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. Cleaning the systems is essential to prevent possible security leaks in future.

Protecting against this goal essentially means achieving security against several non-cooperating adversaries, where only one adversary is active, representing the real attacker, and each other adversary is passive, corrupting 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 some honest party to learn some other honest party's private data. The adversary's inability to learn such data itself does not imply the impossibility of such attacks.

In formalizing such attacks and security against them, we want to cover only leaks that are due to the protocol itself. A protocol always runs in the context of

some larger system and we must be careful to exclude the side channels unrelated to the protocol from the security definition.

The security of protocols is often proved in the universal composability (UC) framework [2] which ensures that the protocol is secure not only when considered alone, 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. This model is not well-suited for defining the property we have in mind because an honest party trying to not learn other honest parties' data should really be modeled as passively corrupted, but independent from the "real" adversary.

If we care about the views of honest parties, we could treat each honest party as an 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]. These models are used to prove the protocol property of *preserving collusions*, meaning that the parties cannot use the protocol to exchange more information that they could do without the protocol. Treating each honest party as a separate adversary, collusion-preserving property would be sufficient to protect against leaking information to honest parties. However, technical details prevent us from using CP or LUC as the basis for defining when the adversary cannot make one honest party's secrets leak to another honest party. Namely, CP and LUC consider the joint view of all the adversaries as the environment output. If each party is controlled by an adversary, then the environment eventually gets the joint view of all the parties on the protocol. Hence a number of techniques are unusable as the building blocks of protocols deemed secure. Threshold secret sharing is one of the techniques ruled out, since the environment gets all the shares and may reconstruct the shared secret. We need a model where we can state that an honest party will never collude with the other parties its view may be treated as being completely separated from the other adversaries' views. For this, we need a model weaker than CP or LUC.

Our contribution We define a "weak CP" (WCP), which splits the adversary into mutually exclusive coalitions. The motivation behind splitting the adversary to coalitions is to treat each honest party and the attacker as separate entities that are not trying to collaborate. Instead of bounding the total number of corrupted parties, we only bound the sizes of coalitions. Our model does not focus on preventing the attacker from sending arbitrary data directly to the honest parties, but rather on detecting the flaws in protocols where 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 $\{\mathcal{A}_1^H, \ldots, \mathcal{A}_n^H\}$ and $\mathcal{A}^L$, each $\mathcal{A}_i^H$ representing a separate adversarial coalition. Only $\mathcal{A}_i^H$ may get messages from the parties corrupted by it, 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 $\mathcal{A}_j^H$. This allows to capture the attacks where both $\mathcal{A}_i^H$ and $\mathcal{A}_j^H$ represent the views of some honest parties.

After reviewing some preliminaries in Sec. 2 and related work in Sec. 3, we give a formal definition of WCP and prove its composability in Sec. 4. In Sec. 5 we give examples of new attacks that WCP detects. In Sec. 6 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. We also present some transformations that make a protocol that is secure in UC model also secure in WCP model.

2 Preliminaries

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. Throughout this work, 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 it computes exactly 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 $\mathcal{A}$.

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 [5]:** 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 [6]:** the corrupted party may misbehave, but only as far as it will not be caught.
- **Active (malicious):** the corrupted party does whatever it wants.

3 Related Work

The problem of leaking a secret to an honest party is not new. The multiparty computation protocol of [7] 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. Very shortly, one considers three parties where at most one can be maliciously corrupted. In the protocol, the first party generates a key and sends it to the second party, which uses it to encrypt a secret and send it to the third party. If the first party maliciously generates a weak key then the third party will learn the second party's secret. 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 [8] does take into account multiple adversaries. The more concrete frameworks [9–11] 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 has 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.

In CP and LUC, the environment gets the joint view of all the adversaries. Assigning an adversary to each honest party results in leaking all the data of honest parties to the environment, and so an honest party gets turned into a passively corrupted party. A secure protocol would have to be secure in the setting where all the parties are corrupted.

One way to prevent the communication between the honest and the corrupted parties is to assume that the environment is split into distinct parts with constrained information movement. For example, [12] 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. This property needs to be checked in addition to ordinary UC security. We use a simpler and cleaner solution in this paper, putting constraints onto the adversary instead of the environment. This allows to embed the confinement property into the definition of emulation.

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.

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 $EXEC_{\pi,\mathcal{A},\mathcal{Z}} \approx EXEC_{\phi,(S\|\mathcal{A}),\mathcal{Z}}$).*

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.

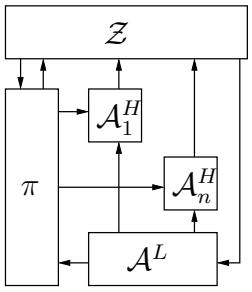

Fig. 1: t-coalition split adversary

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.

- 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

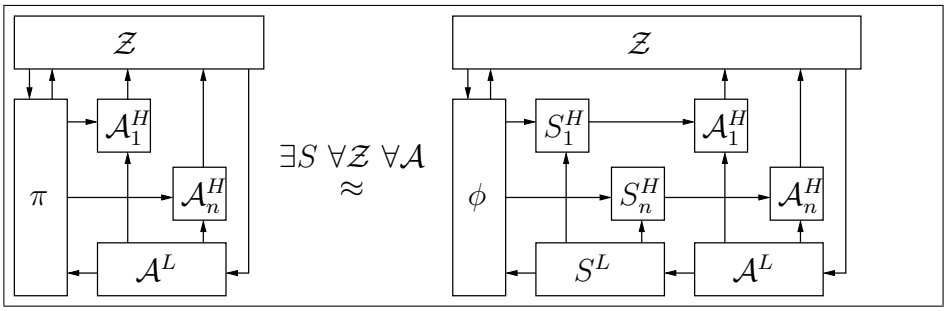

Fig. 2: t-WCP emulation

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 $\mathsf{As} = \{(S_1^H\|\mathcal{A}_1^H), \ldots, (S_n^H\|\mathcal{A}_n^H), (S^L\|\mathcal{A}^L)\}$ is also a t-coalition split adversary.*

The proof of Lemma 1 can be found in the full version of this paper [14].

Definition 5 (t-WCP emulation). *Let π and ϕ be n-party protocols. We say that π WCP-emulates ϕ if there is 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 $\mathsf{As} = \{(S_1^H\|\mathcal{A}_1^H), \ldots, (S_n^H\|\mathcal{A}_n^H), (S^L\|\mathcal{A}^L)\}$, the probability ensembles $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ and $EXEC_{\phi,\mathsf{As},\mathcal{Z}}$ are indistinguishable.*

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.

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. In WCP model, 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 the full version of this paper [14].

WCP Composition Theorem We prove that WCP definition is composable, similarly to UC.

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 the full version of this paper [14].

4.3 Relations to the Existing Notions

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 the full version of this paper [14].

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 the full version of this paper [14].

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 the full version of this paper [14].

5 Attacks Detected in WCP Model

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 the full version of this paper [14].

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 key shares 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 [7] related to bad key generation, and another example of an attack captured by WCP model, are given in the full version of this paper [14].

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 Adversaries Weaker than Active

First, 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 the full version of this paper [14].

A fail-stop adversary [5] follows the protocol as the honest parties do, but it also may force the corrupt parties to abort the protocol. Differently from the passive adversary case, the measures taken in the case when some party attempts to stop the protocol may result in leaking a secret to some honest party.

As a simple example, let us take the transmission functionality $\mathcal{F}_{tr}$ that has been used in [15, 16] 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. The realization of $\mathcal{F}_{tr}$ works on the assumption that the majority of parties follows the protocol. If there is a

Let $[n] = \{1, \ldots, n\}$, where n is the number of parties. Let $\mathcal{As} = \{\mathcal{As}_1^H, \ldots, \mathcal{As}_n^H, \mathcal{As}^L\}$ 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 $\mathcal{As}_i^H$. 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 $\mathcal{As}_i^H$. 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 $\mathcal{As}_i^H$; mark (mid, m, bc) as delivered.

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

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.

Compared to [15, 16], we need to modify the realization of $\mathcal{F}_{tr}$ in such a way that it would be secure in t-WCP model. Namely, P_i may no longer broadcast the message to all the parties, since some honest 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 $\mathcal{F}_{tr}$ given in [15]. 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 not anyone 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 $m'_{ij} \oplus q_{ij}^m$.

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 the full version of this paper [14].

If all the communication in the protocol is performed using $\mathcal{F}_{tr}$, then UC security implies WCP security for any fail-stop adversary. This result can be easily extended to any covert adversary [6] that will not cheat if it will be caught with a non-negligible probability. If the initial protocol is able to detect any covert adversary in the UC model, we may assume that a covert adversary will act as passive anyway, and $\mathcal{A}^L$ will not attempt to modify the flow of π since otherwise it will be detected. Hence we may be sure that, if a covert adversary will not attempt to cheat, then UC-emulation implies t-WCP emulation. Nevertheless,

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. 6.2.

We conclude our discussion about the weaker than active adversary with the following theorem.

Theorem 5. *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.*

The proof of Thm. 5 can be found in the full version of this paper [14].

6.2 Active Adversary

For constructing a multiparty protocol secure against active adversaries, we follow the general pattern used in other related works [17, 18]. 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.

On Fig. 4, we present a functionality $\mathcal{F}_{vmpc}$ that we use to compute one protocol round. In the full version of this paper [14], we give a protocol that t-WCP realizes $\mathcal{F}_{vmpc}$. The implementation relies on Byzantine agreement, and so it works only under (semi)honest majority assumption.

We use $\mathcal{F}_{vmpc}$ to construct a protocol transformation WCP-Comp (Fig. 5) that uses $\mathcal{F}_{vmpc}$ to compute each round. The transformation is analogous to Comp of [18, 19]). Having WCP-Comp, we may prove the following theorem.

Theorem 6. *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.*

The proof of Thm. 6 can be found in the full version of this paper [14].

7 Conclusions

We have defined WCP model, which a stronger version of UC that additionally allows to capture the cases where the information leaks to some honest party. It makes the protocol reliable not on some participants' unconditional honestness, but rather on their non-collusion which seems a more realistic assumption. The definition is weak enough to make WCP security relatively easily achievable. We have proposed a scheme transforming passively secure protocols with one adversary up to actively secure protocols with semihonest majority and multiple adversaries. Our transformation relies on semihonest majority assumption.

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 the party P_i and $(\mathsf{commit\_input}, sid, i)$ from all (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 $\mathcal{B}_0 = \{j \mid (\mathsf{corrupt}, j) \text{ has been sent by } \mathcal{A}\mathsf{s}^L\}$.

Message commitment On input $(\mathsf{commit\_msg}, sid, i, j, \ell, \boldsymbol{m})$ from the party 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 $\mathcal{B}_0 = \{j \mid (\mathsf{corrupt}, j) \text{ has been sent by } \mathcal{A}\mathsf{s}^L\}$.

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

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*

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 $\mathsf{WCP\text{-}Comp}(\pi)$

5-7, 2012. Proceedings, volume 7485 of *Lecture Notes in Computer Science*, pages 281–301. Springer, 2012.

5. 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.
6. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.
7. 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.

8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
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. Peeter Laud and Alisa Pankova. Securing multiparty protocols against the exposure of data to honest parties. Cryptology ePrint Archive, Report 2016/650, 2016. `http://eprint.iacr.org/2016/650`.
15. 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.
16. 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/`.
17. 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.
18. 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.
19. 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.