Securing Multiparty Protocols against the Exposure of Data to Honest Parties

Peeter Laud[1] and Alisa Pankova[1,2,3]

[1] Cybernetica AS
[2] Software Technologies and Applications Competence Centre (STACC)
[3] University of Tartu
{peeter.laud|alisa.pankova}@cyber.ee

Abstract. We consider a new adversarial goal in multiparty protocols, where the adversary may corrupt some parties. The goal is to manipulate the view of some honest party in a way, that this honest party learns the private data of some other honest party. The adversary itself might not learn this data at all. This goal, and such attacks are significant because they create a liability to the first honest party to clean its systems from second honest party's data; a task that may be highly non-trivial.
Protecting against this goal essentially means achieving security against several non-cooperating adversaries, where all but one adversary are passive and corrupt only a single party. We formalize the adversarial goal by proposing an alternative notion of universal composability. We show how existing, conventionally secure multiparty protocols can be transformed to make them secure against the novel adversarial goal.

1 Introduction

Data is a toxic asset [1]. If it has been collected, then it has to be protected from leaking. Hence one should not collect data that one has no or a little use of. To make sure that one is not collecting such data, one should try to never learn that data in the first place. In existing models of multiparty protocols, the security goals of a party are not violated if it learns too much: according to the model, an honest party may simply ignore the messages not meant to it or the data it has learned because of the misbehaviour of some other party. In practice, such forgetting of data may be a complex and expensive process, involving thorough scrubbing or destruction of storage media.

An honest party's attempt to not learn the data that it is not supposed to learn, brings about an adversarial goal that has not been considered so far. The adversary may deliberately try to cause an honest party to learn some other honest party's private data, i.e. make the second honest party's data to be derivable from the first honest party's view. The adversary's inability to learn such data itself does not imply the impossibility of such attacks.

When formalizing security against such attacks, we have to be careful to not make this property unachievable. There are certain related attacks that cannot be excluded. For example, we cannot prevent the adversary from publishing secrets or their parts through some side channels that are not related to the protocol. Hence it would be more interesting to study the attacks where the adversary is able to force one party to leak a secret to another party even without seeing this secret himself.

The security of protocols is usually proved in the universal composability (UC) framework [2] which ensures that the protocol is secure not only in stand-alone model, but also when run in several sessions or in parallel with some other protocols. This framework assumes that there is a single monolithic adversary that controls all the corrupted parties. Construction of a simulator for a UC security proof often relies on the assumption that a value may indeed be leaked since the adversary knows it anyway. In practice, it may be still unpleasant to leak a secret value to some honest party even if some other corrupted party has already seen it. If an attacker has broken into a user's mailbox, it still does not imply that the user is now ready to publish his e-mails to everyone since some attacker has seen them anyway. Moreover, if some secret leaks from one honest user to another honest user, this secret may just remain unnoticed by the adversary.

If we care about the views of honest parties, we could treat each honest party as some kind of independent adversary. There exist some alternative definitions of UC that support multiple adversaries, such as CP (Collusion Preserving) computation [3] or LUC (Local UC) [4]. However, these models are too strong, and they are used to prove strong properties that are not necessary for our purposes. If we treat each honest party as an adversary, we get a setting in which all the parties (except the one that we are formally protecting) are corrupted, and we will quite unlikely come up with a reasonable multiparty protocol whose security we will be able to prove in this model.

Our contribution In this work, we define a "weak CP" (WCP), which splits the adversary into mutually exclusive coalitions. Instead of bounding the total number of corrupted parties, we only bound the the sizes of coalitions. Our model does not focus on preventing the malicious parties from sending their secrets directly to other parties, but rather on detecting the flaws in protocols where it may happen that an honest party is obliged to leak its secret to another honest party at some point. More formally, we split the adversary $\mathcal{A}$ into two parts $\{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H\}$ and $\mathcal{A}^L$, each $\mathcal{A}_i^H$ representing a separate adversarial coalition. Only $\mathcal{A}_i^H$ may get messages from the corrupted parties, but the attacks on the protocol are performed by $\mathcal{A}^L$. We are interested in attacks that can be performed by $\mathcal{A}^L$ without taking into account the messages that $\mathcal{A}_i^H$ received from the protocol. We see if $\mathcal{A}^L$ succeeds in leaking information received by $\mathcal{A}_i^H$ to *another* adversary $\mathcal{A}_j^H$ which may represent the view of a coalition of corrupted parties as well as the view of an honest party.

After reviewing related work in Sec. 2 and giving some preliminaries in Sec. 3, we give a formal definition of WCP and prove its composability in Sec. 4. We discuss the applicability of WCP in Sec. 5. We show that, although UC emulation implies WCP emulation in presence of a passive adversary, it is not the case for fail-stop, covert, and active adversaries. In Sec. 6 we present some transformations that make a protocol that is secure in UC model also secure in WCP model.

2 Related Work

The problem of leaking a secret to an honest party is not new. The multiparty computation protocol of [5] is provided with a description of an attack that allows the malicious party to leak a secret value of one honest party to a different honest party. This attack remains unnoticed by the traditional UC framework [2], and it could be detected using some other model that assumes the existence of two distinct adversaries: the malicious one and the semihonest one.

The abstract cryptography framework [6] does take into account multiple adversaries. The more concrete frameworks [7–9] study the collusion-freeness property of protocols whose main goal is to prevent smaller adversarial coalitions from forming larger coalitions using subliminal channels. A *collusion-free* protocol prevents the parties from any communication. A *collusion-preserving* protocol ensures that the parties cannot exchange more information that they could without executing the protocol.

Extending the traditional UC framework [2] to multiple adversaries have been considered in [3, 4]. In CP (Collusion Preserving computation) [3], there is a separate adversary $\mathcal{A}_i$ for each party P_i. The adversaries communicate with the protocol π using a communication resource R which in turn contributes to defining the adversarial behaviour. The idea is that, in the real protocol, the adversaries should be able to exchange only as much information as they could in the ideal protocol. In LUC (Local UC) [4], each party P_i may be corrupted by $n-1$ adversaries $\mathcal{A}_{(i,j)}$ that can deliver messages to the party P_i where the sender identity of the delivered messages must be P_j. This model can be used to express more interesting properties than CP allows.

Unfortunately, these security models are too strong for our problem. In [3, 4], all the adversaries eventually send all their data to the environment. Therefore, even if we know that there are no coalitions of size larger than t, we still cannot use $(t+1)$-threshold sharing (a sharing where at least $t+1$ parties need to collaborate to reconstruct the secret) since the environment may combine the shares of different coalitions.

One way to solve this issue is to assume that the environment is split into pieces with constrained information movement. For example, [10] formalizes *information confinement* property

of a protocol. It splits the environment $\mathcal{Z}$ into *high* and *low* subenvironments $\mathcal{Z}_\mathcal{H}$ and $\mathcal{Z}_\mathcal{L}$ where data is allowed to move from $\mathcal{Z}_\mathcal{L}$ to $\mathcal{Z}_\mathcal{H}$, but not the other way around. The confinement property is formally achieved if $\mathcal{Z}_\mathcal{L}$ cannot guess a bit generated by $\mathcal{Z}_\mathcal{H}$ with non-negligible advantage. However, this property needs to be checked in addition to ordinary UC security, and the $(t+1)$ secret sharing with t adversaries would be insecure anyway. We use a bit simpler solution in our protocol, and instead of putting constraints on the environment, we put constraints onto the adversary. This allows to embed the confinement property into the definition of emulation.

3 Basics of Universal Composability

First of all, we give a brief review of the basic UC model [2]. UC considers systems of Interactive Turing Machines (ITM) connected to each other by input and output communication tapes. On the figures, ITMs are represented by boxes, and the communication tapes by arrows.

A protocol π consists of ITMs M_i (i is a unique identifier in the given protocol session) that mutually realize some functionality $\mathcal{F}$. They may be connected to each other, and may also use some "trusted" resource ITM R to mediate their communication or even compute something for them. A special ITM $\mathcal{A}$ represents the *adversary* that may corrupt some M_i and get access to their internal states. There is a special ITM $\mathcal{Z}$, the *environment*, that chooses the inputs for each M_i and receives their outputs. This $\mathcal{Z}$ may contain the parties P_i sitting behind the machines M_i, or any other protocols running in parallel or sequentially with π, probably even some other sessions of π. $\mathcal{Z}$ also communicates with $\mathcal{A}$ and sees which information it has extracted from the protocol.

In security proofs, one defines a functionality $\mathcal{F}$ represented by a "trusted" ITM and describes what exactly it computes and which data is insensitive enough to be output to the adversary deliberately. On the other hand, there is a protocol π that has exactly the same communication ports with $\mathcal{Z}$ as $\mathcal{F}$ has, but that consists of untrusted machines M_i and optionally some other smaller resource R. Since π is usually more realistic than $\mathcal{F}$, the goal is to show that π is secure enough to be used instead of $\mathcal{F}$, and this can be done by proving that any attack (represented by $\mathcal{A}$) against π can be converted to an attack (represented by some $\mathcal{A}\mathsf{s}$) against $\mathcal{F}$. Formally, one proves that no environment $\mathcal{Z}$ is able to distinguish whether π (with $\mathcal{A}$) or $\mathcal{F}$ (with $\mathcal{A}\mathsf{s}$) is running, regardless of the adversary $\mathcal{A}$ used.

In our model, we treat different kinds of adversaries:

- **Passive (honest-but-curious):** the corrupted party follows the protocol as an honest party would do, but it shares all its internal state with $\mathcal{A}$.
- **Fail-Stop [11]:** the corrupted party follows the rules, but at some moment it may try to stop the protocol, so that the computation fails. In this paper, we use the definition where the party may stop the protocol only if it will not be caught (by being caught we mean that all the honest parties of the protocol consistently agree that this party is guilty).
- **Covert [12]:** the corrupted party may misbehave, but only as far as it will not be caught.
- **Active (malicious):** the corrupted party does whatever it wants.

4 Weak Collusion Preservation

In this section we present a model that allows to formalize the problems we presented in Sec. 1. We need to define more formally what it means that the protocol does not allow sensitive information to be leaked to honest parties.

Two possible models from which we could start are LUC [4] and CP [3]. At first glance, the LUC approach seems more interesting since it clearly distinguishes the cases where an uncorrupted machine M_i has received a message from another uncorrupted machine M_j, or from a corrupted machine M_k. However, many interesting properties are lost after merging the adversaries into coalitions. Hence we base our work on the collusion preserving (CP) computation of [3], since the construction is simpler in this case.

4.1 Definitions

In this subsection we first repeat some definitions of UC and CP, and then adjust them to WCP. In this paper, the simulation does not mean the transformation of the adversary as $S(\mathcal{A})$, but the parallel composition $(S\|\mathcal{A})$, meaning that the simulator S translates the messages moving between the real adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}$, but S does not get access to the other communication ports of $\mathcal{A}$. The reason is that although there is no difference for UC and CP definitions, in our model getting control over all the ports of $\mathcal{A}$ may give too much power to the simulator. We discuss it in more details when we define WCP.

Let $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ be the probability ensemble of outputs of the environment $\mathcal{Z}$ running the protocol π with the adversary $\mathcal{A}$. Recall the definition of standard UC emulation.

Definition 1 (UC emulation [2]). *Let π and ϕ be PPT (probabilistic polynomial time) protocols. We say that π UC-emulates ϕ if there exists a PPT machine S, such that for any PPT adversary $\mathcal{A}$, and for any PPT environment $\mathcal{Z}$, the probability ensembles $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ and $EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}$ are indistinguishable (denoted $EXEC_{\pi,\mathcal{A},\mathcal{Z}} \approx EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}$, or $|EXEC_{\pi,\mathcal{A},\mathcal{Z}} - EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}| < \varepsilon$).*

If the protocol ϕ is defined in a way that executing some ideal functionality $\mathcal{F}$ is the only thing that the parties do, we may also say that the protocol π UC-realizes $\mathcal{F}$.

Since Def. 1 does not specify the adversary type, we will further explicitly specify whether a protocol emulates the functionality passively, covertly, or actively.

We base our work on the collusion preserving (CP) computation of [3]. Although CP is based on *generalized universal composability* (GUC) [13], which assumes that the protocols may use some shared global setup, we first give a simplified definition based on common UC. Differently from Def. 1, instead of one monolithic adversary there are n adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_n$, one for each party. It is assumed that they do not interact with the protocol directly, but use some kind of communication resource. All the adversaries are connected with the environment $\mathcal{Z}$, and hence potentially may use it for communication.

We give the definition of CP emulation in its simplified form (without shared resources and the global setup).

Definition 2 (CP emulation [3]). *Let π and ϕ be PPT n-party protocols. We say that π CP-emulates ϕ if there exist mutually isolated PPT machines $S_1, \ldots, S_n$, such that for any PPT adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_n$ for any PPT environment $\mathcal{Z}$, for $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, $\mathcal{A}_S = \{(S_1\|\mathcal{A}_1), \ldots, (S_n\|\mathcal{A}_n)\}$, the probability ensembles $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ and $EXEC_{\phi,\mathcal{A}_S,\mathcal{Z}}$ are indistinguishable.*

In CP model, all the adversaries may still communicate through the environment, and so the values seen by any corrupted party may eventually get there. We want to modify the construction in such a way that it would take into account that the distinct adversarial coalitions will never use $\mathcal{Z}$ to communicate. Instead of assigning an adversary to each *party*, we assign an adversary to each *coalition*. We put some additional constraints on the adversary that ensure that the outputs of *only one* of these coalitions reach the environment.

Definition 3 (t-coalition split adversary). *Let n be the number of parties, and let $[n] = \{1, \ldots, n\}$. A t-coalition split adversary $\mathcal{A}$ is a set of PPT machines $\{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ defined as follows.*

1. *The adversary $\mathcal{A}$ is defined as a set PPT ITMs $\{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H\}$ ("high") and $\mathcal{A}^L$ ("low") where $\mathcal{A}_i^H$ [resp. $\mathcal{A}^L$] does not receive inputs from $\mathcal{Z}$ [resp. π] nor give outputs to π [resp. $\mathcal{Z}$]. Any communication inside $\mathcal{A}$ goes from ITM $\mathcal{A}^L$ to ITMs $\mathcal{A}_i^H$.*
2. *The active adversary $\mathcal{A}_1^H$ may corrupt up to t parties. Each party P_i that is not corrupted by $\mathcal{A}_1^H$ is corrupted by some passive adversary $\mathcal{A}_j^H$.*
3. *There is some $j \in [n]$, such that for all $i \in [n]\backslash\{j\}$, the internal state of $\mathcal{A}_i^H$ does not depend on the inputs coming from π. We call $\mathcal{A}_j^H$ the* true *adversary and the other $\mathcal{A}_i^H$-s the* false *adversaries.*

The t-coalition split adversary is depicted on Fig. 1.

The property (1) lets the information moving from $\mathcal{Z}$ to π to be controlled by a single adversary $\mathcal{A}^L$, and it splits the information moving from π to $\mathcal{Z}$ amongst different receiving adversaries. The property (2) constructs an actively corrupted coalition of size at most t, and lets each honest party be controlled by a separate passive adversary. The property (3) guarantees that the views of different coalitions will not be merged.

Let $C(k)$ be the set of party indices corrupted by $\mathcal{A}_k^H$. The execution model of a t-coalition split adversary is the following.

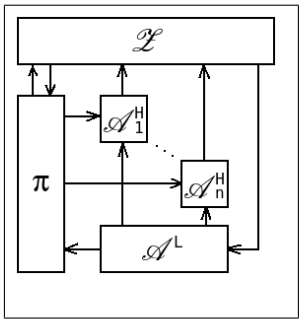

Fig. 1: t-coalition split adversary

- The corruption of a machine M_i into the coalition handled by $\mathcal{A}_j^H$ is determined by $\mathcal{A}^L$, which sends a message $(\mathsf{corrupt}, i, j)$ to the protocol. After the machine M_i receives that message, it forwards its internal state and all further received messages to the adversary $\mathcal{A}_j^H$.
- Any message $\mathtt{m}$ sent by M_i for $i \in C(1)$, can be substituted by $\mathcal{A}^L$ with an arbitrary message $\mathtt{m}^*$. Alternatively, $\mathcal{A}^L$ may substitute $\mathtt{m}$ with $\perp$, which denotes cancelling delivery of $\mathtt{m}$, or with $\top$, which denotes that $\mathtt{m}$ remains unchanged. The message $\top$ is need to enable $\mathcal{A}^L$ to proceed with honest protocol execution even if does not receive $\mathtt{m}$.

We could define WCP emulation analogously to Def. 2, just replacing *any* adversary with a *t-coalition split* adversary. However, we now need to be careful with the simulator definition. If we allow S to be an arbitrary PPT machine, then it may happen that $(S\|\mathcal{A})$ is no longer a t-coalition split adversary. Hence we need to constrain the class of simulators.

Definition 4 (split simulator). *A split simulator $S = \{S_1^H, \ldots, S_n^H, S^L\}$ consists of PPT machines S_i^H and S^L where*

- *the communication is allowed from S^L to S_i^H for all $i \in [n]$, but not the other way around;*
- *the input ports of S_i^H are connected to π, and its output ports to $\mathcal{A}_i^H$;*
- *the input ports of S^L are connected to $\mathcal{A}^L$, and its output ports to π.*

We need to ensure that $(S\|\mathcal{A}) = \{(S_1^H\|\mathcal{A}_1^H), \ldots (S_n^H\|\mathcal{A}_n^H), (S^L\|\mathcal{A}^L)\}$ is also a t-coalition split adversary, since otherwise it may happen that we give more power to the adversary that attacks an ideal functionality than to the adversary that attacks a real functionality, and that would result in weaker security proofs.

Lemma 1. *Let $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ be a t-coalition-split adversary, and let $S = \{S_1^H, \ldots, S_n^H, S^L\}$ be a split simulator. Then the parallel simulation $\mathcal{A}\mathsf{s} = \{(S_1^H\|\mathcal{A}_1^H), \ldots, (S_n^H\|\mathcal{A}_n^H), (S^L\|\mathcal{A}^L)\}$ is also a t-coalition split adversary.*

Proof. By Def. 3, there exist channels only from $\mathcal{A}^L$ to $\mathcal{A}_i^H$ for all $i \in [n]$, but not the other way around. By Def. 4, S_i^H delivers messages from the protocol π to $\mathcal{A}_i^H$, and S^L delivers messages from $\mathcal{A}^L$ to π, and similarly there exist channels only from S^L to S_i^H for all $i \in [n]$. If $\mathcal{A}_i^H$ is the true adversary, we define $\mathcal{A}_i'^H = (S_i^H\|\mathcal{A}_i^H)$. If $\mathcal{A}_i^H$ is a false adversary, then it does not listen to its inputs from S_i^H anyway. We may drop S_i^H entirely since it is just a ITM that does not send any outputs to anyone, and take $\mathcal{A}_i'^H = \mathcal{A}_i^H$ (which can be now directly connected to π) and $\mathcal{A}'^L = (S^L\|\mathcal{A}^L)$. By construction, the resulting adversary $\mathcal{A}' = \{\mathcal{A}_1'^H, \ldots, \mathcal{A}_n'^H, \mathcal{A}'^L\}$ is a t-coalition split adversary, and its behaviour is exactly the same as of $\mathcal{A}\mathsf{s}$. □.

Definition 5 (t-WCP emulation). *Let π and ϕ be n-party protocols. We say that π WCP-emulates ϕ if there exists a PPT split simulator $S = \{S_1^H, \ldots, S_n^H, S^L\}$, such that for any PPT t-coalition split adversary $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and for any PPT environment $\mathcal{Z}$, for a t-coalition split adversary $\mathcal{A}\mathsf{s} = \{(S_1^H\|\mathcal{A}_1^H), \ldots, (S_n^H\|\mathcal{A}_n^H), (S^L\|\mathcal{A}^L)\}$, the probability ensembles $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ and $EXEC_{\phi,\mathcal{A}\mathsf{s},\mathcal{Z}}$ are indistinguishable.*

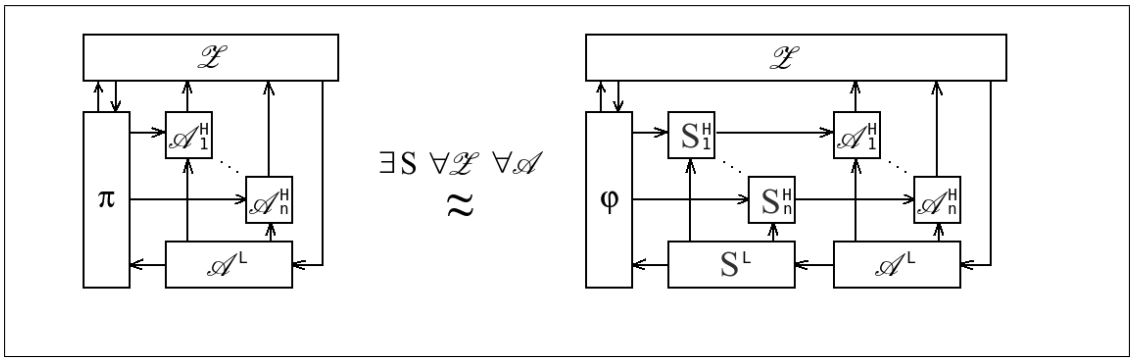

Fig. 2: t-WCP emulation

The definition is correct by Lemma 1. The t-WCP emulation is depicted on Fig. 2.

We emphasize that we intentionally require blackbox simulatability, i.e the same simulator S must be suitable for an arbitrary adversary $\mathcal{A}$. Intuitively, in this case the simulator does not know which $\mathcal{A}_i^H$ is the true adversary, and hence each S_i^H needs to simulate a proper view to all $\mathcal{A}_i^H$, not only to the true one. This is one reason why we use the parallel composition $(S_i^H \| \mathcal{A}_i^H)$ for the simulation, and not the transformation $S_i^H(\mathcal{A}_i^H)$ where the code of $\mathcal{A}_i^H$ could potentially tell S_i^H directly whether $\mathcal{A}_i^H$ is true or false adversary.

What happens if during the protocol simulation, some S_i^H somehow gets to know that $\mathcal{A}_i^H$ is a false adversary? It may happen that $\mathcal{A}^L$ directly tells to S^L which $\mathcal{A}_i^H$ is true, or $\mathcal{A}_i^H$ even proves that it is true by forwarding some challenge of S_i^H through $\mathcal{Z}$ and ϕ back to S_i^H (a false adversary $\mathcal{A}_i^H$ cannot forward a challenge from S_i^H to anyone, because it does not listen to the inputs coming from S_i^H). However, since S should work for an arbitrary $\mathcal{A}$ and any $\mathcal{Z}$, in general $\mathcal{Z}$ is not supposed to forward the messages from $\mathcal{A}$ to the protocol, and even if $\mathcal{A}^L$ gives some direct hints to S^L, they are not necessarily believable. We conclude that all the simulators $S_1^H, \ldots, S_n^H$ will have to work for all $\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H$ in any case.

4.2 Composition Theorem

Dummy Lemma The composition proofs of UC are simpler if instead of an arbitrary adversary $\mathcal{A}$ we consider the dummy adversary $\mathcal{D}$ that only forwards the messages between the protocol and the environment. This kind of adversary is in some sense the strongest one since it delegates all the attacks to the environment $\mathcal{Z}$, and it just gives to $\mathcal{Z}$ the entire view of the corrupted parties. The *dummy lemma* has been proven in [2] and it holds also in LUC and CP models. In our WCP definition, we could also substitute the true adversary with a dummy adversary, similarly to UC. However, the false adversaries are not allowed to forward the messages. If we replace a false adversary with $\mathcal{D}$, it will be too strong since the environment $\mathcal{Z}$ becomes able to forward its inputs through $\mathcal{D}$. We conclude that the dummy lemma of UC that works also for CP and LUC is not straightforwardly applicable to WCP. Nevertheless, it holds if $\mathcal{D}$ satisfies the t-coalition adversary definition.

Definition 6 (k-dummy t-coalition split adversary). *Let n be the number of parties, and let $k \in [n]$. The k-dummy t-coalition split adversary $\mathcal{D}\mathsf{k} = \{\mathcal{D}\mathsf{k}_1^H, \ldots, \mathcal{D}\mathsf{k}_n^H, \mathcal{D}\mathsf{k}^L\}$ is a t-coalition split adversary, where:*

- *$\mathcal{D}\mathsf{k}^L = \mathcal{D}$ is just a message forwarding ITM;*
- *$\mathcal{D}\mathsf{k}_k^H = \mathcal{D}$ is also a message forwarding ITM, but $\mathcal{D}\mathsf{k}_i^H$ for $i \neq k$ does not forward the inputs that come from π (this is a actually a part of Def. 3).*

For n parties, there are n different k-dummy adversaries $\mathcal{D}\mathsf{1}, \ldots, \mathcal{D}\mathsf{n}$.

Lemma 2 (t-dummy lemma). *Let π and ϕ be n-party protocols. Then π t-WCP-emulates ϕ according to Def. 5 if and only if it t-WCP-emulates ϕ with respect to all k-dummy t-coalition split adversaries for all $k \in [n]$.*

The proof of Lemma 2 can be found in App. A.1.

WCP Composition Theorem We need to prove that our WCP definition is composable. Although it is impossible to achieve collusion-freeness for the adversaries if they are allowed to use embedding protocols as side channels, we have ensured by the definition of t-coalition split adversary that they will not try to do it. The composition theorem nevertheless remains useful for the ordinary attacks that are not related to subliminal channels, an that is why it still makes sense to define a composition theorem for WCP.

Theorem 1 (WCP composition theorem). *Let ρ, ϕ, π be protocols such that ρ uses ϕ as subroutine, and π t-WCP-emulates ϕ. Then protocol $\rho[\phi \to \pi]$ t-WCP-emulates ρ.*

The proof of Thm. 1 can be found in App. A.2.

4.3 Relations to the Existing Notions

We need to show that no attack that UC model detects remains unnoticed by WCP model. Namely, we show that t-WCP-emulation implies UC-emulation, and hence our security definition is stronger. However, failure in achieving t-WCP-specific properties does not provide an immediate UC security fallback in general (as in the case of CP), but on the assumption that only t parties remain corrupted.

Since the ports between π and $\mathcal{A}$ are different for UC and WCP, we need to define a transformation between UC and WCP functionalities, as it was done for CP and LUC. The transformation is analogous, and it either splits the monolithic adversary to distinct coalitions, or merges the coalitions into one monolithic adversary. The formal definitions of these transformations are given in App. B.

Theorem 2. *Let π be a protocol that t-WCP emulates a protocol ϕ. Then π also UC emulates ϕ in presence of at most t corrupted parties. However, there exists protocols π and ϕ, such that π UC-emulates ϕ in presence of at most t corrupted parties, but does not t-WCP emulate it.*

The proof of Thm. 2 can be found in App. A.3.

We would also like to compare WCP and CP. In general, CP security is stronger since a t-coalition split adversary is an instance of CP adversary where the entire $\mathcal{A}^L$ can be pushed into $\mathcal{Z}$, and the collaboration of coalitions can be also arranged through $\mathcal{Z}$. The simulators S_i of CP could be used as S_i^H in WCP. The only problem is that the simulator S_i of CP translates the messages between $\mathcal{A}_i$ and ϕ in both directions, while WCP allows S_i^H to only forward messages from ϕ to $\mathcal{A}_i^H$. Using a single S^L for simulating the other direction may fail without knowing certain inputs that S_i^H has got from ϕ.

Hence we could straightforwardly use only such functionalities ϕ that do not give to the adversary any outputs before they have already received from it all the inputs.

Definition 7 (one-time input protocol). *A protocol ϕ is called one-time input if all the inputs that it gets from the adversary $\mathcal{A}$ are obtained before any output is given by ϕ to $\mathcal{A}$.*

We show that, assuming that the number of corrupt parties is the same, and ϕ is one-time input protocol, then CP emulation implies WCP emulation. However, depending on the choice of t, it may happen that t-WCP is strictly weaker than CP.

Theorem 3. *Let t be the total number of corrupted parties. Let π be a protocol that CP emulates a one-time input protocol ϕ. Then π also t'-WCP emulates ϕ for any $t' \leq t$. However, there exists a $t' < t$ and protocol π and ϕ, such that π t'-WCP emulates ϕ, but does not CP emulate it.*

The proof of Thm. 3 can be found in App. A.4.

5 Applicability of WCP Security

In this section we show why WCP is a suitable model for pointing out the problems we mentioned in Sec. 1. We present some properties related to leaking information to an honest party that can be captured by t-WCP, but not by UC, CP, LUC. Since CP lets the adversaries to communicate through an arbitrary resource R, the security in CP model may be dependent on the particular choice of R, which allows it to be stronger as well as weaker than the other models. In order to make the definitions similar, we assume that R delivers to $\mathcal{A}_i$ the internal state of M_i, and the adversary $\mathcal{A}_i$ may also replace any message m sent by M_i by a message m^* of $\mathcal{A}_i$'s own choice.

The relations of our protocols and functionalities with the adversaries are described as $\mathcal{A}(i)$, where i is some party identifier, and $\mathcal{A}(i)$ corresponds to *all i-related adversaries*, which is just $\mathcal{A}$ for UC, $\mathcal{A}_i$ for CP, $\mathcal{A}_{c(i)}$ for WCP, and $\mathcal{A}_{i,1}, \ldots, \mathcal{A}_{i,n}$ for LUC. More details about transformations between different adversaries can be found in App. B.

We now present an ideal functionality $\mathcal{F}_0$ and two of its possible realizations π_1 and π_2. We see that, while for UC, CP, LUC these realizations either both realize or do not realize $\mathcal{F}_0$, they are different in t-WCP model.

Let $Enc(key, message)$ be some symmetric computationally secure encryption scheme that is secure with respect to a uniformly distributed key.

Ideal. The ideal functionality $\mathcal{F}_0$ takes a secret s from a certain party P_i. If P_i is actively corrupted, then $\mathcal{F}_0$ outputs s to each $\mathcal{A}(j)$ for $j \in [n]$. The adversary is allowed to abort the protocol. If it does not, $\mathcal{F}_0$ outputs 0 to each party.

Protocol 1. Consider the protocol π_1 where a (symmetric) key is generated as $k = \sum_{\ell \in \mathcal{I}} k_\ell$ where $\mathcal{I}$ is a set of arbitrarily chosen t parties that are supposed to generate k_ℓ from uniform distribution. All k_ℓ are sent to the party M_i that encrypts a secret s with this key and sends $Enc(k, s)$ to some party M_j. If any party refuses to send its message, the protocol aborts.

Protocol 2. Consider an analogous protocol π_2 which works in exactly the same way, but where M_i itself generates one more share k_{t+1} of k, and sends it to all other parties.

We now compare these protocols in various models.

- **UC** Assuming that the total number of corrupted parties is at most t, both π_1 and π_2 UC-realize $\mathcal{F}_0$. If M_i is corrupted, then S gets s from $\mathcal{F}_0$ and can simulate everything. Otherwise, the adversary either gets only the key k (if P_j is not corrupted), or it gets $Enc(k, s)$ and up to all shares of k except one (if P_j is corrupted). If the number of corrupted parties is at least $t+1$, then both protocols are insecure since all the shares of the key and the $Enc(k, s)$ may leak to $\mathcal{Z}$.
- **CP, LUC** If M_i is corrupted, then the key generating parties may use their shares of k as side channels for collaborating with $\mathcal{A}(i)$, and hence neither π_1 nor π_2 does not realize $\mathcal{F}_0$. Let M_i be honest. Assuming that the total number of corrupted parties is at most t, the functionalities π_1 and π_2 both realize $\mathcal{F}_0$. If at least one key generating party is honest, the simulator $S(j)$ only needs to simulate $Enc(k, s)$ as if the key was uniform. If all the key generating parties are corrupt, then k might not be uniform, but in this case P_j is uncorrupted, and S_j does not have to simulate anything. If the total number of corrupted parties is at least $t+1$, then both the k and $Enc(k, s)$ may leak to $\mathcal{Z}$, and hence π_1 and π_2 are both insecure, similarly to UC.
- **WCP** The protocol π_2 does t-WCP-realize $\mathcal{F}_0$, but π_1 does not. If M_i is corrupted, then all S_j^H get s from $\mathcal{F}_0$, and S^L gets from $\mathcal{A}^L$ all the shares of k that S^L delivers to all S_j^H, so these side-channels are not taken into account by WCP. Let M_i be honest. In π_1, if all the t key generating parties are corrupted, then S_j^H has to simulate $Enc(k, s)$ based on the bad key k that no longer comes from uniform distribution and might be known by $\mathcal{Z}$. Although S^L might have sent the bad key k to S_j^H, it still does not know s, and hence cannot simulate $Enc(k, s)$. In π_2, the key k comes from a uniform distribution in any case, since at least one share is generated by the uncorrupted M_i itself. The question is whether k may leak to $\mathcal{Z}$ if all the key generating parties are controlled by an adversarial coalition of size t, as they also get the final share k_{t+1} at some moment. We care about the simulation by S_j^H only if $\mathcal{A}_j^H$ is the true adversary. In this case, the entire key generating coalition has been controlled by a false adversary that never leaks the final share k_{t+1} to $\mathcal{Z}$.

An analysis of a particular multiparty computation protocol of [5] related to bad key generation is given in App. D. Another example of an attack captured by WCP model is given in App. C.

6 Achieving t-WCP Security

We start from a protocol that is secure against $t < n/2$ passively corrupted parties. In this section, we show how such a protocol can be made secure against $t < n/2$ actively corrupted parties, allowing up to all the other parties to be passively corrupted (i.e "semihonest majority" assumption).

6.1 Passively Corrupted Coalitions

First of all, we show that UC and t-WCP emulations are equivalent definitions if the UC model allows at least t parties to be corrupt, and the adversary is passive. This shows that it does not make sense to define a special transformation for making a protocol passively secure in t-WCP model.

Theorem 4. *Let π be a protocol that passively UC-emulates a protocol ϕ in presence of t corrupted parties. Then π also passively t-WCP emulates ϕ.*

The proof of Thm. 4 is based on the fact that a passive adversary will not interact with the protocol, and so all the false adversaries do not interact with the protocol at all. The only true adversary is handled as in the UC model. A more formal proof be found in App. A.5.

6.2 Fail-Stop Coalitions

A fail-stop adversary [11] follows the protocol as the honest parties do, but it also may force the corrupt parties to abort the protocol. In this case, the protocol may still be secure in t-WCP model if noone attempts to stop the protocol. However, the functionalities used to prevent the protocol from stopping may explicitly require to leak a secret to some honest party.

As a simple example, let us take the transmission functionality $\mathcal{F}_{tr}$ that has been used in [14,15] to prevent the protocol from aborting by pointing out the exact party that has aborted the protocol. This helps against a fail-stop adversary that does not want to be accused in cheating. Suppose that a party P_i should be sending a message m_{ij} to another party P_j. If P_i refuses to send the message to P_j, then there is no way for neither party to prove whether P_i is indeed silent, or P_j has already received m_{ij} but just accuses P_i without reason. The realization of $\mathcal{F}_{tr}$ works on the assumption that the majority of parties follows the protocol. If there is a fail-stop conflict between P_i and P_j, then the message should just be broadcast by P_i to all the parties, so that they get the evidence that P_j indeed received it. Now if P_i decides to abort the protocol, then it will be blamed by everyone. The definition of $\mathcal{F}_{tr}$ is given in Fig. 3 (the functions *reshare received message* and *reveal received message* will be needed for stronger adversaries).

Compared to [14, 15], we need to modify the realization of $\mathcal{F}_{tr}$ in such a way that it would be secure in t-WCP model. In the single adversary case, if there is a conflict between P_i and P_j, then at least one of them is corrupted, and hence the adversary already knows that message anyway. Therefore, P_i may broadcast the message to all the parties, so that they may prove or disprove that P_i has sent the message. However, this does not work for multiple adversaries, since some other party P_k may receive a message that P_i and P_j would exchange privately.

We propose a slight modification to the realization of the functionality $\mathcal{F}_{tr}$ given in [14]. Now for each message bitstring m_{ij} transmitted from P_i to P_j, there is a random bit mask q_{ij}^m that is known by both P_i and P_j, but to noone else (this can be done by sharing a common randomness between each pair of parties). In the case of conflict, P_i signs and broadcasts $m'_{ij} = m_{ij} \oplus q_{ij}^m$ to all the parties, and P_j computes $m'_{ij} \oplus q_{ij}^m$. Masking messages by randomness can be viewed as a weaker version of the physical envelopes used in the construction collusion-free protocols of [9].

Let $[n] = \{1, \ldots, n\}$, where n is the number of parties. Let $\mathcal{A}\mathsf{s} = \{\mathcal{A}\mathsf{s}_1^H, \ldots, \mathcal{A}\mathsf{s}_n^H, \mathcal{A}\mathsf{s}^L\}$ be the ideal t-coalition split adversary. Let $c(i)$ be the index of the coalition to which the party P_i belongs.
$\mathcal{F}_{tr}$ works with unique message identifiers mid, encoding a sender $s(mid) \in [n]$ and a receiver $r(mid) \in [n]$. Some (n, t) threshold sharing scheme is defined.
Secure transmit: Receiving $(\mathsf{transmit}, mid, m)$ from $P_{s(mid)}$ and $(\mathsf{transmit}, mid)$ from all (semi)honest parties, store $(mid, m, r(mid))$, mark it as undelivered, and output $(mid, |m|)$ to all $\mathcal{A}\mathsf{s}_i^H$. If the input of $P_{s(mid)}$ is invalid (or there is no input), and $P_{r(mid)}$ is (semi)honest, then output $(\mathsf{corrupt}, s(mid))$ to all parties.
Secure broadcast: Receiving $(\mathsf{broadcast}, mid, m)$ from $P_{s(mid)}$ and $(\mathsf{broadcast}, mid)$ from all (semi)honest parties, store (mid, m, bc), mark it as undelivered, output $(mid, |m|)$ to all $\mathcal{A}\mathsf{s}_i^H$. If the input of $P_{s(mid)}$ is invalid, output $(\mathsf{corrupt}, s(mid))$ to all parties.
Synchronous delivery: At the end of each round, for each undelivered (mid, m, r) send (mid, m) to P_r; mark (mid, m, r) as delivered. For each undelivered (mid, m, bc), send (mid, m) to each party and all $\mathcal{A}\mathsf{s}_i^H$; mark (mid, m, bc) as delivered.
Reshare received message: On input $(\mathsf{reshare}, mid)$ from the party $P_{r(mid)}$ which at any point received (mid, m), share m to shares m^k and output (mid, m^k) to the party P_k. If both $P_{s(mid)}$ and $P_{r(mid)}$ are corrupt, let $\mathcal{A}\mathsf{s}^L$ choose an arbitrary m.
Reveal received message: On input (reveal, mid) from the party $P_{r(mid)}$ which at any point received (mid, m), output (mid, m) to each party P_k. If both $P_{s(mid)}$ and $P_{r(mid)}$ are corrupt, let $\mathcal{A}\mathsf{s}^L$ choose an arbitrary m.

Fig. 3: Ideal functionality $\mathcal{F}_{tr}$

Lemma 3. *Assuming that the majority of parties are at least semihonest, there exists an realization of $\mathcal{F}_{tr}$ that is secure in t-WCP model.*

Lemma 3 is proven by construction of a certain realization of $\mathcal{F}_{tr}$ in App. E. Using this result, we can state and prove the following theorem.

Theorem 5. *Let π be a protocol where the parties use the functionality $\mathcal{F}_{tr}$ for communication. Let π passively UC-emulate a protocol ϕ in presence of t corrupted parties. If the majority of parties is at most semihonest, then π also t-WCP emulates ϕ, assuming that the strongest adversaries are at most fail-stop.*

Proof. As long as the parties behave according to the protocol, the protocol π emulates ϕ by Thm. 4. The only other thing that a fail-stop adversary may try to do is to stop the protocol. This means that at some point there is a situation where some machine M_i refuses to send a message to some other machine M_j. If P_k for $k \in \{i, j\}$ is corrupted, then $\mathcal{F}_{tr}$ ensures that either all the (honest) parties unanimously agree that P_k is corrupted, or P_i delivers the message to P_j as intended by the original protocol. □

6.3 Covertly Corrupted Coalitions

A covert adversary [12] will not cheat if it will be caught with a non-negligible probability. Assuming that a covert adversary will act as passive anyway, we can extend the result of fail-stop adversaries to covert adversaries since $\mathcal{A}^L$ will not attempt to modify the flow of π. Hence we may be sure that, if a covert adversary will not attempt to cheat, then UC-emulation implies t-WCP emulation.

It is more difficult to reason about fallback security, i.e what happens if the adversary does not follow the protocol regardless of being punished. There may be still more attacks in the t-WCP model than in the UC model, and this will be discussed in more details in Sec. 6.4.

Theorem 6. *Let π be a protocol where the parties use the functionality $\mathcal{F}_{tr}$ for communication. Let π UC-emulate a protocol ϕ in presence of t covertly corrupted parties. If the majority of parties is at least semihonest, then π also t-WCP emulates ϕ, assuming that the strongest adversaries are at most covert.*

Let $[n] = \{1, \ldots, n\}$, where n is the number of parties. Let $\mathcal{A}\mathsf{s} = \{\mathcal{A}\mathsf{s}_1^H, \ldots, \mathcal{A}\mathsf{s}_n^H, \mathcal{A}\mathsf{s}^L\}$ be the ideal t-coalition split adversary. Let $c(i)$ be the index of the coalition to which the party P_i belongs. By Def. 3, let 1 be the index of the actively corrupted coalition (in this way, $C(1)$ is the set of indices of actively corrupted parties). $\mathcal{F}_{vmpc}$ works with session identifiers sid, where $\boldsymbol{r}_i[sid]$ is the randomness of P_i, $\bar{\boldsymbol{x}}_i[sid]$ are all the inputs of P_i committed so far, and $\bar{\boldsymbol{m}}_i[sid]$ are all the messages received by P_i so far, and $\boldsymbol{m}_{ij}[sid, \ell]$ are the committed outputs of P_i to P_j (there can be several such outputs for the same sid, representing different rounds).

Random tape generation On input $(\mathsf{gen\_rnd}, sid, i)$ from all (semi)honest parties, $\mathcal{F}_{vmpc}$ randomly generates $\boldsymbol{r}_i$. It outputs $\boldsymbol{r}_i$ to P_i and also sends $(\mathsf{randomness}, i, \boldsymbol{r}_i)$ to $\mathcal{A}\mathsf{s}_{c(i)}^H$. $\mathcal{F}_{vmpc}$ treats $\boldsymbol{r}_i$ as the committed randomness for P_i's computation. Alternatively, a message $\perp$ may come from $\mathcal{A}\mathsf{s}^L$, and in this case the randomness generation fails.

Input commitment On input $(\mathsf{commit\_input}, sid, i, \boldsymbol{x}_i)$ from P_i and $(\mathsf{commit\_input}, sid, i)$ from all other (semi)honest parties, $\mathcal{F}_{vmpc}$ appends $\boldsymbol{x}_i$ to $\bar{\boldsymbol{x}}_i[sid]$. For $i \in C(1)$, it sends $(\mathsf{input}, i, \boldsymbol{x}_i)$ to $\mathcal{A}\mathsf{s}_{c(i)}^H$. Alternatively, a message $(\mathsf{corrupt}, j)$ may come from $\mathcal{A}\mathsf{s}^L$ with $j \in C(1)$. $\mathcal{F}_{vmpc}$ defines $\mathcal{B}_0 = \{j \mid (\mathsf{corrupt}, j)$ has been sent by $\mathcal{A}\mathsf{s}^L\}$.

Message commitment On input $(\mathsf{commit\_msg}, sid, i, j, \ell, \boldsymbol{m})$ from P_i and $(\mathsf{commit\_msg}, (sid, \ell), i, j)$ from all (semi)honest parties, $\mathcal{F}_{vmpc}$ stores $\boldsymbol{m}_{ij}[sid, \ell] = \boldsymbol{m}$. Alternatively, a message $(\mathsf{corrupt}, j)$ may come from $\mathcal{A}\mathsf{s}^L$ with $j \in C(1)$. $\mathcal{F}_{vmpc}$ defines $\mathcal{B}_0 = \{j \mid (\mathsf{corrupt}, j)$ has been sent by $\mathcal{A}\mathsf{s}^L\}$.

Verification On input $(\mathsf{verify}, sid, C, i, j, \ell)$ from all (semi)honest parties, where C is the description of circuit that corresponds to the computation of a message for P_j by P_i, $\mathcal{F}_{vmpc}$ checks if $\boldsymbol{m}_{ij}[sid, \ell]$ and all the values $\bar{\boldsymbol{x}}_i[sid]$, $\boldsymbol{r}_i[sid]$, $\bar{\boldsymbol{m}}_i[sid]$ necessary for computing $C(\bar{\boldsymbol{x}}_i[sid], \boldsymbol{r}_i[sid], \bar{\boldsymbol{m}}_i[sid])$ are committed. If they are, $\mathcal{F}_{vmpc}$ computes $\boldsymbol{m}'_{ij} = C(\bar{\boldsymbol{x}}_i[sid], \boldsymbol{r}_i[sid], \bar{\boldsymbol{m}}_i[sid])$. If $\boldsymbol{m}'_{ij} = \boldsymbol{m}_{ij}[sid, \ell]$, then $\mathcal{F}_{vmpc}$ outputs $(\mathsf{approved}, sid, C, i, j, \ell, \boldsymbol{m}_{ij}[sid, \ell])$ to P_j and $(\mathsf{approved}, sid, C, i, j)$ to all other parties. It appends $\boldsymbol{m}_{ij}$ to $\bar{\boldsymbol{m}}_j[sid]$ and outputs $\boldsymbol{m}_{ij}$ to $\mathcal{A}\mathsf{s}_{c(j)}^H$. If $j \in C(1)$, then $\mathcal{F}_{vmpc}$ appends $\boldsymbol{m}'_{ij}$ to $\bar{\boldsymbol{m}}_j[sid]$ even if $\boldsymbol{m}'_{ij} \neq \boldsymbol{m}_{ij}$. In any case, it outputs C to each adversary $\mathcal{A}_k^H$.

$\mathcal{F}_{vmpc}$ defines $\mathcal{M} = \mathcal{B}_0 \cup \{i \in [n] \mid \exists j : \boldsymbol{m}'_{ij} \neq \boldsymbol{m}_{ij}[sid, \ell]\}$. For all $i \notin C(1)$, $\mathcal{A}\mathsf{s}^L$ sends $(\mathsf{blame}, i, \mathcal{B}_i)$ to $\mathcal{F}_{vmpc}$, with $\mathcal{M} \subseteq \mathcal{B}_i \subseteq \mathcal{C}$. $\mathcal{F}_{vmpc}$ outputs $(\mathsf{blame}, sid, \ell, \mathcal{B}_i)$ to P_i.

Fig. 4: The ideal functionality for verifiable computations

Proof. As long as the parties behave according to the protocol, the protocol π emulates ϕ by Thm. 4. In the attempts to stop the protocol, π emulates ϕ by Thm. 6. Since the protocol is secure against covert adversaries, all the attempts to cheat in any other way will be detected, and hence the adversary will just not try cheat in any other way, so we do not need to check if π still emulates ϕ in any active attacks. □

6.4 Actively Corrupted Coalitions

For constructing a multiparty protocol secure against active adversaries, we follow the general pattern used in other related works [16, 17]. In this approach, the protocol is constructed in several steps. Initially, there is a multiparty protocol secure only against a passive adversary. In order to make it secure against an active adversary, on each round, each party needs to provide a zero-knowledge proof that it has followed the protocol rules.

Our protocol transformation is similar to the transformation Comp of [18] (the full version of [17]). We present a functionality $\mathcal{F}_{vmpc}$ that can be viewed as a set of ideal functionalities called by Comp on different steps. It is given in Fig. 4.

In App. F, we give a protocol that t-WCP realizes $\mathcal{F}_{vmpc}$. Since our implementation relies on Byzantine agreement, it works under (semi)honest majority assumption. We use $\mathcal{F}_{vmpc}$ to construct a protocol transformation WCP-Comp that is analogous to Comp. It is given in Fig. 5.

Having access to WCP-Comp, we may prove the following theorem.

Theorem 7. *Let π be a protocol that passively UC-emulates a protocol ϕ in presence of t corrupted parties. Assuming that the majority of parties is at least semihonest, the protocol* WCP-Comp(π) *t-WCP emulates ϕ in presence of a coalition of t active adversaries.*

Proof. We prove that the protocol WCP-Comp(π) indeed t-WCP emulates ϕ in presence of t active adversaries.

Let $\boldsymbol{x}_i$ be the vector of inputs of the party P_i in the protocol π. Let $\boldsymbol{r}_i$ be the randomness used by P_i.

1. *Random tape generation.* When activating $\mathsf{WCP\text{-}Comp}(\pi)$ for the first time with session identifier sid, all (semi)honest parties send $(\mathsf{gen\_rnd}, sid, i)$ to $\mathcal{F}_{vmpc}$ for all $i \in [n]$.
2. *Activation due to new input.* When activated with input $(sid, \boldsymbol{x}_i)$, party P_i proceeds as follows.
   (a) *Input commitment:* At any moment when a party P_i should commit an input, all the (semi)honest parties send $(\mathsf{commit\_input}, sid, i)$ to $\mathcal{F}_{vmpc}$. P_i sends $(\mathsf{commit\_input}, sid, i, \boldsymbol{x}_i)$ to $\mathcal{F}_{vmpc}$ and adds $\boldsymbol{x}_i$ to the list of inputs $\bar{\boldsymbol{x}}_i$ (this list is initially empty and contains P_i's inputs from the previous activations of π). P_i then proceeds to the next step.
   (b) *Protocol computation:* Let $\bar{\boldsymbol{m}}_i$ be the series of messages that were transmitted to P_i in all the activations of π until now ($\bar{\boldsymbol{m}}_i$ is initially empty). P_i runs the code of π on its input list $\bar{\boldsymbol{x}}_i$, messages $\bar{\boldsymbol{m}}_i$, and random tape $\boldsymbol{r}_i$. If π instructs P_i to transmit a message, P_i proceeds to the next step.
   (c) *Outgoing message transmission:* Let $\boldsymbol{m}_{ij}^{\ell}$ be the outgoing message that P_i sends in π to P_j on ℓ-th round. As soon as the ℓ-th round starts, all the (semi)honest parties send $(\mathsf{commit\_msg}, sid, i, j, \ell)$ to $\mathcal{F}_{vmpc}$ for all $i, j \in [n]$. P_i sends $(\mathsf{commit\_msg}, sid, i, j, \ell, \boldsymbol{m}_{ij}^{\ell})$ to $\mathcal{F}_{vmpc}$.
3. *Activation due to incoming message* Let C_{ij}^{ℓ} be the description of the arithmetic circuit representing the computation of P_i on the ℓ-th round that finally outputs $\boldsymbol{m}_{ij}^{\ell}$ to P_j. As soon as each party has finished with its computation of the ℓ-th round, it sends $(\mathsf{verify}, sid, C_{ij}^{\ell}, i, j, \ell)$ to $\mathcal{F}_{vmpc}$. Upon receiving a message $(\mathsf{approved}, sid, C_{ij}^{\ell}, i, j, \ell, \boldsymbol{m}_{ij}^{\ell})$ from $\mathcal{F}_{vmpc}$, P_j appends $\boldsymbol{m}_{ij}^{\ell}$ to $\bar{\boldsymbol{m}}_j$ and proceed with the Step 2b above. All the other (semi)honest parties wait for the message $(\mathsf{approved}, sid, C_{ij}^{\ell}, i, j, \ell)$ from $\mathcal{F}_{vmpc}$ to proceed with the Step 2b.
   In addition, $\mathcal{F}_{vmpc}$ outputs a message $(\mathsf{blame}, sid, \ell, \mathcal{B}_i)$ to each (semi)honest $\mathcal{B}_i$. The way in which (semi)honest parties handle the set $\mathcal{B}_i$ depends on the particular protocol π.
4. *Output:* Whenever π generates an output value, $\mathsf{WCP\text{-}Comp}(\pi)$ generates the same output value.

Fig. 5: The compiled protocol $\mathsf{WCP\text{-}Comp}(\pi)$

As far as all the messages are generated according to the rules, the protocol is secure by Thm. 4. We need to show that no other information will be leaked due to verification procedures. By definition, for all $i \in [n]$, $\mathcal{F}_{vmpc}$ stores all the inputs $\boldsymbol{x}_i$, the randomness $\boldsymbol{r}_i$, and all the communication $\boldsymbol{m}_i$ received by P_i. $\mathcal{F}_{vmpc}$ does not output any message $\boldsymbol{m}_{ij}^{\ell}$ to the party P_j unless it has indeed been computed by the protocol rules defined by the circuit C_{ij}^{ℓ}.

As the side-effect, $\mathcal{F}_{vmpc}$ only outputs to $\mathcal{A}\mathsf{s}_{c(i)}^{H}$ the inputs $\boldsymbol{x}_i$, the randomness $\boldsymbol{r}_i$, and the messages $\boldsymbol{m}_{ji}^{\ell}$ generated for P_i according to the protocol rules. All these messages would be sent by a corrupted party to $\mathcal{A}\mathsf{s}_{c(i)}^{H}$ also in the protocol π. □

7 Conclusions

We have defined an alternative version of UC that is motivated by real world applications and allows to prove that the protocol is protected also against honest users. It helps to avoid some attacks that are not covered by the standard UC definition and makes the protocol reliable not on some participants' unconditional honestness, but rather on their non-collusion which seems a more realistic assumption. Compared to the other similar models, the security defined in our model is weaker and thus easier to achieve. The definition is nevertheless stronger than the standard UC security definition, and hence we do not lose in security using this new model.

We have proposed some schemes transforming passively secure protocols with one adversary up to actively secure protocols with semihonest majority and multiple adversaries. While CP and LUC models require a special mediator party to eliminate any subliminal channels that is essential for achieving their definitions, in our model it is sufficient to eliminate only the non-deliberate subliminal channels that could be accidentally used by the parties that follow the protocol. However, we note that we made some additional assumptions when defining transformations for actively secure protocols. Although our proposed protocols are obviously insecure in CP and LUC models due to missing coalition splitting, we think that also CP and LUC models would benefit from making some assumptions about the behaviour of semihonest parties.

References

1. Bruce Schneier. Data is a toxic asset, March 2016. `https://www.schneier.com/blog/archives/2016/03/data_is_a_toxic.html`.
2. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.
3. Joël Alwen, Jonathan Katz, Ueli Maurer, and Vassilis Zikas. Collusion-preserving computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 124–143. Springer, 2012.
4. Ran Canetti and Margarita Vald. Universally composable security with local adversaries. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, volume 7485 of *Lecture Notes in Computer Science*, pages 281–301. Springer, 2012.
5. Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 591–602, New York, NY, USA, 2015. ACM.
6. Ueli Maurer and Renato Renner. Abstract cryptography. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 1–21. Tsinghua University Press, 2011.
7. Joël Alwen, abhi shelat, and Ivan Visconti. Collusion-free protocols in the mediated model. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 497–514. Springer, 2008.
8. Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, abhi shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 524–540. Springer, 2009.
9. Matt Lepinski, Silvio Micali, and abhi shelat. Collusion-free protocols. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 543–552. ACM, 2005.
10. Shai Halevi, Paul A. Karger, and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. *IACR Cryptology ePrint Archive*, 2005:169, 2005.
11. Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model (extended abstract). In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO 87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155. Springer Berlin Heidelberg, 1988.
12. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.
13. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.
14. Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.
15. Peeter Laud and Alisa Pankova. Preprocessing-based verification of multiparty protocols with honest majority. Cryptology ePrint Archive, Report 2015/674, 2015. `http://eprint.iacr.org/`.
16. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229. ACM, 1987.
17. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 494–503. ACM, 2002.
18. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. *IACR Cryptology ePrint Archive*, 2002:140, 2002.
19. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

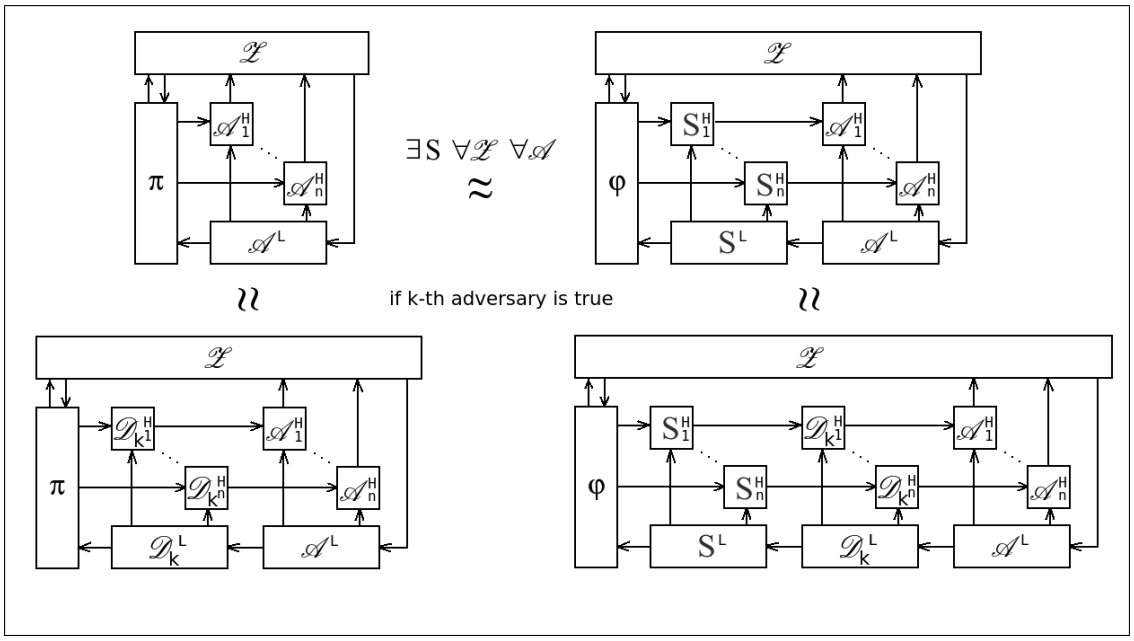

Fig. 6: t-dummy lemma

20. Peeter Laud and Alisa Pankova. Verifiable Computation in Multiparty Protocols with Honest Majority. In Sherman S. M. Chow, Joseph K. Liu, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2014.

A Proofs

A.1 Proof of t-Dummy Lemma

One proof direction is trivial since a t-dummy adversary is just an instance of a t-coalition split adversary. The other direction is more interesting. Let S be the split simulator for any k-dummy adversary $\mathcal{D}\mathsf{k}$ guaranteed by Def. 5 (that is, S satisfies $EXEC_{\phi,(S\|\mathcal{D}\mathsf{k}),\mathcal{Z}'} \approx EXEC_{\pi,\mathcal{D}\mathsf{k},\mathcal{Z}'}$ for all $\mathcal{Z}'$.) We show that π t-WCP emulates ϕ according to Def. 5. We claim that S is a suitable simulator for an arbitrary t-coalition split adversary $\mathcal{A}$.

Assume by contradiction that S is a bad simulator, and there is a t-coalition split adversary $\mathcal{A}$ (let $\mathcal{A}_k^H$ be the true one) and an environment $\mathcal{Z}$ such that $|EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}} - EXEC_{\pi,\mathcal{A},\mathcal{Z}}| \geq \varepsilon$. We use it to construct $\mathcal{Z}'$ that breaks $EXEC_{\phi,(S\|\mathcal{D}\mathsf{k}),\mathcal{Z}'} \approx EXEC_{\pi,\mathcal{D}\mathsf{k},\mathcal{Z}'}$. First, we define $\mathcal{Z}' = (\mathcal{A}\|\mathcal{Z})$. Since $\mathcal{Z}'$ drops all the inputs coming to false $\mathcal{A}_i^H$ anyway, and $\mathcal{D}_i^H$ have no other outputs, we could as well put $\mathcal{D}k$ between the protocol and $\mathcal{Z}'$, getting $|EXEC_{\phi,(S\|\mathcal{D}\mathsf{k}),\mathcal{Z}'} - EXEC_{\pi,\mathcal{D}\mathsf{k},\mathcal{Z}'}| \geq \varepsilon$.
The quantities used in the proof are depicted on Fig. 6.

A.2 Proof of WCP Composition Theorem

We take the simpler proof variant of UC composition theorem of [2] that proves the claim for one instance of ϕ and then extends it to polynomially many calls of ϕ by induction (taking into account that simulation quality is lost). We could base the proof on t-dummy lemma and make it similar to the proofs of UC,CP,LUC compositions, but since we do not want our proofs to be too dependent on each other, we present an alternative proof here.

Consider the protocol ρ that uses ϕ once as subroutine. We need to prove that there exists a split simulator $S = \{S_1^H, \ldots, S_n^H, S^L\}$ such that, for any t-coalition split adversary $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ and any environment $\mathcal{Z}$ we have $EXEC_{\rho[\phi\to\pi],\mathcal{A},\mathcal{Z}} \approx EXEC_{\rho,(S\|\mathcal{A}),\mathcal{Z}}$.

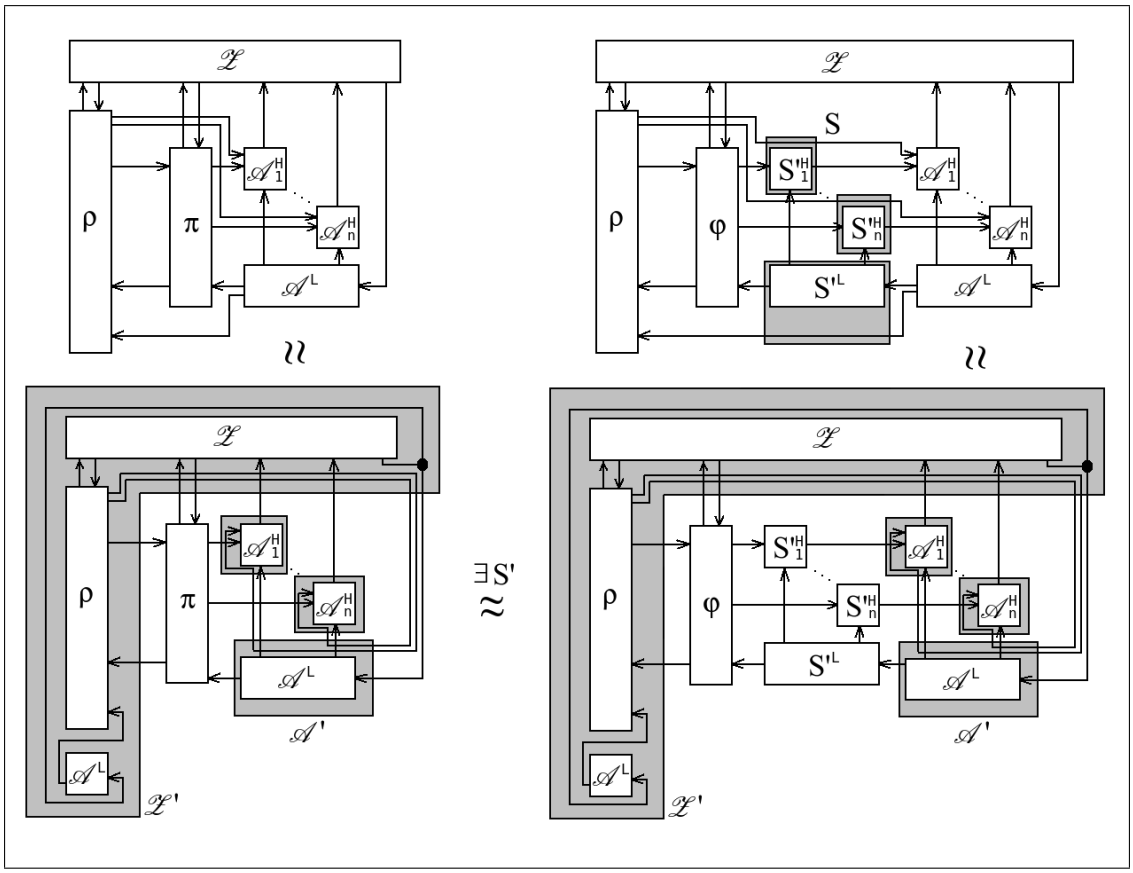

Fig. 7: WCP composition theorem

Since π t-WCP-emulates ϕ, there exists $S' = \{S'^H_1, \ldots, S'^H_n, S'^L\}$ such that for any t-coalition split adversary $\mathcal{A}'$ and any environment $\mathcal{Z}'$ we have $EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} \approx EXEC_{\phi,(S'\|\mathcal{A}'),\mathcal{Z}'}$. We define the simulator $S = \{S^H_1, \ldots, S^H_n, S^L\}$ for the protocol $\rho[\phi \mapsto \pi]$ in such a way that S^H_i acts as S'^H_i when receiving messages from ϕ, and S^L acts as S'^L when receiving messages for ϕ from $\mathcal{A}^L$, and just forwarding all the other messages. We want to show that S is a suitable simulator for $\rho[\phi \to \pi]$.

Assume by contradiction that S is not suitable. That is, there exist an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ such that $|EXEC_{\rho[\phi\to\pi],\mathcal{A},\mathcal{Z}} - EXEC_{\rho,(S\|\mathcal{A}),\mathcal{Z}}| \geq \varepsilon$. We can use these $\mathcal{A}$ and $\mathcal{Z}$ to construct $\mathcal{A}'$ and $\mathcal{Z}'$ to break $EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} \approx EXEC_{\phi,(S'\|\mathcal{A}'),\mathcal{Z}'}$. First, we construct $\mathcal{A}'^H_i$ from $\mathcal{A}^H_i$:

- The adversary $\mathcal{A}^H_i$ cannot see ρ and $\mathcal{Z}$ together as a larger environment $\mathcal{Z}'$ since the messages of ρ are received by $\mathcal{A}^H_i$, while the messages of $\mathcal{Z}$ are received by $\mathcal{A}^L$. Our construction of $\mathcal{Z}'$ thus has to be enhanced by some kind of forwarding functionality $\mathcal{D}$ forwarding messages from $\mathcal{A}^H_i$ to $\mathcal{A}^L$. Otherwise, let us take $\mathcal{A}'^H_i = \mathcal{A}^H_i$.
- Since $\mathcal{A}^L$ gets its inputs only from $\mathcal{Z}$, we may create two copies of $\mathcal{A}^L$, leaving one connected to ρ and $\mathcal{Z}$ only, and the other one to π and $\mathcal{Z}$ only. In order to ensure that $\mathcal{Z}$ sends the same data to both copies, we will again put a forwarding functionality $\mathcal{D}$ between $\mathcal{Z}$ and the two copies of $\mathcal{A}^L$. Let $\mathcal{A}'^L$ be exactly the same as $\mathcal{A}^L$, except that it does not output anything to ρ.

Now we define ρ, $\mathcal{Z}$, $\mathcal{A}^L$, and all the necessary forwarding functionalities $\mathcal{D}$ together as a new environment $\mathcal{Z}'$. We get that $|EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} - EXEC_{\phi,(S'\|\mathcal{A}'),\mathcal{Z}'}| \geq \varepsilon$, which is a contradiction with the assumption that π t-WCP emulates ϕ. □

The quantities used in the proof are depicted on Fig. 7.

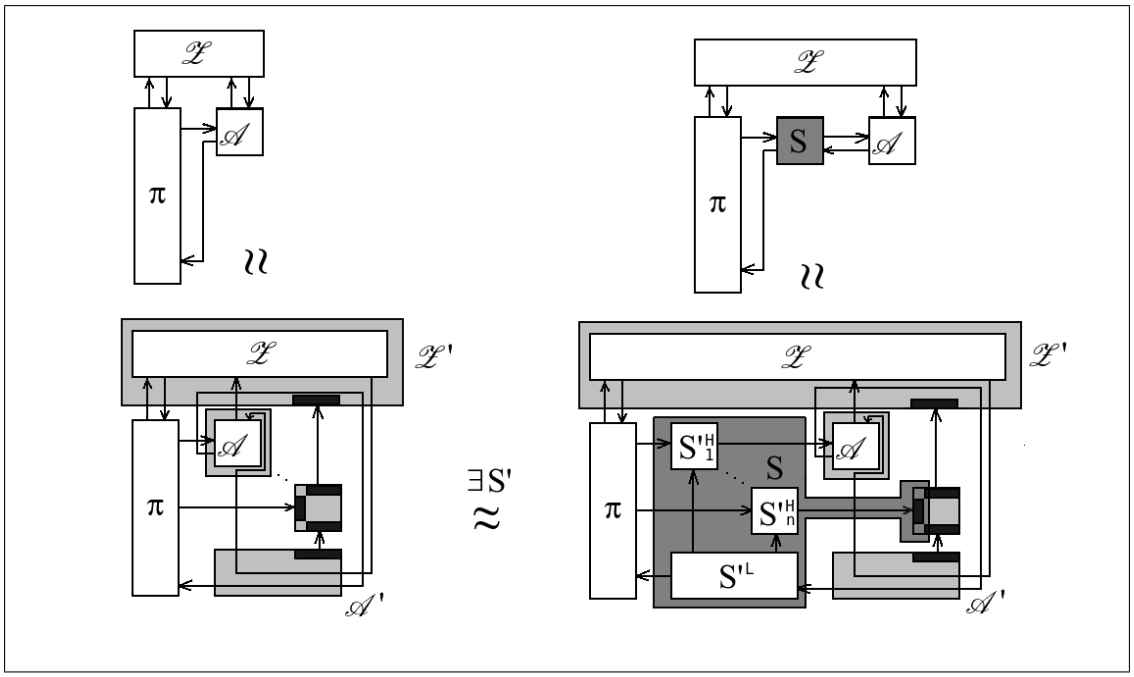

Fig. 8: WCP emulation implies UC emulation

A.3 WCP emulation implies UC emulation

Suppose that π t-WCP emulates ϕ. Let $\{S_1^H, \ldots, S_n^H, S^L\}$ be the set of simulators that exists due to t-WCP emulation. We show that $S = (S_1^H, S^L)$ is a suitable simulator for the UC security. Assume by contradiction that there exist $\mathcal{A}$ and $\mathcal{Z}$ such that $|EXEC_{\pi,\mathcal{A},\mathcal{Z}} - EXEC_{\phi,(S_1\|\mathcal{A}),\mathcal{Z}}| \geq \varepsilon$. Let $\blacksquare$ be a black hole ITM that just consumes all the inputs and does not output anything. If we take $S = \{S_1^H, (S_2^H\|\blacksquare), \ldots, (S_n^H\|\blacksquare), S^L\}$ (where S_i^H for $i \neq 1$ can be connected to ϕ since all their output will be lost in $\blacksquare$ anyway), then still $|EXEC_{\pi,\mathcal{A},\mathcal{Z}} - EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}| \geq \varepsilon$ since all $(S_j^H\|\blacksquare)$ for $j \neq 1$ are isolated from $\mathcal{A}$.

We now define $\mathcal{A}' = \{\mathcal{A}_1'^H, \ldots, \mathcal{A}_n'^H, \mathcal{A}'^L\}$ where $\mathcal{A}_1'^H = \mathcal{A}$, and $\mathcal{A}_j'^H = \blacksquare$ for $j \neq 1$. The only difference between $\mathcal{A}$ and $\mathcal{A}_1^H$ is that $\mathcal{A}_1^H$ receives inputs only from π and sends outputs only to $\mathcal{Z}$. Hence we extend $\mathcal{Z}$ to a new environment $\mathcal{Z}'$ by adding an ITM $\mathcal{D}$ that forwards messages from $\mathcal{A}_1^H$ to $\mathcal{A}^L$, and define $\mathcal{A}^L$ to be a functionality that just forwards messages coming from $\mathcal{Z}$ to $\mathcal{A}_1^H$ or π (we may assume that these messages are labelled, so $\mathcal{A}^L$ can distinguish these two types). Since $\blacksquare$ does not output anything, we may connect $(S_j^H\|\blacksquare)$ directly to $A_j'^H = \blacksquare$ without changing anything, getting $|EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} - EXEC_{\phi,(S\|\mathcal{A}'),\mathcal{Z}'}| \geq \varepsilon$.

Simple counterexamples of the other implication direction are given in Sec. 5. □

The quantities used in the proof are depicted on Fig. 8.

A.4 CP emulation implies WCP emulation

The CP emulation gives a simulator $S' = \{S_1', \ldots, S_n'\}$ such that, for any $\mathcal{Z}$ and for the any set of adversaries $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, $EXEC_{\pi,\mathcal{A},\mathcal{Z}} \approx EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}$.

Let $\mathbf{C} = \{C(1), \ldots, C(n)\}$ be the set of coalitions. Take $S_j^H = \{S_i' \mid i \in C(j)\}$ for all $C(j) \in \mathbf{C}$.

Suppose that this choice of S is not good, and we have found $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ and $\mathcal{Z}$ that can break t-WCP, i.e $|EXEC_{\pi,\mathcal{A},\mathcal{Z}} - EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}| \geq \varepsilon$. Let a CP adversary $\mathcal{A}'$ be defined as $\mathcal{A}_i' = \mathcal{A}_j^H$ for all $i \in C(j)$. Let $\mathcal{Z}'$ contain a forwarding functionality $\mathcal{D}$ that allows all the adversaries $\mathcal{A}_i'$ for $i \in [n]$ to communicate with each other. We show that $\mathcal{A}'$ and $\mathcal{Z}'$ may be used to break CP.

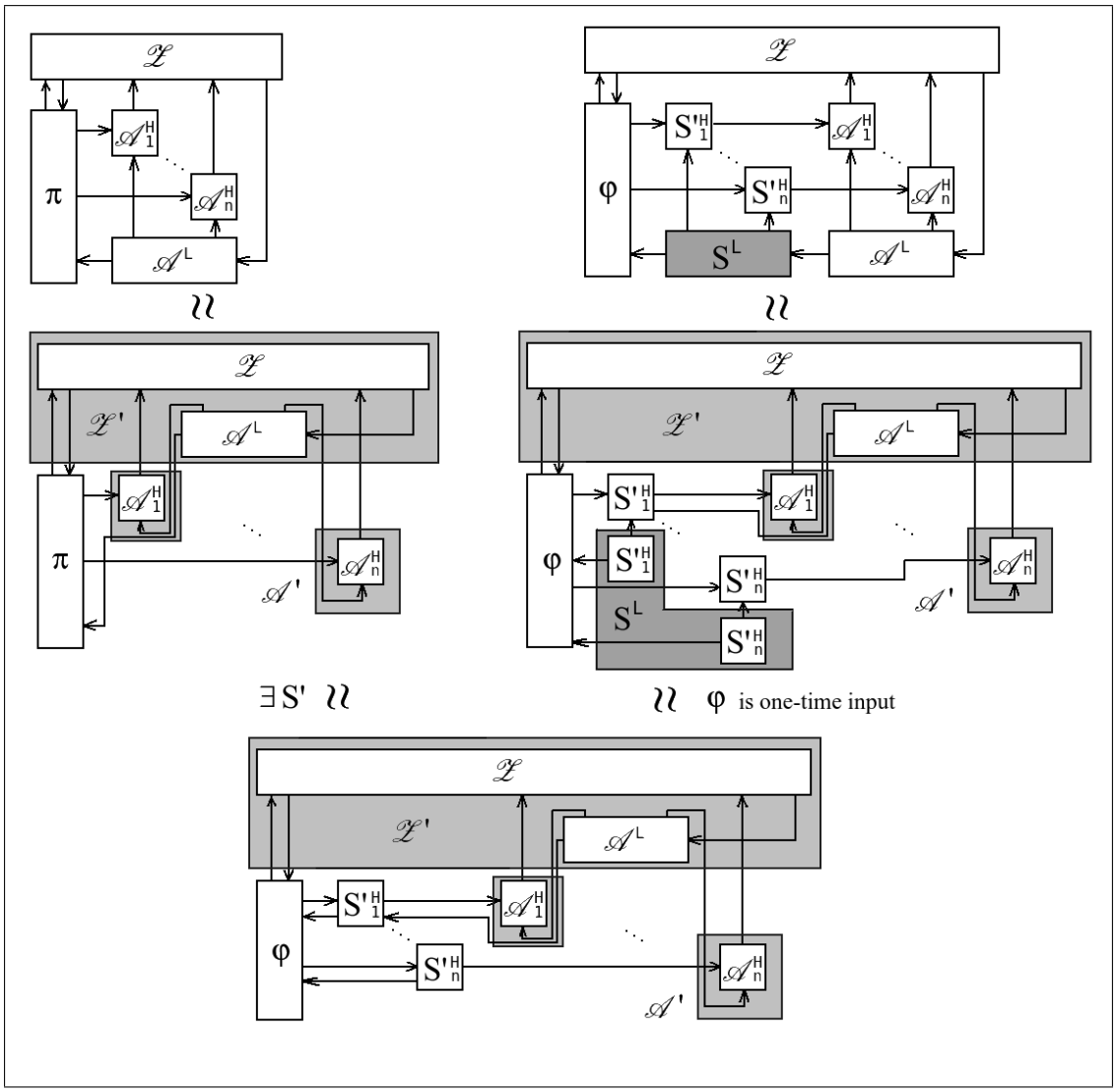

Fig. 9: CP emulation implies WCP emulation

- Since S^L is allowed to deliver everything to S_j^H, S_j^H is as powerful as S'_j and hence all $\{S'_i \mid i \in C(j)\}$. The distribution on the ports exiting S_j^H and entering $\mathcal{A}_j^H$ is the same as the distribution of ports exiting S'_i and entering $\mathcal{A}'_i$ for $i \in C(j)$ in the CP model.
- Although S^L is not allowed to get anything from S_j^H, since ϕ is one-time input, S^L should provide inputs to ϕ before anything is output by ϕ to S_j^H, and hence S_j^H would not tell S^L anything useful anyway, so S^L is able to simulate the behaviour of all S'_i for $i \in [n]$.

Since $\mathcal{A}'_i$ may choose to behave in exactly the same way as $\mathcal{A}_i^H$ and $\mathcal{A}^L$ did (this is possible since $\mathcal{Z}'$ enables communication of colluding parties), and $\mathcal{Z}'$ as $\mathcal{Z}$ did, then, since S produces exactly the same outputs as S' would, we have $|EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} \approx EXEC_{\phi,(S'\|\mathcal{A}'),\mathcal{Z}'}| \geq \varepsilon$.

Simple counterexamples of the other implication direction are given in Sec. 5. □

The quantities used in the proof are depicted on Fig. 9.

A.5 Proof that UC is Equivalent to WCP in Passive Adversary Case

Let $\mathcal{A}$ be the adversary of the UC model. Let S' be the simulator that translates the messages between $\mathcal{A}$ and ϕ to simulate π. We show that a suitable choice for $S = \{S_1^H, \ldots, S_n^H, S^L\}$ for t-WCP emulation is just taking $S^L = S_i^H = S'$ for all i.

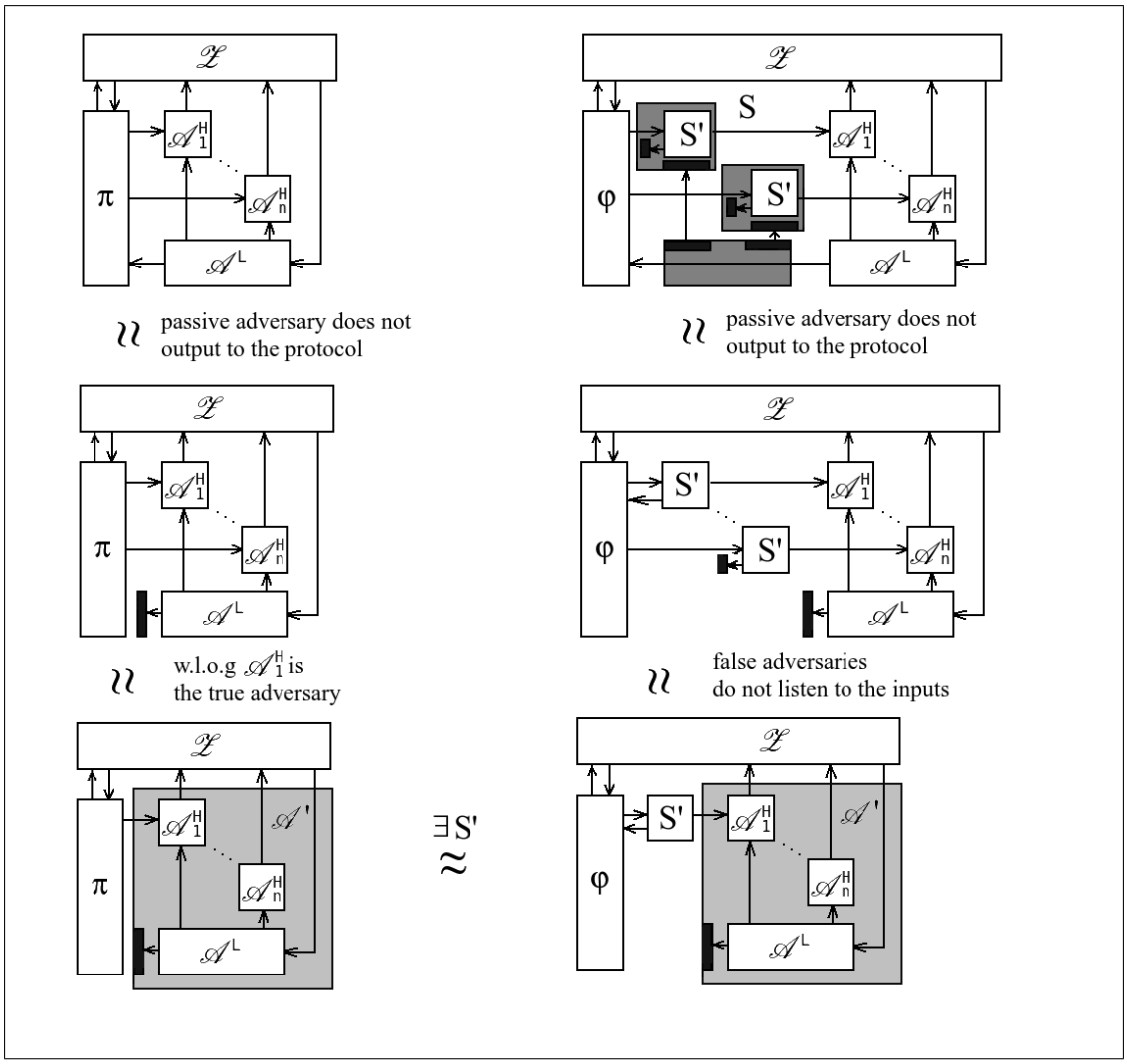

Fig. 10: UC $\approx$ WCP for a passive adversary

Suppose that S is a bad choice and there exist $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ and $\mathcal{Z}$ such that $|EXEC_{\pi,\mathcal{A},\mathcal{Z}} \approx EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}| \geq \varepsilon$. Let $\mathcal{A}_i^H$ be the true adversary.

We show that $\mathcal{A}' = \{\mathcal{A}_i^H, (S_j^H \| \mathcal{A}_j^H)_{j\in[n]\setminus i}, (S^L \| \mathcal{A}^L)\}$ and $\mathcal{Z}' = \mathcal{Z}$ will break UC security.

Since the adversary is passive, π and ϕ do not expect any inputs from it anyway, and so there is no difference whether we connect S^L with the protocol or not. For the false adversaries S_j^H, there is no difference whether S_j^H is connected to the protocol or not since $\mathcal{A}_j^H$ drops all inputs coming from S_j^H anyway. We get $|EXEC_{\pi,\mathcal{A}',\mathcal{Z}'} - EXEC_{\phi,(S'\|\mathcal{A}'),\mathcal{Z}'}| \geq \varepsilon$. □

The quantities used in the proof are depicted on Fig. 10, where ■ (a black rectangle of various shapes) denotes ITMs that do not input/output anything, and they are used to substitute missing ports which formally cannot just be removed.

B Transformations between Functionalities in WCP and UC, CP, LUC

In this section we formally define how an ideal functionality $\mathcal{F}^{WCP}$ defined in WCP model can be mapped to/from the corresponding functionality $\mathcal{F}^{UC}$, $\mathcal{F}^{CP}$, $\mathcal{F}^{LUC}$. We use the notation $\downarrow_Y^X$ for the transformation from a functionality of model X to the functionality of model Y.

B.1 WCP and UC

The transformations between WCP and UC are similar to the merger and splitter transformations between UC and LUC defined in [4]. We assume that there is a mapping $c(\cdot)$ such that $C = c(i)$ is the set of all party indices that belong to the same coalition to which i belongs. Let $\{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ be the t-coalition split adversary.

Definition 8 (WCP to UC). *Let $\mathcal{F}^{WCP}$ be a functionality in the WCP model. The functionality $\mathcal{F}^{UC} =\downarrow_{UC}^{WCP} (\mathcal{F}^{WCP})$ behaves the same as $\mathcal{F}^{WCP}$ with the following differences in the interface:*

1. *Upon receiving an input (in) from the external adversary $\mathcal{A}$, $\mathcal{F}^{UC}$ behaves as $\mathcal{F}^{WCP}$ would upon receiving input (in) from the adversary $\mathcal{A}^L$.*
2. *Whenever $\mathcal{F}^{WCP}$ generates an output (out) to the adversary $\mathcal{A}_k^H$, $\mathcal{F}^{UC}$ gives (out, k) to the external adversary $\mathcal{A}$.*

Definition 9 (UC to WCP). *Let $\mathcal{F}^{UC}$ be a functionality in the UC model. The functionality $\mathcal{F}^{WCP} =\downarrow_{WCP}^{UC} (\mathcal{F}^{UC})$ behaves the same as $\mathcal{F}^{UC}$ with the following differences in the interface:*

1. *Upon receiving an input (in) from the external adversary $\mathcal{A}^L$, $\mathcal{F}^{WCP}$ behaves as $\mathcal{F}^{UC}$ upon receiving input (in) from $\mathcal{A}$.*
2. *Whenever $\mathcal{F}^{UC}$ generates an output (out) to $\mathcal{A}$, $\mathcal{F}^{WCP}$ gives (out) to all external adversaries $\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H$.*

B.2 WCP and LUC

Since LUC contains more adversaries than WCP, the transformations are still related to splitting and merging. The idea is to first merge all the adversaries $\mathcal{A}_{i,j}$ into one adversary $\mathcal{A}_i$, since $\mathcal{A}_{i,j}$ models the communication between P_i and P_j as seen by P_i. These adversaries are in turn merged into coalitions.

Definition 10 (LUC to WCP). *Let $\mathcal{F}^{LUC}$ be a functionality in the LUC model. The functionality $\mathcal{F}^{WCP} =\downarrow_{WCP}^{LUC} (\mathcal{F}^{LUC})$ behaves the same as $\mathcal{F}^{LUC}$ with the following differences in the interface:*

1. *Upon receiving an input $(in, (i, j))$ from the external adversary $\mathcal{A}^L$, $\mathcal{F}^{WCP}$ behaves as $\mathcal{F}^{LUC}$ would upon receiving input (in) from the adversary $\mathcal{A}_{i,j}$.*
2. *Whenever $\mathcal{F}^{LUC}$ generates an output (out) to the adversary $\mathcal{A}_{i,j}$, $\mathcal{F}^{WCP}$ gives $(out, (i, j))$ to the external adversary $\mathcal{A}_{c(i)}^H$.*

Definition 11 (WCP to LUC). *Let $\mathcal{F}^{WCP}$ be a functionality in the WCP model. The functionality $\mathcal{F}^{LUC} =\downarrow_{LUC}^{WCP} (\mathcal{F}^{WCP})$ behaves the same as $\mathcal{F}^{UC}$ with the following differences in the interface:*

1. *Upon receiving an input (in) from some external adversary $\mathcal{A}_{i,j}$, $\mathcal{F}^{LUC}$ behaves as $\mathcal{F}^{WCP}$ upon receiving input (in) from $\mathcal{A}^L$.*
2. *Whenever $\mathcal{F}^{WCP}$ generates output (out) to $\mathcal{A}_k^H$, $\mathcal{F}^{LUC}$ gives (out) to the external adversaries $\mathcal{A}_{i,j}$ $(\forall\ i \in C(k),\ j \in [n],\ j \neq i)$.*
3. *Upon receiving input $(Deliver, m, (\ell, k))$ from some external adversary $\mathcal{A}_{i,j}$, $\mathcal{F}^{LUC}$ outputs $(Delivered, m, (i, j)$ to $\mathcal{A}_{\ell,k}$.*

B.3 WCP and CP

If each party is corrupted by its own adversary, then the definitions of $\mathcal{F}^{CP}$ and $\mathcal{F}^{WCP}$ are exactly the same, and all the difference between CP and WCP is now coming from different adversary definitions. In general, some of the adversaries in WCP model can be merged into coalitions, and that is what where merger and splitter functionalities make the difference.

Definition 12 (CP to WCP). *Let $\mathcal{F}^{CP}$ be a functionality in the CP model. The functionality $\mathcal{F}^{WCP} =\downarrow_{WCP}^{CP} (\mathcal{F}^{CP})$ behaves the same as $\mathcal{F}^{CP}$ with the following differences in the interface:*

1. *Upon receiving an input (in, i) from the external adversary $\mathcal{A}^L$, $\mathcal{F}^{WCP}$ behaves as $\mathcal{F}^{CP}$ would upon receiving input (in) from the adversary $\mathcal{A}_i$.*
2. *Whenever $\mathcal{F}^{CP}$ generates an output (out) to the adversary $\mathcal{A}_i$, $\mathcal{F}^{WCP}$ gives (out) to the external adversary $\mathcal{A}^H_{c(i)}$.*

Definition 13 (WCP to CP). *Let $\mathcal{F}^{WCP}$ be a functionality in the WCP model. The functionality $\mathcal{F}^{CP} =\downarrow_{CP}^{WCP} (\mathcal{F}^{WCP})$ behaves the same as $\mathcal{F}^{WCP}$ with the following differences in the interface:*

1. *Upon receiving an input (in) from some external adversary $\mathcal{A}_i$, $\mathcal{F}^{CP}$ behaves as $\mathcal{F}^{WCP}$ upon receiving input in from $\mathcal{A}^L$.*
2. *Whenever $\mathcal{F}^{WCP}$ generates output (out) to $\mathcal{A}^H_k$, $\mathcal{F}^{CP}$ gives (out) to all the external adversaries $\mathcal{A}_j$ such that $j \in C(k)$.*

C WCP Applicability: Bad Sharing

This attack is somewhat similar to the bad key attack of Sec. 5. However, now the shared value itself remains the same, but it will be shared in such a bad way, so that a subset of parties smaller than the official threshold will be able to reconstruct the secret. This attack would be more interesting if there were several larger adversarial coalitions. We present its particular case where a secret gets leaked entirely to some other party.

Ideal. We take the same ideal functionality $\mathcal{F}_0$ of Sec. 5.

Protocol 1. In the protocol π_1, a subset $\mathcal{I}$ of t parties and a subset $\mathcal{J}$ of $(t+1)$ parties ($i \notin \mathcal{I}$, $\mathcal{I} \cap \mathcal{J} = \emptyset$) are fixed. First, each M_j for $j \in \mathcal{I}$ sends a share s_j to M_i. M_i just generates the last $(t+1)$-th share in such a way that the result would be s, and distributes these shares amongst the $(t+1)$ parties of $\mathcal{J}$. If any party refuses to send its message, the protocol aborts.

Protocol 2. The protocol π_2 is analogous to π_1 with the only difference that this time M_i generates all the shares $(\{s_\ell |\ \ell \in \mathcal{I}\}$ by itself from uniform distribution.

- **UC** Assuming that the total number of corrupted parties is at most t, both π_1 and π_2 UC-realize $\mathcal{F}_0$. If M_i is corrupted, then S gets s from $\mathcal{F}_0$ and can simulate everything. Otherwise, the adversary may get up to t shares of s. If the number of corrupted parties is at least $t+1$, then both protocols are insecure since all the shares may leak.
- **WCP** The protocol π_2 does t-WCP-realize $\mathcal{F}_1$, but π_1 does not. In π_1, the adversary may set up to t shares to a value 0, so that the remaining share will be exactly the secret s that now leaks to some honest party that has not known s yet. At the same time, in π_2 an honest M_i generates all the shares from uniform distribution, and each simulator needs to simulate at most t shares that are distributed uniformly.
- **CP, LUC** Both π_1 and π_2 realize $\mathcal{F}_1$ with at most t parties. Similarly to t-WCP, in π_1 the adversary $\mathcal{A}(i)$ may use the bad sharing as a subliminal channel to leak s to some other party, but if $\mathcal{A}(i)$ falsifies some k shares, there are at most $t-k$ corrupted parties left to receive the other shares, and hence t corruptions are not sufficient for the attack. If the number of parties is at least $(t+1)$, then both π_1 and π_2 are insecure since even if no shares are falsified, in both protocols it may happen that the $t+1$ corrupted parties hold all the $(t+1)$ shares of s.

D The Analysis of a Multiparty Computation Protocol in WCP Model

In this section we analyze the multiparty computation protocol of [5]. This is a 3-party protocol with one malicious party, where the parties P_1, P_2, P_3 compute some function f on inputs x_1, x_2, x_3^*. A high-level description of the protocol is the following. The party P_3 shares its input as $x_3^* =$

$x_3 \oplus x_4$, and sends x_3 to P_1 and x_4 to P_2. The parties P_1 and P_2 agree on a common randomness r. They use it to construct a garbled circuit F that computes $f'(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3^*)$, and to garble the inputs x_1, x_2, x_3, x_4 to X_1, X_2, X_3, X_4. The parties send (F, X_1, X_2, X_3, X_4) to P_3 who evaluates the circuit. As far as P_3 does not know r, it cannot infer x_1 and x_2 from X_1 and X_2.

The authors of [5] mention that the security would be indeed broken if a malicious P_1 sends r to P_3, allowing it to extract x_2 from X_2. This attack is not covered by UC, and indeed we may assume that an honest P_3 will not communicate with P_1 using any side-channels. That attack would also not be noticed by 1-WCP due to the use of side-channels. However, depending on how r is generated, P_1 may perform this attack quietly, without using any side-channels.

Suppose that r is generated by P_1 alone, and P_1 just delivers this r to P_2. In UC model, P_1 has no reason to choose a bad r since it does not help P_1 to gain any information anyway. Moreover, a bad r may leak P_1's own secret. However, P_1 may still sacrifice its own input secrecy and intentionally choose an r that produces low-entropy garbled inputs X_1 and X_2. If we look at the view of P_3 (that is not covered by UC if P_1 is corrupted), we see that it contains x_2 which is not supposed to be there.

This attack is detected in 1-WCP model. Let P_1 be corrupted by an active $\mathcal{A}_1^H$, and P_3 corrupted by a passive $\mathcal{A}_3^H$. Suppose that $\mathcal{A}_3^H$ is the true adversary that just forwards all its data to $\mathcal{Z}$. At some moment, $\mathcal{A}^L$ chooses a bad randomness r that will be delivered to M_2 that uses it to produce a low-entropy X_2. Let $\mathcal{Z}$ be the environment expecting that, in the real protocol execution, the view of $\mathcal{A}_3^H$ will contain the X_2 which is related to x_2 in a certain way defined by the choice of r. This means that S_3^H has to simulate X_2 that is indeed related to x_2, but it does not know x_2 by default. Since the ideal functionality just computes $f(x_1, x_2, x_3^*)$ and outputs just x_1 to $\mathcal{A}_1^H$ and x_3^* to $\mathcal{A}_2^H$, there is no way for S_3^H to simulate x_2.

At the same time, if we assume that P_1 and P_2 mutually generate a good randomness r running some secure protocol, then $\mathcal{A}_1^H$ still knows the values r, $F(r)$, $X_1(r)$, and $X_3(r)$, but it does not generate r itself. The simulator S_3^H should simulate the same values. Since S_1^H and S_3^H are not allowed to communicate, S_3^H generates r' from the same distribution as r, and then computes itself $F(r')$, $X_1(r')$, $X_2(r')$, $X_3(r')$, and $X_4(r')$. Although in general $r \neq r'$, the definition of WCP ensures that only one of the views of $\mathcal{A}_1^H$ or $\mathcal{A}_3^H$ reaches the environment, but not both, and this inconsistency will not be noticed by $\mathcal{Z}$.

E The implementation of $\mathcal{F}_{tr}$ for WCP Model

We give a protocol that t-WCP realizes the communication functionality $\mathcal{F}_{tr}$ given in Sec. 6.2. Compared to [14, 15, 20], our implementation of $\mathcal{F}_{tr}$ ensures that the message that moves from M_i to M_j will not be seen by any M_k that colludes neither with M_i nor M_j. The modification includes a preprocessing phase that just generates a sufficient number of random masks and shares them amongst the n parties using any $(n, t+1)$ threshold sharing.

E.1 Real Functionality

Similarly to [14, 15, 20], we let the implementation $\pi_{transmit}$ of the message transmission functionality, consisting of machines $M_1, \ldots, M_n$ and a public key infrastructure, to have a "cheap mode" (which does not prevent the protocol from stopping), and the "expensive mode" for fallback. The "cheap mode" is the same as in [14, 15, 20]. The difference is that, taking into account multiple adversaries, it is not possible to later prove the authenticity of the messages sent in the "cheap mode" without leaking them to an honest party, and hence this mode is sufficient only for a covert adversary. In "expensive mode", $\pi_{transmit}$ works as follows.

- On input $(\mathsf{preprocess}, mid)$, the machines $M_{s(mid)}$, $M_{r(mid)}$ generate a random element $q(mid)$ by sending (mrnd, mid) to $\mathcal{F}_{pre}$ that generates random elements q_1 and q_2. All the parties M_k get from $\mathcal{F}_{pre}$ two shares $q_1^k(mid)$ and $q_2^k(mid)$, while $M_{s(mid)}$ and $M_{r(mid)}$ get all the shares $q_1^k(mid)$ and $q_2^k(mid)$ for all $k \in [n]$. The machines $M_{s(mid)}$ and $M_{r(mid)}$ take $q(mid) = q_1(mid) + q_2(mid)$.

– On input $(\mathsf{broadcast}, mid, m)$ the machine $M_{s(mid)}$ signs (mid, m) to obtain signature σ_s and sends (mid, m, σ_s) to each other machine.
– On input $(\mathsf{broadcast}, mid)$ each (uncorrupted) machine M_i waits for one round and then expects a message (mid, m, σ_s) from $M_{s(mid)}$, where σ_s is a valid signature from $P_s(mid)$ on (mid, m). If no message arrives or the signature is invalid, it signs and sends $(\mathsf{corrupt}, s(mid))$ to each other machine. Otherwise, it forwards the message (m, mid, σ_s) to each other machine. If any machine receives (mid, m, σ_s) and (mid, m', σ'_s) for $m \neq m'$, it sends $(mid, m, m', \sigma_s, \sigma'_s)$ to each other machine. If indeed $m \neq m'$ and the signatures are valid, the uncorrupted machine M_i receiving them outputs $(\mathsf{corrupt}, s(mid))$ to P_i. But if M_i receives only messages (mid, m, σ_s) with valid σ_s and no message (mid, m', σ'_s) with $m \neq m'$ and valid σ'_s, then it outputs (mid, m) to P_i.
– On input $(\mathsf{transmit}, mid, m)$ the machine $M_{s(mid)}$takes $m' = m \oplus q(mid)$, signs (mid, m') to obtain signature σ_s and sends (mid, m', σ_s) to each other machine.
– On input $(\mathsf{transmit}, mid)$, the machines act in exactly the same way as on input $(\mathsf{broadcast}, mid)$. Let m' be the broadcast value. The machine $M_{r(mid)}$ additionally computes $m = m' \oplus q(mid)$ and outputs (mid, m) to $P_{r(mid)}$.
– On input (commit, mid), we assume that the message (mid, m') has been transmitted from $s(mid)$ to $r(mid)$, and $m' = m \oplus q(mid)$ broadcast to each party. We present an implementation for the particular case where $\oplus$ is not a general masking operation, but a more concrete addition operation in a finite field $\mathbb{F}$, and m, $q(mid)$, m' are all some elements of $\mathbb{F}$. Each machine M_k computes the shares $m_1^k = m' - q_1^k$ and $m_2^k = m' - q_2^k$, obtaining the shares (m_1^k, m_2^k) of m.

Similarly to [14], it can be easily shown that this is a UC secure implementation under the restrictions that the inputs of honest parties are synchronized (this is ensured by the embedding protocol), and that even the messages of corrupted parties are delivered, and that their signatures are valid. We ensure correct delivery in exactly the same way as Damgård et al. [14]. We prove that our implementation is secure in the WCP model defined in Sec. 4.

E.2 Simulator for $\pi_{transmit}$

We have to show that there exists a simulator that can translate the between the messages the ideal functionality $\mathcal{F}_{tr}$ exchanges with the ideal adversary, and the messages the protocol $\pi_{transmit}$ exchanges with the real adversary over the network. We present the work of the simulator $S = \{S_1^H, \ldots, S_n^H, S^L\}$ for the t-coalition split adversary $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$. Let $c(i)$ be the index of the coalition that corrupts the machine M_i.

– On input $(\mathsf{preprocess}, mid)$, S first simulates $\mathcal{F}_{pre}$, and then simulates the transmission of old $\mathcal{F}_{tr}$ of [15]. At this point, $\mathcal{A}^H_{c(s(mid))}$ and $\mathcal{A}^H_{c(r(mid))}$ get all the shares of $q_1(mid)$ and $q_2(mid)$. Each other $\mathcal{A}^H_{c(k)}$ holds at most t shares $q_1^k(mid)$ and $q_2^k(mid)$.
– On input $(\mathsf{broadcast}, mid, m)$, first of all $S^H_{c(s(mid))}$ simulates the broadcast to $\mathcal{A}^H_{c(s(mid))}$. All the other simulators are waiting. On the first step, if $s(mid) \in C(1)$, and $\mathcal{A}^L$ decides that no (semi)honest party receives a properly signed value m_k, then S^L sends $(\mathsf{corrupt}, s(mid))$ to $\mathcal{F}_{tr}$. Otherwise, $S^H_{c(s(mid))}$ presents complaints from all the (semi)honest parties (S^L also knows which parties should complain), and if $\mathcal{A}^L$ adds a sufficient number of complaints from the malicious parties (so that there are at least $t+1$ complaints in total), then S^L also sends $(\mathsf{corrupt}, s(mid))$ to $\mathcal{F}_{tr}$. Otherwise, there are no sufficient votes against $s(mid)$, and hence the broadcast proceeds further. Now m'_k for the next round are chosen, and again $\mathcal{A}^L$ chooses m'_k for the malicious parties. Now S^L checks if there exists a signed $m'_k \neq m_k$ that $\mathcal{A}^L$ has planned for at least one (semi)honest M_k. If it is so, then S^L sends $(\mathsf{corrupt}, s(mid))$ to $\mathcal{F}_{tr}$ and $S^H_{c(s(mid))}$ simulates the same to $\mathcal{A}^H_{c(s(mid))}$. Otherwise, all m'_k and m_k are equal to the same value m, and each (semi)honest party has now received at least one instance of m that was received on the first round by some (semi)honest party, and hence the views of all (semi)honest parties are now consistent. If $s(mid) \in C(1)$, then S^L sends m to $\mathcal{F}_{tr}$.

All the other simulators S_i^H for $i \neq c(s(mid))$ have now received all possibly malicious m_k and m'_k from S^L, and also the message m that was finally broadcast by $\mathcal{F}_{tr}$, so they can simulate the entire broadcast in the same way as $S^H_{c(s(mid))}$ did.

– On input $(\mathsf{transmit}, mid, m)$ the machine $M_{s(mid)}$ computes $m' = m \oplus q(mid)$, and S simulates the broadcast of m'. Similarly to the broadcast case, $S^H_{c(s(mid))}$ first simulates the broadcast to $\mathcal{A}^H_{c(s(mid))}$, and S^L either sends $(\mathsf{corrupt}, s(mid))$ to $\mathcal{F}_{tr}$ or finds out that the same value m' has been accepted by all (semi)honest parties in the real functionality. Now the problem is that, if $s(mid)$ acts honestly, then S^L does not necessarily see m' at all, and this time m' cannot be broadcast using the ideal functionality $\mathcal{F}_{tr}$ since it accepts only $(\mathsf{transmit}, mid, m)$ or $(\mathsf{corrupt}, s(mid))$ on the current step. The only simulators that get m' are $S^H_{c(s(mid))}$ and $S^H_{c(r(mid))}$, and each other S_i^H has to generate m' by itself. Since all coalitions are of size at most t, and no other adversary except $\mathcal{A}^H_{c(s(mid))}$ and $\mathcal{A}^H_{c(r(mid))}$ knows $q(mid)$, the other simulators may just generate a random value m' assuming that this is $m \oplus q(mid)$.
– On input (commit, mid), S_i^H computes $m_1^k = m' - q_1^k$ and $m_2^k = m' - q_2^k$ for $k \in C(i)$. The problem is that, if $s(mid), r(mid) \notin C(i)$, then S_i^H has generated its own random m' during simulation of $\mathsf{transmit}$. Assuming that no coalition that does not include $s(mid)$ or $r(mid)$ has got any additional information about $q(mid)$ so far, m' still comes from the same distribution as $m \oplus q(mid)$ for the view $\mathcal{Z}$. Since each simulator other than $S^H_{c(s(mid))}$ and $S^H_{c(r(mid))}$ has generated up to t random shares (m_1^k, m_2^k) of m, only one adversary will output the shares to $\mathcal{Z}$, and there is no way for (m_1^k, m_2^k) to get to $\mathcal{Z}$ through the protocol for false adversaries, $\mathcal{Z}$ does not notice any inconsistencies.

F Verification functionality for WCP model

In F.1 we present the protocol π_{vmpc} that t-WCP realizes the functionality $\mathcal{F}_{vmpc}$ defined in Sec. 6.4 (Fig. 4). The implementation tightly follows [15], with a slight modification that makes the protocol secure in t-WCP model. We prove in F.2 that it is indeed so.

In our proofs, we assume that the computation takes place over an arithmetic circuit C_{ij}^{ℓ} over a finite field $\mathbb{F}$. Although [15] allows verification of circuits over several rings $\mathbb{Z}_{n_1}, \ldots, \mathbb{Z}_{n_K}$, we will have some problems with the WCP security on the last steps of verification, where a malicious prover is able to leak some information about his secret using a low-level channel only, without actually knowing these secret values. The problem of rings is that a product of a constant and a uniformly distributed value is not necessarily distributed uniformly.

F.1 The Protocol

The protocol π_{vmpc} implementing $\mathcal{F}_{vmpc}$ consists of n machines $M_1, \ldots, M_n$ doing the work of parties $P_1, \ldots, P_n$, and the functionalities $\mathcal{F}_{tr}$ of Sec. 6.2 and $\mathcal{F}_{pre}$ of App. G (the existence of t-WCP implementations of $\mathcal{F}_{pre}$ and $\mathcal{F}_{tr}$ are proven in App. G and App. E). The internal state of each M_i contains a bit-vector mlc_i of length n where M_i marks which other parties are acting maliciously. The goal of the prover is to prove its honestness to all the other parties that act as verifiers.

Randomness generation On input $(\mathsf{gen\_rnd}, sid, i)$, the machines need to generate randomness $\boldsymbol{r}_i$ for the party P_i. For this, they send $(\mathsf{rnd}, (sid, i))$ to $\mathcal{F}_{pre}$. As the result, $\boldsymbol{r}_i$ is shared to $(\boldsymbol{r}_{1i}^k, \boldsymbol{r}_{2i}^k)_{k\in[n]}$, where $\boldsymbol{r}_i = \mathsf{declassify}((\boldsymbol{r}_{1i}^k)_{k\in[n]}) + \mathsf{declassify}((\boldsymbol{r}_{2i}^k)_{k\in[n]})$, and the share $(\boldsymbol{r}_{1i}^k, \boldsymbol{r}_{2i}^k)$ is issued to M_k.

In addition, the machines send $(\mathsf{prec}, (sid, i))$ to $\mathcal{F}_{pre}$ to generate a sufficient number of multiplication triples that will be needed in the verification. Each machine M_k gets the share $(\boldsymbol{s}_{1i}^k, \boldsymbol{s}_{2i}^k)$ of the vector of triples $\boldsymbol{s}_i = \mathsf{declassify}((\boldsymbol{s}_{1i}^k)_{k\in[n]}) + \mathsf{declassify}((\boldsymbol{s}_{2i}^k)_{k\in[n]})$.

If $\perp$ comes from $\mathcal{F}_{pre}$, the preprocessing fails, and each machine outputs $\perp$.

If $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{pre}$, each M_k writes $mlc_k[j] := 1$ and goes to the accusation phase.

Input commitment On input $(\mathsf{commit\_input}, sid, i, \boldsymbol{x}_i)$, the machine M_i sends the message $(\mathsf{commit}, (sid, i), \boldsymbol{x}_i)$ to $\mathcal{F}_{pre}$ to commit $\boldsymbol{x}_i$ as shares $\boldsymbol{x}_i^k = (\boldsymbol{x}_{1i}^k, \boldsymbol{x}_{2i}^k)$, where $\boldsymbol{x}_i = \mathsf{declassify}((\boldsymbol{x}_{1i}^k)_{k\in[n]}) + \mathsf{declassify}((\boldsymbol{x}_{2i}^k)_{k\in[n]})$.

Public values: The prover M_i constructs the vector $\hat{\boldsymbol{s}}_i$ containing the values that have to be published, as in [15]. M_i sends $(\mathsf{broadcast}, (\mathsf{public}, i), \hat{\boldsymbol{s}}_i)$ to $\mathcal{F}_{tr}$.
Local computation: After receiving all the messages $((\mathsf{public}, i), \hat{\boldsymbol{s}}_i)$ from $\mathcal{F}_{tr}$, each verifying party M_k now holds the shares $\boldsymbol{p}^{\ell k}_{1ij} = (1^k \| \boldsymbol{v}^{\ell k}_{1ij} \| \boldsymbol{s}^k_{1i})$ and $\boldsymbol{p}^{\ell k}_{2ij} = (0^k \| \boldsymbol{v}^{\ell k}_{2ij} \| \boldsymbol{s}^k_{2i})$, so $\boldsymbol{p}^{\ell k}_{1ij} + \boldsymbol{p}^{\ell k}_{2ij} = \boldsymbol{p}^{\ell k}_{ij} = (1^k \| \boldsymbol{v}^{\ell k}_{ij} \| \boldsymbol{s}^k_i)$. Each party M_k computes and publishes $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij}$ and $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$. Everyone computes the shares $\boldsymbol{d}^{\ell k}_{ij} = A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij} + A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$ and checks if $\mathsf{declassify}(\boldsymbol{d}^{\ell k}_{ij}) = \boldsymbol{0}$.
Complaints and final verification: The prover M_i knows how a correct verification should proceed and, hence, it may compute the values $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij}$ and $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$ itself. If some of them is wrong, M_i may argue and open one of the $\boldsymbol{p}^{\ell k}_{1ij}$ and $\boldsymbol{p}^{\ell k}_{2ij}$, for which the computation was wrong. All the honest parties may now repeat the computation on these shares and compare the result. If M_k was guilty, then M_i is allowed to commit its own $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{bij}$ also for the remaining $\boldsymbol{p}^{\ell k}_{bij}$ that was not opened. If the shares $\boldsymbol{d}^{\ell k}_{ij}$ finally correspond to $\boldsymbol{0}$, then the proof of M_i for C^ℓ_{ij} is accepted. Otherwise, each honest party M_h now immediately sets $mlc_h[i] := 1$.

Fig. 11: Verification phase of the real functionality

If $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{pre}$, each M_k writes $mlc_k[j] := 1$ and goes to the accusation phase.

Message commitment. On input $(\mathsf{commit\_msg}, sid, i, j, \ell, \boldsymbol{m}^\ell_{ij})$ the machine M_i sends the message $(\mathsf{mcommit}, (sid, i, j, \ell), \boldsymbol{m}^\ell_{ij})$ to $\mathcal{F}_{pre}$. As the result, $\boldsymbol{m}^\ell_{ij}$ is now shared as $\boldsymbol{m}^{\ell k}_{ij} = (\boldsymbol{m}^{\ell k}_{1ij}, \boldsymbol{m}^{\ell k}_{2ij})$, where $\boldsymbol{m}^\ell_{ij} = \mathsf{declassify}((\boldsymbol{m}^{\ell k}_{1ij})_{k\in[n]}) + \mathsf{declassify}((\boldsymbol{m}^{\ell k}_{2ij})_{k\in[n]})$, and $\boldsymbol{m}^\ell_{ij}$ may at be at some moment opened to M_j.

If $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{pre}$, each M_k writes $mlc_k[j] := 1$ and goes to the accusation phase.

Verification. On input $(\mathsf{verify}, C^\ell_{ij}, i, j, \ell, \boldsymbol{m}^\ell_{ij})$, the machines need to verify if $\boldsymbol{m}^\ell_{ij}$ is a valid output of the circuit C^ℓ_{ij}. Let $\boldsymbol{v}^\ell_{ij} = (\bar{\boldsymbol{x}}_i \| \boldsymbol{r}_i \| \bar{\boldsymbol{m}}_i)$ be the vector of inputs and outputs to the circuit C^ℓ_{ij} that M_i uses to compute the ℓ-th message to M_j, where $\bar{\boldsymbol{x}}_i$ and $\bar{\boldsymbol{m}}_i$ are all the inputs and messages received by M_i that have been committed so far. At this point, M_i has shared $\boldsymbol{v}^\ell_{ij}$ among all n parties. Let $\boldsymbol{v}^{\ell k}_{ij} = (\boldsymbol{v}^{\ell k}_{1ij}, \boldsymbol{v}^{\ell k}_{2ij})$ be the share of $\boldsymbol{v}^\ell_{ij}$ given to machine M_k.

Now the actual verification starts. An overview of this phase is given in Fig. 11.

Public values: The prover M_i publishes certain values $\hat{\boldsymbol{s}}_i$ that allow the verifiers to repeat the computation of the circuit locally on shares. This is done using the broadcast functionality of $\mathcal{F}_{tr}$. The details of construction of $\hat{\boldsymbol{s}}_i$ can be seen in [15], and they are computed locally by M_i. If the circuits are defined over $\mathbb{F}$, then $\hat{\boldsymbol{s}}_i$ consists of the values $x' = x - r_x$ and $y' = y - r_y$ revealed for the multiplication triple (r_x, r_y, r_{xy}) so that $x \cdot y = x'r_y + y'r_x + x'y' + r_{xy}$ could be computed locally on shares. This is sufficient to make any computation linear.

If $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{tr}$, each M_k writes $mlc_k[j] := 1$ and goes to the accusation phase.

Local computation: After all the values are committed and published, the verifiers continue with local computation. Their goal is to verify if $\mathsf{declassify}(A^\ell_{ij}\boldsymbol{p}^{\ell k}_{ij})_{k\in[n]} = \boldsymbol{0}$, where $A^\ell_{ij} = A(C^\ell_{ij}, \hat{\boldsymbol{s}}_i)$ is a matrix whose rows correspond to all the linear subcircuits of C^ℓ_{ij} and all the checks $x - x' - r_x = 0$ for the multiplication triples (this is needed to check if M_i has cheated with $\hat{\boldsymbol{s}}_i$), and $\boldsymbol{p}^{\ell k}_{ij} = (1^k \| \boldsymbol{v}^{\ell k}_{ij} \| \boldsymbol{s}^{\ell k}_j)$, where $\boldsymbol{s}^{\ell k}_i$ are the multiplication triple shares generated in the preprocessing phase by $\mathcal{F}_{pre}$. In [15], the parties just computed and opened $\boldsymbol{d}^{\ell k}_{ij} = A^\ell_{ij}\boldsymbol{p}^{\ell k}_{ij}$ which leak no more than $\boldsymbol{0}$ if M_i is honest. However, if some of these shares is invalid and the prover wants to complain about some $\boldsymbol{d}^{\ell k}_{ij}$, the value $\boldsymbol{p}^{\ell k}_{ij}$ should be revealed. This is not allowed in multiple adversary case. We propose that the parties instead work with a sum $\boldsymbol{p}^\ell_{ij} = \boldsymbol{p}^\ell_{1ij} + \boldsymbol{p}^\ell_{2ij}$, where the authenticity of both $\boldsymbol{p}^\ell_{1ij}$ and $\boldsymbol{p}^\ell_{2ij}$ can be proven. All the entries of $\boldsymbol{p}^\ell_{ij}$ collected so far are indeed represented as a sum of two vectors whose authenticity can be proven by $\mathcal{F}_{pre}$. The machine M_k computes and publishes $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij}$ and $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$ separately. Now everyone may compute by itself $\boldsymbol{d}^{\ell k}_{ij} = A^\ell_{ij}\boldsymbol{p}^{\ell k}_{ij} = A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij} + A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$ and check if $\mathsf{declassify}((\boldsymbol{d}^{\ell k}_{ij})_{k\in[n]}) = \boldsymbol{0}$. If some malicious verifier wants to accuse M_i, it may present something wrong instead of $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij}$ or $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$. In this case, M_i opens either $\boldsymbol{p}^{\ell k}_{1ij}$ or $\boldsymbol{p}^{\ell k}_{2ij}$ (if the error is in both, it still suffices to open just one to detect the cheating), and everyone may check $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{1ij}$ or $A^\ell_{ij}\boldsymbol{p}^{\ell k}_{2ij}$ and discover cheating without actually revealing $\boldsymbol{p}^{\ell k}_{ij}$. If M_k was indeed cheating, then M_i may present its own version of *both* $\boldsymbol{d}^{\ell k}_{1ij}$ and $\boldsymbol{d}^{\ell k}_{2ij}$.

If $(\mathsf{corrupt}, j)$ comes from $\mathcal{F}_{pre}$, each M_k writes $mlc_k[j] := 1$ and goes to the accusation phase. If $\mathsf{declassify}(A^{\ell}_{ij}\boldsymbol{p}^{\ell k}_{ij})_{k\in[n]} = \mathbf{0}$ even after M_i was allowed to complain, each M_k writes $mlc_k[i] := 1$ and goes to the accusation phase.

Accusation. Finally, after all the messages of the ℓ-th round have been verified, each uncorrupted machine M_h outputs to P_h the message $(\mathsf{blame}, sid, \ell, \mathcal{M})$ for $\mathcal{M} = \{k \mid mlc_h[k] = 1\}$. If $mlc_h[i] = 0$, the parties send $(\mathsf{open}, (sid, i, j, \ell))$ to $\mathcal{F}_{pre}$ which outputs the previously committed $\boldsymbol{m}^{\ell}_{ij}$ to M_j. M_j outputs $(\mathsf{approved}, C^{\ell}_{ij}, i, j, \ell, \boldsymbol{c}^{\ell}_{ij})$ to P_j, and each other (semi)honest M_h outputs $(\mathsf{approved}, C^{\ell}_{ij}, i, j, \ell)$ to P_h.

F.2 Simulator for $\mathcal{F}_{vmpc}$

In this section we prove that our protocol π_{vmpc} defined in App. F.1 is as secure as $\mathcal{F}_{vmpc}$ defined in Sec. 6.4. We have to show that there exists a simulator that can translate the between the messages $\mathcal{F}_{vmpc}$ exchanges with the ideal adversary, and the messages the protocol π_{vmpc} exchanges with the real adversary over the network. We present the work of the simulator $S = \{S^H_1, \ldots, S^H_n, S^L\}$ in phases, coinciding with the phases of real functionality that it simulates to the adversary $\mathcal{A} = \{\mathcal{A}^H_1, \ldots, \mathcal{A}^H_n, \mathcal{A}^L\}$. Let $c(i)$ denote the index of the adversarial coalition to which M_i belongs. Let 1 be the actively corrupted coalition, as agreed in Def. 3. Let $\mathcal{H} \subseteq [n]\backslash C(1)$ be some fixed subset of semihonest parties of size exactly $t + 1$. We will treat this set as a "commitment group" that holds the shares of the value that is considered "committed".

Throughout the protocol, the simulators will have to simulate $\mathcal{F}_{pre}$ several times. First of all, S^H_1 simulates its interaction with $\mathcal{F}_{pre}$ to check if it succeeds, or an error message $\perp$ (this may come only once in the randomness commitment phase where we allow aborting) or $(\mathsf{corrupt}, j)$ should be output. At any time when a message $(\mathsf{corrupt}, j)$ should be output from $\mathcal{F}_{pre}$, S^L sends $(\mathsf{corrupt}, j)$ to $\mathcal{F}_{vmpc}$ which collects all such messages sent on one round, defines $\mathcal{B}_0 = \{j \mid (\mathsf{corrupt}, j) \text{ has been output}\}$. Each simulator S^H_k also gets $(\mathsf{corrupt}, j)$ from S^L and simulates $(\mathsf{corrupt}, j)$ coming from $\mathcal{F}_{pre}$ for all $j \in \mathcal{B}_0$ in its own simulation. This position corresponds to the direct jump to the accusation phase in the real functionality.

Randomness Commitment. On input $(\mathsf{gen\_rnd}, sid, i, \boldsymbol{r}_i)$, the randomness $\boldsymbol{r}_i$ is generated and shared to $(\boldsymbol{r}^k_{1i}, \boldsymbol{r}^k_{2i})$ using $\mathcal{F}_{pre}$. Each simulator S^H_k simulates its own copy of $\mathcal{F}_{pre}$ to generate the shares of $\boldsymbol{r}_i$ without actually knowing $\boldsymbol{r}_i$ (only $S^H_{c(i)}$ gets $\boldsymbol{r}_i$). Since $\mathcal{F}_{pre}$ generates $\boldsymbol{r}_i$ internally, here we must assume that all S^H_k synchronize the randomness tape of their copies of $\mathcal{F}_{pre}$ in such a way that it uses the same $\boldsymbol{r}_i$ that $S^H_{c(i)}$ has obtained from $\mathcal{F}_{vmpc}$. By definition of $\mathcal{F}_{pre}$, running it leaks to $\mathcal{A}^H_k$ no more than the t shares of $\boldsymbol{r}_i$. Since each simulator S^H_k has to run its own instance of $\mathcal{F}_{pre}$, each $\mathcal{A}^H_k$ gets its own set of t shares ($\mathcal{A}^H_{c(i)}$ even gets all n shares), but since there is only one true adversary, the shares that reach $\mathcal{Z}$ are either the same $\boldsymbol{r}_i$ that $\mathcal{F}_{vmpc}$ chose (if $\mathcal{A}^H_{c(i)}$ is the true adversary), or are just t random uniformly distributed values.

Input commitments: On input $(\mathsf{commit\_input}, sid, i, \boldsymbol{x}_i)$, the vector $\boldsymbol{x}_i$ should be shared to $(\boldsymbol{x}^k_{1i}, \boldsymbol{x}^k_{2i})$ using $\mathcal{F}_{pre}$. Similarly to the randomness commitment, each simulator S^H_k has to run its own instance of $\mathcal{F}_{pre}$, each $\mathcal{A}^H_k$ (except $\mathcal{A}^H_{c(i)}$) gets its own set of t shares, but since there is only one true adversary, the t shares that reach $\mathcal{Z}$ are just random uniformly distributed values.

Message commitment: On input $(\mathsf{commit\_msg}, sid, i, j, \ell, \boldsymbol{m}^{\ell}_{ij})$, the vector $\boldsymbol{m}^{\ell}_{ij}$ should be shared to $(\boldsymbol{m}^{\ell k}_{1ij}, \boldsymbol{m}^{\ell k}_{2ij})$. At this point, the value $\boldsymbol{m}^{\ell}_{ij}$ is known only by $S^H_{c(i)}$. Again, the parties need to simulate sharing of $\boldsymbol{m}^{\ell}_{ij}$ to $(\boldsymbol{m}^{\ell k}_{1ij}, \boldsymbol{m}^{\ell k}_{2ij})$ using $\mathcal{F}_{pre}$. Similarly to the randomness and input commitment, each simulator S^H_k has to run its own instance of $\mathcal{F}_{pre}$, each $\mathcal{A}^H_k$ (except $\mathcal{A}^H_{c(i)}$) gets its own set of t shares, but since there is only one true adversary, the t shares that reach the environment are just random uniformly distributed values.

Verification.

On input $(\mathsf{verify}, C^{\ell}_{ij}, i, j, \ell, \boldsymbol{m}^{\ell}_{ij})$, the machines need to verify if $\boldsymbol{m}^{\ell}_{ij}$ is a valid output of the circuit C^{ℓ}_{ij}. At this point, each simulator S^H_k holds up to t shares $(\boldsymbol{v}^{\ell k}_{1ij}, \boldsymbol{v}^{\ell k}_{2ij})$ of $\boldsymbol{v}^{\ell}_{ij} = (\bar{\boldsymbol{x}}_i \| \boldsymbol{r}_i \| \bar{\boldsymbol{m}}_i)$. Let $\boldsymbol{v}^{\ell k}_{ij} = (\boldsymbol{v}^{\ell k}_{1ij}, \boldsymbol{v}^{\ell k}_{2ij})$

Public values: $\mathcal{A}^L$ comes up with the public values $\hat{\boldsymbol{s}}$ for malicious parties. Each simulator S^H_k has to think out $\hat{\boldsymbol{s}}$ for the honest parties. Similarly to [15], all these values look random to all the parties except the coalition that includes the prover M_i. Hence $S^H_{c(i)}$ already knows how

these values should look like, and it simulates the broadcast of actual $\hat{\boldsymbol{s}}$ to $\mathcal{A}^H_{c(i)}$. Each other simulator S^H_k generates $\hat{\boldsymbol{s}}$ from a uniform distribution.

The simulators now have different ideas of what $\hat{\boldsymbol{s}}$ of (semi)honest should be since they have generated $\hat{\boldsymbol{s}}$ themselves, but the inconsistency remains unnoticed by $\mathcal{Z}$ since only one $\mathcal{A}^H_k$ is the true adversary.

Local computation: Each party computes the circuits of the proving party M_i on its local shares $\boldsymbol{p}^{\ell k}_{ij} = (1^k \| \boldsymbol{v}^{\ell k}_{ij} \| \boldsymbol{s}^k_i)$. $\mathcal{A}^L$ decides on the values $\boldsymbol{d}^{*\ell k}_{ij}$ for $k \in C(1)$. Now the remaining shares $\boldsymbol{d}^{\ell k}_{ij}$ should be computed.

It is easy for $S^H_{c(i)}$ to simulate revealing $\boldsymbol{d}^{\ell k}_{ij}$ of (semi)honest M_k since it already knows $\boldsymbol{p}^{\ell k}_{ij}$ and $\hat{\boldsymbol{s}}$. The other simulators S^H_c for $c \neq c(i)$ need to wait a bit since they do not have enough data for simulation yet, and each of them holds at most t shares. They do not simulate publishing $\boldsymbol{d}^{\ell k}_{ij}$ before S^L tells them for which $\boldsymbol{d}^{\ell k}_{bij}$ there will be presented complaints.

Complaints and final verification: Let M_i be the prover. First of all, $S^H_{c(i)}$ simulates the complaint phase to $\mathcal{A}^H_{c(i)}$, and only then the other simulators may proceed with computation of $\boldsymbol{d}^{\ell k}_{1ij}$ and $\boldsymbol{d}^{\ell k}_{2ij}$.

- If $i \notin c(A)$, then if $\mathcal{A}^L$ decides that some verifier M_k for $k \in C(1)$ refuses to broadcast $\boldsymbol{d}^{\ell k}_{1ij}$ or $\boldsymbol{d}^{\ell k}_{2ij}$, or at least one of them is invalid, then $S^H_{c(i)}$ simulates M_i broadcasting a complaint. Now M_i has the right to reveal $\boldsymbol{p}^{\ell k}_{bij}$ for one of $b \in \{1, 2\}$ for which $\boldsymbol{d}^{\ell k}_{bij} = A(\hat{\boldsymbol{s}}_i)\boldsymbol{p}^{\ell k}_{bij}$ does not hold. In this way, $S^H_{c(i)}$ finally replaces all falsified $\boldsymbol{d}^{*\ell k}_{ij}$ with $\boldsymbol{d}^{\ell k}_{ij}$ that actually correspond to the shares of a (semi)honest M_i.
- If $i \in C(1)$, but $k \notin C(1)$, and $\mathcal{A}^L$ decides that M_i should complain against M_k, then $S^H_{c(i)}$ simulates the response of M_k to the complaint of M_i, and reveals $\boldsymbol{p}^{\ell k}_{bij}$ that M_k should hold. It finally simulates the judgement of M_i as the (semi)honest parties would do. All the pairs (k, b) for the opened $\boldsymbol{p}^{\ell k}_{bij}$ will be delivered by $\mathcal{A}^L$ through S^L, so all S^H_c know which $\boldsymbol{p}^{\ell k}_{bij}$ have to be opened.
- If both $i, k \in C(1)$, then $\mathcal{A}^L$ may still introduce malicious final shares for M_i through M_k. $\mathcal{A}^L$ may still wants to force M_i to complain about M_k, and the corresponding pairs (k, b) for the opened $\boldsymbol{p}^{\ell k}_{bij}$ will be delivered by $\mathcal{A}^L$ through S^L, so all S^H_c know which $\boldsymbol{p}^{\ell k}_{bij}$ have to be opened.

The other simulators S^H_c now finally have to simulate both the publishing of $(\boldsymbol{d}^{\ell k}_{1ij}, \boldsymbol{d}^{\ell k}_{2ij})$ and the further complaints. They wait until S^L tells them which parties of $c(A)$ were attempting to cheat with which values, and for which b the shares $\boldsymbol{p}^{\ell k}_{bij}$ have to be opened. If $i \notin C(1)$, then S^L might not know against which (k, b) the prover M_i has complained. We claim that in this case all the other simulators may choose precisely those (k, b) for which $\mathcal{A}^L$ has chosen some malicious share $\boldsymbol{p}^{*\ell k}_{bij} \neq \top$ (recall from Def. 3 that sending $\top$ is another possibility for $\mathcal{A}^L$ to take $\boldsymbol{p}^{*\ell k}_{bij} = \boldsymbol{p}^{\ell k}_{bij}$ honestly, according to the protocol rules). If it falsified both $\boldsymbol{p}^{*\ell k}_{bij}$ and $\boldsymbol{p}^{*\ell k}_{bij}$, choose any of them. Although $\boldsymbol{p}^{*\ell k}_{bij} \neq \top$ does not necessarily mean that $\boldsymbol{p}^{*\ell k}_{bij} \neq \boldsymbol{p}^{\ell k}_{bij}$, we claim that this choice is sufficient:

- If $\mathcal{A}^H_1$ is the true adversary, then $\mathcal{A}^H_c$ is false, and hence the possibly wrong simulation of S^H_c will never reach $\mathcal{Z}$ anyway.
- If $\mathcal{A}^H_1$ is a false adversary, then it does not know $\boldsymbol{p}^{\ell k}_{bij}$, and hence $\boldsymbol{p}^{*\ell k}_{bij} \neq \top$ will be invalid with overwhelming probability (depending on the size of $\mathbb{F}$). Hence M_i would indeed have a right to argue about $\boldsymbol{p}^{*\ell k}_{bij}$.

At this point, each S^H_c holds the following values:

- up to t shares $\boldsymbol{p}^{\ell k}_{1ij}$ and $\boldsymbol{p}^{\ell k}_{2ij}$ for $k \in C(c)$ that it has simulated to $\mathcal{A}^H_c$;
- the vector $\hat{\boldsymbol{s}}$ that it has simulated to $\mathcal{A}^H_c$;
- up to t additional shares $\boldsymbol{d^*}^{\ell k}_{1ij}$ and $\boldsymbol{d^*}^{\ell k}_{2ij}$ for $k \in C(1)$ that have come from S^L;
- up to t pairs (k, b) obtained via S^L from $\mathcal{A}^L$, that denote which shares $\boldsymbol{p}^{\ell k}_{bij}$ were opened.

In total, S_c^H already has up to $2t$ shares $(\boldsymbol{d}_{1ij}^{\ell k}, \boldsymbol{d}_{1ij}^{\ell k})$, which can be inconsistent since the shares of $k \in C(c)$ depend on previous shares $\boldsymbol{p}_{1ij}^{\ell k}$ and $\boldsymbol{p}_{2ij}^{\ell k}$ that have been generated by S_c^H itself, and for $k \in C(1)$ the shares have been chosen by $\mathcal{A}^L$. We will see that it is not a problem. S_c^H now generates its own version of the remaining shares $\boldsymbol{d}_{1ij}^{\ell k}$ and $\boldsymbol{d}_{2ij}^{\ell k}$ for $k \in [n] \setminus (C \cup C(1))$. Since it already knows for which b it will need to open $\boldsymbol{p}_{bij}^{\ell k}$, it generates $\boldsymbol{p}_{bij}^{\ell k}$ first and then takes $\boldsymbol{d}_{1ij}^{\ell k} = A(\hat{\boldsymbol{s}}_{ij}^{\ell}) \boldsymbol{p}_{bij}^{\ell k}$. The remaining $\boldsymbol{d}_{bij}^{\ell k}$ that will not be opened are generated randomly in such a way that they comprise the share $\mathbf{0}$ if $i \notin C(1)$ (without taking into account the shares $\boldsymbol{d^*}_{1ij}^{\ell k}$ and $\boldsymbol{d^*}_{2ij}^{\ell k}$ for $k \in C(1)$). This is always possible since S_c^H has sent up to t shares to $\mathcal{A}_c^H$ so far. If $i \in C(1)$, then it is possible that the value $\boldsymbol{d}_{ij}^{\ell}$ based on the shares committed so far is not $\mathbf{0}$. We claim that it in this case $\boldsymbol{d}_{ij}^{\ell}$ can be just a uniformly distributed value:

- If $\mathcal{A}_1^H$ is the true adversary, then $\mathcal{A}_c^H$ is false, and hence the possibly wrong simulation of S_c^H will never reach $\mathcal{Z}$ anyway.
- If $\mathcal{A}_1^H$ is a false adversary, then the only way for it to make $\boldsymbol{d}_{ij}^{\ell} \neq \mathbf{0}$ is to provide malicious $\boldsymbol{p}_{ij}^{*\ell}$ and $\hat{\boldsymbol{s}}_{ij}^{*}$. The value $\hat{\boldsymbol{s}}_{ij}^{*}$ is either uniformly distributed (if it was not falsified) or has already been shown to S_c^H. It is more difficult with $\boldsymbol{p}_{ij}^{*\ell}$ since it is possible that only some of its entries were falsified.
  By construction, the elements of $\boldsymbol{p}_{ij}^{\ell}$ are either some already well-committed constants or randomness (1, $\boldsymbol{r}_i$ and $\boldsymbol{s}_i$) that $\mathcal{A}^L$ cannot modify, or are chosen by M_i broadcasting $\boldsymbol{p'}_{ij}^{\ell} = (\boldsymbol{p}_{ij}^{\ell} - \boldsymbol{r})$, where $\boldsymbol{r}$ is generated by $\mathcal{F}_{pre}$ and will never reach $\mathcal{Z}$ if $\mathcal{A}_1^H$ is false. As the result, each entry of $\boldsymbol{p}_{ij}^{\ell}$ is either generated honestly, or looks like a uniformly distributed value to $\mathcal{Z}$.
  In this way, the entries of $A_{ij}^{\ell} \boldsymbol{p^*}_{ij}^{\ell} = \boldsymbol{d}_{ij}^{\ell}$ where the corresponding row of A_{ij}^{ℓ} does not use the corrupted entries of $\boldsymbol{p^*}_{ij}^{\ell}$ will be 0, and the entries that depend on the corrupted entries will be some random values. If we are computing values in a finite field, then these random entries are distributed uniformly since a uniformly distributed value multiplied by a non-zero element is still uniformly distributed. One problem is that $A_{ij}^{\ell} = A(\hat{\boldsymbol{s}})$, and $\hat{\boldsymbol{s}}$ may be important in deciding which entries of A_{ij}^{ℓ} are 0. However, if $\mathcal{A}_1^H$ is a false adversary, then an honestly generated $\hat{\boldsymbol{s}}$ will not reach $\mathcal{Z}$, and any malicious $\hat{\boldsymbol{s}}^*$ is already known by S_c^H.

If the proof of M_i has not failed yet, then all the shares $\boldsymbol{d}_{1ij}^{\ell k}$ and $\boldsymbol{d}_{2ij}^{\ell k}$ for all $k \in [n]$ are published, and are consistent with the view of a (semi)honest M_i since for any conflict with M_k it was allowed to publish its own $\boldsymbol{d}_{1ij}^{\ell k}$ and $\boldsymbol{d}_{2ij}^{\ell k}$. S_j checks if the published values are the shares of $\mathbf{0}$. If the check does not pass, S_j writes $mlc_h[i] := 1$ for each (semi)honest party P_h.

Accusation. $\mathcal{F}_{vmpc}$ computes all the outputs $\boldsymbol{m'}_{ij}^{\ell}$ of C_{ij}^{ℓ}. Let $\mathcal{M} = \{i \mid \exists j : \boldsymbol{m'}_{ij}^{\ell} \neq \boldsymbol{m}_{ij}^{\ell}\}$, and $\mathcal{B}_0 = \{i \mid (\mathsf{corrupt}, i)$ has come from $\mathcal{A}\mathsf{s}^L\}$. It is waiting from $\mathcal{A}\mathsf{s}^L$ for $(\mathsf{blame}, i, \mathcal{B}_i)$, such that $\mathcal{M} \subseteq \mathcal{B}_i \subseteq \mathcal{C}$. Let $\mathcal{B}'_i = \{j \mid mlc_i[j] = 1\}$. S defines $\mathcal{B}_i = \mathcal{B}_0 \cup \mathcal{B}'_i$.

First, we prove that $\mathcal{B}_i \subseteq C(1)$, i.e no (semi)honest party will be blamed.

1. For each $j \in \mathcal{B}_0$, a message $(\mathsf{corrupt}, j)$ has come from $\mathcal{F}_{pre}$ at some moment. Due to properties of $\mathcal{F}_{pre}$, no $(\mathsf{corrupt}, j)$ can be sent for $j \notin C(1)$. Hence $j \in C(1)$.
2. For each $j \in \mathcal{B}'_i$, the proof of M_j has not passed the final verification. For each $j \notin C(1)$, S has chosen $\boldsymbol{d}_{ji}^{\ell k}$ such that $\mathsf{declassify}((\boldsymbol{d}_{ji}^{\ell k})_{k \in [n]}) = \mathbf{0}$, and the check passes. Hence for a (semi)honest M_j the proof would always succeed, so $j \in C(1)$.

Secondly, we prove that $\mathcal{M} \subseteq \mathcal{B}_i$, i.e all malicious parties will be blamed.

1. The first component of $\mathcal{M}$ is $\mathcal{B}_0$ for which the message $(\mathsf{corrupt}, j)$ has been sent to $\mathcal{F}_{vmpc}$. Each simulator S_k^H has simulated $(\mathsf{corrupt}, j)$ to $\mathcal{A}_k^H$, so $\mathcal{B}_0 \subseteq \mathcal{B}_i$ for all $i \in [n]$.
2. The second component $\mathcal{M}'$ of $\mathcal{M}$ are the machines M_i for whom $\boldsymbol{m}_{ij}^{\ell} \neq C_{ij}^{\ell}(\bar{\boldsymbol{x}}_i \| \boldsymbol{r}_i \| \bar{\boldsymbol{m}}_i)$ happens in $\mathcal{F}_{vmpc}$.
   We show that if $M_i \notin \mathcal{B}_i$, then $M_i \notin \mathcal{M}'$. Suppose by contradiction that there is some $M_i \in \mathcal{M}$, $M_i \notin \mathcal{B}_i$. If $M_i \notin \mathcal{B}_i$, then the verification of M_i has succeeded, i.e M_i has come up with $\boldsymbol{d}_{ij}^{\ell k}$ such that $\mathsf{declassify}((\boldsymbol{d}_{ji}^{\ell k})_{k \in [n]}) = \mathbf{0}$.

Recall that there is a subset $\mathcal{H}$ of $t+1$ (semi)honest parties who indeed use the shares of values $\bar{\boldsymbol{x}}_i$, $\boldsymbol{r}_i$, $\bar{\boldsymbol{m}}_i$ that have been committed to $\mathcal{F}_{vmpc}$, and the shares of the precommitted multiplication triples $\boldsymbol{s}_i$. Since $|\mathcal{H}| = t+1$, also $\mathsf{declassify}((\boldsymbol{d}_{ji}^{\ell k})_{k\in\mathcal{H}}) = \mathbf{0}$ holds, this immediately implies $A(C_{ij}^{\ell}, \hat{\boldsymbol{s}})\boldsymbol{p}_{1ij}^{\ell} + A(C_{ij}^{\ell}, \hat{\boldsymbol{s}})\boldsymbol{p}_{2ij}^{\ell} = \mathbf{0}$, where $\boldsymbol{p}_{ij}^{\ell} = \boldsymbol{p}_{1ij}^{\ell} + \boldsymbol{p}_{2ij}^{\ell} = (1\|\bar{\boldsymbol{x}}_i\|\boldsymbol{r}_i\|\bar{\boldsymbol{m}}_i\|\boldsymbol{s}_i)$, so $\hat{\boldsymbol{s}}$ proves that C_{ij}^{ℓ} have been computed correctly with respect to $\bar{\boldsymbol{x}}_i$, $\boldsymbol{r}_i$, $\bar{\boldsymbol{m}}_i$ of $\mathcal{F}_{vmpc}$. Hence $\boldsymbol{m}_{ij}^{\ell} = C_{ij}^{\ell}(\bar{\boldsymbol{x}}_i, \boldsymbol{r}_i, \bar{\boldsymbol{m}}_i)$ for all $M_i \notin \mathcal{M}'$.

Since all parties see all the conflicts, all the sets $\mathcal{B}_k = \mathcal{B}$ are equal for all $k \in [n]$. Now both $\mathcal{F}_{vmpc}$ and π_{vmpc} output $(\mathsf{blame}, i, j, \ell)$ for $i \in \mathcal{B}$. For $i \notin \mathcal{B}$, both $\mathcal{F}_{vmpc}$ and π_{vmpc} output $(\mathsf{approved}, C_{ij}^{\ell}, i, j, \ell, \boldsymbol{m}_{ij}^{\ell})$.

G Preprocessing Phase for WCP Model

Let $\mathbb{F}$ be a finite field where the computation takes place, equipped with addition $(+)$ and multiplication $(\cdot)$ operations. Let $M_1, \ldots, M_n$ be the machines performing the computation of parties $P_1, \ldots, P_n$, $\mathcal{C}$ the set of indices of corrupted machines, and $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ the t-coalition split adversary.

The preprocessing phase uses a linearly homomorphic verifiable $(n, t+1)$-threshold sharing scheme, such that any $t+1$ parties are able to reconstruct the secret, but to any t parties the shares look completely random. We write $(a^k)_{k\in[n]} = \mathsf{classify}(a)$ to denote the sharing of a, and $a = \mathsf{declassify}((a^k)_{k\in[n]})$ to denote the reconstruction of a from shares.

G.1 Ideal Functionality

The verification of [15] relies on a preprocessing phase that generates a sufficient number of preshared randomness and special preshared tuples. We modify the ideal functionality a bit, so that it can be used to make the verification secure in WCP model. Instead of sharing a value s directly, we first represent s as a sum $s = s_1 + s_2$, and then share s_1 and s_2 to s_1^k and s_2^k respectively.

The ideal functionality presented on Fig. 12 is a bit simplified compared to [15]. Namely, we assume that the only preshared tuples are the multiplication triples [19], as they are sufficient for computing an arbitrary circuit over a finite field $\mathbb{F}$. While [15] proposes various kinds of helpful tuples that allow to verify efficiently computations over multiple rings, proving that their generation is secure in WCP model would be analogous.

We note that the randomness distribution function of $\mathcal{F}_{pre}$ is used also to generate the preshared randomness for message transmission protocol $\pi_{transmit}$ (see App. E) that forces these random elements to be distributed correctly even if both the sender M_i and the receiver M_j are actively corrupted.

G.2 Real Protocol

For modelling the communication of the randomness distribution function of $\mathcal{F}_{pre}$, we cannot use the *new implementation* $\pi_{transmit}$ of Sec. E, since $\pi_{transmit}$ is in turn based on the randomness distribution of $\mathcal{F}_{pre}$. Instead, we may assume that all the communication takes place using the *old implementation* of $\mathcal{F}_{tr}$ of [15] in its "cheap mode" which does not prevent the protocol from stopping. We use it to enable simple authentication and forwarding of messages. We note that in the "cheap mode" both the message transmission and the broadcast are secure also in WCP model.

For the functions other than randomness distribution, we are free to use the new implementation $\pi_{transmit}$ of Sec. E. For the functions commit and preveal, we actually *must* use $\pi_{transmit}$ of Sec. E since this function will be called not in the preprocessing phase, but at some point later.

- On input (rnd, id), the randomness shares $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ have to be generated for $M_{p(id)}$. Each machine M_j generates two random vectors $\boldsymbol{r}_{1j}$ and $\boldsymbol{r}_{2j}$ of length $m(id)$, computes $(\boldsymbol{r}_{1j}^k)_{k\in[n]} =$

$\mathcal{F}_{pre}$ works with unique identifiers id, encoding the vector length $m(id)$ and the party $p(id)$ that gets all the shares of the vector. It stores an array val of already shared vectors of $\mathbb{F}$, and a boolean matrix rev that denotes which shares have already been partially revealed. Let $\mathcal{A}\mathsf{s} = \{\mathcal{A}\mathsf{s}_1^H, \ldots, \mathcal{A}\mathsf{s}_n^H, \mathcal{A}\mathsf{s}^L\}$ be the ideal t-coalition split adversary.

Initialization: On input (init) from the environment, initialize all the arrays to empty.

Randomness distribution: On input (rnd, id) from all (semi)honest parties, check if $\mathsf{val}[id]$ exists. If it does, take $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)_{k\in[n]} := \mathsf{val}[id]$.

Otherwise, generate field vectors $\boldsymbol{r}_1, \boldsymbol{r}_2 \xleftarrow{\$} \mathbb{F}$ of length $m(id)$, share them as $(\boldsymbol{r}_1^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{r}_1)$ and $(\boldsymbol{r}_2^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{r}_2)$, and assign $\mathsf{val}[id] := (\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)_{k\in[n]}$.
Output $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)_{k\in[n]}$ to $P_{p(id)}$, and for all $i \neq p(id)$ output $(\boldsymbol{r}_1^i, \boldsymbol{r}_2^i)$ to P_i. Output $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)$ also to $\mathcal{A}\mathsf{s}_{c(k)}^H$. Output $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)_{k\in[n]}$ to $\mathcal{A}\mathsf{s}_{c(p(id))}^H$.

Mutual randomness distribution: On input (mrnd, id) from all (semi)honest parties, $\mathcal{F}_{pre}$ acts in exactly the same way as on input (rnd, id), except that there are now two parties $p1(id)$ and $p2(id)$ that get all the n shares $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)$.

Precomputed triple distribution: On input (prec, id) from all (semi)honest parties, check if $\mathsf{val}[id]$ exists. If it does, take $(\boldsymbol{s}_1^k, \boldsymbol{s}_2^k)_{k\in[n]} := \mathsf{val}[id]$.

Otherwise, generate the random values $\boldsymbol{r}_x \xleftarrow{\$} \mathbb{F}$, $\boldsymbol{r}_y \xleftarrow{\$} \mathbb{F}$ of length $m(id)$, compute $\boldsymbol{r}_{xy} = \boldsymbol{r}_x \cdot \boldsymbol{r}_y$, and take $\boldsymbol{s} = (\boldsymbol{r}_x \| \boldsymbol{r}_y \| \boldsymbol{r}_{xy})$. Generate a random vector $\boldsymbol{s}_1$ and compute $\boldsymbol{s}_2 = \boldsymbol{s} - \boldsymbol{s}_1$. Share these vectors as $(\boldsymbol{s}_1^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{s}_1)$ and $(\boldsymbol{s}_2^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{s}_2)$. Assign $\mathsf{val}[id] := (\boldsymbol{s}_1^k, \boldsymbol{s}_2^k)_{k\in[n]}$.
Output $(\boldsymbol{s}_1^k, \boldsymbol{s}_2^k)_{k\in[n]}$ to $P_{p(id)}$, and output $(\boldsymbol{s}_1^i, \boldsymbol{s}_2^i)$ to P_i for all $i \neq p(id)$. Output $(\boldsymbol{s}_1^k, \boldsymbol{s}_2^k)$ also to $\mathcal{A}\mathsf{s}_{c(k)}^H$. Output $(\boldsymbol{s}_1^k, \boldsymbol{s}_2^k)_{k\in[n]}$ to $\mathcal{A}\mathsf{s}_{c(p(id))}^H$.

Commit: On input $(\mathsf{commit}, id, \boldsymbol{x})$ from $P_{p(id)}$ and (commit, id) from all the other (semi)honest parties, check if $\mathsf{val}[id]$ exists. If it does, and if $|\boldsymbol{x}| = m(id)$, take $(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)_{k\in[n]} := \mathsf{val}[id]$. Otherwise, generate a vector $\boldsymbol{x}_1 \xleftarrow{\$} \mathbb{F}$ of length $m(id)$, compute $\boldsymbol{x}_2 = \boldsymbol{x} - \boldsymbol{x}_1$ and share them as $(\boldsymbol{x}_1^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{x}_1)$ and $(\boldsymbol{x}_2^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{x}_2)$. Assign $\mathsf{val}[id] := (\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)_{k\in[n]}$.
Output $(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)_{k\in[n]}$ to $P_{p(id)}$, and for all $i \neq p(id)$ output $(\boldsymbol{x}_1^i, \boldsymbol{x}_2^i)$ to P_i. Output $(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)$ also to $\mathcal{A}\mathsf{s}_{c(k)}^H$. Output $(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)_{k\in[n]}$ to $\mathcal{A}\mathsf{s}_{c(p(id))}^H$.

Mutual commit: On input $(\mathsf{mcommit}, id, \boldsymbol{x})$ from $P_{p(id)}$ and $(\mathsf{mcommit}, id)$ from all the other (semi)honest parties, $\mathcal{F}_{comm}$ acts in exactly the same way as on input $(\mathsf{commit}, id, \boldsymbol{x})$, except that it remembers another party $p2(id)$ to which all the n shares $(\boldsymbol{x}_1^k, \boldsymbol{x}_2^k)$ will be output later.

Partially revealing shares: On input $(\mathsf{preveal}, id, i, b)$ from all (semi)honest parties, where $b \in \{1, 2\}$, if $\mathsf{rev}[id][i] == false$, check if $\mathsf{val}[id]$ exists. If it does, take the shares of precomputed values $(\boldsymbol{v}_1^k, \boldsymbol{v}_2^k)_{k\in[n]} := \mathsf{val}[id]$ and output $(\boldsymbol{v}_b^i)$ to each party P_j and each adversary. Set $\mathsf{rev}[id][i] = true$. Alternatively, $\mathcal{A}\mathsf{s}^L$ may choose to output $(\mathsf{corrupt}, p(id))$ to all (semi)honest parties.

Revealing mutual commitment to the second party: On input (open, id) from all (semi)honest parties, if $(\mathsf{mcommit}, id, \boldsymbol{x})$ has been input at some moment, take the shares of precomputed values $(\boldsymbol{v}_1^k, \boldsymbol{v}_2^k)_{k\in[n]} := \mathsf{val}[id]$ and output $(\boldsymbol{v}_1^k, \boldsymbol{v}_2^k)_{k\in[n]}$ to the party $P_{p2(id)}$ and the adversary $\mathcal{A}_{c(p2(id))}^H$. Alternatively, $\mathcal{A}\mathsf{s}^L$ may choose to output $(\mathsf{corrupt}, p(id))$ to all (semi)honest parties.

Stopping: On input (stop) from $\mathcal{A}\mathsf{s}^L$, stop the computation and output $\perp$ to all parties.

Fig. 12: Ideal functionality $\mathcal{F}_{pre}$

$\mathsf{classify}(\boldsymbol{r}_{1j})$, $(\boldsymbol{r}_{2j}^k)_{k\in[n]} = \mathsf{classify}(\boldsymbol{r}_{2j})$, and transmits $(\boldsymbol{r}_{1j}^k, \boldsymbol{r}_{2j}^k)$ to M_k for all $k \in [n]$. M_k forwards the received shares to $M_{p(id)}$. $M_{p(id)}$ verifies that the shares are consistent, computes $\boldsymbol{r}_1^k = \sum_{j\in[n]} \boldsymbol{r}_{1j}^k$ and $\boldsymbol{r}_2^k = \sum_{j\in[n]} \boldsymbol{r}_{2j}^k$. Since the secret sharing scheme is linear, the obtained shares also correspond to a $(n, t+1)$ sharing, so $M_{p(id)}$ takes $\boldsymbol{r}_1 = \mathsf{declassify}(\boldsymbol{r}_1^k)$, $\boldsymbol{r}_2 = \mathsf{declassify}(\boldsymbol{r}_2^k)$. If something goes wrong, the preprocessing aborts, some (semi)honest party sends a complaint to each other party, and $M_{p(id)}$ outputs $\perp$ to P_i for $i \notin C(1)$.

- On input (mrnd, id), the machines first act in the same way as on input (rnd, id), generating a shared randomness $(\boldsymbol{r}_1, \boldsymbol{r}_2)$ that is given to the machine $M_{p1(id)}$. We now want to give it also to the machine $M_{p2(id)}$. For this, $M_{p1(id)}$ forwards all the shares $\boldsymbol{r}_{1j}^k$ using the forwarding functionality of old $\mathcal{F}_{tr}$ of [15]. By definition of forwarding, $M_{p2(mid)}$ gets all the rights for proving the authenticity of all $\boldsymbol{r}_{1j}^k$ which are sufficient for reconstructing $(\boldsymbol{r}_1^k, \boldsymbol{r}_2^k)$.
- On input $(\mathsf{prec}, id, \eta, \kappa)$, the multiplication triples $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of length $m = m(id)$ need to be generated for the machine $M_{p(id)}$, where η and κ are some security parameters. Their generation is done in several steps.

1. $M_{p(id)}$ generates and shares 3 random vectors $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of length $m\eta + \kappa$ exactly in the same way as on input (rnd, id). It then computes and broadcasts $\boldsymbol{c}' = \boldsymbol{c} - \boldsymbol{a} \cdot \boldsymbol{b}$ (here $M_{p(id)}$ just computes $\boldsymbol{c}'$ itself and noone checks yet if it is valid). Each machine M_k outputs the vectors $\boldsymbol{s}_1^k = (\boldsymbol{a}_1^k \| \boldsymbol{b}_1^k \| \boldsymbol{c}_1^k - \boldsymbol{c}')$ and $\boldsymbol{s}_2^k = (\boldsymbol{a}_2^k \| \boldsymbol{b}_2^k \| \boldsymbol{c}_2^k)$ to P_k. If the broadcast fails, the preprocessing aborts, and $M_{p(id)}$ outputs $\perp$ to P_i for $i \notin C(1)$.
2. The machines agree on the common randomness that will be used in the verification of the triples of $M_{p(id)}$. Each machine M_k broadcasts a random number r^k, and they take the seed $r = \sum_{k \in [n]} r^k$. If some party refuses to participate, its share is ignored by (semi)honest parties.
3. Some κ of $M_{p(id)}$'s triples are published (by just broadcasting the shares without presenting the signatures obtained before). The choice of which triples have to be published is determined by the seed r on which the parties agreed before. If any party refuses to participate, its share is ignored since there are still at least $(t+1)$ shares available. If any shares are inconsistent, M_i outputs $\perp$ to P_i for $i \notin C(1)$.
4. Some $2m$ of $M_{p(id)}$'s triples are divided to pairs according to the seed r. For each pair (a, b, c), (a', b', c'), the parties publish $\hat{a} = a - a'$ and $\hat{b} = b - b'$ (by just broadcasting the shares $a^k - a'^k$ and $b^k - b'^k$, without presenting the signatures obtained before), compute locally the shares $a'^k \cdot \hat{b} + b^k \cdot \hat{a} + c'^k - c^k$, publish them (again, just using broadcast), and check if the resulting shares comprise 0. If any party refuses to participate, its share is ignored since there are still at least $(t+1)$ shares available. If any broadcast shares are inconsistent, M_i outputs $\perp$ to P_i for $i \notin C(1)$.
   If the zero check passes, the triple (a', b', c') is discarded, and (a, b, c) is now again paired with some other triple (a'', b'', c'') from the remaining triples. This pairing repeats η until there are m triples left, which are output to the parties.

– On input (commit, id), the machines first generate a fresh shared randomness $(\boldsymbol{r}_1, \boldsymbol{r}_2)$ for $M_{p(id)}$, similarly to (rnd, id). The machine $M_{p(id)}$ that has received the input $(\mathsf{commit}, id, \boldsymbol{x})$, broadcasts $\boldsymbol{x}' = (\boldsymbol{r}_1 + \boldsymbol{r}_2) - \boldsymbol{x}$. Each (semi)honest machine M_k outputs $(\boldsymbol{r}_1^k - \boldsymbol{x}', \boldsymbol{r}_2^k)$ to P_k. If the broadcast fails, each (semi)honest M_i outputs $(\mathsf{corrupt}, p(id))$ to P_i.
– On input $(\mathsf{mcommit}, id)$, the machines act exactly in the same way as on input (commit, id), and only remember that the shares should be revealed to another party $p2(id)$ at some moment.
– On input $(\mathsf{preveal}, id, k, b)$ where $b \in \{1, 2\}$, the machine M_k broadcasts the message $(\boldsymbol{s}_b^k, \sigma)$, where σ is the signature of $M_{p(id)}$ on $\boldsymbol{s}_b^j$. If M_j refuses to broadcast or does not provide a valid signature, then its share is ignored since there are still at least $(t+1)$ shares available. If any shares are inconsistent (and nevertheless are all signed by $p(id)$), each (semi)honest M_i outputs $(\mathsf{corrupt}, p(id))$ to P_i.
– On input (open, id), each machine M_k sends its share of the randomness that was used when committing a value on input (commit, id) to $M_{p2(id)}$. At least $(t+1)$ shares will be delivered by (semi)honest parties. Since each randomness share is accepted iff it contains the signatures of all parties, the valid shares cannot be inconsistent. $M_{p2(id)}$ reconstructs $\boldsymbol{r}_1 + \boldsymbol{r}_2$ and computes $\boldsymbol{x} = (\boldsymbol{r}_1 + \boldsymbol{r}_2) - \boldsymbol{x}'$, where $\boldsymbol{x}'$ is the value that was broadcast on input (commit, id).

We note that, since we are working with $\mathbb{F}$, the verification of triples can be optimized using some polynomial-based approach that allows to verify several triples succinctly at once. We do not provide the optimization details here, as it is not our primary goal.

G.3 Simulator for π_{pre}

We construct a simulator $S = \{S_1^H, \ldots, S_n^H, S^L\}$ that translates the between the messages $\mathcal{F}_{pre}$ exchanges with the ideal adversary, and the messages the protocol of Sec. G.2 exchanges with the real adversary $\mathcal{A} = \{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H, \mathcal{A}^L\}$ over the network.

All the communication is modeled using the "cheap mode" of $\mathcal{F}_{tr}$. That is, if for any malicious sender $M_{s(mid)}$, $\mathcal{A}_L$ delivers a message $\perp$, or a message m^* that is not provided by a valid signature of $P_{s(mid)}$, then S^L sends (stop) to $\mathcal{F}_{pre}$ and forwards $\perp$ to all S_i^H that simulate to

$\mathcal{A}_i^H$ a complaint on behalf of the receiver $M_{r(mid)}$. If $\mathcal{A}^L$ delivers a complaint for any malicious receiver $M_{r(mid)}$, then S^L sends (stop) to $\mathcal{F}_{pre}$.

First of all, S^L sends (init) to $\mathcal{F}_{pre}$.

- On input (rnd, id), each simulator S_i^H simulates generation of shares $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ of length $m = m(id)$. First, S^L receives the shares $\boldsymbol{r}_{1j}^k$ and $\boldsymbol{r}_{2j}^k$ for all $k \in [n]$, $j \in C(1)$ from $\mathcal{A}_L$. It checks if they are consistent. If not, then S^L sends (stop) to $\mathcal{F}_{pre}$. Otherwise, it forwards the values provided by $\mathcal{A}^L$ to all S_i^H. S^L sends (rnd, id) for id s.t $p(id) = i$ to $\mathcal{F}_{pre}$.
  - For $k \in C(i)$, S_i^H receives the shares $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ whose generation now needs to be simulated. Up to t shares $\boldsymbol{r}_{1j}^k$ and $\boldsymbol{r}_{2j}^k$ for $j \in C(1)$ have already been chosen. Each S_i^H now has to simulate the remaining shares $\boldsymbol{r}_{1j}^k$ and $\boldsymbol{r}_{2j}^k$ of $j \notin C(1)$ in such a way that the resulting $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ are the same as chosen by $\mathcal{F}_{pre}$. This is always possible due to the fact that at least one of the parties that generate the randomness is (semi)honest.
  - For $k \notin C(i)$, S_i^H does not know the values $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ generated by $\mathcal{F}_{pre}$. However, $\mathcal{A}_i^H$ waits for up to t shares $\boldsymbol{r}_{bj}^k$ for $j \in C(i)$. In this case, S_i^H just generates up to t random shares $\boldsymbol{r}_{bj}^k$ for $j \in C(i)$. Together with the shares $\boldsymbol{r}_{bj}^k$ of $j \in C(1)$, since $t < n/2$, there are still less than n shares of $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$ given to $\mathcal{A}_i^H$ now.

  In general, different simulators S_i^H may now have different opinions on what the share $\boldsymbol{r}_{bj}^k$ of some $j \notin C(1)$ would be, but the inconsistency is not noticed since only one coalition leaks $\boldsymbol{r}_{bj}^k$ to $\mathcal{Z}$, and $\mathcal{Z}$ does not know the precise values of $\boldsymbol{r}_{bj}^k$ generated inside of the protocol.
- On input (mrnd, id), the simulator S acts similarly to the input (rnd, id) to generate $\boldsymbol{r}_1^k$ and $\boldsymbol{r}_2^k$. It then simulates forwarding these values to $M_{p2(id)}$. If $p1(id) \in C(1)$, and $\mathcal{A}^L$ decides that $M_{p1(id)}$ does not provide the forwarded values with valid signatures, then S^L sends (stop) to $\mathcal{F}_{pre}$.
- On input $(\mathsf{prec}, id, \eta, \kappa)$, S needs to simulate generation of $m = m(id)$ multiplication triples for $M_{p(id)}$.
  1. For $p(id) \notin C(1)$, S simulates the generation of $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ in the same way it has simulated (rnd, id), this time sending m times (prec, id) to $\mathcal{F}_{pre}$. In addition, S^L generates $m(\eta - 1) + \kappa$ more random triple shares $(\boldsymbol{a}^k, \boldsymbol{b}^k, \boldsymbol{c}^k)$ and sends them to all S_i^H.
     After the triples are generated, each S_i^H shuffles the triples according to a common randomness r provided by S^L. Now only the simulators know where are the m triples generated by $\mathcal{F}_{pre}$ and where are their own triples, and $\mathcal{Z}$ does not know it.
     After shuffling, all the k-th shares are sent by $S_{c(k)}^H$ to $\mathcal{A}_{c(k)}^H$. Then $S_{c(p(id))}^H$ simulates to $\mathcal{A}_{c(p(id))}^H$ the broadcast of $\boldsymbol{d} = \boldsymbol{c} - \boldsymbol{a} \cdot \boldsymbol{b}$. If $p(id) \in C(1)$, then it might have cheated and broadcast a malicious $\boldsymbol{d}^*$. This $\boldsymbol{d}^*$ is sent by S^L to all S_i^H, and they use it in their simulation.
     If $M_{p(id)}$ has not cheated, then the other simulators do not know $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ that were generated by $\mathcal{F}_{pre}$, and they just simulate broadcast of a random value $\boldsymbol{c}'$ that comes from uniform distribution.
  2. The machines should agree on the common randomness that they will use to choose which triples have to be opened. $\mathcal{A}^L$ contributes the values of the malicious parties that S^L sends to all simulators S_i^H. They now generate the remaining shares r^k of $k \notin C(1)$ in such a way that $r = \sum_{k \in [n]} r^k$ is the same r that the simulators used before for the random shuffle.
  3. The simulators need to reveal the triples based on the randomness r they generated before. Recall that they have chosen r in such a way that the shares $(\boldsymbol{a}^k, \boldsymbol{b}^k, \boldsymbol{c}^k)$ to be revealed are already known to all the simulators. $\mathcal{A}^L$ decides on the values that the malicious parties publish. S^L sends them to all S_i^H. Since S^L already holds all the shares, it may check if $\mathcal{A}^L$ has cheated and any shares are inconsistent. If they are, S^L sends (stop) to $\mathcal{F}_{pre}$.
     - If $p(id) \notin C(1)$, then all S_i^H simulate honest behaviour of $M_{p(id)}$. All the valid shares are already held by all S_i^H, so it simulates publishing the shares of honest parties. It is now important that $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$ would hold. The simulators have generated these triples themselves, and since they have simulated a valid broadcast of $\boldsymbol{c}' = \boldsymbol{c} - \boldsymbol{a} \cdot \boldsymbol{b}$ for a (semi)honest M_i, the equality check passes.

- If $p(id) \in C(1)$, it may additionally happen that $\boldsymbol{d}^* \neq \boldsymbol{c} - \boldsymbol{a} \cdot \boldsymbol{b}$. This attack is also easy to simulate since $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ and $\boldsymbol{d}^*$ are already known by S_i^H.

4. In the real protocol, the parties should start verifying the triples pairwise. The pairing depends on the randomness r agreed on before, and the simulators have ensured that all the triples except the last one (that will be output to the parties) are already known to all of them. Since simulating already known shares is simpler, let us consider the last step, where the shares of (a', b', c') are known to all simulators, but the shares of (a, b, c) are not.
   First, S^L sends to all S_i^H the shares of malicious parties provided by $\mathcal{A}^L$. Only S_1^H may check if the shares provided by $\mathcal{A}^L$ are incorrect. Each other simulator treats any $a^{*k} \neq \top$ and $a'^{*k} \neq \top$ (analogously for $b^{*k}, b'^{*k}, c^{*k}, c'^{*k}$) as incorrect shares, generating the remaining shares independently from them. We claim that it is sufficient for the simulation.
   - If $\mathcal{A}_1^H$ is a false adversary, then the shares a^{*k} and a'^{*k} that were chosen by $\mathcal{A}_L$ cannot depend on the shares a^k and a'^k, or even the shares a and a', that were actually issued by $\mathcal{F}_{vmpc}$ for $k \in C(1)$. Hence the shares of (semi)honest parties may indeed be generated independently from a^{*k} and a'^{*k}.
   - If $\mathcal{A}_1^H$ is the true adversary, then generating independent shares a^k and a'^k for (semi)honest parties is not a good idea since the malicious shares of $\mathcal{A}_L$ may strongly depend on the shares a^k and a'^k generated for $k \in C(1)$ by $\mathcal{F}_{vmpc}$. However, in this case we do not care about the view that S_i^H simulates to $\mathcal{A}_i^H$ since if $\mathcal{A}_{c(A)}^H$ is the true adversary, then $\mathcal{A}_i^H$ is a false adversary.

   The simulation of opening the shares of (semi)honest parties proceed as follows:
   - If $p(id) \notin C(1)$, S_i^H needs to simulate the remaining shares for honest parties.
     * The first component are values $\hat{a} = a - a'$ and $\hat{b} = b - b'$. Since a' and b' are generated by the simulators and are never leaked to anyone, both $\hat{a}$ and $\hat{b}$ are distributed uniformly, so it is also easy to simulate them.
     * The second component of these published values are always 0 for a (semi)honest party, and hence are easy to simulate.
   - If $p(id) \in C(1)$, then the simulation is trivial for S_1^H since it has all the values already, so let us consider S_i^H for $i \neq 1$. First, S_i^H needs to simulate the values $\hat{a} = a - a'$ and $\hat{b} = b - b'$. From S^L, it has received up to t shares $\hat{a}^k$ and $\hat{b}^k$. It also holds the shares a^k and a'^k of the parties belonging to the i-th coalition, and since there can be more than t shares, they may already be inconsistent. Since S_i^H has shown up to t shares to $\mathcal{A}_i^H$ so far, it just does not take into account the malicious shares sent by S^L and defines the remaining shares in such a way that $\hat{a}$ and $\hat{b}$ are uniformly distributed values. We claim that it is a good simulation (the reasoning is similar to the inconsistent share case):
     * If $\mathcal{A}_1^H$ is a false adversary, then the shares of $\hat{a}^k$ and $\hat{b}^k$ for $k \in C(1)$ that were sent by $\mathcal{A}_L$ cannot depend on the actual values a, a', b, b', and the actual values of $\hat{a}$ and $\hat{b}$ still are uniformly distributed in the view of $\mathcal{Z}$. Each share provided by $\mathcal{A}_L$ is either completely independent from the shares of the other parties (and hence is inconsistent with them with overwhelming probability), or is equal to $\top$ which means that S_i^H generates the corresponding share itself.
     * If $\mathcal{A}_1^H$ is the true adversary, then we do not care about the view that S_i^H simulates to $\mathcal{A}_i^H$, similarly to the inconsistent share case.

     We can use a similar discussion to show that the shares $a'^k \cdot \hat{b} + b^k \cdot \hat{a} + c'^k - c^k$ for $k \notin C(1)$ can also be generated uniformly. However, even if $\mathcal{A}^L$ decided not to falsify any shares at all, there is still the question from which distribution the resulting value $z = a' \cdot \hat{b} + b \cdot \hat{a} + c' - c$ should come. Differently from the $p(id) \notin C(1)$ case, these shares do not necessarily comprise 0 since $\mathcal{A}_L$ may have published false d^* or d'^* before. We claim that, unless $d^* = \top$ and $d'^* = \top$, the value z can be sampled from a uniform distribution.
     * If $\mathcal{A}_1^H$ is a false adversary, then the values d^* and d'^* that were sent by $\mathcal{A}_L$ cannot depend on the initial values (a, b, e) and (a', b', e') for which $d = e - a \cdot b$

and $d' = e' - a' \cdot b'$ should have been computed. The random values e and e' have not been used anywhere else, and hence $c = e - d^*$ and $c' = e' - d'^*$ are uniformly distributed value regardless of the choice of d^* and d'^*, and so is $z = a' \cdot \hat{b} + b \cdot \hat{a} + c' - c$.

  - If $\mathcal{A}_1^H$ is the true adversary, we do not care about the view that S_i^H simulates to $\mathcal{A}_i^H$, similarly to the $\hat{a} = a - a'$ and $\hat{b} = b - b'$ case.

  The simulators should also ensure that no invalid triples are finally accepted by $\mathcal{F}_{pre}$. The zero check of S_1^H may be passed only if either both triples are correct or both are incorrect. Two incorrect triples may get into the same pair only with a small probability due to the previous cut-and-choose. Repeating this check η times makes the probability negligible.

- On input (commit, id) and $(\mathsf{commit}, id, \boldsymbol{x})$, the simulators first simulate the generation of $(\boldsymbol{r}_1, \boldsymbol{r}_2)$ in the same way they have simulated (rnd, id). Since the distribution of $\boldsymbol{x}'$ is random if $p(id)$ is (semi)honest, they may simulate the broadcast of $\boldsymbol{x}' = (\boldsymbol{r}_1 + \boldsymbol{r}_2) - \boldsymbol{x}$ similarly to the broadcast of $\boldsymbol{c}'$ in (prec, id).
- On input $(\mathsf{mcommit}, id)$, the simulation is the same as on input (commit, id).
- On input $(\mathsf{preveal}, id, j, b)$ where $b \in \{1, 2\}$, the simulators first use $\mathcal{F}_{pre}$ to actually reveal the shares to each other. Then the broadcasts of M_j of the message $(\boldsymbol{s}_b^j, \sigma_{jb})$ have to be simulated. First of all, S^L sends the possibly malicious shares generated by $\mathcal{A}^L$ to all S_i^H. Now each S_i^H may compare these shares with what it has got from $\mathcal{F}_{pre}$ and simulate the attack sending the shares of remaining (semi)honest parties to $\mathcal{A}_i$. Here we assume that any simulator S_i^H is allowed to generate signatures of (semi)honest parties on any messages.
  If id is such that $(\mathsf{commit}, id, \boldsymbol{x})$ has been input at some moment, then one needs to be careful since $\boldsymbol{x}'$ has already been leaked to $\mathcal{Z}$. Since at most one r_b will be revealed, the value $\boldsymbol{r} = \boldsymbol{r}_1 + \boldsymbol{r}_2$ can still be arbitrary for the parties that have not known $\boldsymbol{r}$.
- On input (open, id), each S_k^H should simulate transmitting the randomness of (semi)honest parties to $M_{p2(id)}$. In general, this randomness is not known by S_k^H. However, the simulation is needed only for $S_{p2(id)}^H$ that gets the value $\boldsymbol{x}$ to be opened from $\mathcal{F}_{pre}$. Sine it already knows $\boldsymbol{x}'$ that has been broadcast on input $(\mathsf{mcommit}, id)$, it computes $\boldsymbol{r} = \boldsymbol{x} = \boldsymbol{x}'$ and simulates the shares of (semi)honest parties in such a way that they comprise $\boldsymbol{r}$.