fs}$ and $IDEAL^{fs}$ represents that they are defined for **f**airness and **s**ecurity.

We employ Definition 2 when we prove security and fairness of our protocols with the TTP. We use Definition 1 in our proofs when we show that a malicious TTP which can also collude with the other parties can only harm fairness but cannot harm security, by considering it as another corrupted party. Canetti [26] gives general security and fairness definition for multi-party protocols without considering TTP. Similar definitions were employed in the two-party case [47, 50, 58].

Optimistic Multi-Party Fair Exchange: The participants are $P_1, P_2, ..., P_n$. Each participant P_i has some item f_i to exchange, and wants to exchange his own item f_i with some or all of the other parties' items $\{f_j\}_{j \neq i}$, , where $i, j \in \{1, ..., n\}$. 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 standard ring topology, it would be defined as $\phi_i = (f_{i-1 \mod n})$. The topology information specifies this.) Therefore, we can use Definition 2 as the security and fairness definition of a multi-party fair exchange protocol, using the ϕ_i representing the desired topology. To the best of our knowledge, we are the first to employ fair SMPC definition for proving fairness and security of multi-party fair exchange protocols.

Hybrid Model: Assume that we have a protocol π^ψ for the functionality ψ which realizes an ideal ψ functionality U^ψ and have protocol π^ϕ for functionality ϕ that is using π^ψ as a sub-protocol. In the hybrid model, while proving security of π^ϕ based on Definition 2 or Definition 1, we can use the ideal ψ functionality U^ψ instead of the real protocol π^ψ to simplify the proof. Basically, it is sufficient to consider the parties in π^ϕ interact with U^ψ whenever they need to compute functionality ψ [26]. This is called as $\psi-$hybrid model.

3.2 Preliminaries

In this section, we give some cryptographic primitives that we use in our protocols.

Definition 3 (Shared-Key Deniable Encryption Scheme [27, 81])**.** *It is a symmetric key encryption scheme with the message space $\mathcal{M}$ with the following PPT algorithms:*

- $\mathsf{SymGen}(1^\ell) \rightarrow K$: *It takes the security parameter ℓ in unary as input and outputs a shared key K.*
- $\mathsf{SymEnc}(K, m, r_S, r_R) \rightarrow E$: *It takes the shared key K, a message $m \in \mathcal{M}$ and some randomness r_S from sender and r_R from the receiver as input and outputs the ciphertext E.*
- $\mathsf{SymDec}(K, r_S, r_R, E) \rightarrow m$: *It takes the shared key K, randomness r_S, r_R and a ciphertext E as input and outputs a message m.*
- $\mathsf{SymFake}(m_1, m_0, r_S, r_R, E, K) \rightarrow K', \tilde{r}_S$: *It takes $m_1, m_0 \in \mathcal{M}$ where m_0 is the fake message, randomness r_S, r_R, ciphertext $E = \mathsf{SymEnc}(K, m_1, r_S, r_R)$ and key K as input and outputs a key K' and a randomness $\tilde{r}_S$ such that $\mathsf{SymDec}(K', \tilde{r}_S, r_R, E) \rightarrow m_0$.*

Note that a regular (non-deniable) symmetric encryption scheme would have only the first three algorithms, and without the randomness r_R of the receiver.

Definition 4 (Security of Sender Deniability [82]). *Sender deniability is shown via a CPA-like game where the adversary sends two messages m_0, m_1 to the faking-algorithm challenger. The challenger picks $b = \{0, 1\}$ and computes $E = \mathsf{SymEnc}(K, m_b, r_S, r_R)$. If $b = 0$, it sends $E, (K, r_S)$. Otherwise, it sends $E, (K', \tilde{r}_S) = \mathsf{SymFake}(m_1, m_0, r_S, r_R, E)$ (which generates a key K' such that the decryption of E with $K', \tilde{r}_S$ gives m_0). The adversary wins if he guesses b correctly. If the advantage of the adversary is negligible in ℓ, then the scheme has security of sender deniability.*

A simple deniable encryption scheme is the one time pad. The faking algorithm generates a key by computing the bitwise XOR of the message and the ciphertext. For example, a sender encrypts a message m as $e = m \oplus k$ with the secret key k. Then, she sends e to the receiver. Later, if she decides that the receiver should learn the message m' from e, then she sends $k' = e \oplus m'$ to the receiver, who uses it to decrypt e and obtain $m' = e \oplus k'$.

Endorsed E-cash: Endorsed e-cash [24] consists of two pieces: unendorsed coin '$coin^u$' and endorsement 'e'. $coin^u$ cannot be used as a coin without e. In addition, no one except the owner of $coin^u$ can construct a valid endorsement e for $coin^u$. The endorsement can be efficiently verifiably escrowed and can be verified to endorse the matching unendorsed coin. Note that in our protocols, we can employ any electronic payment scheme, as long as it can be efficiently verifiably escrowed (e.g., electronic checks [29]). For the formal security definitions, we refer the reader to [24], since those are not necessary to understand our paper. The payment system is used only in our CMFE protocol for fair exchange of large files in Section 5, and in both the protocol and the security proof, honest parties' endorsements' are never given to any other party.

Definition 5 ((k, n)-Threshold Encryption [83]). *It consists of following PPT algorithms assuming that the number of parties is n and the threshold is k:*

- $\mathsf{ThGen}(1^\ell, k, n) \to (\mathsf{pk}, v, \{x_i\}_{1 \le i \le n})$*: It takes a security parameter ℓ in unary and parameters k, n as input and outputs a public key* pk*, a verification key v, and a secret key x_i for each party P_i.*
- $\mathsf{ThEnc}(\mathsf{pk}, m) \to E$ *: It takes the public key* pk *and a message m as input and outputs the ciphertext E.*
- $\mathsf{ThDShare}(x_i, \mathsf{pk}, E) \to d_i$ *: It takes a private key x_i, the public key* pk*, and a ciphertext E as input and outputs the decryption share d_i.*
- $\mathsf{ThDSProve}(x_i, \mathsf{pk}, E) \to \mathsf{DSproof}_i$ *: It takes as input a private key x_i, a ciphertext E, the public key* pk*, a decryption share d_i, and outputs a proof* $\mathsf{DSproof}_i$ *that the decryption share is valid.*
- $\mathsf{ThDSVerify}(v, \mathsf{pk}, E, d_i, \mathsf{DSproof}_i) \to \mathsf{valid/invalid}$ *: It takes the verification key v, the public key* pk*, a ciphertext E, and a decryption share d_i with its proof* $\mathsf{DSproof}_i$ *as input and outputs either valid or invalid based on the verification of the proof.*
- $\mathsf{ThDec}(DS, \mathsf{pk}, E) \to m$ *: It takes a list of decryption shares DS where $|DS| \ge k$, the public key* pk*, and a ciphertext E as input and outputs a plaintext m.*

While Shoup and Gennaro [83] define a (k, n)-threshold encryption scheme with $\mathsf{ThDSProve}$ and $\mathsf{ThDShare}$ combined into a single algorithm, for the sake of clarity, we separately define them. Then we use the ideal functionality $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ below for the security of $\mathsf{ThDSProve}$ and $\mathsf{ThDSVerify}$ with the following functionality.

Ideal zero-knowledge functionality: Defined below between two parties P_i as a prover and P_j as a verifier:

$U^{\mathsf{ZK}\text{-}\mathcal{R}}$ with a relation $\mathcal{R}$

- $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ receives $(\mathsf{prove}, id||P_i||P_j, w, \delta)$ from party P_i.
- If $(w, \delta) \in \mathcal{R}$, then $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ outputs $(\mathsf{proof}, id||P_i||P_j, \delta)$ to P_j. Otherwise, $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ outputs $(\mathsf{disproof}, id||P_i||P_j, \delta)$ to P_j.

For the threshold encryption scheme, the prover P_i in $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ corresponds to the party who wants to prove that the decryption share d_i is correctly constructed for the encryption E (using $\mathsf{ThDSProve}$) and the verifier P_j in $U^{\mathsf{ZK}\text{-}\mathcal{R}}$ is the party who wants to verify that d_i is a correct decryption share for E (using $\mathsf{ThDSVerify}$).

Definition 6 (Deniable (k, n)-Threshold Encryption)**.** *It consists of all algorithms in Definition 5 and also the PPT algorithm below. Assume that we have $E = \mathsf{ThEnc}(\mathsf{pk}, m)$ where its corresponding valid decryption shares are $DS = \{d_{i_1}, d_{i_2}, ..., d_{i_t}\}$ where $\{i_1, i_2, ..., i_t\} \subseteq \{1, 2, ..., n\}$ and $t = |DS| < k$.*

- $\mathsf{ThDeny}(E, DS, \mathcal{I}, m', \mathsf{pk}) \rightarrow DS'$ *: It takes as input a ciphertext E, decryption shares DS, an index set $\mathcal{I}$ such that $|\mathcal{I}| = k - |DS|$ and $\{i_1, i_2, ..., i_t\} \notin \mathcal{I}$, a fake plaintext m', and the public key* pk*. It outputs a set of fake decryption shares DS' such that $\mathsf{ThDec}(DS' \cup DS, \mathsf{pk}, E) \rightarrow m'$.*

Remark that ThDeny only returns a set of decryption shares. Because of the security of $\mathsf{ThDSProve}$ and $\mathsf{ThDSVerify}$, it is not possible in a real protocol execution to provide fake decryption shares that verify as valid. Thus, this algorithm will only be used in the simulation for the sake of the security proof.

Definition 7 ((Labeled) Public Key Encryption)**.** *A (labeled) public key encryption scheme with security parameter ℓ consists of the following PPT algorithms:*

- $\mathsf{PkGen}(1^\ell) \rightarrow (\mathsf{sk}, \mathsf{pk})$ *: It takes a security parameter ℓ in unary as input and outputs a public key* pk *and a secret key* sk*.*
- $\mathsf{PkEnc}(\mathsf{pk}, m; lbl) \rightarrow \mathsf{VE}$ *: It takes the public key* pk*, a message $m \in \mathcal{M}$ and a label lbl as input and outputs the ciphertext* VE*.*
- $\mathsf{PkDec}(\mathsf{sk}, \mathsf{VE}) \rightarrow m$ *: It takes the secret key* sk *and a ciphertext* VE *as input and outputs the plaintext m.*

Verifiable Encryption: It is a public key encryption scheme that enables the recipient to verify, using a public-key, that the plaintext satisfies some relation, without performing any decryption [25, 23]. A public non-malleable label *lbl* can be attached to a verifiable encryption [83].

Definition 8 (Verifiable Encryption [25, 22])**.** *Let $\psi = [\mathcal{R}, W, \Delta]$ be a description of a binary relation $\mathcal{R}$ on $W \times \Delta$ and $\mathcal{M}$ be a message space. A verifiable encryption scheme is a public key encryption scheme with the following two additional potentially interactive PPT algorithms* $\mathsf{ProveEnc}$ *run by a prover who encrypts the message and* $\mathsf{VerifyEnc}$ *run by a verifier who receives the ciphertext.*

- $\mathsf{ProveEnc}(\mathsf{pk}, w, \delta; lbl) \rightarrow (\mathsf{VE}, \mathsf{VEproof})$ *: It takes as input* pk*, a witness $w \in W$, and a statement $\delta \in \Delta$, and outputs the ciphertext $\mathsf{VE} = \mathsf{PkEnc}(\mathsf{pk}, w; lbl)$ and a proof* $\mathsf{VEproof}$ *that $(w, \delta) \in \mathcal{R}$.*

- $\mathsf{VerifyEnc}(\mathsf{pk}, \delta, \mathsf{VE}, \mathsf{VEproof}) \rightarrow \mathsf{valid/invalid}$: *It takes as input* $\mathsf{pk}, \delta, \mathsf{VE}$ *and the proof* $\mathsf{VEproof}$*, and outputs valid if* $(w, \delta) \in \mathcal{R}$*, or invalid if* $(w, \delta) \notin \mathcal{R}$*.*

We give the security of $\mathsf{ProveEnc}$ and $\mathsf{VerifyEnc}$ with the following functionality which satisfies the security properties of these algorithms such as completeness, soundness, and zero knowledge [22, 25].

Ideal verifiable encryption/escrow functionality: It is defined below between two parties P_i as a prover and P_j as a verifier:

$U^{\mathsf{VE}\text{-}\mathcal{R}}$ with a relation $\mathcal{R}$

- $U^{\mathsf{VE}\text{-}\mathcal{R}}$ receives $(\mathsf{VEprove}, id||P_i||P_j, w, \delta, \mathsf{pk}, item, label)$ from party P_i.
- If $(w, item, \delta) \notin \mathcal{R}$ or pk is not a valid public-key for the encryption algorithm Enc, then $U^{\mathsf{VE}\text{-}\mathcal{R}}$ outputs $(\mathsf{VEdisproof}, id||P_i||P_j, \delta, \mathsf{pk}, label, \perp)$ to the party P_j. Otherwise, $U^{\mathsf{VE}\text{-}\mathcal{R}}$ computes the ciphertext $\mathsf{Enc}(\mathsf{pk}, item) \rightarrow E$ and outputs $(\mathsf{VEproof}, id||P_i||P_j, \delta, \mathsf{pk}, label, E)$ to P_j.

The ideal verifiable *escrow* functionality $U^{\mathsf{VS}\text{-}\mathcal{R}}$ works the same as $U^{\mathsf{VE}\text{-}\mathcal{R}}$, except that it checks that the pk is the public-key of the TTP instead of checking whether pk is valid or not.

For our purposes, we need a verifiable deniable (n,n)-threshold encryption scheme which satisfies Definitions 5, 6, and 8. In Appendix B, we show how to efficiently instantiate it using the Decisional Diffie-Hellman assumption via the ElGamal [37] threshold scheme.

We discuss in Section 7 how to efficiently initialize these functionalities using previous work as primitives, and present concrete performance numbers.

Finally, we need a signature scheme that is existentially-unforgeable under adaptive chosen message attack.

Definition 9 (Signature Scheme [48])**.** *A signature scheme with security parameter* ℓ *consists of the following PPT algorithms:*

- $\mathsf{SgGen}(1^\ell) \rightarrow (s, v)$: *It takes a security parameter* ℓ *in unary as input and outputs a public verification key* v *and a secret signing key* s*.*
- $\mathsf{SgSign}(s, m) \rightarrow \sigma$: *It takes the secret signing key* s *and a message* m *and outputs the signature* σ*.*
- $\mathsf{SgVerify}(v, m, \sigma) \rightarrow \mathsf{accept/reject}$: *It takes the public verification key* v*, a message* m*, and a signature* σ *as input and outputs accept only if the signature is valid on the given message.*

3.3 Notation

The n parties and their names in the protocol are represented by P_i, where $i \in \{1, ..., n\}$. $\mathcal{P}_h$ denotes the set of honest parties, and the set of $\mathcal{P}_c$ denotes the corrupted parties controlled by the adversary $\mathcal{A}$ where $\mathcal{P}_c \cup \mathcal{P}_h = \{P_1, P_2, ..., P_n\}$.

The topology of the protocol is described as $\mathsf{topology} = \{(P_1, \epsilon_1), ..., (P_n, \epsilon_n)\}$ where ϵ_i is a set which includes the list of items with their owners that P_i expects an item from. Each element of ϵ_i is in the following form: $(P_j; F_j)$ where F_j is the list of items that P_i expects from P_j (e.g. $\epsilon_i = \{(P_{i-1}; f_{i-1})\}$ for the ring topology, $\epsilon_i = \{(P_j; f_j)\}_{1 \leq j \neq i \leq n}$ for the complete topology). F_j can have more items in the case that P_j has more than one item to give a party.

$E_i^{u_v}$ is used to denote a ciphertext where the plaintext is u_i and the key is v. $\mathsf{Enc}(\mathsf{pk}, m; lbl)$ is used to show an encryption of a message m with a key pk where lbl is a public non-malleable label.

VE_i and VS_i are used to show the verifiable encryption and verifiable escrow prepared by P_i, respectively.

pk and pk_T show the public key of a threshold-encryption scheme and the public key of a TTP, respectively. We let d_i^k denote a decryption share of the encryption E_i which is generated by a party P_k (i.e., $\mathsf{ThDShare}(x_k, \mathsf{pk}, E_i) \rightarrow d_i^k$). Usually we denote simulated values separately. For example, the encryption of a random message is shown with $\tilde{E}_i$ and its corresponding valid decryption shares are shown with $\tilde{d}_i$.

The notation ⓩ denotes the number z in the Figures 1 and 2. ℓ is the security parameter.

In MFE and CMFE, we use the ideal functionalities, $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}, U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$ and $U^{\mathsf{VS}\text{-}\mathcal{R}_{coin}}$ where the relations are:

$$\mathcal{R}_{ds} = \{((x_i, r), (\mathsf{pk}, d_i, v) | \mathsf{ThDShare}^r(x_i, \mathsf{pk}, E) \rightarrow d_i\},$$

where $\mathsf{ThDShare}^r$ runs $\mathsf{ThDShare}$ with random coins r, and

$$\mathcal{R}_{vs\text{-}ds} = \{((\{d_i, \mathsf{DSproof}_i\}), (\mathsf{pk}, \mathsf{pk}_T, \{E_i\}, v)) | \forall i, \mathsf{ThDSVerify}(v, \mathsf{pk}, E_i, d_i, \mathsf{DSproof}_i) \rightarrow \mathsf{valid}\}$$

which shows that the verifiable escrow VS is the encryption of valid decryption shares and their proofs.

For exchanging items, we have the following relation to verify that the item is the desired one:

$$\mathcal{R}_{item} = \{(f, (F, c)) | F(f, c) = \text{VALID}\}$$

For example, in $\mathcal{R}_{item}$, if f is a signature on a contract, then c contains the signature verification key together with the contract, and F is the signature verification algorithm.

Finally, in CMFE we have

$$\mathcal{R}_{coin} = \{(e, coin^u) | e \text{ is the endorsement of unendorsed coin } coin^u\}$$

which denotes that a valid e-coin payment is escrowed.

4 Multi-Party Fair Exchange Protocol (MFE)

Our aim in MFE is to create an efficient multi-party fair exchange protocol that works in every topology. The most difficult challenges of this type of protocols are the following:

- Even if $n - 1$ parties are colluding, the protocol has to guarantee fairness. Consider a simple optimistic exchange protocol for the complete topology: each party first sends the verifiable escrow of his/her item to the other parties. After all verifiable escrows are received, each of them sends their (plaintext) items to each other. If one of the parties contacts the TTP for a resolution, the TTP decrypts the verifiable escrow(s) and stores the contacting party's item for the other parties (in case they request it later).
  Assume now that parties P_i and P_j are colluding, and P_i receives verifiable escrow of the honest party P_h, but P_j did not send his verifiable escrow to P_h yet. Next, P_i contacts the TTP, receives P_h's item f_h via the decryption of the verifiable escrow of P_h, and gives his item f_i to the TTP in return. At this moment, if P_i and P_j leave the protocol,

then fairness is violated because P_h never gets the item of P_j, whereas, by colluding with P_i, P_j receives f_h.

Thus, it is important *not* to let any party to learn any item *before all the parties are guaranteed* that they will get their requested 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-trusted third party. Hence, privacy against the TTP needs to be satisfied. In the simple protocol above, the privacy against the TTP is violated since the TTP learns the items of the parties. This requirement becomes even more important when one tries to employ a multi-party fair exchange protocol on top of SMPC protocols to achieve fairness. If the extensions used to provide fairness leak the outputs of the parties (even to the TTP), then the privacy against the TTP is violated. As mentioned before, we only need TTP for fairness even in the fair SMPC setting, and **security and privacy are always satisfied even when the TTP misbehaves and colludes with participants** in our solution.
- The parties do not receive or send any item from/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 [4], 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 he sends her item to P_3, while it may be the case that P_4 did not receive the item of P_3 (see Section 2). 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 6. All zero-knowledge proof of knowledge protocols below are executed non-interactively in the random oracle model [19, 39].

4.1 Description of MFE

MFE consists of n parties and a trusted third party (TTP) that is involved in the protocol only when a dispute arises between the participants related to fairness. The parties use a secure channel in step ④. MFE is constructed on top of a verifiable and deniable (n, n)-threshold encryption scheme with $(\mathsf{ThGen}, \mathsf{ThDShare}, \mathsf{ThDSProve}, \mathsf{ThDSVerify}, \mathsf{ThEnc}, \mathsf{ThDec}, \mathsf{ThDeny})$ and another verifiable encryption scheme with $(\mathsf{PkGen}, \mathsf{PkEnc}, \mathsf{PkDec})$. Both encryption schemes have $\mathsf{ProveEnc}$ and $\mathsf{VerifyEnc}$ algorithms since they are verifiable. The TTP has the secret/public key pair $(\mathsf{sk}_T, \mathsf{pk}_T)$ generated by $\mathsf{PkGen}(1^\ell)$ where ℓ is the security parameter. We assume that pk_T is known by every participant.

Overview: The protocol has three phases. In the first phase (Setup Phase), parties 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 (encrypted item exchange -EIE- Phase), they send to each other the verifiable encryptions of

[4] Topology satisfied means that all parties received all the items described in the topology.

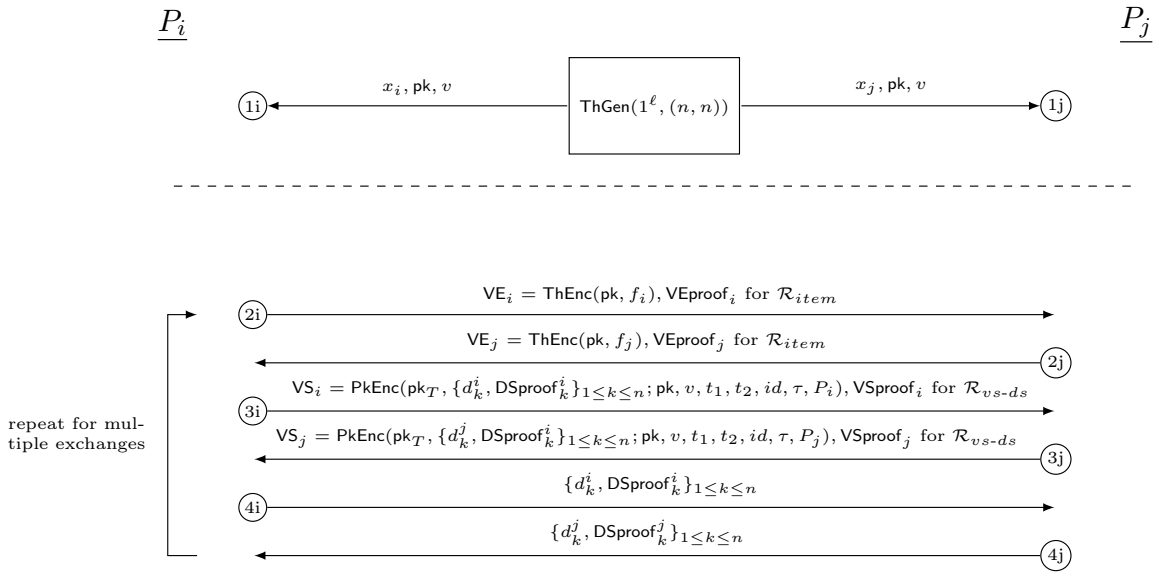

Figure 1: Our MFE Protocol. Each (i, j) message pair can be performed in any order or in parallel within a step.

the items that they want to exchange. If anything goes wrong up till here, the protocol is simply aborted. In the final phase (decryption share exchange -DSE- Phase), they exchange the decryption shares for each item. If something goes wrong during the final phase, resolutions with the TTP are performed. Because the proofs need to be transferred to the TTP, they need to be instantiated as non-interactive proofs, which can be achieved efficiently in the random oracle or common reference string model. Between the same set of participants, after one setup phase, the other phases can be re-executed multiple times to perform multiple exchanges with possibly varying topologies. The details are below (see also Figure 1).

Below we describe MFE based on the complete topology. Section 6 shows how MFE works in any topology. Since our protocol is symmetric for all participants, we describe the actions of some P_i.

Setup Phase (① in Figure 1): All parties jointly run the key generation algorithm ThGen of a threshold encryption scheme with the threshold n. At the end, each party P_i receives its secret share x_i, the public key pk, and the verification share v of a deniable (n, n)-threshold encryption. Appendix B discusses an instantiation based on ElGamal and how to achieve this in a distributed manner in a single round of communication [77].

The Setup Phase is executed only once. Afterward, the same set of parties can exchange as polynomially many items as they want without re-executing the Setup Phase.

Encrypted Item Exchange (EIE) Phase (② in Figure 1): Firstly, parties agree on two time parameters t_1 and t_2 where $t_1 > \alpha$ and $t_2 - t_1 > \alpha$ (hence allowing parties to reach the TTP even with adversarial delay; see the adversarial model in Section 3), the $\mathsf{topology}$, and the identification id of the protocol.[5] Each party P_i does the following:

- P_i generates the verifiable encryption of its item f_i with the public key pk by running $\mathsf{ProveEnc}(\mathsf{pk}, f_i, c_i)$. Then, he sends the encryption $\mathsf{VE}_i = \mathsf{ThEnc}(\mathsf{pk}, f_i)$ and the proof $\mathsf{VEproof}_i$ showing that the encrypted item is valid $(f_i, c_i) \in \mathcal{R}_{item}$ where c_i is a public

[5]Time parameters and topology can also be agreed in the Setup Phase if they will remain constant.

value.

Shortly, P_i proves that he encrypts the requested item (e.g., if f_i is a signature on a contract, then c_i contains the signature verification key of P_i together with the contract, and $\mathcal{R}_{item}$ is the relation that f_i is a valid signature with respect to c_i). VE_i does not have any label.

- Then, P_i waits for the verifiable encryptions of the other parties. When he receives a VE_j and $\mathsf{VEproof}_j$ from a party P_j, he verifies it with $\mathsf{VerifyEnc}(\mathsf{pk}, c_j, \mathsf{VE}_j, \mathsf{VEproof}_j)$. If any party does not respond or responds with an invalid proof, P_i simply aborts the protocol.

Note that without knowing n decryption shares, no party can decrypt any VE_j. So, no party learns the items. Thus, if anything goes wrong up to this point, the parties can locally abort the protocol. After this point, they need to obtain all the decryption shares. This is done in the following phase.

Decryption Share Exchange (DSE) Phase (③ and ④ in Figure 1): No party begins this phase without completing the EIE Phase and receiving all verifiable encryptions VE_j correctly.

- P_i generates all necessary decryption shares by running the algorithm $\mathsf{ThDShare}(x_i, \mathsf{pk}, \mathsf{VE}_j)$ and $\mathsf{ThDSProve}(x_i, \mathsf{pk}, \mathsf{VE}_j)$ for all $j \in \{1, 2, ..., n\}$. Then, he obtains decryption shares $\{d_j^i\}_{1 \leq j \leq n}$ and their corresponding proofs $\{\mathsf{DSproof}_j^i\}_{1 \leq j \leq n}$. After the generation of decryption shares, he runs $\mathsf{ProveEnc}(\mathsf{pk}_T, \{d_j^i, \mathsf{DSproof}_j^i\}_{1 \leq j \leq n}, \delta; lbl)$ where $\delta = (\mathsf{pk}, \mathsf{pk}_T, \{\mathsf{VE}_i\}, v)$ and obtains the verifiable escrow $\mathsf{VS}_i = \mathsf{PkEnc}(\mathsf{pk}_T, \{d_j^i, \mathsf{DSproof}_j^i\}; lbl)$ together with its proof $\mathsf{VSproof}_i$. $\mathsf{VSproof}_i$ proves that $((\{d_j^i, \mathsf{DSproof}_j^i\}), \delta) \in \mathcal{R}_{vs\text{-}ds}$. At the end, P_i sends VS_i and $\mathsf{VSproof}_i$ to the other parties.

  In simple terms, the verifiable escrow VS_i includes the encryption of the decryption shares of P_i that is used to decrypt the encrypted items of all parties and their proofs. The other parties verify it with $\mathsf{VerifyEnc}(\mathsf{pk}_T, \delta, \mathsf{VS}_i, \mathsf{VSproof}_i)$. In addition, only the TTP can decrypt it. The public non-malleable label $lbl = \{\mathsf{pk}, v, \tau, t_1, t_2, id, P_i\}$, where τ represents the topology of the protocol along with the names of the parties and corresponding verification shares: $\tau = \{\mathsf{topology}, P_1, ..., P_n\}$. Here, we assume that each party knows the other parties' names/identifiers.

  Remark: The name P_i is necessary to show the VS_i belongs to him. It is not beneficial to put a wrong name in a verifiable escrow's label, since otherwise another party can convince the TTP to decrypt VS_i by showing P_i is dishonest. Furthermore, the party names can be random and distinct in each exchange, as long as the parties know each others' names, and so it does not violate the privacy of the parties. The other labels id, t_1, t_2, τ are to show the protocol parameters to the TTP. τ is necessary for TTP to know who needs whose share. Unique (random) exchange identifier id is necessary to prevent corrupted parties to induce TTP to decrypt VS_j for an another exchange. Consider that some exchange protocol ended unsuccessfully, which means nobody received any item. The corrupted party could go to the TTP as if VS_j was 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.

- 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 simply continues with the next step without waiting.

- P_i sends all decryption shares $\{d_j^i\}_{1\leq j\leq n}$ and their proofs $\{\mathsf{DSproof}_j^i\}_{1\leq j\leq n}$ to the other parties. Then, he waits for $(d_1^j, d_2^j, ..., d_n^j)$ together with their proofs from each P_j. If one of the values that he receives is not as expected (i.e., if there exists j, t such that $\mathsf{ThVerify}(v, \mathsf{pk}, d_t^j, \mathsf{DSproof}_t^j) \rightarrow \mathsf{invalid}$) or if he does not receive them from some P_j, he performs **Resolve 2** protocol with the TTP, before t_2 and after t_1. Otherwise, P_i continues with the next step without waiting.
- After receiving all the necessary values, P_i can decrypt each VE_t and get all the items by running $\mathsf{ThDec}(\{d_t^j\}_{1\leq j\leq n}, \mathsf{pk}, \mathsf{VE}_t)$.

Resolve 1: The goal of Resolve 1 is to *record* the corrupted parties that did *not* send their verifiable escrow in ③. 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 the parameters t_1, t_2, id, τ. The *complaintList* contains the names of pairs of parties having a dispute because of missing or wrong VS. If the ***complainant*** does not expect a file from the ***complainee*** according to $\mathsf{topology} \in \tau$, the TTP does not add him to the *complaintList*. Otherwise, TTP saves the complainant as the first part of the tuple, and the complainee as the second part of the tuple. As a third part of the tuple, the TTP saves the names of the expected shares from the complainee. The name of the expected share d_t^j is $share_t^j$. The TTP saves also *complainee's verification share* which is in τ; 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

1: P_i sends $\mathsf{pk}, v, t_1, t_2, id, \tau, P_j$ to the TTP where P_j is the party that did not send his proper VS_j to P_i. The TTP does the following:
2: **if** $currenttime > t_1$ **or** $(P_j;.) \notin \mathsf{topology}[P_i] = \epsilon_i$ **or then**
3: **send** *msg* "Abort Resolve 1"
4: **else**
5: $complaintList$ = `GetComplaintList`$(\mathsf{pk}, v, t_1, t_2, id, \tau)$
6: **if** $complaintList ==$ NULL **then**
7: $complaintList$ = `EmptyList`$(\mathsf{pk}, v, t_1, t_2, id, \tau)$ // initialize empty list
8: $solvedList$ = `EmptyList`$(\mathsf{pk}, v, t_1, t_2, id, \tau)$ // will be used in Resolve 2
9: **end if**
10: $complaintList$.`add`$(P_i, P_j, \{share_t^j\}_{(P_{t;:})\in\epsilon_i})$
11: **send** *msg* "Come after t_1 for **Resolve 2**"
12: **end if**

Remark that cheating on one of the labels t_1, t_2, id, τ causes to have a completely different *complaintList* which has no relation with a *complaintList* which are created by the correct labels given by an honest party. Therefore, it is not helpful for a malicious party who wants an honest party's item.

Resolve 2: It is the resolution protocol where the parties contact 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 and their proofs inside them to solve the complaints in the *complaintList* according to the topology, and populate *solvedList*. 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 and get all the decryption shares together with their

proofs that he requests immediately. Hence, in our model, the latest time an honest party would have Resolve 3 performed is $t_2 + \alpha$ (including adversarial delay).

Algorithm 2 Resolve 2

1: P_i gives $\mathcal{V}$, which is the set of verifiable escrows that P_i has. P_i also provides the protocol information $(\mathsf{pk}, v, t_1, t_2, id, \tau)$. The TTP does the following:
2: **if** $t_1 < currenttime < t_2$ **then**
3: $complaintList$ = `GetComplaintList`$(\mathsf{pk}, v, id, t_1, t_2, \tau)$
4: **for all** VS_j in $\mathcal{V}$ **do**
5: **if** $(*, P_j, *) \in complaintList$ AND `VerifyEnc`$(\mathsf{pk}_T, \delta, \mathsf{VS}_j, \mathsf{VSproof}_j) \rightarrow$ valid **then**
6: $shares$ = `PkDec`$(\mathsf{sk}_T, \mathsf{VS}_j) = \{d_j^t, \mathsf{DSproof}_j^t\}_{1 \le t \le n}$
7: $P_j.solvedList$.`Append`$(shares)$
8: **for all** $(P_t, P_j, *) \in complaintList$ **do**
9: l_t = $complaintList$.`GetShare`(P_t, P_j) // it returns $\{share_k^j\}_{(P_k,.) \in \epsilon_i}$
10: **if** $l_t \subseteq$ `GetShareNames`$(shares)$ **then**
11: $complaintList$.`Remove`$(P_t, P_j, *)$ // solve complaints according to the topology
12: **end if**
13: **end for**
14: **end if**
15: **end for**
16: **end if**
17: **if** $complaintList$ is empty **then**
18: **send** msg "Perform **Resolve 3**"
19: **else**
20: **send** msg "Come after t_2 for **Resolve 3**"
21: **end if**

Resolve 3: If the *complaintList* still has parties, even after t_2, then 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*, then 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 and proofs. See Algorithm 3. When a party P_i receives the decryption shares and their proofs from the TTP, he verifies the proofs via $\mathsf{ThDSVerify}(v, \mathsf{pk}, VE_j, d_i^j, \mathsf{DSproof}_i^j)$ for each received $1 \le j \ne i \le n$ and outputs $\perp$ if any verification fails.[6]

Algorithm 3 Resolve 3

1: P_i gives $\mathcal{C}$, which is the set of parties that did not perform step ③ or ④ with P_i, and $\mathcal{V}$, which is the set of verifiable escrows that belongs to parties in $\mathcal{C}$ who performed step ③ properly. P_i also provides the protocol information $(pk, v, d, t_1, t_2, \tau)$. The TTP does the following:
2: $complaintList$ = `GetComplaintList`$(\mathsf{pk}, v, t_1, t_2, id, \tau)$
3: **if** $complaintList$ is empty **then**
4: **for all** P_j in $\mathcal{C}$ **do**
5: **if** $\mathsf{VS}_j \in \mathcal{V}$ and has correct label **then**
6: **send** $\mathsf{PkDec}(\mathsf{sk}_T, \mathsf{VS}_j)$
7: **else**
8: **send** $solvedList$.`GetShares`(P_j)
9: **end if**
10: **end for**
11: **else if** $currenttime > t_2$ **then**
12: **send** msg "Protocol is aborted"
13: **else**
14: **send** msg "Try after t_2"
15: **end if**

Remark that the TTP does not give any decryption shares until he is sure that some parties received their expected VS in step ③ and the rest of the parties' expected decryption shares are in *solvedList*.

4.2 Fairness Proof of MFE

Theorem 1. *The MFE protocol in the complete topology is fair in* $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, *and* $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ *hybrid models according to Definition 2,*

[6]This is only necessary against a potentially malicious TTP and used to ensure that the corrupted TTP cannot affect correctness when used over secure multi-party computation.

assuming that the verifiable deniable (n,n)-threshold encryption scheme with $(\mathsf{ThGen}, \mathsf{ThDShare}, \mathsf{ThDSProve}, \mathsf{ThDSVerify}, \mathsf{ThEnc}, \mathsf{ThDec}, \mathsf{ThDeny})$ *is IND-CPA secure and the verifiable encryption scheme with (*$\mathsf{PkGen}, \mathsf{PkEnc}, \mathsf{PkDec}$*) is IND-CCA secure.*

Proof. We prove the fairness in the $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ hybrid model. Assume that the set of parties corrupted by an adversary $\mathcal{A}$ is $\mathcal{P}_c$ and the set of uncorrupted (honest) parties is $\mathcal{P}_h$ and $\mathcal{P} = \mathcal{P}_h \cup \mathcal{P}_c$ with $\mathcal{P}_h \cap \mathcal{P}_c = \emptyset$. The simulator $\mathcal{S}$ simulates the honest parties in $\mathcal{P}_h$ in the real world, and the corrupted parties in $\mathcal{P}_c$ in the ideal world. $\mathcal{S}$ also simulates the TTP in the real world if any resolution protocol occurs, since TTP is an honest party. First, $\mathcal{S}$ generates $(\mathsf{sk}_T, \mathsf{pk}_T)$ as TTP does and publishes pk_T.

We first show the simulation, and then prove it is indistinguishable from the adversary's view. $\mathcal{S}$ does the following:

Setup Phase: $\mathcal{S}$ behaves as ThGen and generates all secret keys, the public key pk, and the verification key v. $\mathcal{S}$ distributes each secret key x_j to each corrupted party P_j together with pk, v.

EIE Phase: $\mathcal{S}$ simulates $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}$ as below:

- $\mathcal{S}$ picks random items $\{\tilde{f}_i\}_{P_i \in \mathcal{P}_h}$ for the corresponding honest parties since $\mathcal{S}$ does not know the actual items $\{f_i\}_{P_i \in \mathcal{P}_h}$ at this point. Then, $\mathcal{S}$ sends $(\mathsf{VEproof}, id||P_i||P_j, c_i, \mathsf{pk}, \emptyset, \tilde{\mathsf{VE}}_i)$ on behalf of each $P_i \in \mathcal{P}_h$ to each corrupted party P_j where $\tilde{\mathsf{VE}}_i = \mathsf{ThEnc}(\mathsf{pk}, \tilde{f}_i)$.
- $\mathcal{S}$ behaves as $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}$ when it receives a message $(\mathsf{VEprove}, id||P_j||P_i, f_j, c_j, \mathsf{pk}, \emptyset, \mathsf{VE}_j)$ from a corrupted party P_j. It learns each item f_j whenever it receives a $\mathsf{VEprove}$ message from a corrupted party P_j. $\mathcal{S}$ does not continue on behalf of any $P_i \in \mathcal{P}_h$ to the next step if some of the corrupted parties does not provide valid inputs to $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}$ and sends ABORT message to U. $\mathcal{S}$ outputs $\perp$ on behalf of real honest parties and whatever $\mathcal{A}$ outputs on behalf of ideal corrupted parties and stops.
  If $\mathcal{S}$ continues to the next step, it means that it learned all items of corrupted parties.

DSE Phase: $\mathcal{S}$ simulates $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$. It encrypts and sends random ciphertexts to the corrupted parties on behalf of honest parties. Whenever a corrupted party contacts with $\mathcal{S}$, it behaves as $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$ and learns their decryption shares.

At this point, one of the following situations must have happened for each party $P_i \in \mathcal{P}_h$ that $\mathcal{S}$ simulates:

(1) All corrupted parties have already submitted correct verifiable escrows to at least one honest party P_i before t_1.

(2) Some of the corrupted parties did not send any message to S or sent incorrect decryption shares or sent invalid labels.

We explain how the simulation works in these two cases:

Case (1) means it is guaranteed that the real honest party P_i would obtain her desired items via step ④ of MFE or resolutions. As long as at least one honest party P_i is in case (1), we do not consider the case (2) for the other honest parties because normally in MFE, P_i would perform Resolve 2 and give all necessary decryption shares for these parties. At the end, they would learn their items.

Therefore, $\mathcal{S}$ sends $\{f_j\}_{P_j \in \mathcal{P}_c}$ to U and also sends $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$ as TTP, even if only one honest party is in case (1). As a response, U sends $\{f_i\}_{P_i \in \mathcal{P}_h}$ to $\mathcal{S}$. At this point, $\mathcal{S}$ should send his decryption shares to the corrupted parties in the real world. However, $\mathcal{S}$

had sent the encryption of random items $\{\tilde{f}_i\}_{P_i \in \mathcal{P}_h}$ in the EIE phase. So, it cannot send the correct decryption shares of random ciphertexts $\{\tilde{\mathsf{VE}}_j\}_{j \in \mathcal{P}_h}$. Instead, for each $\tilde{\mathsf{VE}}_j$ where $P_j \in \mathcal{P}_c$, $\mathcal{S}$ runs ThDeny algorithm to obtain decryption shares that make decryption of $\tilde{\mathsf{VE}}_j$ be f_j. Assume that the correct decryption shares of a random ciphertext $\tilde{\mathsf{VE}}_j$ is $\{\tilde{d}_j^i = \mathsf{ThDShare}(x_i, \mathsf{pk}, \tilde{\mathsf{VE}}_j)\}_{P_j \in \mathcal{P}}$ and the index set of honest parties is $\mathcal{I}$ such that for all $i \in \mathcal{I}$, $P_i \in \mathcal{I}$.

For all $\tilde{\mathsf{VE}}_j$ where $P_j \in \mathcal{P}_h$, S obtains fake decryption shares $\{d_j^i\}_{P_i \in \mathcal{P}_h}$ by running $\mathsf{ThDeny}(\tilde{\mathsf{VE}}_j, \{\tilde{d}_j^t\}_{P_t \in \mathcal{P}_c}, \mathcal{I}, f_i, \mathsf{pk})$ and uses $\{d_j^i\}_{i \in \mathcal{P}_h}$ when simulating the step ④ for each encryption $\tilde{\mathsf{VE}}_j$. For the encryptions of corrupted parties $\{\mathsf{VE}_j\}_{j \in \mathcal{P}_c}$, it again uses ThDeny. For each encryption VE_j where $P_j \in \mathcal{P}_c$ (the encryptions generated by the corrupted parties), S runs $\mathsf{ThDeny}(\mathsf{VE}_j, \{d_j^i\}_{P_i \in \mathcal{P}_c}, \mathcal{I}, f_j, \mathsf{pk})$ and obtains valid decryption shares $\{d_j^i\}_{i \in \mathcal{P}_h}$ for VE_j. Remark that S has already learned the items of the corrupted parties $\{f_j\}_{P_j \in \mathcal{P}_c}$ and the decryption shares of corrupted parties $\{d_j^i\}_{P_i \in \mathcal{P}_c}$ generated for the encryptions $\{\mathsf{VE}_j\}_{j \in \mathcal{P}_c}$ of the corrupted parties.

$\mathcal{S}$ acts as $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ and sends $(\mathsf{proof}, id||P_i||P_j, (\mathsf{pk}, \tilde{\mathsf{VE}}_i, \{d_t^i\}_{P_t \in \mathcal{P}}, v))$ to each corrupted party P_j on behalf of each honest party P_i as a proof. $\mathcal{S}$ waits for the decryption shares from the corrupted parties. If all of them send their valid decryption shares, then the simulation ends by $\mathcal{S}$ outputting received items on behalf of real honest parties and whatever $\mathcal{A}$ outputs on behalf of ideal corrupted parties.

If some parties did not send their decryption shares to $\mathcal{S}$ before t_2, $\mathcal{S}$ simulates Resolve 2 with itself as the TTP as in the real protocol, and clears the *complaintList* (even if some corrupted parties also performed Resolve 1 before t_1) because it has all the correct verifiable escrows of the corrupted parties.

Case (2) requires $\mathcal{S}$ 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(s) would have complained about them before t_1 in Resolve 1. In addition, if a corrupted party performs Resolve 1, $\mathcal{S}$ behaves like the TTP and adds complainant and his complainee to the *complaintList*.

Moreover, $\mathcal{S}$ does not send any of P_i's decryption shares to others, as in the real protocol. If some of the corrupted parties come for Resolve 2, $\mathcal{S}$ behaves exactly as the TTP and clears the parties from the *complaintList* according to the given verifiable escrows. Each time it clears the *complaintList*, it learns the decryption shares of the complainee. It can perform Resolve 2 by using all decryption shares sent by the corrupted parties itself and clear the parts in *complaintList* where it has P_i's name as complainee. In the end, if the *complaintList* is empty, it means that S learned all the decryption shares of the corrupted parties. If so, it sends $\{f_j\}_{P_j \in \mathcal{P}_c}$ as ideal adversary and also $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$ as TTP to U. Then, U sends $\{f_i\}_{P_i \in \mathcal{P}_h}$ to $\mathcal{S}$. $\mathcal{S}$ calculates the decryption shares as described in case (1). In this case, $\mathcal{S}$ outputs the received items on behalf of real honest parties.

If *complaintList* is not empty at time t_2, $\mathcal{S}$ sends message ABORT to U and will return an abort message to all Resolve 3 attempts. $\mathcal{S}$ outputs $\perp$ on behalf of real honest parties.

In all cases, $\mathcal{S}$ simulates the resolutions by replacing the lines between 5-7 in Resolve 2 and 5-6 in Resolve 3 (they correspond where normally TTP needs to decrypt VS) as follows:

Resolve 2:

if $(*, (P_j, h_j), *) \in$ *complaintList* $\wedge$ $\mathtt{VerifyEnc}(\mathsf{pk}_T, \delta, \mathsf{VS}_j, \mathsf{VSproof}_j) \rightarrow \mathsf{valid}$

  if $P_j \in \mathcal{P}_h$

    $shares = \{d_t^j\}_{P_t \in \mathcal{P}}$ || $(\mathsf{proof}, id||P_j||P_i, (\mathsf{pk}, \tilde{\mathsf{VS}}_j, \{d_t^j\}_{P_t \in \mathcal{P}}, v))$

else:

$shares = \mathtt{PkDec}(\mathsf{sk}_T, \mathsf{VS}_j)$

$P_j.solvedList.\mathtt{Append}(shares)$

Resolve 3:

if $\mathsf{VS}_j \in \mathcal{V}$ and has correct label

if $P_j \in \mathcal{P}_h$

send $shares = \{d_t^j\}_{P_t \in \mathcal{P}} \mid\mid (\mathsf{proof}, id||P_j||P_i, (\mathsf{pk}, \tilde{\mathsf{VS}}_j, \{d_t^j\}_{P_t \in \mathcal{P}}, v))$

else:

send $\mathtt{PkDec}(\mathsf{sk}_T, \mathsf{VS}_j)$

Remark that this change is necessary for resolutions because $\mathcal{A}$ received $\{\tilde{\mathsf{VS}}_i\}_{P_i \in \mathcal{P}_h}$ which do not include the real decryption shares. Therefore, if $\mathcal{S}$ decrypted them, $\mathcal{A}$ would have received random decryption shares and could distinguish the simulator.

Finally, the simulator outputs whatever the adversary $\mathcal{A}$ outputs on behalf of ideal corrupted parties. This finishes our description of the simulator. We now show the simulator's actions remain indistinguishable from the adversary's view.

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

We prove this claim via a sequence of hybrid games. The initial game corresponds to the real protocol, whereas the final game corresponds to the simulator $\mathcal{S}$ described above. In each game, we change one (or more) step of MFE with the steps which are different in the simulation above. We start our first game with the items of honest parties as in MFE. Through these, in the last two games, we are able to simulate MFE only using random encryptions and random decryption shares The last game corresponds to the simulator above, without knowing the actual items of the honest parties. The details of the games with their reductions are below:

Game 1 : The adversary $\mathcal{A}$ who corrupts the parties in $\mathcal{P}_c$ in MFE wants to break the fairness. We honestly simulate the honest parties $\mathcal{P}_h$ and TTP as in MFE in the hybrid models $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$.

In this game, we use the items of parties as inputs of our simulation. Our simulation is identical to MFE.

Game 2 It is the same game as the previous game except that we simulate $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ such that they always output $(\mathsf{VEproof}, id||P_i||., ., \mathsf{pk}, \emptyset, \mathsf{VE}_i)$, $(\mathsf{VSproof}, id||P_i||., ., \mathsf{pk}_T, \emptyset, \mathsf{VS}_i)$, $(\mathsf{proof}, id||P_i||., .)$, respectively, for each honest party P_i. Because of the correctness of these functionalities, the game is identical to the previous one.

Game 3 : It is the same as the previous game except that we simulate the TTP where it needs to decrypt VS of honest parties; instead of decrypting, the simulator just returns the (already known) decryption shares of the honest parties, as below:

Resolve 2:

if $(^*, (P_j, h_j), ^*) \in complaintList \wedge \mathtt{VerifyEnc}(\mathsf{pk}_T, \delta, \mathsf{VS}_j, \mathsf{VSproof}_j) \rightarrow \mathsf{valid}$

if $P_j \in \mathcal{P}_h$

$shares = \{d_i^j\}_{1 \le i \le n} \mid\mid (\mathsf{proof}, id||P_j||P_i, (\mathsf{pk}, \tilde{\mathsf{VS}}_j, \{d_t^j\}_{P_t \in \mathcal{P}}, v))$

else:

$shares = \mathtt{PkDec}(\mathsf{sk}, \mathsf{VS}_j)$

$P_j.solvedList.\mathtt{Append}(shares)$

Resolve 3:

if $\mathsf{VS}_j \in \mathcal{V}$ and has correct label
  if $P_j \in \mathcal{P}_h$
    send $\{d_i^j\}_{1\leq i\leq n}$ || $(\mathsf{proof}, id||P_j||P_i, (\mathsf{pk}, \tilde{\mathsf{VS}}_j, \{d_t^j\}_{P_t\in\mathcal{P}}, v))$
  else:
    send $\mathtt{PkDec}(\mathsf{sk}, \mathsf{VS}_j)$

Remark that at this point, $\{d_i^j\}$'s are correct, since we did not start putting fake decryption shares yet. Because of the correctness of the verifiable encryption scheme, this game is indistinguishable from the previous game.

Game 4 : It is the same as the previous game except that the honest provers send encryption of random decryption shares in step ③. Intuitively, they are indistinguishable by the IND-CCA security of the verifiable escrow. The reduction from Game 3 to Game 4 is the following:

We define hybrid game $H_{3,i}$ where first i parties behave as in Game 3 and the rest of the simulated parties behaves as in Game 4. Again, for the sake of clarity of the hybrid argument, assume without loss of generality that $\mathcal{P}_h = \{P_i\}_{1\leq i\leq m}$. $H_{3,0}$ is equivalent to Game 4 and $H_{3,m}$ is equivalent to Game 3. We use the hybrid argument to show the indistinguishability of $H_{3,0}$ and $H_{3,m}$. If the adversary manages to distinguish $H_{3,0}$ and $H_{3,m}$ with non-negligible advantage, it must distinguish $H_{3,i}$ and $H_{3,i+1}$ for some i. If so, we can construct an adversary $\mathcal{B}$ which breaks the IND-CCA security of the verifiable escrow scheme, as follows:

The IND-CCA challenger sends public-key pk, and $\mathcal{B}$ publishes it as the public-key of the TTP. Then $\mathcal{B}$ guesses i randomly in range $[0, m-1]$, and does the following:

- For each party $P_j \in \{P_1, ..., P_i\}$, she encrypts $\{d_t^j, \mathsf{DSproof}_t^j\}_{1\leq t\leq n}$ for VS_j.
- For each party $P_j \in \{P_{i+2}, ..., P_m\}$, she encrypts $\{r_t\}_{1\leq t\leq n}$ for some random r_t from the same distribution of decryption shares and proofs for VS_j.

As the challenge query, $\mathcal{B}$ sends $m_0 = \{d_t^{i+1}, \mathsf{DSproof}_t^{i+1}\}_{1\leq t\leq n}$ and sets m_1 randomly and obtains back VS_{i+1}. It then continues interacting with the adversary as prescribed. Remember that at this point, all proofs regarding these verifiable escrows are simulated. Observe that if m_0 is encrypted by the IND-CCA challenger, then this corresponds to hybrid $H_{3,i+1}$, and if m_1 is encrypted, then this is hybrid $H_{3,i}$.

If corrupted parties do not send the decryption shares in step ④, then $\mathcal{B}$ calls decryption oracle to decrypt the corresponding encryption in VS values to learn the missing decryption shares and continues simulation. Note that we do not need to query the decryption oracle for VS_{i+1} during resolutions, since we changed the simulation of Resolve 2 and Resolve 3 since Game 3.

If the guess of i was correct and the adversary distinguishes between Game 3 and Game 4 with $\mathsf{adv}(\ell)$ advantage, then $\mathcal{B}$ can guess whether m_0 or m_1 was encrypted with the same advantage. The guess of i is correct with at least $1/m$ probability, and hence the IND-CCA security of the verifiable escrow is broken with at least $\mathsf{adv}(\ell)/m$ advantage. Since $\mathsf{adv}(\ell)/m$ must be negligible if we use a secure verifiable escrow scheme, $\mathsf{adv}(\ell)$ must be negligible as well, meaning that this behavior of our simulator remains indistinguishable to the adversary.

Game 5 : It is the same as the previous game except that we simulate the honest provers by encrypting random items in step ② and using decryption shares outputted by ThDeny algorithm. More specifically, each honest prover P_i encrypts a random item $\tilde{f}_i$ and

sends the encryption $\tilde{\mathsf{VE}}_i = \mathsf{ThEnc}(\mathsf{pk}, \tilde{f}_i)$ to all other parties. Normally, the valid decryption shares of the honest parties and the corrupted parties for $\tilde{\mathsf{VE}}_i$ are $\{\tilde{d}_i^j = \mathsf{ThDShare}(x_j, \mathsf{pk}, \tilde{\mathsf{VE}}_i)\}_{P_j \in \mathcal{P}}$. Instead of the valid ones, at step ④, each honest party P_j uses fake decryption shares as a decryption shares of $\{\tilde{\mathsf{VE}}_i\}_{P_i \in \mathcal{P}_h}$. The fake decryption shares $\{d_i^j\}_{P_j \in \mathcal{P}_h}$ for each $\tilde{\mathsf{VE}}_i$ are the output of $\mathsf{ThDeny}(\tilde{\mathsf{VE}}_i, \{\tilde{d}_i^j\}_{P_j \in \mathcal{P}_c}, \mathcal{I}, f_i, \mathsf{pk})$. Intuitively, Game 4 and Game 5 are indistinguishable because of the IND-CPA security of the deniable threshold encryption scheme. The reduction is as follows:
We define hybrid game $H_{4,i}$ where first i honest parties behave as in Game 4 and the rest of the simulated parties behave as in Game 5. For the sake of clarity of the hybrid argument, assume without loss of generality that $\mathcal{P}_h = \{P_i\}_{1 \leq i \leq m}$. $H_{4,0}$ is equivalent to Game 5 and $H_{4,m}$ is equivalent to Game 4. We use the hybrid argument to show the indistinguishability of $H_{4,0}$ and $H_{4,m}$. Against the adversary, $\mathcal{B}$ plays the honest parties $\{P_i\}_{1 \leq i \leq m}$. Against the IND-CPA challenger, $\mathcal{B}$ plays the honest parties $\{P_i\}_{m+1 \leq i \leq n}$.
If the adversary manages to distinguish $H_{4,0}$ and $H_{4,m}$ with non-negligible advantage, it must distinguish $H_{4,i}$ and $H_{4,i+1}$ for some i. If so, we can construct an adversary $\mathcal{B}$ which breaks the IND-CPA security of the threshold encryption scheme, as follows:
$\mathcal{B}$ picks i randomly in range $[1, m]$. Then, $\mathcal{B}$ obtains the secret keys $\{x_{m+1}, x_{m+2}, ..., x_n\}$, the public key pk, and the verification key v from the IND-CPA challenger. As the challenge query, $\mathcal{B}$ sends the actual item f_{i+1} and a random item $\tilde{f}_{i+1}$ and receives VE^*_{i+1}, which either encryption of f_{i+1} or $\tilde{f}_{i+1}$.
Then, $\mathcal{B}$ simulates each party $P_j \in \{P_1, ..., P_i\}$ as encrypting the correct item f_j, each party $P_j \in \{P_{i+2}, P_{i+3}, ..., P_m\}$ as encrypting a random item $\tilde{f}_j$, and P_{i+1} using VE^*_{i+1} as the item encryption of P_{i+1}. During the simulation of $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, it learns the decryption shares of the corrupted parties from the adversary.
$\mathcal{B}$ does not know the secret keys $x_1, x_2, ..., x_m$ but it can generate the decryption shares of the honest parties as follows:

- For the decryption shares for the encrypted items of the corrupted parties and decryption shares of the encrypted items of $\{P_j\}_{1 \leq j \leq i}$:

$$\{d_j^t\}_{P_t \in \mathcal{P}_h} = \mathsf{ThDeny}(\mathsf{VE}_j, \{d_j^t\}_{P_t \in \mathcal{P}_c}, \mathcal{I}, f_j, \mathsf{pk})$$

- For the decryption shares of the encrypted items of $\{P_j\}_{i+2 \leq j \leq m}$:

$$\{d_j^t\}_{P_t \in \mathcal{P}_h} = \mathsf{ThDeny}(\tilde{\mathsf{VE}}_j, \{\tilde{d}_j^t\}_{P_t \in \mathcal{P}_c}, \mathcal{I}, f_j, \mathsf{pk})$$

- For the decryption shares of the challenge ciphertext VE^*_{i+1} of P_{i+1}:

$$\{d_{i+1}^t\}_{P_t \in \mathcal{P}_h} = \mathsf{ThDeny}(\mathsf{VE}^*_{i+1}, \{d_{i+1}^t\}_{P_t \in \mathcal{P}_c}, \mathcal{I}, f_{i+1}, \mathsf{pk})$$

It simulates the proofs of all decryption shares via $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$.
Observe that if VE^*_{i+1} is not the encryption of f_{i+1} then $\mathcal{B}$ simulates $H_{4,i+1}$. Otherwise, it simulates $H_{4,i}$. Therefore, if the guess of i was correct and the adversary distinguishes between Game 5 and Game 4 with $\mathsf{adv}(\ell)$ advantage, then $\mathcal{B}$ can guess whether the actual item f_{i+1} or a random item was encrypted with the same advantage. The guess of i is correct with at least $1/m$ probability, and hence the IND-CPA security of the threshold encryption scheme is broken with at least $\mathsf{adv}(\ell)/m$ advantage. Since $\mathsf{adv}(\ell)/m$ must be negligible if we use a secure verifiable deniable threshold encryption scheme, $\mathsf{adv}(\ell)$ must be negligible as well, meaning that this behavior of our simulator remains indistinguishable to the adversary.

Claim 2. *The distributions of the outputs of honest and corrupted players in ideal and real worlds are indistinguishable:*

$$\{IDEAL^{fs}_{\phi,\mathcal{S}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, \ell)\}_{\ell\in\mathbb{N}} \equiv_c \{REAL^{fs}_{\pi,\mathsf{TTP},\mathcal{A}(aux),\mathcal{P}_c}(f_1, f_2, ..., f_n, \ell)\}_{\ell\in\mathbb{N}}$$

Proof. We showed in Claim 1 that the simulator's actions, on behalf of the honest parties in the real world, are indistinguishable from the MFE protocol. We also need to show the joint output of honest and corrupted parties are indistinguishable in the real and ideal worlds.
$\mathcal{S}$ sends the message $\{f_i\}_{P_i\in\mathcal{P}_h}$ as ideal adversary and $\{b_i = \text{CONTINUE}\}_{P_i\in\mathcal{P}_h}$ as TTP to U whenever it is guaranteed that the honest participants would obtain their desired items. At these cases, the honest parties output all the items $\{f_j\}_{P_j\in\mathcal{P}}$ in the ideal world. Similarly, at these cases, the simulated honest parties also obtain $\{f_j\}_{P_j\in\mathcal{P}}$ in the real world as discussed during the simulation.
$\mathcal{S}$ sends ABORT message to U after the end of the EIE phase when *complaintList* is not empty at time t_2. After the end of the EIE phase, if $\mathcal{S}$ sends ABORT message, it means that the adversary in the real world has only encryptions of random items, which are independent from the real items. Hence, the adversary in the real world does not output the items just as the adversary in the ideal world. Indeed, $\mathcal{S}$ outputs in the ideal world on behalf of corrupted parties whatever $\mathcal{A}$ outputs in the real world; hence adversarial parties' outputs in both worlds will always be indistinguishable. Since the protocol is aborted, both ideal and real (simulated) honest parties output $\perp$.

□

At the end of the simulation, the outputs of the parties in the ideal world are identically distributed to the outputs of the parties in the real protocol. This completes the proof of Theorem 1. □

Security of multiple exchanges with single Setup: Once the setup is performed, all the remaining actions of our simulator can be repeated for each set of item exchanges. Observe that the simulator never uses the secret keys in the simulation, which means that the interaction with the adversary cannot leak any useful information about the secret keys. Therefore, we can simulate multiple exchanges with a single setup phase.

4.3 Fair Secure Multi-Party Computation

In this section, we show how to adapt the MFE protocol to **any** secure multi-party computation (SMPC) protocol that computes general functionalities [17, 46, 32, 14, 84], to achieve fairness. Gordon et al. [51] show that given a fair secret-sharing reconstruction functionality that outputs the secret fairly to all parties given non-malleable shares of the parties, one can construct a fair and secure two-party computation protocol for any functionality. The idea is to employ a secure computation protocol that outputs encrypted outputs of the desired functionality together with the shares of the key used in the encryption. Then, the parties construct the key using the fair reconstruction functionality. We follow the same methodology in the multi-party case, where we employ the DSE phase of our MFE protocol to efficiently initialize this fair reconstruction functionality in the real world.

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 the party P_i. The protocol π^ϕ that allows parties to compute ϕ *securely and fairly*

is described below. π^{ϕ} is a composition of the Setup and DSE phases of MFE and an SMPC protocol which eliminates the need for the EIE phase of MFE. Therefore, as in MFE, we have a TTP which has the secret/public key pair $(\mathsf{sk}_T, \mathsf{pk}_T)$ generated by $\mathsf{PkGen}(1^{\ell})$ and every party knows pk_T.

- Each party P_i runs the Setup phase of MFE and obtains x_i, pk, v for the deniable (n,n)-threshold encryption scheme.
- All parties run an SMPC protocol π^{ψ} where the functionality $\psi = (\psi_1(w_1, w_2, ..., w_n), \psi_2(w_1, w_2, ..., w_n), ..., \psi_n(w_1, w_2, ..., w_n))$ and $\psi_i(w_1, w_2, ..., w_n) = \mathsf{ThEnc}(\mathsf{pk}, \phi_i)$ is the output of P_i. This corresponds to a functionality encrypting the outputs of the original function ϕ using the public key pk, which covers the EIE phase of the MFE.
  It is expected that everyone learns the output of ψ before a fair exchange occurs. If some party did not receive ψ 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 π^{ψ}, then they execute the DSE Phase of MFE. Differently from the DSE phase of MFE in the complete topology, each party P_i generates $n-1$ different verifiable escrows to send to each other party. For each party P_j, P_i sends the verifiable escrow VS_j of *only* the decryption share d_j^i and its proof $\mathsf{DSproof}_j^i$. Then, he sends *only* d_j^i and $\mathsf{DSproof}_j^i$ (see Figure 8). The reason for this is that P_j needs to decrypt *only* ψ_j to learn ϕ_j.

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 (d_i^i) to anyone else. Indeed, if a symmetric functionality is desired for the *underlying* SMPC protocol π^{ψ}, $\psi(z_1, z_2, ..., z_n)$ may be computed symmetrically (the difference is that all output ciphertexts are given to all parties), and since P_i does not give the decryption share of ϕ_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 and securely using our solution. Note also that we view π^{ψ} as black box.

Our overhead over just securely computing ϕ is minimal. Even though the input and output sizes are extended additionally by $O(n)$ values for n parties and the functionality is extended to perform encryptions, these are independent of the circuit size needed for the original computation, and can be computed efficiently especially if π^{ψ} works over *arithmetic circuits* (e.g., [14, 84]) when used with the ElGamal-based instantiation of the deniable threshold encryption scheme in Appendix B.

We proceed by proving first that our protocol is a fair and secure protocol for computing ϕ by employing a TTP to similarly prove the security with a corrupted TTP. Then, we prove (again via ideal-real simulation) that our protocol remains secure (though becomes unfair) even when the TTP acts maliciously and colludes with the malicious participants. Therefore, we argue that by employing our extensions on top of any SMPC protocol, **we efficiently gain fairness and lose nothing about security**.

4.3.1 Proof of Fairness and Security

Theorem 2. *π^{ϕ} is fair and secure in $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$, $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and U_s^{ψ} hybrid models according to Definition 2, assuming that the verifiable deniable (n,n)-threshold encryption scheme with* $(\mathsf{ThGen}, \mathsf{ThDShare}, \mathsf{ThDSProve}, \mathsf{ThDSVerify}, \mathsf{ThEnc}, \mathsf{ThDec}, \mathsf{ThDeny})$ *is IND-CPA secure*

and the verifiable encryption scheme with ($\mathsf{PkGen}, \mathsf{PkEnc}, \mathsf{PkDec}$*) is IND-CCA secure.*

In our proof, we use U_s^{ψ} to denote the ideal party in Definition 1 for the functionality ψ and U_{fs}^{ϕ} to denote the ideal party in Definition 2 for the functionality ϕ. We do our proof in the $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$, $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and U_s^{ψ} hybrid model.

Proof. We use the same notations for the corrupted parties $\mathcal{P}_c$ by $\mathcal{A}$ and the honest parties $\mathcal{P}_h$ and $\mathcal{P} = \mathcal{P}_c \cup \mathcal{P}_h$ as in the proof of Theorem 1. The simulator $\mathcal{S}$ simulates the honest parties and the TTP in the real world, and the corrupted parties in the ideal world. $\mathcal{S}$ does the following:

- $\mathcal{S}$ behaves as ThGen and gives the secret shares $\{x_i\}_{P_i \in \mathcal{P}_c}$ along with pk and v. Then, $\mathcal{S}$ simulates U_s^{ψ} and waits for the message of the corrupted parties. If $\mathcal{S}$ receives the message ABORT or invalid inputs then $\mathcal{S}$ aborts the protocol in the real world and sends the message ABORT to U_{fs}^{ϕ} in the ideal world. If $\mathcal{S}$ receives valid inputs $\{w_i\}_{P_i \in \mathcal{P}_c}$, it encrypts random messages with pk and obtains random encryptions $\{\tilde{\psi}_i\}_{i \in \mathcal{P}_c}$. Then, it gives $\{\tilde{\psi}_i\}_{i \in \mathcal{P}_h}$ to the corrupted parties. At this point, during the simulation of U_s^{ψ}, if $\mathcal{S}$ receives the message ABORT from $\mathcal{A}$ in the real world, it aborts the protocol and sends the message ABORT to U_{fs}^{ϕ} in the ideal world. If $\mathcal{S}$ receives the message CONTINUE it continues as below.
- The rest of the simulation is the same as the simulation of the DSE phase of MFE by replacing $\mathsf{VE}_i/\tilde{\mathsf{VE}}_j$ with $\psi_i/\tilde{\psi}_j$, U with U_{fs}^{ϕ}, where the output of U_{fs}^{ϕ} for each party is ϕ_i. When $\mathcal{S}$ contacts U_{fs}^{ϕ} to give the inputs of the corrupted parties, she gives $\{w_i\}_{P_i \in \mathcal{P}_c}$.

Claim 3. *The view of adversary $\mathcal{A}$ in his interaction with the simulator $\mathcal{S}$ is indistinguishable from the view in his interaction with real honest parties.*

The proof of Claim 3 is very similar to that of Claim 1. Remark that the only difference of the simulation of SMPC from the simulation of MFE is between the Setup and the DSE phase. Here, $\mathcal{S}$ behaves as U_s^{ψ} and provides encryptions of random outputs. This is the same as encrypting random items in MFE. So, the games in Claim 1 directly apply here, with the mentioned notation changes.

At the end of the simulation, the outputs of the parties in the ideal world are identically distributed to the outputs of the parties in the real protocol. The output indistinguishability can be shown just as in Claim 2. This completes the proof of Theorem 2.

□

4.3.2 Proof of Security Against Malicious TTP

In our fair SMPC solution, the privacy against the TTP is preserved. We formally prove this by providing a simulator for the case where the TTP is adversarial. Hence, this shows that even when the TTP is malicious and colluding, he can break fairness, but not privacy or correctness.

Theorem 3. *π^{ϕ} is secure in $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$, $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and U_s^{ψ} hybrid models according to Definition 2 when the **TTP is adversarial**, assuming that the verifiable deniable (n,n)-threshold encryption scheme with (*$\mathsf{ThGen}, \mathsf{ThDShare}, \mathsf{ThDSProve}, \mathsf{ThDSVerify}, \mathsf{ThEnc}, \mathsf{ThDec}, \mathsf{ThDeny}$*) is IND-CPA secure and the verifiable encryption scheme with (*$\mathsf{PkGen}, \mathsf{PkEnc}, \mathsf{PkDec}$*) is IND-CCA secure.*

Proof. The simulation is same as the proof of Theorem 2 until the simulation of the DSE phase. Before beginning this phase, $\mathcal{S}$ sends $\{w_i\}_{P_i \in \mathcal{P}_c}$ to U_s^{ϕ}. Here, $\mathcal{S}$ does not wait for being sure that honest parties will receive their outputs in the ideal world because we have the assumption that TTP is not honest meaning that fairness is not necessarily possible. Then, it receives $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ from U_s^{ϕ}. At this point, U_s^{ϕ} does not send the outputs of honest parties in the ideal world because it is waiting for the message of the TTP in the ideal world. The DSE phase simulation is the same as the DSE phase of MFE except that $\mathcal{S}$ does not simulate the resolve protocols.

Instead, remember that the TTP is now under adversarial control. Thus, in the real world, the TTP actions are controlled by the adversary, and the simulator must simulate the TTP's actions in the ideal world. Thus, $\mathcal{S}$ continues simulating honest party behavior for resolution protocols with the TTP, as directed by the protocol description. For each honest party P_t, there are three options:

- The malicious TTP indeed resolves with P_t correctly and hence the honest party P_t obtains its correct output. Similarly, it could be that P_t obtained its correct output during the protocol via interaction with the adversary. In these cases, $\mathcal{S}$ sends $b_t = \text{CONTINUE}$ to U_{fs}^{ϕ} and outputs the correct output of P_t in the real world.
- Until time $t_2 + \alpha$ (the latest time an honest resolution with an honest TTP would succeed) there is no response from the TTP. Then, $\mathcal{S}$ sends $b_t = \perp$ to U_{fs}^{ϕ} and outputs $\perp$ on behalf of P_t in the real world.
- The malicious TTP resolves with P_t incorrectly, trying to affect correctness. $\mathcal{S}$ can detect this, since when given any decryption share d_i^j of P_j for the encryption ψ_i and its proof $\mathsf{DSproof}_i^j$ by the TTP, $\mathcal{S}$ runs $\mathsf{ThDSVerify}(v, \mathsf{pk}, \psi_i, d_i^j, \mathsf{DSproof}_i^j)$. If any such $\mathsf{ThDSVerify}$ outputs $\mathsf{invalid}$, then $\mathcal{S}$ sends $b_t = \perp$ to U_{fs}^{ϕ} and outputs $\perp$ on behalf of P_t in the real world.

Note that honest parties cannot receive incorrect outputs from the TTP in the ideal and real worlds, since the output of the real TTP is verifiable with the $\mathsf{ThDSVerify}$ algorithm. Therefore, each honest party either receives its correct output or no output. This means, even when the TTP is malicious and colludes with the adversarial participants, it can selectively break fairness, but cannot harm privacy or correctness.

Finally, on behalf of the malicious parties in the ideal world, $\mathcal{S}$ outputs whatever the adversary in the real world outputs.

Similar simulator and output indistinguishability claims apply here. Indeed, $\mathcal{S}$ does not do anything as interesting as before in this proof. Therefore, we do not repeat another lengthy indistinguishability claim here.

At the end of the simulation, the outputs of the parties in the ideal world are identically distributed to the outputs of the parties in the real protocol. This completes the proof of Theorem 3. □

5 Coin-based Multi-Party Fair Exchange Protocol (CMFE)

MFE protocol (Section 4) is efficient if and only if the item that is exchanged can be efficiently verifiably encrypted. This can be very practical, for example, in contract signing scenarios, where a signature in the encryption can be verified easily with the corresponding verification key [8]. But, in scenarios where verifiable encryption of the items cannot be performed

efficiently (e.g., file exchanges), we cannot efficiently use our MFE protocol (neither can the previous MFE protocols). Therefore, to achieve fairness efficiently, we consider a different fairness definition (coin based fairness) for the exchange of these types of items. In this definition, if a party receives a wrong item, then (s)he can get the coin of the party who provided the wrong item. Note that, some existing two-party protocols (e.g., [16, 61, 65]) also apply this type of fairness definitions. They consider scenarios such as peer-to-peer file sharing where the hash of the file to be exchanged is known beforehand (e.g., via a trusted Tracker [30]).

Coin based fairness: The functionality ϕ in the coin based definition is as follows:

$$\phi(z_1, z_2, ..., z_n) = (\phi_1(z_1, z_2, ..., z_n), \phi_2(z_1, z_2, ..., z_n), ..., \phi_n(z_1, z_2, ..., z_n))$$

where $z_i = (coin_i, f_i)$ and $\phi_i = \{\rho_1, ..., \rho_{i-1}, \rho_{i+1}, ..., \rho_n\}$ (for complete topology). Each ρ_j is defined as:

$$\rho_j = f_j, \text{if } H(f_j) = H_{f_j} coin_j, \text{otherwise.}$$

Here, $coin_j$ is the electronic payment [28, 29]) and f_j is the item of P_j. H_{f_j} is the publicly known hash value of the item f_j, where H is a hash function. In BitTorrent [30], H_{f_j} is already obtained in the torrent file before the exchange begins; hence is known by every participant. Note that, the general condition for ρ_j values is that the actual item will be obtained as long as it is the agreed-upon item (e.g., hashes may be used for verifying files, a public-key infrastructure may be used for verifying signatures, see [61] for details), and the payment will be obtained otherwise. This condition applies to each ρ_j independently. Thus, it is possible that some item f_j is obtained from P_j, while some payment $coin_i$ is obtained from P_i. It is even possible in cases that P_i obtains f_j from P_j and P_k obtains $coin_j$ from P_j.

While relaxing our fairness definition this way to allow exchange of files and other general items, we want to minimize our changes to the MFE protocol so that we can keep enjoying its efficiency and privacy benefits. One problem occurs due to the public-key threshold encryption scheme we employed. Since it can encrypt only the elements in a certain set (e.g., elements in $\mathbb{Z}_p$ where p is a prime), we cannot use it to efficiently encrypt any general item (e.g., a large file). Therefore, we solve this problem by performing a symmetric key encryption of the file, and encrypting this key with a public-key threshold encryption.

But, this does not directly solve all the problems. Remember that our item (e.g., file) is still not verifiably encrypted. Assume we now want to verifiably encrypt the symmetric key that encrypts our file. What does it mean to have a correct key? What should the receiving party verify? This is where the electronic payments come into the picture. We can verifiably encrypt the payment, but not the file (or its key) efficiently. Thus, we also add a new resolution procedure that provides the party's payment in case the file is not obtained correctly. Full details of CMFE follow.

5.1 Description of CMFE

We assume that $\{H_{f_i}\}_{1 \leq i \leq n}$ are publicly known. For the sake of simplicity of presentation, the underlying electronic payment scheme is the Endorsed E-cash [24] scheme, as explained in the preliminaries. CMFE protocol also works in every exchange topology as explained in Section 6, but we explain CMFE in the complete topology for the sake of simplicity. As in MFE, CMFE is constructed on top of a verifiable and deniable (n,n)-threshold encryption scheme with (ThGen, ThDShare, ThDSProve, ThDSVerify, ThEnc, ThDec, ThDeny) and another verifiable encryption scheme with (PkGen, PkEnc, PkDec). In addition, CMFE employs a signature scheme

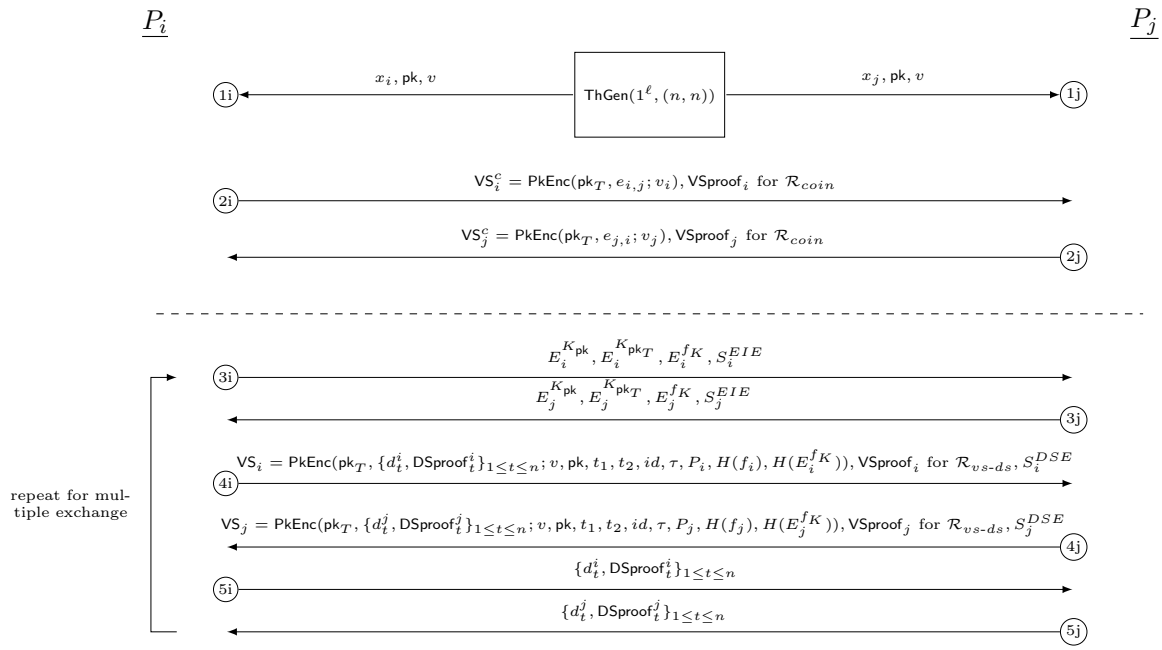

Figure 2: Our CMFE Protocol. Each (i, j) message pair can be performed in any order or in parallel within a step.

$(\mathsf{SgGen}, \mathsf{SgSign}, \mathsf{SgVerify})$ and a symmetric key encryption scheme $(\mathsf{SymGen}, \mathsf{SymEnc}, \mathsf{SymDec})$. The TTP has the secret/public key pair $(\mathsf{sk}_T, \mathsf{pk}_T)$ generated by $\mathsf{PkGen}(1^\ell)$. Our CMFE protocol has the following phases:

Setup Phase (① and ② in Figure 2): The parties obtain their secret shares x_i, the public key pk, and the verification key v for $\mathsf{ThGen}(1^\ell, (n, n))$ as in the Setup Phase of MFE. Additionally, each P_i generates a signature signing-verification key-pair (s_i, v_i) by running $\mathsf{SgGen}(1^\ell)$. Then, each party P_i generates a verifiable escrow of the endorsement e_i of an unendorsed coin $coin_i^u$ with the TTP's public-key pk_T, and sends the encryption $\mathsf{VS}_i^c = \mathsf{PkEnc}(\mathsf{pk}_T, e_i; v_i)$ proving that the payment is valid $((e_i, e_i, coin_i^u) \in \mathcal{R}_{coin})$ where $coin_i^u$ is sent in clear. The signature verification key v_i is the label of VS_i^c. This label is necessary to link a later escrow (explained below) with this one, so that the TTP can understand that both were sent by the same party. As usual, upon receipt, each party verifies the values. This ensures that the coin will be valid once endorsed with the endorsement in the verifiable escrow.[7]

The parties execute the Setup Phase only once, as in MFE. They can continue exchanging multiple sets of items with the same set of parties by repeating only the following phases. If a party P_i receives the coin of a party P_j during a dispute, to continue with the next item exchange, P_j needs to send a new VS_j^c to only P_i, without the need to repeat the Setup Phase completely (note that electronic payment schemes have methods to either prevent or penalize double spenders, and hence we do not need to worry about some coin being used multiple times in our protocol). In some sense, think of every participant putting down some money; if theirs is spent, they need to renew it to continue.

[7]For efficiency, the verifiable escrow used here can be Camenisch-Shoup verifiable escrow [25], which can be efficiently used to verifiably encrypt Endorsed E-cash coins [24].

Encrypted Item Exchange (EIE) Phase (③ in Figure 2): The EIE phase is different from MFE in that there is no verifiable encryption. Since it is hard to encrypt a large item with public key threshold encryption schemes, each party P_i first encrypts their item f_i (e.g., a file) using a symmetric encryption scheme (e.g., AES) with a key K_i generated by $\mathsf{SymGen}(1^\ell)$ and obtains $E_i^{f_K} = \mathsf{SymEnc}(K_i, f_i)$. Then, P_i encrypts K_i with the threshold encryption public-key pk : $E_i^{K_{\mathsf{pk}}} = \mathsf{ThEnc}(\mathsf{pk}, K_i)$. In addition, he encrypts K_i with the TTP's public-key pk_T and obtains $E_i^{K_{\mathsf{pk}_T}} = \mathsf{PkEnc}(\mathsf{pk}_T, K_i)$. This extra encryption is needed during the resolutions. He also signs with his signing key s_i the ciphertext $E_i^{K_{\mathsf{pk}_T}}$: $S_i^{EIE} = \mathsf{SgSign}(s_i, E_i^{K_{\mathsf{pk}_T}})$. S_i^{EIE} helps the TTP to understand that VS_i^c and $E_i^{K_{\mathsf{pk}_T}}$ are encrypted by the same party (remember, VS_i^c contains the signature verification key in its public non-malleable label). P_i sends $E_i^{f_K}, E_i^{K_{\mathsf{pk}}}, E_i^{K_{\mathsf{pk}_T}}, S_i^{EIE}$ to all parties.

No party continues to the next phase without completing this phase and receiving all messages and verifying signatures. If anything goes wrong, parties can locally abort.

Decryption Share Exchange (DSE) Phase (④ and ⑤ in Figure 2): This is almost the same as the MFE protocol, except P_i additionally puts $H(f_i)$ and $H(E_i^{f_K})$ into the label of VS_i, and also creates and sends the signature S_i^{DSE} of VS_i (again tying VS_i to VS_i^c). Since H_{f_i} is public, the other parties can check it against $H(f_i)$. Since they received $E_i^{f_K}$ in the previous phase, they can also verify the correctness of $H(E_i^{f_K})$ in the label. If no problem occurs while checking the correctness of the label or verifying the verifiable escrow or the signature, they continue with the next step. Otherwise, they do **Resolve 1**.

In the next step, P_i sends the decryption shares $\{d_t^i\}_{1\leq t\leq n}$ with their proofs $\{\mathsf{DSproof}_t^i\}_{1\leq t\leq n}$ as in MFE. If P_i does not receive valid decryption shares, **Resolve 2** is executed. After obtaining the decryption shares either at the end of the protocol, or after **Resolve 3**, P_i gets the key K_j from $E_j^{K_{\mathsf{pk}}}$ and then decrypts $E_j^{f_K}$ to obtain some f_j'. It is possible, since the item was not verifiably encrypted, that K_j is wrong and the decrypted file f_j' has a different hash (i.e. $H(f_j') \neq H_{f_j}$). In that case P_i runs **Resolve 4** (below) with the TTP.

Remark that Resolve 1-2-3 are the same and can be executed after the same phases as in MFE.

Resolve 4: Note that if the parties reach this point, all complaints were already resolved in the Resolve 1-2-3. First, P_i requests the key in $E_j^{K_{\mathsf{pk}_T}}$ from the TTP. The TTP decrypts $E_j^{K_{\mathsf{pk}_T}}$ and sends the resulting K_j to P_i, if the *complaintList* is empty. If the K_j is a wrong key, meaning that it decrypts $E_j^{f_K}$ to some f_j' with $H(f_j') \neq H_{f_j}$, then P_i proves that K_j is not correct using the Belenkiy et al. subprotocol [16] below, and obtains the endorsement e_j in VS_j^c from the TTP, thereby obtaining the payment of P_j. See Algorithm 4. An honest party would have performed Resolve 4 the latest at time $t_2 + 2\alpha$ (one delay for Resolve 3 and another delay for Resolve 4).

Prove Key is not Correct by Belenkiy et al. [16]: Belenkiy et al. showed a communication-efficient and partially-privacy-preserving way of proving that the decryption of $E_j^{f_K}$ with the key K_j does not result in an f_j with hash H_{f_j}. It is equivalent to showing that the key does *not* decrypt properly. Observe that this can be done by simply sending a ciphertext that matches $H(E_j^{f_K})$ in the label of the verifiable escrow VS_j. Then, the TTP can decrypt this with K_j and check if the hash of the plaintext equals H_{f_j}. However, this resolution is very inefficient since the TTP needs to obtain the whole (potentially large) file.

Algorithm 4 Resolve 4

1: P_i sends $v, \mathsf{pk}, t_1, t_2, id, \tau, P_j$ and $E_j^{K_{\mathsf{pk}_T}}, \mathsf{VS}_j^c, S_j^{EIE}$, VS_j, S_j^{DSE} to the TTP where P_j is the party whose item with hash H_{f_j} could not be obtained. The TTP does the following:
2: **if** $currenttime < t_2$ **or** (S_j^{EIE} **or** S_j^{DSE} is not valid when verified with the signature verification key in the label of VS_j^c) **then**
3: **send** msg "Abort Resolve 4"
4: **end if**
5: $complaintList = \texttt{GetComplaintList}\,(\mathsf{pk}, v, id, t_1, t_2, \tau)$
6: **if** $complaintList$ is empty **then**
7: $\mathsf{topology} = \texttt{GetTopology}(\tau)$
8: **if** $P_j \in \mathsf{topology}[P_i]$ **then**
9: $K_j = \texttt{PkDec}(\mathsf{sk}_T, E_j^{K_{\mathsf{pk}_T}})$
10: **send** K_j
11: // Run sub-protocol only if requested by P_i
$output \leftarrow$ **execute** sub-protocol "Prove Key is not Correct" with P_i
12: **if** $output$ is true **then**
13: **send** $\texttt{PkDec}(\mathsf{sk}_T, \mathsf{VS}_j^c)$ // P_i obtains coin of P_j instead of f_j
14: **else**
15: **send** msg "Abort Resolve 4"
16: **end if**
17: **else**
18: **send** msg "Abort Resolve 4"
19: **end if**
20: **else**
21: **send** msg "Abort Resolve 4"
22: **end if**

Thus, Belenkiy et al. solution improves resolution efficiency here.

The TTP knows $H(E_j^{f_K})$ and $H(f_j)$ from the label of the verifiable escrow VS_j given by P_i (remember that the label is non-malleable). He also knows K_j as he decrypted and obtained from $E_j^{K_{\mathsf{pk}_T}}$. He further verified both signatures S_j^{EIE}, S_j^{DSE} against the same verification key in the label of VS_j^c, and hence he is sure that all these belong to the same participant.

Note that for this subprotocol to work efficiently, the hashes need to be Merkle hashes [72], and the file needs to be encrypted block-by-block with the same key. Hence, $H(E_j^{f_K})$ and H_{f_j} values are the root hashes of the corresponding Merkle trees.

The TTP challenges some random blocks of the file. P_i sends the corresponding path proofs of Merkle hash trees of f_j and $E_j^{f_K}$, as well as the associated blocks of f_j and $E_j^{f_K}$. The TTP verifies the Merkle hash tree paths against the hash tree roots $H(f_j)$ and $H(E_j^{f_K})$ that he knows. Then, the TTP decrypts the given blocks of the ciphertext with the key K_j, and checks if they do *not* match the file blocks (so the key is indeed wrong). See [16, 61] for the details and security analysis.

Note that throughout the protocol, the TTP only learns some random blocks of the file, and some hash values in the Merkle tree. Thus, even though it is not fully privacy preserving, the protocol does not reveal the whole item either. This protocol was designed for efficiently verifying even large items without the need to transfer them to the TTP, but this partial privacy comes as a side benefit. Since we directly employ MFE in our fair SMPC solution, this does not affect the privacy there.

5.2 Fairness proof of CMFE

Below, we discuss security of two versions of our CMFE protocol. In the **first version**, for the item encryption $E_i^{f_K}$ we employ an efficient symmetric key encryption scheme such as AES. In the **second version**, the item is encrypted using a sender deniable encryption scheme.

For the first version, we note that none of the previous work in the multi-party fair exchange scenario employed such simulation proofs; they only analyze fairness case-by-case. Thus, we also provide a fairness analysis below and argue that our protocol will achieve fairness.

For the second version, we achieve a much stronger level of security: We provide a full simulation proof for the security and fairness of CMFE according to Definition 2. This version

is less efficient using current schemes [27, 81].[8]

5.2.1 Fairness Analysis for Version 1

Note, first of all, that *fair* here refers to the world where either each party received every other party's item, or no party received any item, where each *item* refers to either the file or the coin of a party.

Correctness: If all parties are honest and no problem occurs in the network, each party will have other parties' items because of the correctness of the threshold encryption scheme, the correctness of the signature scheme, and the correctness of the symmetric encryption scheme.

Correctness of the Resolve Protocols: We first show that the resolve protocols protect fairness in the case that all parties act honestly in Resolve 1-2-3-4. (Note that resolutions may still be necessary due to network issues.) Remember that an honest party P_h would do Resolve 1 after the step ④ and Resolve 2 after the step ⑤.

For the simplicity of the analysis, assume complete topology for now. We define three sets $\mathcal{Y}, \mathcal{Z}, \mathcal{W}$ as follows: $\mathcal{Y}$ includes the parties that did not receive VS from at least one participant, even though they sent their own VS to the other parties in ④. Thus, they did not execute ⑤. $\mathcal{Z}$ includes the parties that did not receive others' decryption shares, even though they sent their own decryption shares in ⑤. It means that the parties in $\mathcal{Z}$ received VS's from all parties in ④. $\mathcal{W}$ includes the parties that did not send their verifiable escrow to at least one party in $\mathcal{Y}$.

The parties in $\mathcal{Y}$ contact the TTP for Resolve 1, and complain about the parties in $\mathcal{W}$. If the parties in $\mathcal{W}$ contact the TTP in Resolve 2, they can provide their decryption shares to the TTP. So, they are removed from *complaintList*. The other way to clear *complaintList* is that some parties in $\mathcal{Z}$ contact the TTP and give all the verifiable escrows that they received, including the parties in $\mathcal{W}$. Since all parties act honestly, the *complaintList* will be empty at time t_2.

When *complaintList* is empty by t_2, it means that the TTP saved all the decryption shares needed by the parties in $\mathcal{Y}$. Thus, they can obtain those in Resolve 3. The parties in $\mathcal{W}$ and $\mathcal{Z}$ also obtain the necessary decryption shares in Resolve 3, if needed. After this point, every party can learn the symmetric keys and then decrypt the ciphertexts E^{f_K} and learn the files. If some files are not the correct ones, then they perform Resolve 4.

Thanks to the security of the Belenkiy et al. subprotocol, Resolve 4 protocol guarantees that the requesting party can receive the endorsement of the coin if he does not receive the correct item. Note that, the correct item means that an item described in the verifiable escrow VS_j (e.g., the file with hash H_{f_j}).

Fairness: We now consider the actions of the malicious (and colluding) parties.

To decrypt E^{f_K}, the parties need to know the corresponding key K because of the security of the symmetric encryption scheme. To learn K, they need to learn all decryption shares so that they can decrypt E^{K_h}, or get help from the TTP in Resolve 4 (discussed later). There are two options to obtain the shares:

[8] We can employ a symmetric-key deniable encryption scheme with a short key (unlike one time pad), or an efficient public-key sender-deniable encryption, if developed in the future. One may also consider employing non-committing encryption schemes [56, 75], but our simulator employs shared-key deniable encryption as we defined in the preliminaries.

1. All the malicious parties sent their verifiable escrows to the honest parties in step ④, and thus the honest parties will send their decryption shares in step ⑤.
2. Some malicious parties did not send their verifiable escrows to some honest parties in step ④. In this case, the only way to obtain the decryption shares would be via resolution with the TTP, since those honest parties would not proceed to step ⑤.

Note that before step ④, no party has useful information about the decryption shares. We consider the two cases above in detail:

Case 1: In this case, the honest party (or parties) all have received all the correct verifiable escrows. Therefore, they will learn all the decryption shares from the TTP in Resolve 3 in the worst case, as follows: The honest parties (who correspond to set $\mathcal{Z}$ above) can have all the verifiable escrows decrypted during the resolutions when *complaintList* is empty. But, malicious parties may perform unnecessary Resolve 1 operations with the TTP to populate the *complaintList*. Yet, remember that the honest parties already have all the verifiable escrows. Thus, in Resolve 2, they will give all the verifiable escrows to the TTP, and the TTP will save the decryption shares in *solvedList*, removing everyone from the *complaintList*. Then, when the honest parties perform Resolve 3, the *complaintList* is guaranteed to be empty, and they can obtain the decryption shares for all the items. Moreover, if malicious parties put wrong keys (or files), after Resolve 4 as discussed above, all honest parties would receive the coins of the corresponding malicious party. Therefore, fairness is guaranteed, at the worst case, immediately after t_2 (indeed, $t_2 + 2\alpha$).

Case 2: In this case, the malicious parties need the TTP's help to be able to obtain the decryption shares of the honest parties. Note that, since some malicious party P_m did not send its verifiable escrow to some honest party P_h, P_h will complain about P_m in Resolve 1. Therefore, the TTP's *complaintList* will not be empty. But, to be able to obtain the decryption of the verifiable escrow of P_h, P_m needs to clear *complaintList*, by giving away his correct shares, since P_h would not reply back with his decryption share without the verifiable escrow of P_m. Note that, if there are multiple concerned honest parties, all will complain about P_m (and all other malicious parties who did not send their verifiable escrows).

Similarly, if some malicious party performs Resolve 3, *complaintList* must be empty for him to obtain the decryption of some verifiable escrow, which means all the malicious parties must have already given their correct decryption shares in Resolve 2. Thus, the honest parties may also perform Resolve 3 and obtain the decryption shares of the malicious parties, ensuring fairness. If the *complaintList* is not empty by time t_2, then no party can obtain all the decryption shares, and hence no party can obtain any item. P_m cannot obtain the key or coin of P_h in Resolve 4 as well, since *complaintList* is not empty.

Additionally, malicious parties may want to do two things by **Resolve 4**:

1. Try to put both the wrong file and an invalid coin, thus not giving away anything to the honest party, from whom she received the correct file.
2. Try to obtain both the correct file and valid coin of an honest party.

The first case is not possible, since the coin endorsement is verifiably escrowed (thus, is correct), and the Belenkiy et al. subprotocol is secure. Remember that if the adversary manages to obtain the correct file of the honest party, by the fairness discussion above, the honest party would also receive the adversary's file (correct or wrong), at the latest in Resolve 3. Thus, if the adversary sent a wrong file, the honest party can prove this via Belenkiy et al. subprotocol, and obtain the verified endorsement from the TTP, thereby obtaining the adversary's valid coin in exchange.

As for the second case, due to the security of the Belenkiy et al. subprotocol, if the key is correct, the adversary would not be able to prove that it is wrong key to the TTP. Thus, the TTP would never decrypt the verifiably escrowed endorsement of an honest party (except with negligible probability). At a high level, such an adversary can be used to break the e-cash protocol or the collision resistance of the hash function (see a similar reduction in [61]).

5.2.2 Simulation Proof for Version 2

Theorem 4. *The CMFE protocol in the complete topology is fair in* $U^{\mathsf{VS}\text{-}\mathcal{R}_{coin}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, *and* $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ *hybrid models according to Definition 2, assuming that the verifiable deniable (n,n)-threshold encryption scheme with* $(\mathsf{ThGen}, \mathsf{ThDShare}, \mathsf{ThDSProve}, \mathsf{ThDSVerify}, \mathsf{ThEnc}, \mathsf{ThDec}, \mathsf{ThDeny})$ *is IND-CPA secure and the verifiable and non-verifiable escrow schemes with (*$\mathsf{PkGen}, \mathsf{PkEnc}, \mathsf{PkDec}$*) are IND-CCA secure, the payment (e-cash) scheme is unforgeable, the signature scheme* $(\mathsf{SgGen}, \mathsf{SgSign}, \mathsf{SgVerify})$ *is existentially-unforgeable under chosen-message attack, the hash function is chosen from a (target) collision resistant family (required for Belenkiy et al. subprotocol), and the symmetric encryption scheme* $(\mathsf{SymGen}, \mathsf{SymEnc}, \mathsf{SymEnc}, \mathsf{SymFake})$ *used for the item encryption is secure sender deniable with the faking algorithm* $\mathsf{SymFake}$.

Proof. We do our proof in the $U^{\mathsf{ZK}\text{-}\mathcal{R}_{dl}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{coin}}$, $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ hybrid model. As in the previous proofs, assume that an adversary $\mathcal{A}$ corrupts the parties in the set $\mathcal{P}_c$ and that the set of uncorrupted (honest) parties is $\mathcal{P}_h$. The simulator $\mathcal{S}$ simulates the honest parties in the real world, and the corrupted parties in the ideal world. $\mathcal{S}$ also acts as the TTP in the protocol if any resolution protocol occurs, since TTP is honest. So, $\mathcal{S}$ generates secret/public key-pair $(\mathsf{sk}, \mathsf{pk})$ and publishes pk as the TTP. $\mathcal{S}$ does the following:

Setup Phase: $\mathcal{S}$ behaves almost the same as the simulator of MFE protocol in the Setup Phase of CMFE. Differently, $\mathcal{S}$ behaves as $U^{\mathsf{VS}\text{-}\mathcal{R}_{coin}}$. It sends $(\mathsf{VSproof}, id||P_i||P_j, coin^u_{i,j}, \mathsf{pk}, label, \tilde{\mathsf{VS}}^c_{i,j})$ on behalf of each $P_i \in \mathcal{P}_h$ where $\tilde{\mathsf{VS}}^c_{i,j}$ includes fake endorsements. Then, it waits for $(\mathsf{VSprove}, id||P_i||P_j, e_{i,j}, coin^u_{i,j}, \mathsf{pk}, v_i)$ from the corrupted parties ($P_i \in \mathcal{P}_c$) obtaining valid endorsed coins $coin_i$ of the corrupted parties.

$\mathcal{S}$ does not proceed the next phase until it receives all necessary values properly.

EIE Phase: $\mathcal{S}$ behaves exactly as in the protocol description, except that it encrypts random items $\{\tilde{f}_i\}_{P_i \in \mathcal{P}_h}$ inside $\{E_i^{\tilde{f}_K}\}_{P_i \in \mathcal{P}_h}$. Then, it waits for the corresponding values from the corrupted parties, and it does not execute the next step before successfully completing this one.

DSE Phase: $\mathcal{S}$ does the same computations as in the step ④ on behalf of honest parties. It also saves the signatures that it sends as $(i, S_i^{EIE}, S_i^{DSE})$ to the list $SignList$. Then, $\mathcal{S}$ as $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$ waits for the values in the step ④ from the corrupted parties. The same cases (Case (1) and Case (2)) as in the simulation of MFE can occur for each $P_j \in \mathcal{P}_h$:

Case (1): By time t_1, if $\mathcal{S}$ received all $\{d_j^i\}_{P_i \in \mathcal{P}_c}$ for all j, it means that all parties may obtain the required items because $\mathcal{S}$ in the real world is now able to learn all decryption shares (or coins) from the corrupted parties via Resolve 1, 2, 3, 4. $\mathcal{S}$ decrypts $\{E_j^{K_{\mathsf{pk}_T}}\}_{P_j \in \mathcal{P}_c}$ (since it obtained all decryption shares behaving as $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$) and learns the keys $\{K_j\}_{P_j \in \mathcal{P}_c}$. Then, it decrypts $\{E_j^{f_K}\}_{P_j \in \mathcal{P}_c}$. It sends all $\{f_j\}_{P_j \in \mathcal{P}_c}$ to U together with all $\{coin_j\}_{P_j \in \mathcal{P}_c}$ (that it obtained during the Setup Phase) as ideal adversary and $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$ as TTP to U. Afterward, U checks the correctness of each item f_i using their public hash values. If some of them is not the correct item, U sends corresponding coins to the parties in the ideal world

instead of the file. Besides, U sends $\{f_i\}_{P_i \in \mathcal{P}_h}$ to $\mathcal{S}$. It is sure that U outputs $\{f_i\}_{P_i \in \mathcal{P}_h}$ to $\mathcal{S}$ because those come from the honest parties in the ideal world; so none can be a coin.

$\mathcal{S}$ uses the deniable encryption scheme's faking algorithm $\mathsf{SymFake}(\tilde{f}_i, f_i, r_S, r_R, E, K_i)$, where r_S and r_R are the randomness used in $E_i^{\tilde{f}_K}$, to calculate the keys $\{K_i'\}_{P_i \in \mathcal{P}_h}$[9]. For each $P_i \in \mathcal{P}_h$, it finds a key K_i' such that when a party decrypts $E_i^{\tilde{f}_K}$, it obtains the correct item f_i.

Remark: This process prevents us from using efficient encryption schemes such as AES because in this case $\mathcal{S}$ cannot find such keys $\{K_i'\}_{P_i \in \mathcal{P}_h}$ after encrypting random items initially, without actually breaking the scheme. Thus, we need a sender deniable encryption scheme (Definition 4) for the simulation to proceed.

Let us assume that the decryption shares of $E_i^{K_{\mathsf{pk}}}$ are $\tilde{DS} = \{\tilde{d}_i^j\}_{P_j \in \mathcal{P}}$ for all $P_i \in \mathcal{P}_h$. For each $P_i \in \mathcal{P}_h$, $\mathcal{S}$ computes $\{d_i^j\}_{P_j \in \mathcal{P}_h}$ by using $\mathsf{ThDeny}(E_i^{K_{\mathsf{pk}}}, \{\tilde{d}_i^j\}_{P_j \in \mathcal{P}_j}, \mathcal{I}, K_i', \mathsf{pk})$ such that the other parties can get the key K_i' from $\mathsf{ThDec}(\{\tilde{d}_i^j\}_{P_j \in \mathcal{P}_c} \cup \{d_i^j\}_{P_j \in \mathcal{P}_h}, \mathsf{pk}, E_i^{K_{\mathsf{pk}}})$ where I is the index set of honest parties.

$\mathcal{S}$ acts as $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ and sends as $\mathsf{DSproof}_i$ $(\mathsf{proof}, id||P_i||P_j, (\mathsf{pk}, \tilde{\mathsf{VE}}_i, \{d_t^i\}_{P_t \in \mathcal{P}}, v))$ to each corrupted party $P_j \in \mathcal{P}_c$ on behalf of each honest party $P_i \in \mathcal{P}_h$. If all of the corrupted parties send their valid decryption shares, then the simulation ends by the simulator outputting the items (if correct item is obtained in the real protocol) or coins of malicious parties (if item is incorrect, then $\mathcal{S}$ uses coins obtained during setup) on behalf of the real honest parties and whatever the adversary outputs on behalf of the ideal corrupted parties.

If some of the parties send wrong decryption shares or do not send any decryption shares, $\mathcal{S}$ simulates the resolutions as in the real protocol (also acting as the TTP) and clears the all *complaintList* if necessary because it has all corrupted parties' verifiable escrows. The simulation ends after all resolutions successfully complete by the simulator outputting the items (if correct item is obtained in the real protocol) or coins of malicious parties (if item is incorrect, then $\mathcal{S}$ uses coins obtained during setup) on behalf of the real honest parties and whatever the adversary outputs on behalf of the ideal corrupted parties.

Case (2): If some of the corrupted parties do not send a verifying message to $U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$ before t_1, $\mathcal{S}$ adds these parties to the *complaintList* (simulating honest parties performing Resolve 1 with the TTP) and continues as the simulator of MFE in the proof of Theorem 1, potentially performing other resolutions. Essentially, if *complaintList* becomes empty, then the simulator sends $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$ as TTP to U. In this case, the simulator outputs the items (if correct item is obtained in the real protocol) or coins of malicious parties (if item is incorrect, then $\mathcal{S}$ uses coins obtained during setup) on behalf of the real honest parties and whatever the adversary outputs on behalf of the ideal corrupted parties. If the *complaintList* is not empty at time t_2, then the simulator sends $\{b_i = \perp\}_{P_i \in \mathcal{P}_h}$ as TTP to U. In this case, the simulator outputs $\perp$ on behalf of the real honest parties and whatever the adversary outputs on behalf of the ideal corrupted parties.

In all cases, $\mathcal{S}$ simulates Resolve 2 and Resolve 3 as described in the proof of Theorem 1. Besides, it simulates Resolve 4 by only modifying the lines 2-4, 8-9 and 11-13 of Resolve 4 as follows:

Resolve 4:

We replace lines 2-4 of Algorithm 4 with the ones below instead of actually verifying the

[9]If an asymmetric key deniable encryption scheme is employed, the faking algorithm generates the fake secret key, which is sent so that the obtainer can decrypt to the desired item.

signatures using $SgVerify$. So, the signatures of the honest parties that are not generated by the simulator are never accepted.

if $P_j \in \mathcal{P}_h$
  if $currenttime < t_2$ **or** $(j, S_j^{EIE}, S_j^{DSE}) \notin SignList$
    send msg "Abort Resolve 4"
else: as in 2-4 in Algorithm 4

We also replace lines 8-9 of Algorithm 4 with the ones below in order to give the correct key K_j that makes $\mathsf{SymDec}(K_j, E_j^{f_K})$ output the valid item f_j.

if $P_j \in \mathsf{topology}[P_i]$
  if $P_j \in \mathcal{P}_h$
    $K_j = K'_j$
  else:
    $K_j = \mathsf{PkDec}(E_j^{K_{\mathsf{pk}_T}}, \mathsf{sk}_T)$

We replace lines 11-13 of Algorithm 4 with the ones below so that the honest parties' fake endorsements are never exposed.

if $P_j \in \mathcal{P}_h$
  send msg "Abort Resolve 4"
else: as in 11-13 in Algorithm 4

This finishes our simulator.

Claim 4. *The view of adversary $\mathcal{A}$ in his interaction with the simulator $\mathcal{S}$ is indistinguishable from the view in his interaction with real honest parties and the TTP.*

As usual, we prove this claim via a sequence of hybrid games. The initial game corresponds to the real protocol, whereas the final game corresponds to the simulator described above. Since there are similarities to the proof of Claim 1, we do not repeat such details.

Game 1 : The adversary $\mathcal{A}$ who corrupts the parties in $\mathcal{P}_c$ in CMFE wants to break the fairness. We simulate the honest parties $\mathcal{P}_h$ and TTP in the real protocol.

Game 2 It is the same game as the previous game except that we simulate $U^{\mathsf{VE}\text{-}\mathcal{R}_{item}}, U^{\mathsf{VS}\text{-}\mathcal{R}_{vs\text{-}ds}}$, and $U^{\mathsf{ZK}\text{-}\mathcal{R}_{ds}}$ such that they always output $(\mathsf{VEproof}, id||P_i||., ., \mathsf{pk}, \emptyset, \mathsf{VE}_i)$, $(\mathsf{VSproof}, id||P_i||., ., \mathsf{pk}_T, \emptyset, \mathsf{VS}_i)$, $(\mathsf{proof}, id||P_i||., .)$, respectively, for each honest party P_i. Because of the correctness of these functionalities, the game is identical to the previous one.

Game 3 : It is the same as the previous game except that we simulate Resolve 2 and 3 as in Game 3 of the proof of Claim 1 and lines 8-9 of Resolve 4 as described in the simulation above where $K'_j = K_j$. Because of the correctness of the verifiable and not verifiable encryption scheme, this game is indistinguishable from the previous game.

Game 4 : It is the same as the previous game except that we simulate lines 2-4 of Resolve 4 as described in the simulation. The only difference between Game 3 and Game 4 is in the case that the TTP receives a valid signature that is not generated by an honest party. One can easily prove that this case happens with negligible probability due to the unforgeability of the signature scheme.

Game 5 : It is the same as the previous game except that we simulate lines 11-13 of Resolve 4 as described in the simulation. The only difference between Game 5 and Game 4 is in the case that the TTP outputs true by running the sub protocol "Prove Key is not Correct" even though the key is correct. We can easily prove that this case happens

with negligible probability via the security of this sub protocol. Therefore, this game is indistinguishable from the previous game.
Note that this reduction shows that a malicious party cannot get both the file and coin of an honest party at the same time.

Game 6 : It is the same as the previous game except that for all encryptions with pk, we encrypt random keys $\tilde{K}$, random decryption shares, and fake endorsements with pk_T to obtain $E^{\tilde{K}_{\mathsf{pk}_T}}, \tilde{\mathsf{VS}}$, and $\tilde{\mathsf{VS}}^c$. They are indistinguishable because of the IND-CCA-security of their encryption scheme (a very similar reduction as in Game 4 of the proof of Claim 1 can be employed here) and $\tilde{\mathsf{VS}}^c, \tilde{\mathsf{VS}}, E^{\tilde{K}_{\mathsf{pk}_T}}$ of honest parties are never decrypted in the resolution protocols as defined in Game 5 (meaning that in the IND-CCA reductions, the challenge ciphertext never needs to be decrypted, just as in Game 4 of the proof of Claim 1).

Game 7 : It is the same as the previous game except that we use the same random keys $\{\tilde{K}_i\}_{P_i \in \mathcal{P}_h}$ for the encryption with pk and send fake decryption shares using ThDeny as described in the simulation. The reduction can be shown as in Game 5 in the proof of Claim 1.

Game 8 : It is the same as the previous game except we encrypt random items $\{\tilde{f}_j\}_{P_i \in \mathcal{P}_h}$ instead of $\{f_j\}_{P_i \in \mathcal{P}_h}$ with keys $\{K_j\}_{P_i \in \mathcal{P}_h}$ and use $\{K'_j\}_{P_i \in \mathcal{P}_h}$ as K_j such that the decryption of $\{E_j^{\tilde{f}_K}\}_{P_i \in \mathcal{P}_h}$ gives $\{f_j\}_{P_i \in \mathcal{P}_h}$. Intuitively, this game is indistinguishable because of the secure sender deniable encryption. The reduction from Game 7 to Game 8 is as follows: We define hybrid game $H_{7,i}$ where first i parties behave as in Game 7 and the rest of the simulated parties behave as in Game 8. For the sake of clarity of the hybrid argument, assume without loss of generality that $\mathcal{P}_h = \{P_i\}_{1 \le i \le m}$. $H_{7,0}$ is equivalent to Game 8 and $H_{7,m}$ is equivalent to Game 7. We use the hybrid argument to show the indistinguishability of $H_{7,0}$ and $H_{7,m}$. If the adversary manages to distinguish $H_{7,0}$ and $H_{7,m}$ with non-negligible advantage, it must distinguish $H_{7,i}$ and $H_{7,i+1}$ for some i. If so, we can construct an adversary $\mathcal{B}$ which breaks the sender deniability of the deniable encryption scheme.
First, $\mathcal{B}$ guesses i randomly in range $[0, m-1]$, and encrypts items $\{f_j\}_{1 \le j \le i}$ and random items $\{\tilde{f}_j\}_{i+2 \le j \le m}$. Then, $\mathcal{B}$ sends f_{i+1} and a random item $\tilde{f}_{i+1}$ to the faking algorithm challenger. The challenger picks $b \in \{0,1\}$. If $b = 0$, it generates $c = E^{f_K} = \mathsf{SymEnc}(K, f_{i+1}, r_S, r_R)$ and sends E^{f_K}, K. Otherwise, it generates $c = E^{\tilde{f}_K} = \mathsf{SymEnc}(K, \tilde{f}_{i+1})$ and sends $E^{f_K}, K = \mathsf{SymFake}(f_{i+1}, \tilde{f}_{i+1}, r_S, r_R, E^{\tilde{f}_K}, K)$. $\mathcal{B}$ uses c as the encryption of f_{i+1} and K as K_{i+1}. Then, it continues as in Game 7 for all parties. Observe that if the challenger picks $b = 0$, $\mathcal{B}$ sends the real item encryption of P_{i+1} corresponding to $H_{7,i+1}$. Otherwise, it sends a random encryption corresponding to $H_{7,i}$. In any case, $\mathcal{B}$ uses given key K to generate $\mathsf{ThDeny}(E_{i+1}^{K_{\mathsf{pk}}}, \{\tilde{d}_{i+1}^j\}_{P_j \in \mathcal{P}_c}, \mathcal{I}, K, \mathsf{pk})$. In the end, if the guess of i was correct and the adversary distinguishes between Game 7 and Game 8 with $\mathsf{adv}(\ell)$ advantage, then $\mathcal{B}$ can guess b with the same advantage. The guess of i is correct with at least $1/m$ probability, and hence the sender deniability of the deniable encryption scheme is broken with at least $\mathsf{adv}(\ell)/m$ advantage. Since $\mathsf{adv}(\ell)/m$ must be negligible if we use a secure sender deniable encryption scheme, $\mathsf{adv}(\ell)$ must be negligible as well, meaning that this behavior of our simulator remains indistinguishable to the adversary.

Game 8 is the same as our simulation of CMFE. Since Game 8 and Game 1 are indistinguishable, our simulation is indistinguishable as well. The output indistinguishability can be shown

as in Claim 2. At the end of the simulation, the outputs of the parties in the ideal world are identically distributed to the outputs of the parties in the real protocol. This completes the proof of Theorem 4.

□

Privacy against the TTP: Similar to MFE, the privacy against the TTP is preserved if no Resolve 4 is executed. Without any Resolve 4, the TTP just learns some decryption shares which are not related to the exchanged items. If Resolve 4 is executed, then the TTP learns some parts of the file but not whole file (due to the Belenkiy et al. subprotocol). Then, the privacy is potentially preserved partially (depends on the nature of the items).

Multiple exchanges with single Setup: Similar to MFE, as long as the coins are not used, multiple sets of items can be exchanged with a single setup. Even when coins are used, it is indeed enough for those parties to renew their coins rather than executing a full setup. Thus, step ① need not be renewed but step ② may need to be (selectively) renewed.

Malicious TTP: A malicious TTP may obviously break fairness. In this context, this means that the TTP can provide both the item and the coin of an honest party to a corrupted party.

6 All Topologies for MFE and CMFE

In this section, we adapt our MFE and CMFE protocols to any 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 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. He needs to receive $d_1^1, d_1^3.d_1^4$ from P_1, P_3, P_4, respectively (see Figure 4), to be able to decrypt $\mathsf{VE}_1 = ThEnc(\mathsf{pk}, f_1)$. 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 exchange any item with each other and P_1 wants two items (f_4, f_5) from P_4 in return for f_1.

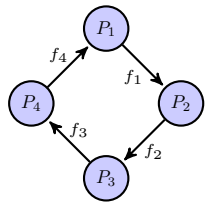

Figure 3: Graph representation of the ring topology.

		Sender			
		P_1	P_2	P_3	P_4
Receiver	P_1		d_4^2	d_4^3	d_4^4
	P_2	d_1^1		d_1^3	d_1^4
	P_3	d_2^1	d_2^2		d_2^4
	P_4	d_3^1	d_3^2	d_3^3	

Figure 4: Decryption shares in ring topology.

In Figure 4 and 6, we show the decryption shares used in step ③ and step ④ in MFE (step ④ and ⑤ in CMFE) by the sender to send them to the corresponding receiver. They clearly show that all parties communicate with each other independently from the topology as a result of all-or-none fairness requirement. The only change based on the topology is the number of decryption shares that a party receives. This number is equal to the number of items that a party expects from the others. For instance, P_1 in Figure 5 wants four items in total from other parties, so it expects four decryption shares from each of them. Thus, the

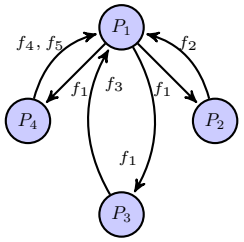

Figure 5: Graph representation of a random topology.

		Sender			
		P_1	P_2	P_3	P_4
Receiver	P_1		d_2^2, d_3^2, d_4^2, d_5^2	d_2^3, d_3^3, d_4^3, d_5^3	d_2^4, d_3^4, d_4^4, d_5^4
	P_2	d_1^1		d_1^3	d_1^4
	P_3	d_1^1	d_1^2		d_1^4
	P_4	d_1^1	d_1^2	d_1^3	

Figure 6: Decryption shares in random topology on the left.

decryption shares are independent of *who sends the item*, but depends only on *who needs to receive that item*. Therefore, the edges in the graph topology represent sending the encrypted items only.

Generic Protocol: The generic algorithm for a sender party P_i to decide which decryption shares are necessary for the receiver party P_j given topology is as follows:

```
FindDS(topology)
  for all P_j = {P_1, ..., P_{i-1}, P_{i+1}, ..., P_n}
        retrieve (P_j, ε_j) ∈ topology
        create DS_j = ∅ // the list of decryption shares that P_j expects from P_i
        for all (P_k; F_k) ∈ ε_j
            for all f_t ∈ F_k
                add d_t^i to DS_j
```

Each party computes FindDS in step ③ of MFE (step ④ of CMFE) to compute the decryption share(s) that they are supposed to encrypt. In step ④ of MFE (step ⑤ of CMFE), they send these decryption shares. The TTP is also aware of the topology in the resolutions and knows who needs which decryption shares from whom.

We remind that we described MFE and CMFE using Figures 1 and 2 in complete topology. The encryptions $\mathsf{VS}_i, \mathsf{VE}_i$ for MFE and CMFE, and $E_i^{K_h}, E_i^{K_{\mathsf{pk}}}, E_i^{f_K}$ for CMFE computed by P_i are all the same for all parties. However, in other topologies they would be all different for different parties because each party may want different items.

Fair SMPC: Finally, we show the fair SMPC topology in Figure 7 (and the related decryption share table in Figure 8). At first, it looks like a strange topology since each party wants his own item. But remember that the encrypted items have already been delivered by the underlying SMPC protocol, and hence they each want the decryption shares related to their encrypted item only. Each item f_i here is indeed ϕ_i, which is the output of the functionality ϕ.

Security: The security proof for any topology can be done very similar to the proofs of Theorem 1 and 4. Below, we highlight the parts that need to depend on the topology in the simulator (assuming the messages sent as in the real protocol already depend on the topology, based on the FindDS algorithm).

In the MFE simulator, the Setup and EIE phases remain the same. In the DSE phase, case (1) would mean that the simulator received VS values required by all honest parties (consider VS values together containing a union of decryption shares expected by the honest

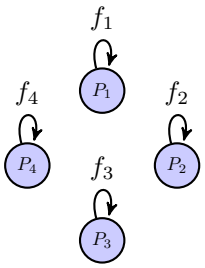

Figure 7: Graph representation of fair SMPC topology.

		Sender			
		P_1	P_2	P_3	P_4
Receiver	P_1		d_1^2	d_1^3	d_1^4
	P_2	d_2^1		d_2^3	d_2^4
	P_3	d_3^1	d_3^2		d_3^4
	P_4	d_4^1	d_4^2	d_4^3	

Figure 8: Decryption shares in fair SMPC topology.

parties for them to be able to decrypt all their expected items). If so, again the simulator can perform Resolve 2 and ensure all parties would learn their items. Therefore, output delivery is guaranteed, and hence the simulator can contact U. The simulated decryption shares would then depend on the topology, and the corresponding shares in the topology matrix (e.g., Figure 4 and 6) would be simulated. Case (2) is similar, where the simulator acts as the TTP based on the topology, and depending on the adversary behavior, the resolutions may succeed and the simulation would then proceed to the fake decryption share generation as above, or the resolutions may be unresolved, and hence the simulator sends ABORT to U. The same reasoning also applies to the CMFE simulator.

7 Performance Analysis

Table 3: Required time for the primitive computations and the message sizes from Cashlib [20] and Charm [1] libraries. Verifiable encryption includes the normal encryption and the associated proof.

Computation	**Time**	**Size**
El Gamal [for item encryption with h] (Enc/Dec)	2 ms/ 1 ms	0.25 KB
Cramer-Shoup [for escrows] (Enc/Dec)	4 ms/ 3 ms	0.44 KB
RSA [for signatures in CMFE] (Sign/Verify)	0.147 ms/0.009 ms	0.13 KB
VE(1 item) with El Gamal (Prove /Verify)	2.24 ms/ 0.38 ms	1.06 KB
$\mathsf{VS}(t$ items) with Cramer-Shoup (Prove /Verify)	$4.13t - 3.85$ ms/ $0.18t + 0.02$ ms	$0.8t - 0.03$ KB
VS^c with Camenisch-Shoup (Prove /Verify)	198.4/60.5 ms	24.40 KB
Endorsed E-cash [for coin in CMFE] (Generate/Verify)	65.80/50.10 ms	22 KB

While we did not implement the full protocol, we provide performance numbers based on the performance of the underlying primitives taken from Cashlib [20] and Charm [1] libraries. The yardstick of the values in Table 3 were obtained from a DELL Latitude E7240 laptop with a 2.10GHz i7 processor and 8GB of RAM running Ubuntu 16.04 LTS. The security parameter for the public-key operations is 1024 bits. KB represents kilobytes, MB is megabytes, and ms

is milliseconds.

For the underlying primitives, we employ El Gamal [37] based verifiable encryptions, Cramer-Shoup [33] based verifiable escrows, and Camenish-Shoup [25] based verifiable endorsement escrows, Endorsed E-cash [24] for coins, and RSA [79] signatures. Zero knowledge proofs of knowledge are taken from Cashlib [20] ZKPDL [71] implementation of efficient Sigma proofs [34] in the random oracle model.

7.1 MFE Analysis

Communication Overhead: Table 4 shows that the total bit overhead of MFE per participant is just around 111 KB in the complete topology and 102 KB in the ring topology with 10 participants. Thus, the overhead is slightly higher in the complete topology, as expected.

Computation Overhead: We analyze time complexity of MFE in Table 5. The total computation per party is just around 88 ms in the complete topology and 60 ms in the ring topology for 10 parties.

Table 4: The size of messages in MFE for each participant. # is the number of parties.

#	Complete Topology **(KB)**	Ring Topology **(KB)**
2	3.48	3.48
4	17.24	15.07
6	39.80	35.47
8	71.11	64.62
10	111.16	102.51

Table 5: Required time for each party for the computations in MFE. # is the number of parties.

#	Complete Topology **(ms)**	Ring Topology **(ms)**
2	9.00	9.00
4	23.73	21.97
6	41.75	34.90
8	63.76	47.84
10	88.70	60.77

7.1.1 Fair SMPC Analysis

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 [14, 84, 32], then there are only $O(n)$ additional exponentiations required for n parties, which does not extend circuit size a lot, since it is independent of the actual computation circuit. If boolean circuits are used, the size of the circuit increases more than an arithmetic circuit would have, but it is still tolerable and independent of the computation circuit, especially considering in comparison to the related work: While gradual release based solutions [42] require many rounds of exchanges, and bitcoin based approaches [18, 2] require broadcast in the bitcoin network and use a payment-based fairness mechanism, our only overhead is (*asymptotically optimally*) a constant number of rounds, and $O(n^2)$ messages, while providing *maximal fairness* (even against $n-1$ malicious and colluding parties). Also remember that even when the TTP becomes malicious, only fairness is lost, and security and privacy still holds

7.2 CMFE Analysis

Optimizations: Observe that, a party's coin is decrypted by the TTP only if during Resolve 4, some other party manages to prove that this party sent a wrong key/file. Therefore, if

a party is honest, her coin will *never* be obtained by another party (via the security of the Belenkiy et al. [16] subprotocol). This means that an honest party needs to prepare only one coin and VS^c only, and can send the same coin to all other parties.[10]

Moreover, if participants may want to change the topology during CMFE repetitions after a single setup, then they need to initially perform a complete topology type of one time setup where every pair of parties exchange unendorsed coins and verifiably encrypted endorsements. But, on the other hand, if a topology will be fixed for all the subsequent exchanges, the setup can be more efficient. Consider the ring topology: Since the only other party who can obtain a party's coin in Resolve 4 is the succeeding party in the ring, it is enough that each party sends her unendorsed coin and verifiably encrypted endorsement only to the succeeding party in the ring. This makes the per-participant cost of the Setup Phase for the ring topology independent of the number of participants.

Below, we provide performance numbers with these optimizations.

Communication Overhead: Table 6 shows that the total bit overhead of CMFE is about 527 KB in the complete topology and 70 KB in the ring topology for 10 participants. Since the Setup Phase of CMFE is executed once among the same set of participants, if we do not consider this phase (for repeated exchanges), the bit complexity for 10 parties is around 105 KB for the complete topology and 20 KB for the ring topology. Remark that, the bit overhead of CMFE *without Setup Phase* is less than the bit overhead of MFE, providing advantage after multiple repetitions.

Computation Overhead: We analyze the time complexity of CMFE in Table 7. The total computation is around 1.3 seconds in the complete topology and 0.4 seconds in the ring topology for 10 participants. The setup is costly due to the electronic coin requirement (and hence would be much faster with non-anonymous electronic checks). If we do not consider the Setup Phase, then the computation time decreases to 87 ms in the complete topology and 59 ms in the ring topology.

Of course, the main benefit of CMFE comes from not only repeated exchanges, but also the fact that CMFE is the *first* multi-party fair exchange protocol that enables efficient fair exchange of items that cannot be efficiently verifiably encrypted.

Table 6: The size of messages in CMFE for each participant. The values in parenthesis show the size of the messages in CMFE without the Setup Phase. The unit of values is KB. # is the number of parties.

#	Complete Topology **(KB)**	Ring Topology **(KB)**
2	49.8 (2.94)	49.8 (2.94)
4	155.9 (15.38)	54.87 (7.20)
6	270.9 (36.62)	59.93 (11.46)
8	394.6 (66.61)	64.98 (15.72)
10	527.1 (105.34)	70.04 (19.98)

Table 7: Required time for each party for the computations in CMFE. The values in parenthesis show the time of computation in CMFE without the Setup Phase. The unit of the values is millisecond. # is the number of parties.

#	Complete Topology **(ms)**	Ring Topology **(ms)**
2	387.5 (12.08)	387.5 (12.08)
4	623.1 (25.70)	400.1 (23.95)
6	861.8 (42.63)	412.4 (35.82)
8	1104.6 (63.52)	424.9 (47.68)
10	1350.3 (87.34)	437.3 (59.55)

Relative Overhead: We analyzed the overhead percentage of CMFE compared to the

[10]Of course, a malicious party can also do so, but remember that offline ecash schemes have penalties for double spenders, which is outside our scope.

simplest *unfair* exchange (just sending files to other parties) for the complete topology in Table 8. For slow connections (1Mbps), our overhead is always small, and is less than 0.2% for files larger than 10 MB, for any number of participants. Indeed, as the number of participants increase, our relative overhead decreases, since file transfer takes the bulk of the process. For fast connections (10Mbps), our overhead is around 1.8% for 10 MB files, but becomes less than 0.2% for files larger than 100 MB. In peer-to-peer settings where movie files or linux distros of sizes in gigabytes are exchanged, our overhead becomes neglectable, for any number of parties.

Table 8: The overhead percentage of CMFE in terms of time: CMFE computations + network time spent for transferring CMFE extra messages versus the time to upload files unfairly using a 1 Mbps / 10 Mbps network between each pair. # is the number of participants. MB is the file size in megabytes. Topology is complete.

# Size	1 MB	10 MB	100 MB	1000 MB
2	5.07% / 32.86%	0.531% / 4.666%	0.0534% / 0.487%	0.0053% / 0.0489%
4	3.02% / 20.93%	0.310% / 2.579%	0.0311% / 0.264%	0.0031% / 0.0264%
6	2.62% / 18.09%	0.268% / 2.161%	0.0269% / 0.220%	0.0026% / 0.0220%
8	2.47% / 16.86%	0.252% / 1.988%	0.0253% / 0.202%	0.0025% / 0.0202%
10	2.40% / 16.20%	0.245% / 1.897%	0.0246% / 0.193%	0.0024% / 0.0193%

8 Conclusion

In this paper, we defined three protocols MFE and CMFE, and fair SMPC. The first two are used as multi-party fair exchange protocols.MFE is useful for the exchanges of items that can be efficiently verifiably encrypted, such as signatures and group elements. For the items that are not efficiently verifiably encryptable (movies, images, other files, etc.), CMFE is the solution for multi-party fair exchange. Indeed, CMFE is the first multi-party fair exchange protocol that enables efficient fair exchange of general files. Our fair SMPC solution is useful for secure multi-party computations which need fairness as well as security.

In all our protocols, we achieve fairness employing an optimistic TTP. We pay attention to use the TTP efficiently and satisfy privacy against the TTP. In particular, we proved, via ideal-real simulation, that even a malicious TTP under adversarial control cannot break security of the SMPC solution, even when colluding with malicious participants, but can only break fairness. Thus, even when the TTP misbehaves, our fair SMPC security reverts back to security with abort. To the best of our knowledge, our solution is the first such fair SMPC protocol.

For all our protocols, we proved their fairness and security by showing ideal-real world indistinguishability. As far as we know, this is the first time that this proof technique is used for multi-party fair exchange protocols. Our proofs show that either all participants receive their desired items, or no participant receives any useful information about any item. We also argued that this should be *the fairness definition* for multi-party exchanges.

Moreover, our exchange protocols are flexible since they can be used for any exchange topology. Our fair exchange protocols have setup phases, and once done, the participants may repeat the remaining steps for further item exchanges, even under differing topologies.

Future Work: For the complete topology, we match the known trivial lower bound in [70], achieving asymptotic optimality. For other topologies, there are more efficient protocols, but they do not satisfy the all-or-none type of fairness. We conjecture that the lower bound of $O(n^2)$ messages and $O(1)$ rounds apply to every topology, when all-or-none fairness is required. Proving this formally is left as future work.

Acknowledgements

The authors acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project numbers 111E019 and 115E766, as well as European Union COST Action IC1306, and the valuable comments of Prof. Serge Vaudenay and Journal of Cryptology referees.

References

[1] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Symposium on Security and Privacy*, pages 443–458. IEEE, 2014.

[3] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC*, pages 291–316. Springer, 2014.

[4] G. Asharov, A. Beimel, N. Makriyannis, and E. Omri. Complete characterization of fairness in secure two-party computation of boolean functions. In *TCC*, pages 199–228, 2015.

[5] G. Asharov, R. Canetti, and C. Hazay. Towards a game theoretic view of secure computation. In *EUROCRYPT*, volume 6632, pages 426–445. Springer, 2011.

[6] G. Asharov, Y. Lindell, and H. Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *ASIACRYPT*, pages 201–220. Springer, 2013.

[7] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for multi-party fair exchange. Technical report, IBM Research RZ2892, 1996.

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

[9] G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *ACM CCS*, pages 138–146. ACM, 1999.

[10] G. Ateniese and C. Nita-Rotaru. Stateless-recipient certified e-mail system based on verifiable encryption. In *CT-RSA*, volume 2, pages 182–199. Springer, 2002.

[11] G. Avoine and S. Vaudenay. Optimistic fair exchange based on publicly verifiable secret sharing. In *ACISP*, volume 3108, pages 74–85. Springer, 2004.

[12] F. Bao, R. Deng, K. Q. Nguyen, and V. Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *DEXA*, pages 858–862. IEEE, 1999.

[13] 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*, pages 77–85, 1998.

[14] C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In *SCN*, pages 175–196. Springer, 2014.

[15] B. Baum-Waidner and M. Waidner. Round-optimal and abuse-free optimistic multi-party contract signing. In *ICALP*, pages 524–535. Springer, 2000.

[16] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making P2P accountable without losing privacy. In *WPES*, pages 31–40. ACM, 2007.

[17] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10. ACM, 1988.

[18] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, pages 421–439. Springer, 2014.

[19] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112. ACM, 1988.

[20] Brownie cashlib cryptographic library. http://github.com/brownie/cashlib.

[21] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO*, pages 93–111. Springer, 2000.

[22] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *ASIACRYPT*, pages 331–345. Springer, 2000.

[23] J. Camenisch and I. B. Damgård. Verifiable encryption and applications to group signatures and signature sharing. *BRICS Report Series*, 5(32), 1998.

[24] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *Security and Privacy*, pages 101–115. IEEE, 2007.

[25] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144. Springer, 2003.

[26] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[27] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *CRYPTO*, pages 90–104. Springer, 1997.

[28] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203. Springer, 1983.

[29] D. Chaum, B. den Boer, E. van Heyst, S. Mjølsnes, and A. Steenbeek. Efficient offline electronic checks. In *EUROCRYPT*, pages 294–301. Springer, 1989.

[30] B. Cohen. Incentives build robustness in bittorrent. In *WEPS*, volume 6, pages 68–72, 2003.

[31] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

[32] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–300. Springer, 2001.

[33] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *CRYPTO*, pages 45–64. Springer, 2002.

[34] I. Damgård. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[35] Y. Dodis, P. Lee, and D. Yum. Optimistic fair exchange in a multi-user setting. In *PKC*, pages 118–133. Springer, 2007.

[36] G. Draper-Gil, J.-L. Ferrer-Gomila, M. F. Hinarejos, and J. Zhou. On the efficiency of multi-party contract signing protocols. In *ISC*, pages 227–243. Springer, 2015.

[37] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[38] S. Even and Y. Yacobi. Relations among public key signature systems. Technical report, Technical Report 175, Technion, Haifa, Israel, 1980.

[39] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. Springer, 1986.

[40] M. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *FC*, pages 90–102. Springer, 1998.

[41] J. Garay, J. Katz, B. Tackmann, and V. Zikas. How fair is your protocol?: A utility-based approach to protocol optimality. In *PODC*, pages 281–290. ACM, 2015.

[42] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. *Journal of Cryptology*, 24(4):615–658, October 2011.

[43] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO*, volume 99, pages 449–466. Springer, 1999.

[44] J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In *DISC*, volume 99, pages 151–165. Springer, 1999.

[45] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[46] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229. ACM, 1987.

[47] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[48] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, April 1988.

[49] N. González-Deleito and O. Markowitch. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *ICISC*, pages 258–267. Springer, 2001.

[50] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *Journal of the ACM (JACM)*, 58, 2011.

[51] S. D. Gordon, Y. Ishai, T. Moran, R. Ostrovsky, and A. Sahai. On complete primitives for fairness. In *TCC*, pages 91–108. Springer, 2010.

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

[53] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. In *EUROCRYPT*, volume 6110, pages 157–176. Springer, 2010.

[54] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. *Journal of cryptology*, 25(1):14–40, 2012.

[55] A. Groce and J. Katz. Fair computation with rational players. In *EUROCRYPT*, pages 81–98. Springer, 2012.

[56] B. Hemenway, R. Ostrovsky, and A. Rosen. Non-committing encryption from ϕ-hiding. In *TCC*, pages 591–608. Springer, 2015.

[57] A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT*, pages 705–734. Springer, 2016.

[58] H. Kılınç and A. Küpçü. Efficiently making secure two-party computation fair. In *FC*, pages 188–207. Springer, 2016.

[59] H. Kılınç and A. Küpçü. Optimally efficient multi-party fair exchange and fair secure multi-party computation. In *CT-RSA*, pages 330–349, 2015.

[60] M. S. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *CT-RSA*, volume 8, pages 88–105. Springer, 2008.

[61] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, pages 50–63, 2012.

[62] A. Küpçü. Distributing trusted third parties. *ACM SIGACT News Distributed Computing Column*, 44:92–112, 2013.

[63] A. Küpçü and A. Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, pages 488–507. Springer, 2010.

[64] A. Küpçü and P. Mohassel. Fast optimistically fair cut-and-choose 2PC. In *FC*, pages 208–228. Springer, 2016.

[65] A. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, pages 121–137. Springer, 2008.

[66] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78. Springer, 2007.

[67] Y. Liu and H. Hu. An improved protocol for optimistic multi-party fair exchange. In *EMEIT*, volume 9, pages 4864–4867. IEEE, 2011.

[68] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In *ICISC*, pages 109–122. Springer, 2000.

[69] S. Mauw and S. Radomirović. Generalizing multi-party contract signing. In *POST*, pages 156–175. Springer, 2015.

[70] S. Mauw, S. Radomirovic, and M. T. Dashti. Minimal message complexity of asynchronous multi-party contract signing. In *CSF*, pages 13–25. IEEE, 2009.

[71] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, 2010.

[72] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378. Springer, 1987.

[73] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, pages 12–19. ACM, 2003.

[74] A. Mukhamedov and M. D. Ryan. Fair multi-party contract signing using private contract

signatures. *Elsevier Information and Computation*, 206:272–290, 2008.

[75] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126. Springer, 2002.

[76] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Darmstadt University of Technology, TUD-BS-1999-02, 1999.

[77] T. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, pages 522–526. Springer, 1991.

[78] B. Pinkas. Fair secure two-party computation. In *EUROCRYPT*, volume 2656, pages 87–105. Springer, 2003.

[79] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *ACM Communications*, 21:120–126, 1978.

[80] O. Ruan, J. Chen, J. Zhou, Y. Cui, and M. Zhang. An efficient fair UC-secure protocol for two-party computation. *Security and Communication Networks*, 7(8):1253–1263, 2014.

[81] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484. ACM, 2014.

[82] M. Šebek. Fully bideniable public-key encryption. *IACR Cryptology ePrint Archive*, 2013:684, 2013.

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

[84] M. Zamani, M. Movahedi, and J. Saia. Millions of millionaires: Multiparty computation in large networks. *IACR Cryptology ePrint Archive*, 2014:149, 2014.

A Equivalence Among Fair and Secure Definitions

In Section 3, we extended the fair and secure computation definition by Cachin and Camenisch [21] to the multi-party case in the spirit of the standard two-party computation definition. When the TTP is honest, we prove that this definition is equivalent to an intuitive and simpler fair and secure computation definition whose ideal world is given below:

Ideal World: The ideal-world execution consists of honest party(s) $\mathcal{P}_h$, an adversary $\mathcal{A}$ that corrupts the parties in set $\mathcal{P}_c$, and the ideal functionality U^{ϕ}_{alt} (*not the TTP*). The ideal protocol is as follows:

U^{ϕ}_{alt} for alternative security and fairness

- U^{ϕ}_{alt} receives inputs $\{w_i\}_{P_i \in \mathcal{P}_c}$ or the message ABORT from $\mathcal{A}$, and $\{w_i\}_{P_i \in \mathcal{P}_h}$ from the honest party(s). If the inputs are invalid or $\mathcal{A}$ sends the message ABORT, then U^{ϕ} sends $\perp$ to all of the parties and halts.
- Otherwise U^{ϕ}_{alt} 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 $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ to $\mathcal{A}$ and each one of $\{\phi_i\}_{P_i \in \mathcal{P}_h}$ to the corresponding honest party.

Realize that in the alternative definition for security and fairness, U^{ϕ}_{alt} does not wait for any adversarial input; instead, it sends the outputs to both adversarial and honest players simultaneously. Intuitively, this does not require any explanation as to why it achieves security

and fairness together. Since the TTP is assumed to be honest, this definition does not allow any TTP input, and does not even represent the TTP in the ideal world. Nevertheless, below, we prove that the U_{fs}^{ϕ} definition in Section 3 is equivalent to the U_{alt}^{ϕ} definition here, when the TTP is honest.

$\mathbf{U_{alt}^{\phi}}$ implies $\mathbf{U_{fs}^{\phi}}$ (for honest TTP): The adversary controls $P_i \in \mathcal{P}_c$ in the real world. If π is secure and fair according to U_{alt}^{ϕ}, then there exists a simulator S_{alt} which simulates $P_i \in \mathcal{P}_c$ in the ideal world defined via U_{alt}^{ϕ}, and simulates the honest parties $P_i \in \mathcal{P}_h$ in the execution of π in the real world. We want to show that a simulator S_{fs} exists according to U_{fs}^{ϕ} in Section 3. S_{fs} runs S_{alt} as a subroutine and acts as U_{alt}^{ϕ} for S_{alt}. Thus, S_{fs} receives inputs $\{w_i\}_{P_i \in \mathcal{P}_c}$ of the corrupted parties or the message ABORT from S_{alt}. S_{fs} then forwards this to U_{fs}^{ϕ}. At this point, unless malformed inputs or ABORT was sent, U_{fs}^{ϕ} sends back outputs $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ of the corrupted parties, and also asks the honest TTP in the ideal world, who simply returns $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$, enabling honest parties to receive their outputs in the ideal world. S_{fs} simply forwards $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ received from U_{fs}^{ϕ} to S_{alt}, perfectly simulating U_{alt}^{ϕ}. At the end, because S_{alt} exists, the ideal and real distributions will be indistinguishable.

$\mathbf{U_{fs}^{\phi}}$ implies $\mathbf{U_{alt}^{\phi}}$ (for honest TTP): The adversary controls $P_i \in \mathcal{P}_c$ in the real world. If π is secure according to U_{fs}^{ϕ}, then there exists a simulator S_{fs} which simulates $P_i \in \mathcal{P}_c$ in the ideal world defined via U_{fs}^{ϕ}, and simulates the honest parties $P_i \in \mathcal{P}_h$ in the execution of π in the real world. We want to show that a simulator S_{alt} exists according to U_{alt}^{ϕ} above. S_{alt} runs S_{fs} as a subroutine and acts as U_{fs}^{ϕ} for S_{fs}. Thus, S_{alt} receives inputs $\{w_i\}_{P_i \in \mathcal{P}_c}$ of the corrupted parties or the message ABORT from S_{fs}. S_{alt} then forwards this to U_{alt}^{ϕ}, obtaining back outputs $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ of the corrupted parties, unless malformed inputs or ABORT was sent. Then, S_{alt} internally simulates the honest TTP by setting $\{b_i = \text{CONTINUE}\}_{P_i \in \mathcal{P}_h}$. S_{alt} also simply forwards $\{\phi_i\}_{P_i \in \mathcal{P}_c}$ received from U_{alt}^{ϕ} to S_{fs}. Thus, S_{alt} perfectly simulates U_{fs}^{ϕ} for S_{fs}. At the end, because S_{fs} exists, the ideal and real distributions will be indistinguishable.

B Sender Deniable (n, n)- Threshold El-Gamal Encryption

In this section, we use a group $\mathbb{G}$ with a generator g of a prime order p with respect to a security parameter ℓ. All operations are group operations. We use multiplicative group notation.

Deniable (n, n)- threshold El-Gamal encryption consists of the probabilistic polynomial time (PPT) protocols below:

- $\mathsf{ThGen}(\ell, n, n)$: It is an interactive protocol. Each party P_i picks $x_i \in \mathbb{Z}_p$ as a secret share. Then, he sends $h_i = g^{x_i}$ as a verification key by proving the relation $\mathcal{R}_{dl} = \{(x_i, (\mathbb{G}, p, g)) | h_i = g^{x_i}\}$ denoting knowledge of the discrete logarithm. At the end, all parties agree on the public key $h = \prod g^{x_i}$ and the verification key $v = \{g^{x_1}, g^{x_2}, ..., g^{x_n}\}$.
- $\mathsf{ThEnc}(h, m) \rightarrow (a, b)$ where $(a, b) = (g^r, mh^r)$ for random $r \in \mathbb{Z}_p$.
- $\mathsf{ThDShare}(x_i, h, E) \rightarrow d_i$ where $d_i = a^{x_i}$ and a is the first part of E.
- $\mathsf{ThDSProve}(x_i, E, h, d_i) \rightarrow p_i$ where $p_i = (c, D)$ such that $c = H(W_1, W_2)$, $W_1 = g^r$, $W_2 = a^r$ and $D = r - x_i c \mod p$. Remark that p_i is the proof of the relation $\mathcal{R}_{dleq} = \{(x_i, (\mathbb{G}, p, g, h_i, a, d_i | x_i = \log_g h_i = \log_a d_i\}$.
- $\mathsf{ThDSVerify}(v, h, E, d_i, p_i)$ outputs valid if $c = H(g^D h_i^c, a^D d_i^c)$. Otherwise it outputs invalid.

- $\mathsf{ThDec}(\{d_i\}_{1\leq i\leq n}, E) \rightarrow m$ where

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

- $\mathsf{ThDeny}(E, DS', \mathcal{I}, m_1, h) \rightarrow DS''$. Assume that $DS' \subset DS$ where DS is the decryption shares of $E = ThEnc(h, m_0)$ with $|DS'| < n$ and $\mathcal{I}$ is the index set of missing decryption shares. ThDeny first picks randomly an index $i \in \mathcal{I}$ and then randomly picks d_j for all $j \in \mathcal{I} \setminus i$. Then, finds a decryption share d_i as follows:

$$d_i = \frac{b}{m_1 \prod_{j\in\mathcal{I}\setminus i} d_j}$$

  and then outputs $DS'' = \{d_t\}_{t\in\mathcal{I}}$.

We can efficiently [23, 21, 77] construct a verifiable encryption for ElGamal threshold encryption scheme for a relation $\mathcal{R}_{item}$. Below, we use the descriptive notation for the verifiable encryption as $\mathsf{V}(E, \mathsf{pk}; l)\{(w, \delta) \in \mathcal{R}\}$. It denotes the verifiable encryption for the ciphertext E whereby w –whose relation $\mathcal{R}$ with the public value δ can be verified– is encrypted under the public-key pk, and labeled by l. In particular, $\mathsf{ProveEnc}$ and $\mathsf{VerifyEnc}$ can be realized via ElGamal as follows:

P_i sends a verifiable encryption of his item f_i as

$$\mathsf{VE}_i = \mathsf{V}((g^{r_i}, f_i h^{r_i}), h; \emptyset)\{(f_i, r_i), (c_i, a_i, b_i, h, g)) \in \mathcal{R}_{item}\}$$

where r_i is randomly selected from $\mathbb{Z}_p$. For the notation simplicity, we denote $(a_i, b_i) = (g^{r_i}, f_i h^{r_i})$. (a_i, b_i) is the ElGamal encryption of the item f_i with the public key h using randomness r_i. It can be verified that the encrypted item f_i and the public value c_i has the relation $\mathcal{R}_{item}$.

It has an efficient realization using Sigma proofs [34] where they are converted to non-interactive zero-knowledge proofs of knowledge [19] using the Fiat-Shamir technique [39] in the random oracle model. Therefore, under the Decisional Diffie-Hellman assumption (and the discrete logarithm assumption for the proofs) we have that ElGamal is a verifiable deniable (n,n)-threshold encryption scheme. It is IND-CPA secure against any PPT adversary who controls less than threshold-many participants (i.e., the adversary knows the public key and less than threshold-many, up to n-1 in our case, secret keys).