Secure Protocols with Asymmetric Trust

Ivan Damgård[1], Yvo Desmedt[2], Matthias Fitzi[3], and Jesper Buus Nielsen[1]

[1] Aarhus University, Dept. of Computer Science, 8200 Aarhus N, Denmark
{ivan,buus}(at)daimi.au.dk

[2] University College London, Dept. of Computer Science, London WC1E 6BT, United Kingdom
y.desmedt(at)cs.ucl.ac.uk

[3] ETH Zürich, Dept. of Computer Science, 8092 Zürich, Switzerland
fitzi(at)inf.ethz.ch

Abstract. In the standard general-adversary model for multi-party protocols, a global adversary structure is given, and every party must trust in this particular structure. We introduce a more general model, the *asymmetric-trust model*, wherein every party is allowed to trust in a different, personally customized adversary structure. We have two main contributions. First, we present non-trivial lower and upper bounds for broadcast, verifiable secret sharing, and general multi-party computation in different variations of this new model. The obtained bounds demonstrate that the new model is strictly more powerful than the standard general-adversary model. Second, we propose a framework for expressing and analyzing asymmetric trust in the usual simulation paradigm for defining security of protocols, and in particular show a general composition theorem for protocols with asymmetric trust.

1 Introduction

In the standard general-adversary model for multi-party computation (MPC) [13], an adversary structure is specified which basically lists all sets of parties that we expect the adversary might be able to corrupt. This model is *symmetric*: every party is required to trust in the same adversary structure $\mathcal{A}$. This is unnatural since there is no inherent reason why the parties should all have the same view on which adversary structure best models the given scenario. For instance, two parties may have completely contradictory beliefs on whether a third party can be corrupted or not. Also, insisting on one global adversary structure may imply that a party must consent to the fact that he himself is completely untrusted. In this paper, we introduce a more natural *asymmetric-trust* model where each party p_i is allowed to trust in his own adversary structure $\mathcal{A}_i$. We then explore the differences between this asymmetric model and the standard one.

Of course, a trivial approach is to try to build a protocol that will be secure even if any set from *any* $\mathcal{A}_i$ is corrupt. However, this may be impossible, namely if the union of all $\mathcal{A}_i$ violates known lower bounds for the symmetric model. Our main conclusion in this paper is that there are cases where the trivial symmetric solution does not work, but where nevertheless broadcast, verifiable secret sharing, or even general secure computation, are possible with asymmetric trust.

As an example, consider the three-party scenario where p_1 distrusts p_2, p_2 distrusts p_3, and p_3 distrusts p_1. In the standard model, MPC requires broadcast channels for this problem. In contrast, in the most natural one of our asymmetric-trust models, MPC does not require broadcast for the same scenario.

1.1 General setting

We assume that n parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ are given who are connected by a complete, synchronous network of pairwise channels. Also present is an adversary who may corrupt some subset of the parties.

We consider both *passive* and *active* corruption. We also consider both *computational* and *unconditional* security; where we may distinguish between *unconditional security with negligible error probability* or *perfect security.* When we do not state the type of security explicitly, positive results mean that the goal can be achieved with unconditional security, and negative results hold even w.r.t. to computational security.

A crucial point is whether the parties are additionally connected by *broadcast channels* and/or share a consistent *public-key infrastructure (PKI).* In the active case, broadcast/PKI typically allow for more resilient protocols than in the setting with only pairwise channels. Note that a PKI can be set up with respect to an unconditional pseudo-signature scheme [2, 17]. Therefore, in the PKI setting, the achievability of a computationally secure task typically implies its feasibility with unconditional security.

1.2 Contributions

General multi-party computation (MPC) [20, 12] typically relies on the two fundamental building blocks *broadcast* [16] (BC, aka Byzantine agreement) and *(verifiable) secret-sharing* [4, 19, 7] ((V)SS). It is thus interesting to know to which extent these tasks can be achieved in a certain model.

We introduce different variants of the asymmetric-trust model and corresponding definitions for broadcast, VSS, and general MPC; and give feasibility and impossibility results for these cases. Most results demonstrate that protocols for the asymmetric model are able to tolerate a strictly stronger adversary than any protocol for the symmetric model. For broadcast and VSS, we come quite close to characterizing the difference between symmetric and asymmetric trust, while the situation is much more open for general MPC.

In addition we give a general framework for augmenting security models with asymmetric trust. For concreteness we describe how to extend the UC framework [5] with asymmetric trust. This seems to be the first simulation-based security model for reasoning about asymmetric trust. Finally, we explore the issue of when UC secure MPC is possible when the parties have asymmetric trust in the setup assumptions.

1.3 Symmetric-trust model

In the symmetric-trust model, a single adversary structure $\mathcal{A}$ is given which is a monotone subset of the power set of $\mathcal{P}$, $\mathcal{A} \subseteq 2^{\mathcal{P}}$. Monotone means that $A \in \mathcal{A}$ and $A' \subset A$ imply that $A' \in \mathcal{A}$.[4] The goal is to achieve secure MPC for the case that an adversary corrupts the parties in exactly one set in $\mathcal{A}$. However, if the adversary manages to corrupt a set $A \notin \mathcal{A}$ then no security is guaranteed. A set $A \in \mathcal{A}$ is called maximal if there is no set $A' \in \mathcal{A}$ that strictly contains A: $\nexists A' \in \mathcal{A} : A' \supset A$.

The tight bounds [13] for multi-party computation in the symmetric model are summarized in the following table where the second column indicates whether broadcast channels or a public-key infrastructure (PKI) are available.

STD	Broadcast/PKI	Unconditional	Computational
Passive	don't care	Q^2	Q^1
Active	available	Q^2	Q^2
Active	not available	Q^3	Q^3

$$Q^k \equiv \left(\forall A_1, \ldots, A_k \in \mathcal{A} : \bigcup_{i=1}^{k} A_i \neq \mathcal{P}\right)$$

1.4 Asymmetric-trust model

In the asymmetric-trust model, every party p_i has its own personalized adversary structure $\mathcal{A}_i \subseteq 2^{\mathcal{P}}$. We denote $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ as the aggregate adversary structure and define $\mathcal{A}^* := \bigcup_{i=1}^{n} \mathcal{A}_i$. We assume that each party p_i trusts itself, i.e., $A \in \mathcal{A}_i \Rightarrow p_i \notin A$. The set of corrupted parties is denoted by F.

We generally assume that all the adversary structures $\mathcal{A}_i$ are publicly known, so that we can use information on them in the code of our protocols. In other words, parties must make their beliefs public. Indeed, this seems necessary for our feasibility results and besides we do not believe this to be problematic: even if we were in the symmetric model and just wanted to agree on one global adversary structure, it would still seem necessary to discuss beliefs in public.

We now introduce some variants of the asymmetric-trust model. The presentation here is somewhat informal; we show later in the paper how to fully formalize it using a variant of the UC framework.

Via symmetry. One approach is to define security for $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ via the usual symmetric notion. It is clear that if party p_i believes that the subsets $\mathcal{A}_i$ could be corrupted, then p_i would only be willing to participate in an $\mathcal{A}$-secure protocol π if $\mathcal{A}_i \subseteq \mathcal{A}$: if $\mathcal{A}_i \setminus \mathcal{A} \neq \emptyset$, then there exists a subset $F \subseteq \{p_1, \ldots, p_n\}$ which p_i thinks might be corrupted and which π might not tolerate being corrupted.

[4] However, we allow for the loose notation of non-monotone structures $\mathcal{A}$ in which case we actually mean the structure's monotone closure, e.g., $\mathcal{A} = \{\{p_1\}\}$ refers to the actual structure $\mathcal{A} = \{\{p_1\}, \emptyset\}$.

The definition going via symmetry insists on still giving a definition of security by specifying some subsets F against which the protocol should be secure. As argued above, to allow all parties to participate in π, any such definition would have to require π simply to tolerate the corruption structure $\mathcal{A}^* = \bigcup_{i=1}^{n} \mathcal{A}_i$. This of course gives no new views at asymmetric trust.

Via allowed consequences. A more interesting approach to trust is to say that in reality all subsets $F \subseteq \{p_1, \ldots, p_n\}$ can imaginably be corrupted. The party p_i having corruption structure $\mathcal{A}_i$ simply means that p_i thinks it very unlikely that a subset $A_i \notin \mathcal{A}_i$ will be corrupted. A reasonable security definition should therefore allow *any* corruption pattern $F \subseteq \{p_1, \ldots, p_n\}$ to occur. The goal is then (similarly to [15]) to specify for each $F \subseteq \{p_1, \ldots, p_n\}$ what consequences the corruption of F is allowed to have. These consequences should ideally be such that all p_i would be willing to participate in an $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$-secure protocol.

STRICT. In the strict notion we take the standard security definitions for broadcast, VSS, and MPC, and require that no matter what subset $F \subseteq \{1, \ldots, n\}$ is corrupted, the protocol must provide full security to all uncorrupted parties. In terms of threshold security this corresponds to $t = n$ and is unattainable for most multi-party tasks. Two-party tasks like secure communication and zero-knowledge however have strictly secure implementations, possibly using setup assumptions.

FULLY RELAXED. At the other extreme from strict security we consider fully relaxed security. From the set $F \subseteq \{p_1, \ldots, p_n\}$ of corrupted parties we define three types of parties: corrupted, naïve, foreseeing. A corrupted party is a party from F. A naïve party p_i is honest (not from F) but it happens that $F \notin \mathcal{A}_i$. A foreseeing party p_i is honest and has $F \in \mathcal{A}_i$. A naïve party is called naïve as it believed it very unlikely that F would be corrupted, yet it was.

The fully relaxed model requires full security (in the usual sense) for the set of foreseeing parties but no security for the naïve parties. That is, a naïve party is treated like a corrupted party (although it is not controlled by the adversary).

If $(\mathcal{A}_1, \ldots, \mathcal{A}_n) = (\mathcal{A}, \ldots, \mathcal{A})$ for some common adversary structure $\mathcal{A}$, then all parties are foreseeing (and thus protected) as long as $F \in \mathcal{A}$ and all parties are naïve (and thus unprotected) as long as $F \notin \mathcal{A}$. In this sense fully relaxed security corresponds to usual $\mathcal{A}$-security.

SEMI-RELAXED. Strict security protects even naïve parties and fully relaxed security gives no security at all to naïve parties. There are different ways to define a semi-relaxed model in-between these extremes. In general, a semi-relaxed model requires full security for the set of foreseeing parties but still some partial security (to be defined) for naïve parties.

The main reason why we consider semi-relaxed models is that, in the fully relaxed model, composition of subprotocols is difficult. A naïve party may, for instance, not be able to consistently broadcast the message it wants although it follows the protocol. Extending some security constraints to the set of all honest parties thus allows to compose protocols more easily.

2 Broadcast, VSS, and MPC with Asymmetric Trust

In this section we focus on implementing broadcast, VSS, and MPC in the point-to-point model with asymmetric trust. A first observation is that in the passive case, as well as in the active case where broadcast or a PKI is given, the symmetric bounds (summarized in the table below[5]) still hold for any of the defined asymmetric-trust models:

Theorem 1. *In the passive case and in the active case with broadcast (or PKI), broadcast, VSS, and MPC in any asymmetric model are achievable with respect to an aggregate adversary structure $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ if and only if they are achievable with respect to the structure $\mathcal{A}^* = \bigcup_{i=1}^{n} \mathcal{A}_i$ in the symmetric-trust model:*

	Passive	*Active (BC/PKI)*
Broadcast	Q^1	Q^1
(V)SS	Q^1	Q^2
MPC	Q^1 / Q^2	Q^2

Proof. The cases where there is a protocol for any structure are trivial. For all remaining cases $Q^2(\mathcal{A}^*)$ is a tight bound in the symmetric model.

- $\Leftarrow$ Trivially, if a task is achievable in the symmetric model for $\mathcal{A}^*$ then it is also achievable in any asymmetric model for aggregate structure $(\mathcal{A}^*, \ldots, \mathcal{A}^*)$ and thus for any $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that $\bigcup_{i=1}^{n} \mathcal{A}_i = \mathcal{A}^*$.
- $\Rightarrow$ Assume any protocol in the asymmetric model for some aggregate structure $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ such that $\neg Q^2(\mathcal{A}^*)$. Since each party trusts itself there must be two distinct parties p_i and p_j and adversary sets $A_i \in \mathcal{A}_i$ and $A_j \in \mathcal{A}_j$ such that $A_i \cup A_j = \mathcal{P}$, and $p_i \in A_j$ and $p_j \in A_i$. From this, we can build a two-party protocol for the same task wherein the parties distrust each other. This is done by having one party simulate p_i (and the parties in A_j) and the other one p_j (and the parties in A_i), and then execute the asymmetric protocol we assumed exists.
  For unconditionally secure MPC in the passive case this implies that two parties can securely compute the logical OR over their input bits, which is impossible [3, 18].
  For VSS (in the active case) this implies that a dealer can secret-share a value in the two-party setting such that the other party can reconstruct it during the reconstruction phase without the help of the dealer — but then it can also do so at any time after the sharing phase, which contradicts security.
  For MPC (and secure function evaluation, in particular) in the active case this implies that two parties can flip a fair coin, which is impossible [8]. □

In view of this theorem, for the rest of this section, we concentrate on the case where the adversary is active and where BC/PKI are not assumed.

[5] The only difference between computational and unconditional security occurs for MPC in the passive case where MPC for any structure is achievable with computational security but Q^2 is necessary for unconditional security.

2.1 Broadcast

Definition 1 (Broadcast w/ full relaxation). *A protocol where sender $p_s \in \mathcal{P}$ inputs $x_s \in \mathcal{D}$, and all $p_i \in \mathcal{P}$ output $y_i \in \mathcal{D}$, achieves* ***broadcast with full relaxation*** *if:*

VALIDITY: *If p_s and p_i are honest and $F \in \mathcal{A}_s \cap \mathcal{A}_i$ then $y_i = x_s$.*
CONSISTENCY: *If p_i and p_j are honest and $F \in \mathcal{A}_i \cap \mathcal{A}_j$ then $y_i = y_j$.* ◇

For the use as a subprotocol in MPC, a helpful additional property of broadcast is to demand validity independently of the sender's adversary structure. In that way, a naïve party can still consistently convey its view. We define semi-relaxed broadcast as broadcast with sender-independent validity in the following way — where we only state the different validity condition.

Definition 2 (Broadcast w/ sender-indep. validity (semi-relaxed)).

VALIDITY: *If p_s and p_i are honest and $F \in \mathcal{A}_i$ then $y_i = x_s$.* ◇

The following theorem is proven in the full version of the paper.

Theorem 2. *Broadcast with sender-independent validity for every sender $p_s \in \mathcal{P}$ is (perfectly) achievable if and only if*

$$\boxed{B^3(\underline{\mathcal{A}}) \;\equiv\; \forall \mathcal{A}_i, \mathcal{A}_j : \forall A_i \in \mathcal{A}_i, A_j \in \mathcal{A}_j, A_{ij} \in \mathcal{A}_i \cap \mathcal{A}_j : \;A_i \cup A_j \cup A_{ij} \neq \mathcal{P}.}$$

Note that $B^3(\underline{\mathcal{A}})$ is a proper relaxation of $Q^3(\underline{\mathcal{A}})$, which is necessary and sufficient in the symmetric framework. In particular, $B^3(\underline{\mathcal{A}})$ is a condition on all pairs of parties, whereas $Q^3(\underline{\mathcal{A}})$ is the condition $\forall A_i \in \mathcal{A}_i, \forall A_j \in \mathcal{A}_j, \forall A_k \in \mathcal{A}_k : A_i \cup A_j \cup A_k \neq \mathcal{P}$ on all triples of parties. Trivially, any semi-relaxed version of broadcast implies broadcast with full relaxation. Achievability under $B^3(\underline{\mathcal{A}})$ thus follows for the fully relaxed case. However, the next two results show, first that $B^3(\underline{\mathcal{A}})$ is not necessary for fully relaxed broadcast, and second, a weaker but necessary condition.

Proposition 1. *There are aggregate structures $\underline{\mathcal{A}}$ such that $\neg B^3(\underline{\mathcal{A}})$ and broadcast with full relaxation is achievable for every selection of a sender $p_s \in \mathcal{P}$.*

Proof. Consider aggregate structure $\underline{\mathcal{A}} = (\{\{p_2\},\{p_3\}\},\{\{p_1\},\{p_3\}\},\emptyset)$ among $\mathcal{P} = \{p_1,p_2,p_3\}$. If the sender is p_1 or p_2 then it can simply multi-send its input value since, with respect to p_3, validity and consistency only have to hold if nobody is corrupted. If the sender is p_3 then p_3 can send its input value to p_1 who in turn sends it to p_2. Again, validity and consistency with respect to p_3 only have to hold if no party is corrupted; parties p_1 and p_2 are trivially consistent. □

Theorem 3. *If there are structures $\mathcal{A}_i$, $\mathcal{A}_j$, and $\mathcal{A}_k$, and sets $A_{ij} \in \mathcal{A}_i \cap \mathcal{A}_j$, $A_{ik} \in \mathcal{A}_i \cap \mathcal{A}_k$, $A_{jk} \in \mathcal{A}_j \cap \mathcal{A}_k$, such that $A_{ij} \cup A_{ik} \cup A_{jk} = \mathcal{P}$ then broadcast with full relaxation is not achievable for any selection of a sender $p_s \in \mathcal{P}$.*

Proof. Along the lines of the lower-bound part of the proof of Theorem 2. □

2.2 Verifiable secret sharing (VSS)

Definition 3 (VSS w/ full relaxation). *A pair of protocols* (`Sh`, `Rec`) *wherein dealer $p_d \in \mathcal{P}$ inputs secret s in protocol* `Sh` *and every $p_i \in \mathcal{P}$ outputs s_i in protocol* `Rec` *achieves VSS with full relaxation if:*

SECRECY: *If p_d is honest and $F \in \mathcal{A}_d$ then the adversary has no information about s as long as protocol* `Rec` *has not started yet.*

CORRECTNESS: *If p_d and p_i are honest and $F \in \mathcal{A}_d \cap \mathcal{A}_i$ then p_i computes output $s_i = s$ in protocol* `Rec`*.*

COMMITMENT: *If p_i and p_j (case $i = j$ included) are honest and $F \in \mathcal{A}_i \cap \mathcal{A}_j$ then, after termination of protocol* `Sh`*, there is a value $s' \in \mathbb{F}$ such that, in protocol* `Rec`*, p_i and p_j compute output $s_i = s_j = s'$.* $\diamond$

It may be tempting to believe that fully relaxed VSS could be obtained by just running a standard VSS protocol that is secure with respect to the dealer's adversary structure $\mathcal{A}_d$. But such a protocol provides no security at all if $F \notin \mathcal{A}_d$, and hence cannot in general guarantee that the commitment property is satisfied.

We define semi-relaxed VSS as VSS with dealer-independent correctness in the following way — where we only state the conditions different from the previous definition.

Definition 4 (VSS w/ dealer-indep. correctness (semi-relaxed)).

CORRECTNESS: *If p_d and p_i are honest and $F \in \mathcal{A}_i$ then p_i computes output $s_i = s$ in protocol* `Rec`*.* $\diamond$

We derive our VSS protocols from the VSS protocol in [14]. Note that, since we are not given full-fledged broadcast, additional measures have to be taken.

Theorem 4. *Perfectly secure VSS with full relaxation is achievable for every selection of a dealer $p_d \in \mathcal{P}$ if*

$$\boxed{V^3(\underline{\mathcal{A}}) \quad \equiv \quad \forall \mathcal{A}_i, \mathcal{A}_j : \ \forall A_i \in \mathcal{A}_i, \forall A_i' \in \mathcal{A}_i, \forall A_j \in \mathcal{A}_j : \quad A_i \cup A_i' \cup A_j \neq \mathcal{P}.}$$

Proof. Follows from the protocol in Fig. 1 which is analyzed in Lemma 1. $\square$

Lemma 1. *For a given V^3-structure, the protocol in Fig. 1 achieves fully relaxed VSS with perfect security.*

Proof. SECRECY: If $F \in \mathcal{A}_d$ then the share s_k with $P_k = \mathcal{P} \setminus F \neq \emptyset$ being all honest does not get opened during the sharing phase by an honest dealer since it receives no complaints from within P_k with respect to this share (all broadcasts are valid with respect to p_d). Share s_k perfectly hides the secret.

Sharing `Sh`*:* For each maximal set $A_k \in \mathcal{A}_d$ the dealer p_d assigns a random share $s_k \in \mathbb{F}$ with the only restriction that $s = \sum_k s_k$. The dealer sends each s_k to all parties in $P_k = \mathcal{P} \setminus A_k$. Each party $p_i \in P_k$ stores s_k and $s'_k := s_k$.
For each s_k, the parties in P_k pairwisely compare their shares. If any inconsistency is detected by a party $p_i \in P_k$ it broadcasts (w/ sender-independent validity) a complaint to all parties in $\mathcal{P}$.
Now, if the dealer receives any complaint, it opens share s_k by broadcast (w/ sender-independent validity) towards $\mathcal{P}$. Party $p_i \in \mathcal{P}$ always adopts any opening by the dealer. If a party $p_i \in \mathcal{P}$ sent or received a complaint but does not see the dealer open s_k, it disqualifies the dealer and defines $s_k := 0$. Note that a party in P_k who disqualifies the dealer still stores the initial share s'_k it is holding — although, from now on, it uses $s_k = 0$ for its own computation.
A party $p_i \in A_k$ who, at this point, neither disqualified the dealer nor saw the dealer open share s_k, is called k-curious.
Reconstruction `Rec`*:* For each share s_k,

- All parties in P_k multi-send s'_k to the parties in A_k.
- All parties who are not k-curious accept s_k as the reconstructed share.
- All parties p_j who are k-curious wait for the parties $p_i \in P_k$ to send their shares. Then they search for a set $A_j \in \mathcal{A}_j$ such that all parties in $P_k \setminus A_j$ sent the same share $\hat{s}'_k$. Party p_j then accepts $s_k := \hat{s}'_k$.

Finally, all shares s_k are summed up in order to compute the reconstructed secret.

Fig. 1. Protocol VSS with full relaxation

Correctness: We show that when parties p_d and p_i are honest and $F \in \mathcal{A}_d \cap \mathcal{A}_i$ then, during reconstruction, p_i opens each share s_k (share with respect to $A_k \in \mathcal{A}$) correctly as distributed by p_d.

First, we observe that p_i does not disqualify the dealer p_d: disqualification implies a complaint sent or received by p_i — and thus also received by p_d. This forces p_d to open s_k, implying that p_i does not disqualify p_d. This implies that either p_d opened s_k during the sharing phase or that all honest parties in P_k agree on the same share $s'_k = s_k$. An opening during the sharing phase is correctly received by p_i (validity of broadcast). If p_i remains k-curious then there is the unique value $\hat{s}'_k = s_k$ such that there exists some $A_i \in \mathcal{A}_i$ with all parties in $P_k \setminus A_i$ opening the same share $\hat{s}'_k$ — since $A_d \cup A_i \cup A'_i \neq \mathcal{P}$.

Commitment: Consider two honest parties p_i and p_j such that $F \in \mathcal{A}_i \cap \mathcal{A}_j$. All information that is broadcast is thus valid and consistent with respect to p_i and p_j. We distinguish three cases.

- $p_i, p_j \in P_k$. Because of broadcast consistency, both parties either disqualify the dealer ($s_k = 0$), or accept the same initial share, or adopt the same share being opened by the dealer.
- $p_i \in P_k$, $p_j \in A_k$. Because of broadcast consistency, p_i and p_j receive exactly the same values that are broadcast. Thus either both disqualify, or both adopt, or p_i stays with his initial share whereas p_j is k-curious. In the latter case, there was no complaint and thus no conflict among

any honest parties in P_k — and thus all honest parties in P_k hold the same share s'_k. As in the correctness argument, p_j thus finds the unique value $\hat{s}'_k = s_k$ such that there is some $A_j \in \mathcal{A}_j$ with all parties in $P_k \setminus A_j$ all opening the same share $\hat{s}'_k$ — which is identical to p_i's share. Thus, commitment is also guaranteed in the latter case.

– $p_i, p_j \in A_k$. The parties both either are k-curious, or adopt the same opening, or disqualify the dealer. If they are k-curious then no honest party in P_k broadcast a complaint and thus, again, all honest parties in P_k hold the same share s'_k. Since $F \in \mathcal{A}_i \cap \mathcal{A}_j$ both parties will determine a set $A_i \subseteq F$ (and $A_j \subseteq F$, respectively) such that the parties in $P_k \setminus A_i$ ($P_k \setminus A_j$) all open the same share $\hat{s}'_k = s'_k$. □

Proposition 2. *There are aggregate structures $\underline{\mathcal{A}}$ such that $\neg V^3(\underline{\mathcal{A}})$ and VSS with full relaxation is achievable for every selection of a dealer $p_d \in \mathcal{P}$.*

Proof. Consider aggregate structure $\underline{\mathcal{A}} = (\{\{p_2\}, \{p_3\}\}, \{\{p_1\}\}, \emptyset)$ among $\mathcal{P} = \{p_1, p_2, p_3\}$. The parties can run the preprocessing protocol from [10] trying to establish a PKI with unconditional security. If it succeeds then the players can simulate broadcast and thus use the VSS protocol for dishonest minorities in, e.g., [9]. If it fails then it suffices that the dealer always reconstructs his input value whereas the other parties reconstruct some default value. □

The following theorem is proven in the full version of the paper.

Theorem 5. *Unconditionally secure VSS with dealer-independent correctness is achievable if $V^3(\underline{\mathcal{A}})$. Additionally, secrecy with respect to any $F \in \mathcal{A}^*$ can be guaranteed.*

Theorem 6. *If $\neg V^3(\underline{\mathcal{A}})$ then perfectly secure VSS with dealer-independent correctness is not achievable for every selection of a dealer $p_d \in \mathcal{P}$.*

Proof. If $n = 2$ then $\neg V^3(\underline{\mathcal{A}})$ and self-trust imply $\neg Q^2(\mathcal{A}^*)$, and impossibility follows from Theorem 1. We can therefore assume that $n \geq 3$.

With $\neg V^3(\underline{\mathcal{A}})$ there are structures $\mathcal{A}_i$ and $\mathcal{A}_j$, and sets $A_i, A'_i \in \mathcal{A}_i$ and $A_j \in \mathcal{A}_j$ with $A_i \cup A'_i \cup A_j = \mathcal{P}$. We show that there is no VSS with respect to dealer $p_d = p_j$. Note that $A_d \cup A_i \cup A'_i = \mathcal{P}$ and self-trust imply, wlog, that $p_d \in A_i$ and $p_i \in A_d$.

If such a VSS protocol existed then three parties p_δ, p_ι and p_κ could use it to simulate VSS among themselves with dealer p_δ where $(\mathcal{A}_\delta, \mathcal{A}_\iota, \mathcal{A}_\kappa) = \{\{p_\iota\}, \{\{p_\delta\}, \{p_\kappa\}\}, \emptyset\}$: p_δ simulates all parties in A_i, p_ι simulates all parties in A_d, and p_κ simulates all parties in A'_i. Now the share s_ι is not allowed to give any information about secret s but any triplet $(s_\delta, s_\iota, \cdot)$ perfectly reveals an honest dealer's correct secret and any triplet $(\cdot, s_\iota, s_\kappa)$ perfectly reveals the value a corrupted dealer was committed to. This is not possible. □

Finally, note that impossibility of broadcast implies impossibility of VSS. Thus all impossibility results for broadcast naturally extend to VSS.

2.3 Multi-party computation (MPC)

We now argue informally that, also with respect to general MPC, the asymmetric-trust model allows to tolerate strictly more than in the symmetric case. We only consider fully relaxed security, i.e., privacy and correctness only hold for foreseeing parties. This notion of MPC security is formalized in the following section.

Theorem 7. *In the fully relaxed model, there exist infinite many aggregate structures* $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ *with* $\neg Q^3(\mathcal{A}^*)$ *for which unconditionally secure MPC is achievable.*

Proof. We construct an aggregate adversary structure $\mathcal{A}$ for $n = 3T$ parties, where each individual $\mathcal{A}_i$ is such that its maximal sets have size T, but where no set of size T occurs in more than one $\mathcal{A}_i$. Clearly, for each n, there are several such structures and several of them are not Q^3. For such a structure, we can implement MPC by first running a preprocessing protocol from [11] that aims at establishing a PKI with unconditional security (as discussed earlier). This protocol in its most general form has parameters T and t, where $2T + t < n$; we choose $t = T - 1$. The protocol guarantees success if there are at most t corruptions. If there are at most T, there will be agreement on the result which is “success” or “failure.” Our solution is that, if the preprocessing is successful, we run a standard MPC protocol secure against T corruptions based on the PKI constructed. If the preprocessing fails, each party computes its output locally using its own input and default values for the other parties. As for security, note first that if there are more than T corruptions, all parties are naïve or corrupted, and security is guaranteed. If there are at most $T-1 = t$ corruptions, the preprocessing succeeds, and the protocol is secure. If there are T corruptions, either the preprocessing succeeds, in which case we are fine, as before. Otherwise, all honest parties agree that it failed. Since the corrupted set occurs in at most one of the $\mathcal{A}_i$, at most one party is foreseeing, and it may securely compute its output locally since the fully relaxed requirement only forces foreseeing parties to be consistent. All other parties are naïve or corrupt. $\square$

Again, the impossibility results for broadcast naturally extend to MPC.

3 A Generic Framework for Asymmetric Trust

Until now we gave ad-hoc definitions of asymmetric security for VSS and broadcast. We now develop a general framework for augmenting security models with asymmetric trust. The asymmetric security notions introduced above can be derived as special cases. The exposition is meant as a framework for adding asymmetric trust to protocol security models phrased via ideal functionalities and corruptions. For concreteness we consider the UC framework.

3.1 Basic UC framework

We consider protocols π for a party set $\mathcal{P} = \{p_1, \ldots, p_n\}$. A corruption pattern is a pair of subsets $\text{Pat} = (\text{Act}, \text{Pas})$, $\text{Act}, \text{Pas} \subseteq \mathcal{P}$, where $\text{Act} \subseteq \text{Pas}$; The interpretation is that the parties $p_i \in \text{Pas}$ are passively corrupted and the parties $p_i \in \text{Act}$ actively corrupted. We write $(\text{Act}, \text{Pas}) \subseteq (\text{Act}', \text{Pas}')$ to mean $\text{Act} \subseteq \text{Act}'$ and $\text{Pas} \subseteq \text{Pas}'$. An adversary structure is a set $\mathcal{A} = \{(\text{Act}, \text{Pas})\}$ of corruption patterns, where $\text{Pat} \in \mathcal{A} \wedge \text{Pat}' \subseteq \text{Pat} \Rightarrow \text{Pat}' \in \mathcal{A}$.

An ideal functionality is an ITM $\mathcal{F}$. It can receive inputs from each $p_i \in \mathcal{P}$ (p_i-inputs) and deliver outputs to each $p_i \in \mathcal{P}$ (p_i-outputs). Besides this, it can receive aux-inputs and deliver aux-outputs, thought of as inputs coming from the adversary respectively values leaked to the adversary. As an example an ideal functionality $\mathcal{F}_{\text{Com}}$ for bit commitment can be phrased as follows: On p_i-input $(\texttt{commit}, cid, p_i, p_j, m \in \{0,1\})$ produce aux-output $(\texttt{commit}, cid, p_i, p_j)$; Here cid is a commitment identifier. On a later aux-input $(\texttt{deliver}, cid, p_i, p_j)$, output $(\texttt{receipt}, cid, p_i, p_j)$ to p_j. On p_i-input $(\mathsf{open}, cid, p_i, p_j)$ after receiving p_i-input $(\texttt{commit}, cid, p_i, p_j, m)$, produce aux-output $(\mathsf{open}, cid, p_i, p_j, m)$. On a later aux-input $(\mathsf{open}, cid, p_i, p_j)$, output $(\mathsf{open}, cid, p_i, p_j, m)$ to p_j.

A protocol π consists of n parties $p_1, \ldots, p_n$ and some ideal functionalities $\mathcal{G}$, which might, e.g., model point-to-point lines or commitment. We write $\mathcal{G} \in \pi$ and $\pi[\mathcal{G}]$ to mean that π uses the ideal functionality $\mathcal{G}$. An environment $\mathcal{Z}$ for π is a ITM which gives inputs to the parties and gets outputs from the parties. We denote an execution of π in $\mathcal{Z}$ by $\text{Exec}_{\pi,\mathcal{Z}}$. The environment $\mathcal{Z}$ also corrupts parties.[6] For a corruption pattern $\text{Pat} = (\text{Act}, \text{Pas})$ the environment is allowed to see the internal state of $p_i \in \text{Pas}$ and control $p_i \in \text{Act}$: When $p_i \in \text{Act}$, then in $\text{Exec}_{\pi,\mathcal{Z}}$ it is $\mathcal{Z}$ which determines all p_i-inputs to $\mathcal{G} \in \pi$ and receives all p_i-outputs from $\mathcal{G} \in \pi$. The party p_i is not run at all. Besides this, $\mathcal{Z}$ receives all aux-outputs from all $\mathcal{G} \in \pi$ and can give aux-inputs to all $\mathcal{G} \in \pi$. As an example, in $\text{Exec}_{\pi[\mathcal{F}_{\text{Com}}],\mathcal{Z}}$ the environment sees when commitments are made and determines when to deliver receipts and openings.

The execution $\text{Exec}_{\pi,\mathcal{Z}}$ is compared to a simulation $\text{Sim}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. Here the simulator $\mathcal{S}$ must simulate an execution of π. E.g., $\mathcal{S}$ simulates aux-outputs to $\mathcal{Z}$ from all $\mathcal{G} \in \pi$ and receives aux-inputs from $\mathcal{Z}$ to $\mathcal{G} \in \pi$. The simulator itself receives aux-outputs from $\mathcal{F}$ and gives aux-inputs to $\mathcal{F}$. When $\mathcal{Z}$ gives a p_i-input for $p_i \notin \text{Act}$, it is given to $\mathcal{F}$. The simulator gives all p_i-inputs to $\mathcal{F}$ for $p_i \in \text{Act}$. When $\mathcal{F}$ produces a p_i-output for $p_i \notin \text{Act}$, it is given to $\mathcal{Z}$, but when $\mathcal{F}$ produces a p_i-output for $p_i \in \text{Act}$, it is not given to $\mathcal{Z}$. When $\mathcal{Z}$ gives a p_i-input to $\mathcal{F}$ for $p_i \in \text{Pas}$, it is shown to $\mathcal{S}$, and when $\mathcal{F}$ produces a p_i-output for $p_i \in \text{Pas}$, it is shown to $\mathcal{S}$.

A protocol π is called a UC secure implementation of $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that $\text{Sim}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{Exec}_{\pi,\mathcal{Z}}$ for all $\mathcal{Z}$. It is possible to restrict $\mathcal{Z}$ to corrupting according to some $\text{Pat} \in \mathcal{A}$, in which case we say that π is $\mathcal{A}$-secure (in the symmetric sense).

[6] We use the formulation of the UC framework without an explicit adversary, see full version of [5].

$\mathcal{F}^{(\mathrm{Xtn})}$ runs a copy of $\mathcal{F}$. When $\mathcal{F}$ produces aux-output z, $\mathcal{F}^{(\mathrm{Xtn})}$ produces aux-output z, and when $\mathcal{F}^{(\mathrm{Xtn})}$ receives aux-input z it gives $\mathcal{F}$ the aux-input z. When the (current) corruption pattern is $\textsc{Pat}$ and $\mathrm{Xtn}(\textsc{Pat}) = (\textsc{ActIn}, \textsc{ActOut}, \textsc{PasIn}, \textsc{PasOut})$, the remaining inputs and outputs are handled as follows:

- For $p_i \in \textsc{PasIn}$: On p_i-input x, produce aux-output $(\mathtt{in}, p_i, x)$, and then give $\mathcal{F}$ the p_i-input x.
- For $p_i \in \textsc{PasOut}$: On p_i-output x from $\mathcal{F}$, produce aux-output $(\mathtt{out}, p_i, x)$, and then produce the p_i-output x.
- For $p_i \in \textsc{ActIn}$: Ignore all p_i-inputs, and on an aux-input $(\mathtt{in}, p_i, x)$, give $\mathcal{F}$ the p_i-input x.
- For $p_i \in \textsc{ActOut}$: Ignore all p_i-outputs from $\mathcal{F}$, and on an aux-input $(\mathtt{out}, p_i, x)$, produce the p_i-output x.

Fig. 2. $\mathcal{F}^{(\mathrm{Xtn})}$

3.2 Modeling the security loss of naïve parties

To define asymmetric trust in the UC framework, we need to model the loss of security we will allow for a party who turns out to have been naïve. To express what we choose to allow, we introduce the concept of a corruption extension Xtn which is a function that maps a corruption pattern $\textsc{Pat}$ to a tuple

$$\mathrm{Xtn}(\textsc{Pat}) = (\textsc{ActIn}, \textsc{ActOut}, \textsc{PasIn}, \textsc{PasOut}),$$

of party subsets, where

$$\textsc{PasIn}, \textsc{PasOut} \subseteq \mathcal{P} \setminus \textsc{Pas} \text{ and } \textsc{ActIn}, \textsc{ActOut} \subseteq \mathcal{P} \setminus \textsc{Act}.$$

These are subsets of parties who are not corrupt but nevertheless have their security violated in some way.

This is modeled in the simulation $\textsc{Sim}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ by giving $\mathcal{S}$ the following extra power over $\mathcal{F}$: For the parties $p_i \in \textsc{PasIn}$, respectively $p_i \in \textsc{PasOut}$, we show $\mathcal{S}$ the p_i-inputs to $\mathcal{F}$, respectively the p_i-outputs from $\mathcal{F}$. For the parties $p_i \in \textsc{ActIn}$, when $\mathcal{Z}$ gives a p_i-input to $\mathcal{F}$, it is not given to $\mathcal{F}$. Instead we allow $\mathcal{S}$ to specify these p_i-inputs. Finally, for the parties $p_i \in \textsc{ActOut}$, when $\mathcal{F}$ produces a p_i-output to $\mathcal{Z}$, it is not given to $\mathcal{Z}$ but we allow $\mathcal{S}$ to specify these p_i-outputs.

Of course, a functionality $\mathcal{F}$ may also be used as an auxiliary functionality in a protocol. In this case the extra power is given to the environment (adversary), see more on this below.

In order to formally incorporate the above into the UC framework without making changes that require us to reprove the composition theorem, we define the following way to extend any ideal functionality: For a functionality $\mathcal{F}$ and any extension Xtn, let $\mathcal{F}^{(\mathrm{Xtn})}$ be the ideal functionality in Fig. 2. We say that π is an Xtn-secure implementation of $\mathcal{F}$ if π is a UC secure implementation of $\mathcal{F}^{(\mathrm{Xtn})}$ (tolerating environments corrupting any subset of parties).

Note that in $\textsc{Sim}_{\mathcal{F}^{(\mathrm{Xtn})},\mathcal{Z}}$ it is $\mathcal{S}$ which has access to the aux-inputs and aux-outputs of $\mathcal{F}^{(\mathrm{Xtn})}$, giving it exactly the desired extra power. Note also that

$\mathcal{F}^{(\mathrm{Xtn})}$ is allowed in the UC framework since it allows functionalities to know which parties are corrupted.

Note that in the above definition we do not restrict in any way how many parties the environment corrupts! We might however use Xtn to specify that for some corruptions $A \notin \mathcal{A}$ the simulator is allowed to corrupt all parties in the simulation. This allows to model that for corruptions $A \notin \mathcal{A}$ no security guarantees are given. The notion of corruption extensions therefore subsumes the normal notion of restricting the environment to certain corruption patterns $\mathcal{A}$.

As mentioned, extensions also apply to a functionality $\mathcal{G}$ used in protocol π. For this purpose we assume that π associates to each $\mathcal{G} \in \pi$ an extension $\mathrm{Xtn}_{\mathcal{G}}$. We then let $\widehat{\pi}$ denote the protocol where each $\mathcal{G} \in \pi$ is replaced by $\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}$. In $\textsc{Exec}_{\widehat{\pi},\mathcal{Z}}$ it is $\mathcal{Z}$ which has access to the aux-inputs and outputs of $\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}$, granting it extra power over $\mathcal{G}$ in the same way as we did for the simulator before.

Definition 5. *Let π be a protocol having an extension* $\mathrm{Xtn}_{\mathcal{G}}$ *associated to each $\mathcal{G} \in \pi$, let* Xtn *be some extension and let $\mathcal{F}$ be some ideal functionality. We say that π is an* Xtn*-secure implementation of $\mathcal{F}$ if $\widehat{\pi}$ is a UC secure implementation of $\mathcal{F}^{(\mathrm{Xtn})}$ (tolerating all corruption patterns).* $\diamond$

We can prove a composition theorem for this notion of security. For a protocol $\pi = \pi[\mathcal{G}]$ and a protocol γ we use $\pi[\gamma/\mathcal{G}]$ to denote the protocol π where the use of $\mathcal{G}$ has been replaced by γ. Let Xtn^{π} (Xtn^{γ}) be the extensions π (γ) associates to its ideal functionalities. For $\mathcal{H} \in \pi[\gamma/\mathcal{G}]$ we associate the extension $\mathrm{Xtn}(\mathcal{H}) = \mathrm{Xtn}^{\pi}(\mathcal{H})$ when $\mathcal{H} \in \pi$ and $\mathrm{Xtn}(\mathcal{H}) = \mathrm{Xtn}^{\gamma}(\mathcal{H})$ when $\mathcal{H} \in \gamma$.

Theorem 8. *Assume that π is an* Xtn*-secure implementation of $\mathcal{F}$ and $\mathcal{G} \in \pi$ with $\mathrm{Xtn}^{\pi}(\mathcal{G}) = \mathrm{Xtn}_{\mathcal{G}}$. Assume furthermore that γ is an $\mathrm{Xtn}_{\mathcal{G}}$-secure implementation of $\mathcal{G}$. Then $\pi[\gamma/\mathcal{G}]$ is an* Xtn*-secure implementation of $\mathcal{F}$.*

Proof. When $\mathrm{Xtn}^{\pi}(\mathcal{G}) = \mathrm{Xtn}_{\mathcal{G}}$ for $\mathcal{G} \in \pi$, then π being an Xtn-secure implementation of $\mathcal{F}$ implies that $\widehat{\pi}[\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}]$ is a UC secure implementation of $\mathcal{F}^{(\mathrm{Xtn})}$. That γ is an $\mathrm{Xtn}_{\mathcal{G}}$-secure implementation of $\mathcal{G}$ implies that $\widehat{\gamma}$ is a UC secure implementation of $\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}$. So, by the UC composition theorem, $\widehat{\pi}[\widehat{\gamma}/\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}]$ is a UC secure implementation of $\mathcal{F}^{(\mathrm{Xtn})}$. Since $\widehat{\pi[\gamma/\mathcal{G}]} = \widehat{\pi}[\widehat{\gamma}/\mathcal{G}^{(\mathrm{Xtn}_{\mathcal{G}})}]$ this implies that $\widehat{\pi[\gamma/\mathcal{G}]}$ is a UC secure implementation of $\mathcal{F}^{(\mathrm{Xtn})}$ which by definition implies that $\pi[\gamma/\mathcal{G}]$ is an Xtn-secure implementation of $\mathcal{F}$. $\square$

3.3 Asymmetric trust

We now use Definition 5 to express asymmetric trust, formalizing the concepts we introduced in Section 1.4. To each p_i we associate an adversary structure $\mathcal{A}_i$ expressing that p_i trusts that only corruption patterns $\textsc{Pat} \in \mathcal{A}_i$ will actually occur. We call $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ an aggregate adversary structure. A symmetric adversary structure $\mathcal{A}$ corresponds to the aggregate adversary structure $\underline{\mathcal{A}}^n = (\mathcal{A}, \ldots, \mathcal{A})$. For an actually occurring corruption pattern $\textsc{Pat} =$

$(\textsc{Act}, \textsc{Pas})$ we let $\textsc{Foreseeing}_{\underline{\mathcal{A}}}(\textsc{Pat}) = \{p_i \in \mathcal{P} \setminus \textsc{Pas} | \textsc{Pat} \in \mathcal{A}_i\}$ and we let $\textsc{Naïve}_{\underline{\mathcal{A}}}(\textsc{Pat}) = \{p_i \in \mathcal{P} \setminus \textsc{Pas} | \textsc{Pat} \notin \mathcal{A}_i\}$. We call $p_i \in \textsc{Foreseeing}_{\underline{\mathcal{A}}}(\textsc{Pat})$ foreseeing and we call $p_i \in \textsc{Naïve}_{\underline{\mathcal{A}}}(\textsc{Pat})$ naïve. We model asymmetric trust by treating the foreseeing honest parties as we normally treat the honest parties in UC security and treating the corrupted parties as we do normally. For the naïve parties we allow the simulator (environment) extra powers, using the concepts we defined earlier. Formally, we say that a corruption extension Xtn is an extension for $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ if it holds for all $\textsc{Pat}$ that $\mathrm{Xtn}(\textsc{Pat}) = (\textsc{ActIn}, \textsc{ActOut}, \textsc{PasIn}, \textsc{PasOut})$ satisfies $\textsc{PasIn}, \textsc{PasOut}, \textsc{ActIn}, \textsc{ActOut} \subseteq \textsc{Naïve}_{\underline{\mathcal{A}}}(\textsc{Pat})$. This gives a lot of granularity in how to treat the honest-but-naïve parties: $\textsc{PasIn}$ specifies the naïve parties for which the inputs are allowed to leak to the adversary, $\textsc{PasOut}$ specifies the naïve parties for which the outputs are allowed to leak to the adversary, $\textsc{ActIn}$ specifies the naïve parties for which the inputs might be controlled by the adversary, and $\textsc{ActOut}$ specifies the naïve parties for which the outputs might by controlled be the adversary.

To get some more structure, we name some special types of extensions, called `relaxed`, `semi-relaxed`, `strong`, `strict`, which are defined as follows:

- Xtn is of type `relaxed` if it is the extension of $\underline{\mathcal{A}}$ that specifies $\textsc{ActIn} = \textsc{ActOut} = \textsc{Naïve}_{\underline{\mathcal{A}}}(\textsc{Pat})$ for all $\textsc{Pat}$, i.e., there is no security for naïve parties.
- Xtn is of type `semi-relaxed` if it is the extension of $\underline{\mathcal{A}}$ that specifies $\textsc{ActIn} = \emptyset, \textsc{ActOut} = \textsc{Naïve} \setminus \textsc{Act}, \textsc{PasIn} = \textsc{Naïve} \setminus \textsc{Pas}, \textsc{PasOut} = \textsc{Naïve} \setminus \textsc{Pas}$. I.e., the honest-but-naïve parties are guaranteed that their inputs are contributed correctly to the computation. They are however not guaranteed to receive correct outputs nor any privacy of their inputs or their outputs.
- Xtn is of type `strong` if it is the extension of $\underline{\mathcal{A}}$ that specifies $\textsc{ActIn} = \textsc{ActOut} = \emptyset,\ \textsc{PasIn} = \textsc{PasOut} = \textsc{Naïve}_{\underline{\mathcal{A}}}(\textsc{Pat})$, i.e., naïve parties have no privacy but may contribute their inputs and get correct results.
- Xtn is of type `strict` if it is the extension of $\underline{\mathcal{A}}$ that specifies $\mathrm{Xtn}_{\mathcal{G}}(\textsc{Pat}) = (\emptyset, \emptyset, \emptyset, \emptyset)$, i.e., there is full security for naïve parties.

If $\textsc{atk}$ is one of `relaxed`, `semi-relaxed`, `strong`, `strict`, we call π an $\textsc{atk}$ $\underline{\mathcal{A}}$-secure implementation of $\mathcal{F}$ if π is an Xtn-secure implementation of $\mathcal{F}$ tolerating $\underline{\mathcal{A}}$, where Xtn is the extension of $\underline{\mathcal{A}}$ of type $\textsc{atk}$. Also, if π makes use of functionality $\mathcal{G}$, we say that $\mathcal{G}$ is an $\textsc{atk}$ functionality if the extension π assigns to $\mathcal{G}$ is of type $\textsc{atk}$.

The following composition theorem is an immediate corollary to Theorem 8.

Corollary 1. *Let* $\textsc{atk}, \textsc{atk}' \in \{$`relaxed`, `semi-relaxed`, `strong`, `strict`$\}$. *If* π *is an* $\textsc{atk}$ $\underline{\mathcal{A}}$*-secure implementation of* $\mathcal{F}$*, where* $\mathcal{G} \in \pi$ *is an* $\textsc{atk}'$ *functionality, and* γ *is an* $\textsc{atk}'$ $\underline{\mathcal{A}}$*-secure implementation of* $\mathcal{G}$*, then* $\pi[\gamma/\mathcal{G}]$ *is an* $\textsc{atk}$ $\underline{\mathcal{A}}$*-secure implementation of* $\mathcal{F}$*.*

Note that the notion of semi-relaxed security as defined here is equivalent to the notions sender-independent validity and dealer-independent correctness in Section 2. Indeed, defining broadcast and VSS by requiring a `semi-relaxed` secure implementation of a corresponding ideal functionality would define exactly these notions.

To see the connection between symmetric security and our notions of asymmetric security, let $\underline{\mathcal{A}}$ be the aggregate adversary structure modeling that all parties trust that at most t parties will be corrupted, and let π be a protocol using only strict functionalities. In this case the simulator is given no extra corruption when at most t parties are corrupted, as all parties are foreseeing. So, as long as at most t parties are corrupted, relaxed, semi-relaxed, strong and strict $\underline{\mathcal{A}}$-security are equivalent to the usual UC t-security. If however more than t parties are corrupted, then all honest parties are naïve, meaning e.g. that strong security allows the simulator to see the inputs and outputs of all parties, and relaxed security allows the simulator to specify the inputs and outputs of all parties. So, when more than t parties are corrupted, strong $\underline{\mathcal{A}}$-security gives no guarantees on the privacy of any party but still guarantees correctness for the honest-but-naïve parties, and relaxed $\underline{\mathcal{A}}$-security gives no guarantees at all. Note that giving no guarantees at all when more than t parties are corrupted is equivalent to normal t-security, where simulation is only required for environments corrupting at most t parties. Therefore relaxed security is a generalization of normal (symmetric) UC security, and strong security is a strengthening.

4 Multi-Party Computation in the UC Framework

In this section we first formalize the notion of secure multi-party computation in the UC framework where the parties have asymmetric trust in each other. In Section 2.3 we already informally looked at this case in the secure-channels model. In Section 4.2 we look at a setting where a number of certificate authorities (or common reference strings) are present and where the parties have asymmetric trust in these certificate authorities (or common reference strings).

4.1 Secure function evaluation

For simplicity we focus on secure function evaluation (SFE). SFE of $f(x_1, \ldots, x_n)$ can be expressed as securely evaluating the ideal functionality $\mathcal{F}_{\mathrm{SFE}}^{f}$ for secure function evaluation of f. Essentially $\mathcal{F}_{\mathrm{SFE}}^{f}$ takes an input x_i from each p_i, computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and outputs y_i securely to p_i. We call π an $\underline{\mathcal{A}}$ SFE w/ full relaxation of f if π is a `relaxed` $\underline{\mathcal{A}}$-secure implementation of $\mathcal{F}_{\mathrm{SFE}}^{f}$. We call π an $\underline{\mathcal{A}}$ SFE w/ contributor-independent correctness of f if π is a `semi-relaxed` $\underline{\mathcal{A}}$-secure implementation of $\mathcal{F}_{\mathrm{SFE}}^{f}$. For concreteness we flesh out these notions below.

Definition 6 (SFE w/ full relaxation). *The simulator has the following extra powers:*

Input Secrecy: *If p_i is honest and $F \notin \mathcal{A}_i$ or p_i is corrupt, then the simulator sees x_i. If party p_i is honest and $F \in \mathcal{A}_i$ then the simulator is not shown x_i.*

Input Correctness: *If p_i is honest and $F \notin \mathcal{A}_i$ or p_i is corrupt, then the simulator can replace x_i by some x'_i. After this, the outputs $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$ are computed, where $x'_i = x_i$ for all honest p_i with $F \in \mathcal{A}_i$.*

Output Secrecy: *If p_i is honest and $F \notin \mathcal{A}_i$ or p_i is corrupt, then the simulator sees y_i. If party p_i is honest and $F \in \mathcal{A}_i$ then the simulator is not shown y_i.*

Output Correctness: *If p_i is honest and $F \notin \mathcal{A}_i$ or p_i is corrupt, then the simulator can replace y_i by some y'_i. After this, $\mathcal{F}^f_{\mathrm{SFE}}$ outputs y'_i on behalf of p_i, where $y'_i = y_i$ for all honest p_i with $F \in \mathcal{A}_i$.*

Robustness: *Robustness is best expressed as a condition on the protocol (as opposed to the simulation), by requiring that all honest parties compute an output, i.e., no honest party aborts the protocol. Alternatively, one can require this only for the foreseeing parties, getting* weak robustness. $\diamond$

Definition 7 (SFE w/ contributor-independent correctness). *The simulator has the following extra powers (listing only differences from Definition 6):*

Input Correctness: *If p_i is corrupt, then the simulator can replace x_i by some x'_i. After this, the outputs $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$ are computed, where $x'_i = x_i$ for all honest p_i.* $\diamond$

These notions can be generalized to MPC w/ full relaxation and MPC w/ contributor-independent correctness by requiring a `relaxed` (`semi-relaxed`) $\underline{\mathcal{A}}$-secure implementation of a more general ideal functionality $\mathcal{F}$.

4.2 With asymmetrically trusted setup

We now consider a setting where some setup is given. We focus on UC security, where setup is needed when there is no trust among the parties. We consider two setup assumptions which have been studied previously: common reference string (CRS) and key registration (KR), and we generalize the study to consider asymmetric trust. Here, we only cover the KR case whereas the CRS case is treated in the full version of the paper, using similar techniques.

Key Registration. In [1] Barak *et al.* gave a feasibility result for UC secure MPC in a network which had a key registration service $\mathcal{F}_{\mathrm{KR}}$ which allows a user U_i to register a public key pk_i while checking that U_i knows a corresponding secret key. We extend this analysis of the power of key registration by analyzing a setting where there are several key registration services (KRS's) in which the users have different partial trust. For completeness we also assume that the users have different, partial trust in each other. We characterize the aggregate adversary structures which allow to securely compute any ideal functionality in this setting. We consider the same type of security as in [1, 6]: polynomial time security and the protocol is only required to deliver outputs if all parties are honest. This is modeled by allowing the simulator to decide when and if honest outputs from $\mathcal{F}$ to $\mathcal{Z}$ are delivered in $\textsc{Sim}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

We model the KRS's as parties $\mathcal{KR} = \{KR_k\}$. We then add n users $\mathcal{U} = \{U_1, \ldots, U_n\}$, making the party set $\mathcal{P} = \mathcal{KR} \cup \mathcal{U}$. The users are the parties which want to compute some ideal functionality $\mathcal{F}_\mathcal{U}$ among them.[7] We also add a strict functionality for authenticated, asynchronous point-to-point communication among the users and between the users and the KRS's, and we add a strict functionality for secure, asynchronous point-to-point communication among the users.[8] Finally we need that each user can give a proof of possession (PoP) of the secret key when it registers a public key. For this purpose we postulate an ideal functionality $\mathcal{POP}$. Let gen be the generator $pk = \text{gen}(r)$ used to generate public keys (we can wlog assume that the randomness r constitutes the private key). We assume that $\mathcal{POP}$ behaves as follows: On input (U_i, KR_k, r_i) from U_i, output $(U_i, KR_k, pk_i = \text{gen}(r_i))$ to KR_k. This models that U_i gives pk_i to KR_k and then somehow proves knowledge of r_i such that $pk_i = \text{gen}(r_i)$.

By U_i registering a public key at KR_k we then mean that U_i samples a random public key $pk_i \leftarrow \text{gen}(r_i)$ and inputs (U_i, KR_k, r_i) securely to $\mathcal{POP}$. The honest behavior of each KR_k is as follows: The first time it sees $\mathcal{POP}$ output (U_i, KR_k, pk_i) for U_i it sends (U_i, KR_k, pk_i) to all users U_j using authenticated point-to-point communication.

Since the behavior of KR_k is fixed and the behavior of $\mathcal{POP}$ is given by gen, we specify a protocol by $\pi = (\text{gen}, U_1, \ldots, U_n)$. For convenience we assume that each U_i starts the protocol by registering some $pk_{i,k}$ with each KR_k. Then U_i waits for each KR_k to send some $pk_{j,k}$ for each $U_j \in \mathcal{U}$ and stores all these keys. After this registration phase the users then proceed to run the actual protocol. We can therefore in the specification of p_i assume that it knows the keys $pk_{k,j}$. We call such a $\pi = (\text{gen}, U_1, \ldots, U_n)$ a KR-protocol.

As for trust, we consider only active corruptions, so that $\textsc{Pat} = (\textsc{Act}, \textsc{Act})$ for all patterns. We therefore write $\textsc{Act} \in \mathcal{A}$ and consider $\mathcal{A} \in 2^{\mathcal{P}}$. We associate no trust to the KRS's. That is, we assume that $\mathcal{A}_{KR_k} = 2^{\mathcal{P}}$ for each KRS. To each U_i we associate a corruption structure $\mathcal{A}_i \subseteq 2^{\mathcal{P}}$. We call $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ complete for the KR setting if it allows to securely compute any efficient ideal functionality $\mathcal{F}_\mathcal{U}$ among the users using a KR protocol.

For $\textsc{Act}_i \in \mathcal{A}_i$ we let $\textsc{Act}_i^{\mathcal{U}} = \textsc{Act}_i \cap \mathcal{U}$ and $\textsc{Act}_i^{\mathcal{KR}} = \textsc{Act}_i \cap \mathcal{KR}$. We say that two users $U_i \neq U_j$ are KR connected if it holds for all $\textsc{Act}_i \in \mathcal{A}_i$ and $\textsc{Act}_j \in \mathcal{A}_j$ that either $\textsc{Act}_i^{\mathcal{KR}} \neq \mathcal{KR}$ or $\textsc{Act}_j^{\mathcal{KR}} \neq \mathcal{KR}$ or $\textsc{Act}_i^{\mathcal{U}} \cup \textsc{Act}_j^{\mathcal{U}} \neq \mathcal{U}$. That is, together, U_i and U_j cannot imagine a scenario where both of them think that all KRS's might be corrupted and where together they think all users might be corrupted.

[7] We say that $\mathcal{F}_{\mathcal{P}'}$ is among $\mathcal{P}'$ if it ignores p_i-inputs for $\mathcal{P} \setminus \mathcal{P}'$ and gives no p_i-outputs for $\mathcal{P} \setminus \mathcal{P}'$.

[8] Since secure, asynchronous point-to-point communication has a normal UC secure implementation given authenticated channels and several standard complexity assumptions, this strict ideal functionality can be replaced with any such implementation to get an equivalent model with only authenticated communication, using Corollary 1.

Theorem 9. *An aggregate adversary structure $\underline{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ is complete for the KR setting iff all pairs of distinct users are KR connected in $\underline{\mathcal{A}}$.*

The proof of Theorem 9 is given in the full version of the paper. A special case of Theorem 9 is when the users have no trust in each other ($\mathcal{A}_i^{\mathcal{U}} = \mathcal{U}$ for all U_i) in which case the condition can be phrased as: There exists at most one user who thinks that all KRS's can be corrupted.

Conclusion

We proposed a notion of asymmetric trust in protocol security and gave a general definition of asymmetric secure MPC and gave specialized definitions of asymmetric secure broadcast, VSS, and SFE. We explored the feasibility of broadcast, VSS, and MPC in various models with asymmetric trust. A tight characterization of the feasibility of broadcast has been found for asymmetric trust, and nontrivial upper and lower bounds for VSS, and we have shown how to tolerate strictly stronger adversaries in MPC than with symmetric trust. It is an open problem to completely characterize the aggregate adversary structures that allow for MPC in the case with active adversaries and no set-up.

References

1. B. Barak, R. Canetti, J. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195. IEEE, 2004.
2. B. Baum-Waidner, B. Pfitzmann, and M. Waidner. Unconditional Byzantine agreement with good majority. In *8th Annual Symposium on Theoretical Aspects of Computer Science*, volume 480 of *LNCS*, 1991.
3. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.
4. G. R. Blakley. Safeguarding cryptographic keys. In *1979 National Computer Conference*, volume 48 of *AFIPS Conference proceedings*. AFIPS Press, 1979.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001. Full version in IACR ePrint Archive 2000/067.
6. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 494–503, 2002.
7. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th IEEE Symposium on the Foundations of Computer Science*, pages 383–395, 1985.
8. R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM Symposium on the Theory of Computing*, pages 364–369, 1986.
9. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multi-party computations secure against an adaptive adversary. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, 1999.

10. M. Fitzi, N. Gisin, U. Maurer, and O. von Rotz. Unconditional Byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *Advances in Cryptology — EUROCRYPT '02*, volume 2332 of *LNCS*, pages 482–501, 2002.
11. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable Byzantine agreement secure against faulty majorities. In *21st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 118–126, 2002.
12. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.
13. M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.
14. U. Maurer. Secure multi-party computation made simple. In *Third Conference on Security in Communication Networks*, volume 2576 of *LNCS*, pages 14–28, 2002.
15. U. Maurer. Towards a theory of consistency primitives. In *Distributed Computing (DISC)*, volume 3274 of *LNCS*, pages 379–389, 2004.
16. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.
17. B. Pfitzmann and M. Waidner. Information-theoretic pseudosignatures and Byzantine agreement for $t >= n/3$. Technical Report RZ 2882 (#90830), IBM Research, 1996.
18. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on the Theory of Computing*, pages 73–85, 1989.
19. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
20. A. Yao. Protocols for secure computations. In *23rd IEEE Symposium on the Foundations of Computer Science*, pages 160–164. IEEE, 1982.