Universally Composable Incoercibility[*]

Dominique Unruh
Saarland University

Jörn Müller-Quade
Universität Karlsruhe

October 27, 2009

Abstract

We present the UC/c framework, a general definition for secure and incoercible multi-party protocols. Our framework allows to model arbitrary reactive protocol tasks (by specifying an ideal functionality) and comes with a universal composition theorem. We show that given natural setup assumptions, we can construct incoercible two-party protocols realising arbitrary functionalities (with respect to static adversaries).

Contents

[*]Partially funded by the Cluster of Excellence "Multimodal Computing and Interaction". Errata will be published on `http://crypto.m2ci.org/unruh/publications/errata/ucc-errata.pdf`.

1 Introduction

Commonly, security of a cryptographic protocol encompasses (very roughly) two aspects: The protocol should guarantee that the private data of the parties stays secret (privacy), and it should ensure that all data transferred or computed is correct (integrity). Most security definitions ensure one or both of these requirements, and many protocols are known to satisfy these definitions (e.g., [GMW87, BOGW88, CCD88, CFGN96, CLOS02]).

There is, however, a requirement that does not fall into either category: coercion resistance (first noted by [Her91, BT94]). To illustrate this property, we use the example of a voting scheme. In a voting scheme, it might be possible for a voter to acquire a receipt that he cast a specific vote. This does not violate the anonymity of the voter since the voter is not required to reveal or even acquire such a receipt. Thus privacy is maintained. And getting a receipt does not allow to falsify the outcome of the election. Thus the integrity of the scheme is maintained. Yet the mere possibility of acquiring a receipt may make a party coercible. A coercive adversary may threaten certain reprisals if the party does not cast a specific vote and proves this by delivering a receipt to the adversary. Thus such an election protocol would not be coercion resistant (short: incoercible).

Incoercibility is an important property in any setting in which some malicious agent has the power to harm and thus threaten other protocol participants. Clearly, this is not restricted to the setting of voting but may be the case in other settings, too (e.g., when financial transactions are involved). Unfortunately, incoercibility turns out to be both difficult to define and to achieve.

Previous definitions of incoercibility are usually restricted to special domains such as voting (e.g., [BT94, JCJ05, DKR09]). An exception are the models by Canetti and Gennaro [CG96] and by Moran and Naor [MN06] which give general definitions of incoercible multi-party computation. Their definitions are, however, restricted to the case of secure function evaluation. That is, they only consider protocols in which all parties need to first contribute their inputs, and then from these inputs the outputs for the parties are computed. Reactive protocols, protocols that have multiple phases and where the inputs in one phase can depend on the outputs of an earlier phase, are excluded. For example, the security of a commitment protocol could not be modelled in their settings.

Besides the problem of reactive protocols, the issue of composability arises. When building a complex protocol, it is often necessary to abstract from certain subprotocols in the analysis to make the analysis manageable. For example, one might first analyse the protocol assuming a perfectly secure mechanism for performing commitments (modelled by a trusted machine), and then later on prove the security of the subprotocol that is actually used for the commitments. To do so, and also to have a guarantee that the protocol does not become insecure when executed together with other protocols or instances of itself, one needs a security notion that comes with a composition theorem.

In the case of normal secure multi-party computation (i.e., without incoercibility) both the problem of modelling reactive protocols and of giving strong compositionality guarantees has been solved by Canetti's UC model [Can01]. In this model, we can define a protocol task by specifying a trusted machine, the ideal functionality, which by definition performs the required protocol task. Since this machine can interact with its environment in arbitrary ways, the security of very general reactive protocols can be modelled. Furthermore, the UC model guarantees that if a protocol is secure when using (as opposed to realising) an ideal functionality, then the protocol stays secure when instead of the ideal functionality, a subprotocol that securely realises the ideal functionality is used. The UC model, however, does not guarantee incoercibility.

Our contribution. We define the Composable Incoercibility framework (UC/c) which is an extension of the UC framework. Like UC, UC/c allows to model very general reactive protocol tasks and gives strong compositionality guarantees (universal composition). Additionally, protocols secure with respect to UC/c are incoercible. To illustrate the model, we show that a voting scheme that is UC/c secure is also incoercible with respect to a definition tailored specifically to voting. Finally, we show that in the restricted case of static coercions/deceptions (all corruptions and coercions happen at the beginning of the protocol), arbitrary UC/c secure two-party computation is possible assuming the availability of secure channels.

Organisation. In Section 1.1, we explain the intuition behind the UC/c framework. In Section 2 we define the UC/c framework and present the universal composition theorem. In Section 3 we illustrate our model by applying it to the setting of voting protocols. In Section 4 we show that UC/c secure two-party protocols exist for arbitrary functionalities. In Section 5 we give directions for further work.

1.1 The intuition behind UC/c

To understand the UC/c model, we first need to get an intuition of how incoercibility is achieved. The goal of an incoercible protocol is the following: When an adversary tries to coerce a party into performing a certain action (such as casting a particular vote v^*), the party should be able to perform the action it originally intended to perform (casting a vote v) without the adversary noticing. That is, the adversary should not be able to tell the difference between a party P that follows the adversary's instructions (a *corrupted* party, casting the vote v^*) and a party P that only tries to make the adversary believe that it follows the adversary's instructions (a *deceiving* party, casting the vote v and giving fake evidence to the adversary that it cast the vote v^*).

The most natural way to define incoercibility would be to require that the adversary cannot distinguish between a coerced and a deceiving party. This, however, usually cannot be achieved. For example, in a voting protocol the adversary will eventually learn the tally. The distribution of the tally will, since there are only polynomially many voters, slightly but noticeably change when the vote of P changes from v to v^*. The adversary can hence distinguish coerced and deceiving parties by observing the tally.

Thus, we have to weaken the requirement. The adversary should not be able to distinguish a coerced and a deceiving party any better than he could do given only information that is "legally" available to him (the tally in our example). In general, however, it is not straightforward to define what information is "legally" available to the adversary in any particular situation. Neither is it straightforward to determine how much distinguishing advantage the adversary would get given only that information.

In order to circumvent this problem, we use a slightly different approach: We first define an ideal model in which the adversary has, by definition, only access to the "legally" available information. In the case of voting, such an ideal model would consist of a trusted machine (the ideal voting functionality $\mathcal{F}$) that collects the votes from all parties and gives only the tally to the adversary. In the ideal model, the distinguishing advantage between a coerced party (that gives v^* to $\mathcal{F}$) and a deceiving party (that gives v to $\mathcal{F}$) is, by definition, exactly the advantage the adversary gets from the "legally" available information (the tally).

To make this definition more formal, we introduce an additional entity, the deceiver [For91]. The task of the deceiver is to instruct a deceiving party what it should do (i.e., how to deceive the adversary). More formally, a deceiving party will not run any program of its own, but instead follow the instructions of the deceiver. (In a sense, the deceiver models the party's free will.) In particular, the deceiver may instruct a party to cast a vote v and to send to the adversary the fake notification that it cast vote v^*. (Since we are in the ideal model, no cryptographic receipts or similar need to be faked.) A corrupted party, on the other hand, will follow the adversaries instructions.

The combination of adversary and deceiver in the ideal model now allows to model any coercion situation that can occur in the ideal model. To define what it means that the real protocol is incoercible (or more precisely, as incoercible as the ideal model), we will use the concept of simulation that underlies many cryptographic definitions such as multi-party computation and zero-knowledge: We show that for any adversary in the real model that performs some coercion attack, there is another adversary in the ideal model (called the adversary-simulator) that performs a corresponding attack with as much success. In other words, we require that for any deceiver (specifying what a party would ideally want to do), and for any adversary in the real model (trying to coerce parties), there is an adversary-simulator in the ideal model such that the real and the ideal model are indistinguishable.

We are, however, missing one ingredient: We need to specify how the ideal deceptions (specified in terms of inputs to the ideal functionalities) translate into real deceptions (specified in terms of faked messages etc.). This is done by introducing a deceiver in the real model, too, called the deceiver-simulator. We then require that for any deceiver in the ideal model (representing a possible deception) there is a deceiver-simulator in the real model (that performs the corresponding real deceptions) such that for any adversary in the real model there is a adversary-simulator in the ideal model such that the two models are indistinguishable.

Finally, to model the indistinguishability of the two models, we follow the ideas from the UC framework and introduce a further machine, the environment, that either communicates with the machines in the real model or with the machines in the ideal model and that has to guess which model it is in. (For details on how this indistinguishability actually ensures that the adversary's advantage in distinguishing corrupted and deceiving parties carries over from the ideal to the real model we refer to the example in Section 3.)

1.2 Related work

We are aware of only two works that tackle the problem of defining incoercibility or a similar property in a general fashion (i.e., not specialised to a particular protocol task such as voting).

Incoercible secure function evaluation. Canetti and Gennaro [CG96] present a model for defining incoercible secure function evaluation which was subsequently refined by Moran and Naor [MN06]. The model by Moran and Naor is based on the so-called stand-alone model [Can00, Gol04, Ch. 7]. In this model, one assumes that the inputs of all honest parties are fixed before the beginning of the protocol. This has several implications: First, reactive protocols where parties may decide on their inputs in later phases cannot be modelled. Second, when actually deploying the protocol, one would have to ensure very strong synchronisation: In order not to introduce possibilities for attacks not covered by the model, we have to ensure that no protocol message is sent until all honest parties have decided on their input. Third, the stand-alone model only guarantees sequential composability.[1] That is, we have no guarantee that the protocol stays secure when running concurrently with other protocols (which usually happens in real-life networks).

Since the model by Moran and Naor is based on the stand-alone model, in this model coerced parties only need to lie about their initial inputs. Because of this, Moran and Naor do not need to introduce an explicit deceiver; any deception a party might want to perform can be encoded by specifying a second input, the so-called "fake input". In contrast, the more complex deceptions that are possible in our setting necessitate the introduction of an explicit machine, the deceiver, to specify the deceptions.

[1]Note that it has not been shown that the variant of the stand-alone model presented by Moran and Naor does compose sequentially. But it does not seem unlikely that this could be shown.

Everything we said about the work by Moran and Naor also applies to the earlier work by Canetti and Gennaro [CG96]. Furthermore, the model by Canetti and Gennaro only models a very weak form of coercion-resistance; the adversary may instruct a coerced party to use a different input, but he may not instruct that party to deviate from the protocol. For a discussion of the difference between the models by Moran and Naor and by Canetti and Gennaro, we refer to [MN06].

Externalized UC. Another approach to define properties similar to incoercibility for general protocols is the Externalized UC (EUC) framework proposed by Canetti, Dodis, Pass, and Walfish [CDPW07] (also known as Generalized UC, UC with global setup, or, proposed independently by Hofheinz, Müller-Quade, and Unruh [HUMQ07], UC with catalysts).

This framework is, like ours, an extension of the UC framework and inherits its support for reactive protocols and its universal composition theorem. The EUC framework differs from the UC framework by allowing the environment to directly access the ideal functionality used in the real protocol. As explained in [CDPW07], security in the EUC framework implies a property called deniability. This means that no (malicious) protocol party P can collect any information during the protocol run that can later be used to prove to an outsider that some party Q participated in the protocol. (An example for such incriminating information would be a message signed by Q.) In other words, Q can plausibly claim that the whole protocol did not take place. Obviously, such a claim is only realistic with respect to an outsider who did not himself communicate with Q during the protocol execution. In contrast, incoercibility as understood by this paper means that a party can lie about its actions towards an insider (e.g., a party could lie even towards another voter about the vote it has cast).

Thus the two models (EUC and UC/c) have very different aims. Technically they are, however, related: In Section 4.2 we show that under certain conditions, EUC security implies UC/c security.

2 The Composable Incoercibility Framework (UC/c)

2.1 The UC framework

Our model is based on the Universal Composability (UC) framwork [Can01]. For self containment and to fix notation, we give a short overview over the UC framework. An interactive Turing machine (ITM) is a Turing machine that has additional tapes for incoming and for outgoing communication. An ITM may be activated by a message on an incoming communication tape. At the end of an activation, the ITM may send a message on an outgoing communication tape to another ITM. The recipient of a message is addressed by the unique identity of that ITM. The actions of an ITM may depend on a global parameter $k \in \mathbb{N}$, the so-called security parameter.

A network is modeled as a (possibly infinite) set of ITMs.[2] We call a network S executable if it contains an ITM $\mathcal{Z}$ with distinguished input and output tape and with the special identity `env`. An execution of S with input $z \in \{0,1\}^*$ and security parameter $k \in \mathbb{N}$ is the following random process: First, $\mathcal{Z}$ is activated with the message z on its input tape. Whenever an ITM $M_1 \in S$ finishes an activation with an outgoing message m addressed to another ITM $M_2 \in S$ on its outgoing communication tape, the other ITM M_2 is invoked with incoming message m on its incoming communication tape (tagged with the identity of the sender M_1). If an ITM terminates its activation without an outgoing message or sends a message to a non-existing ITM, the ITM

[2]In the case of infinite networks we require the network to be uniform in the sense that given the identity of an ITM, we can compute the code of that ITM in deterministic polynomial-time.

$\mathcal{Z}$ is activated. When the ITM $\mathcal{Z}$ sends a message on its output tape (not the communication tape!), the execution of S terminates. The output of $\mathcal{Z}$ we denote by $\mathrm{EXEC}_S(k, z)$. An ITM $\mathcal{Z}$ with identity `env` we call an environment and an ITM $\mathcal{A}$ with identity `adv` we call an adversary. A protocol is a network that does not contain an environment or an adversary.

We call networks S, S' indistinguishable if there is a negligible function μ such that for all $k \in \mathbb{N}$, $z \in \{0,1\}^*$, we have that $|\Pr[\mathrm{EXEC}_S(k,z) = 1] - \Pr[\mathrm{EXEC}_{S'}(k,z) = 1]| \leq \mu(k)$. We call S, S' perfectly indistinguishable if $\mu = 0$.

Using the above network model, security is defined by comparison. We first define an ideal protocol ρthat specifies the intended protocol behaviour. Then we define what it means that another protocol π (securely) emulates ρ:

Definition 1 (UC [Can01]) *Let π and ρ be protocols. We say that π* UC emulates *ρ if for any polynomial-time adversary $\mathcal{A}$ there exists a polynomial-time adversary $\mathcal{S}$ (the adversary-simulator) such that for any polynomial-time environment $\mathcal{Z}$ the networks $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ (called the real model) and $\rho \cup \{\mathcal{S}, \mathcal{Z}\}$ (called the ideal model) are indistinguishable.*

In the UC framework, one can model secure channels (that do not even leak the length of the transmitted message) by direct communication between the ITMs; insecure channels can be modelled by sending messages to the adversary; secure channels that leak the length of the message, as well as authenticated channels can be modelled as an ideal functionality.

Corruptions are modelled as follows: The environment $\mathcal{Z}$ can send special corruption requests to protocol parties (which are ITMs in π). If a protocol party receives such a request, it sends its current state to the adversary and from then on is controlled by the adversary (i.e., it forwards all incoming communication to the adversary and vice versa).

Usually, the ideal model will be described by a so-called ideal functionality. Such an ideal functionality is an incorruptible ITM that can be seen as a trusted third party accessible to the protocol parties. The ideal protocol corresponding to $\mathcal{F}$ consists of $\mathcal{F}$ itself and a so-called dummy-party $\tilde{P}$ for each party P in the real model. The dummy-party $\tilde{P}$ simply forwards all messages received from the environment to $\mathcal{F}$ and vice versa. In slight abuse of notation, we write $\mathcal{F}$ for the ideal protocol corresponding to $\mathcal{F}$. Note that the dummy-parties can be corrupted, hence the inputs and outputs to $\mathcal{F}$ from corrupted parties can be influenced by the adversary-simulator. Using the concept of an ideal functionality, we can express many protocol tasks by first specifying an ideal functionality $\mathcal{F}$ that fulfils the protocol task by definition, and then requiring that the protocol π UC emulates $\mathcal{F}$.

We can also consider real protocols π which contain ideal functionalities $\mathcal{F}$ (e.g., a functionality modelling a CRS). These functionalities can then be accessed by all parties. We then say that π is a protocol in the $\mathcal{F}$-hybrid model.

For more details, we refer the reader to the full version of [Can01].

2.2 The Composable Incoercibility framework (UC/c)

In our framework (UC/c) the possibility of coercions is modelled by the presence of an additional adversarial entity, called the deceiver. Formally, a deceiver is an ITM $\mathcal{D}$ with the special identity `dec`. We further refine the notion of a protocol: A protocol is a network that does not contain an environment, adversary, or deceiver.

A typical network would consist of a protocol π, an adversary $\mathcal{A}$, a deceiver $\mathcal{D}$, and an environment $\mathcal{Z}$ (where the adversary and the deceiver may also be called adversary-simulator and deceiver-simulator for clarity depending on their role in the protocol). Both the adversary and the deceiver may control parties. The exact mechanism of this is the following:

A protocol party may be in one of three corruption states: *Uncontrolled*, *corrupted*, and *deceiving*. We say a party is *controlled* if it is corrupted or deceiving. Initially, all machines are uncontrolled. Uncontrolled parties behave according to the protocol specification. If the environment $\mathcal{Z}$ sends a *corruption request* to an uncontrolled party, the party becomes corrupted. If the environment sends a *deception request* to an uncontrolled or a corrupted party, the party becomes deceiving. When a party becomes corrupted or deceiving, it sends its state to the adversary or the deceiver, respectively. From then on, it is controlled by the adversary or the deceiver, respectively (that is, it forwards all incoming communication to the controlling machine and sends messages as instructed by the controlling machine). The only exception is that if a corrupted machine receives a deception request, it will not forward that request to the adversary, because in that moment, it will become deceiving and hence be under the control of the deceiver.

We assume the existence of a globally readable register that contains the state of each party (whether it is uncontrolled, corrupted, or deceiving). However, when the adversary reads this register, the state of any deceiving machine will be reported as corrupted. (This reflects the fact that the adversary should not be able to know which machine is deceiving.) Protocol parties will not usually read this register; in some cases, however, it might be useful if the behaviour of an ideal functionality can depend on whether a machine is controlled or not.[3]

We are now ready to specify the notion of UC/c security. In this notion, we do not only require the adversary-simulator (in the ideal model) to simulate the adversary's actions (in the real model), but simultaneously require that the deceiver-simulator (in the real model) simulates the actions of the deceiver (in the ideal model).

Definition 2 (UC/c) *Let π and ρ be protocols. We say that π* UC/c emulates *ρ if for any polynomial-time deceiver $\mathcal{D}$ there exists a polynomial-time deceiver $\mathcal{D}_S$ (the deceiver-simulator) such that for any polynomial-time adversary $\mathcal{A}$ there exists a polynomial-time adversary $\mathcal{A}_S$ (the adversary-simulator) such that for any polynomial-time environment $\mathcal{Z}$ the following networks are indistinguishable:*

$$\pi \cup \{\mathcal{A}, \mathcal{D}_S, \mathcal{Z}\} \qquad \text{and} \qquad \rho \cup \{\mathcal{A}_S, \mathcal{D}, \mathcal{Z}\}.$$

Why is the adversary not informed about deceiving parties? The reader may notice an asymmetry in the definition: While the deceiver learns which party is corrupted and which party is deceiving, the adversary will be told that a party is corrupted even if it is deceiving. This is necessary because during a deception, the goal is to cheat the adversary into thinking that one behaves as he instructs (i.e., that one is corrupted). Therefore corrupted and deceiving parties should be indistinguishable from the point of view of the adversary.

Why can deceiving party not become corrupted? Another asymmetry is that a corrupted party can later become deceiving while the model does not allow to corrupt parties that are deceiving. Although formally both directions could be allowed, we have excluded the latter because we could not find an interpretation for such a scenario. For an interpretation of the former direction (bad-guy coercions), see the next section.

2.3 Corruption schedules

The notion of UC/c (Definition 2) allows the environment to corrupt or coerce any party at any point of time. This leads to a very strict definition. To get a definition that is more lenient

[3]A typical example is the key exchange functionality, which returns a random key for both parties [Can05]. If one of the parties is corrupted, the key is instead chosen by the adversary. Thus the functionality needs to know which parties are corrupted.

but easier to fulfil, one can impose certain restrictions on the corruption and deception requests performed by the environment. We call such a restriction a corruption schedule.

Bad-guy coercions. There are no restrictions on the environment (except that the environment cannot corrupt a deceiving party, this is implicit in the modelling of the corruption mechanism).

We call this notion bad-guy coercions because the environment may first corrupt a party (make it a “bad-guy”) and then later coerce it. It is very difficult to design protocols that are secure against bad-guy coercions because a corrupted party may be instructed by the adversary to actively deviate from the protocol to produce evidence against itself and thus thwart its own deniability. (In contrast, a deceiving party would, given the same instructions, only try to make the adversary believe that it follows these instructions.)

For example, in some protocol the ability to deceive the adversary (and thus the incoercibility of the protocol) might be based on the following fact: When the adversary requests a private secret m of some party, that party may send a different secret m' instead which contains a trapdoor. This trapdoor then is later essential for achieving incoercibility. In the setting of bad-guy coercions, a party might first be corrupted and then reveal the true secret m to the adversary. This secret m does not contain a trapdoor. Then later, if the party becomes deceiving, it will be unable to follow its deception strategy because it does not know any trapdoor for m. In a nutshell, while corrupted, a party may actively try to prevent its own incoercibility. Thus we expect that UC/c security with respect to bad-guy coercions is very hard to achieve.

In practise, bad-guy coercions are arguably a very rare event. A possible motivation for bad-guy coercions is the following thought experiment: A member (say, Bob) of a criminal organisation is required by the rules of that organisation to actively produce and deliver some evidence (e.g., certain keys) against himself to that organisation. While Bob still works for the organisation, he will not try to circumvent these rules and will deliver this evidence. But if Bob later decides to leave the criminal organisation and to cooperate with the police (undercover), Bob may have to convincingly act as if he was still following the criminal organisation’s instructions. This is exactly the case that is modelled by bad-guy coercions.

In most cases, however, UC/c with bad-guy coercions will be much to strong a notion, and the notion of good-guy coercions (below) will be preferred.

Good-guy coercions. The environment may corrupt parties at any time and may send deception requests to uncontrolled parties at any time. The environment may not send deception requests to corrupted parties.

Receipt-freeness. The environment may corrupt parties at any time, and may send deception requests to uncontrolled parties after the end of the protocol (so that the adversary gets their state). The environment may not send deception requests to a corrupted party. Receipt-freeness implies that an honest party does not learn any data during the protocol that could later be used to prove after the protocol execution that the party performed a certain action. (Note that with erasing parties, receipt-freeness is probably easy to achieve: an honest party simply erases all intermediate protocol data.)

Static corruptions/deceptions. All corruption and deception requests must be sent at the very beginning of the protocol execution. In particular, this implies that the environment cannot choose which parties to corrupt depending on messages it observes during the protocol execution.

Only corruptions. The environment may not send deception requests. UC/c with only corruptions is equivalent to the UC notion from [Can01].

Combinations. The above corruptions schedules may be combined by requiring that the environment obeys a certain schedule with respect to some parties and another with respect to other parties. For example, one might have protocols that are UC/c secure with receipt-freeness for Alice and good-guy coercions for Bob.

2.4 Erasing and non-erasing parties

We can distinguish two kinds of honest parties: Erasing and non-erasing parties. An erasing party is able to delete information when the protocol instructs it to do so. In contrast, a non-erasing party will make a snapshot of its whole memory in every computation step in a special log; when the party becomes coerced/deceiving, the adversary/deceiver gets the whole log. Non-erasing parties model the fact that it may be difficult to reliably erase information, a secret may end up, e.g., in the swap partition. However, we allow corrupted parties to erase their state. This is due to the fact that we also cannot expect to reliably recover state which the adversary has ordered destroyed. Since a deceiving party will stay in that state forever (the environment is not allowed to send corruption requests to a deceiving party), it does not matter whether a deceiving party may or may not erase information. For concreteness, we fix that a deceiving party may erase information.

Summarising, a non-erasing party stores all its states when uncontrolled, and may erase its state when controlled. An erasing party may erase its state at any point.

In the following, we consider non-erasing parties. We wish to stress, however, that our modelling applies to erasing parties as well. Being able to erase data may be very helpful in the design of incoercible protocols: If a party deletes some data, it may later credibly claim that it cannot reveal that data. However, one should keep in mind that implementing non-erasing protocols may be more difficult because one needs to actively keep track of all copies of a certain datum in memory and on disk.

It is also possible to imagine a setting in which a machine is partially erasing. Such a machine would have a certain memory area that can be erased, while the main part of the memory is assumed to be non-erasable. Such a modelling might be motivated, e.g., by the use of trusted platform modules [Tru07], or by operating system extensions that allocate blocks of memory that are guaranteed never to be written to the disk.

2.5 Basic properties

Transitivity, reflexivity. The following lemma states that UC/c emulation is a reflexive and transitive relation. Reflexivity can be seen as a sanity check for the definition – if a protocol would not UC/c emulate itself, something would probably be wrong. Transitivity is necessary to use the universal composition theorem, see Corollary 8 below.

Lemma 3 (Reflexivity, transitivity) *Let π, ρ, and σ be protocols. Then π UC/c emulates π. If π UC/c emulates ρ, and ρ UC/c emulates σ, then π UC/c emulates σ.*

Proof. For any polynomial-time deceiver $\mathcal{D}$, any polynomial-time adversary $\mathcal{A}$, and any polynomial-time environment $\mathcal{Z}$, we have with $\mathcal{D}_S := \mathcal{D}$, and $\mathcal{A}_S := \mathcal{A}$ that $\pi \cup \{\mathcal{A}, \mathcal{D}_S, \mathcal{Z}\}$ and $\pi \cup \{\mathcal{A}_S, \mathcal{D}, \mathcal{Z}\}$ are equal and thus (perfectly) indistinguishable. Hence π UC/c emulates ρ.

Assume now that π UC/c emulates ρ, and ρ UC/c emulates σ. Then, by definition, for any polynomial-time deceiver $\mathcal{D}^\rho$, there is a polynomial-time deceiver-simulator $\mathcal{D}^\pi_S(\mathcal{D}^\rho)$, and for any polynomial-time deceiver $\mathcal{D}^\rho$ and any polynomial-time adversary $\mathcal{A}^\pi$, there is an adversary-simulator $\mathcal{A}^\rho_S(\mathcal{D}^\rho, \mathcal{A}^\pi)$ such that for all polynomial-time environments $\mathcal{Z}$,

$$\pi \cup \{\mathcal{A}^\pi, \mathcal{D}^\pi_S(\mathcal{D}^\rho), \mathcal{Z}\} \qquad \text{and} \qquad \rho \cup \{\mathcal{A}^\rho_S(\mathcal{D}^\rho, \mathcal{A}^\pi), \mathcal{D}^\rho, \mathcal{Z}\} \tag{1}$$

are indistinguishable. Similarly, for any polynomial-time deceiver $\mathcal{D}^\sigma$, there is a polynomial-time deceiver-simulator $\mathcal{D}^\rho_S(\mathcal{D}^\sigma)$, and for any polynomial-time deceiver $\mathcal{D}^\sigma$ and any polynomial-time adversary $\mathcal{A}^\rho$, there is an adversary simulator $\mathcal{A}^\sigma_S(\mathcal{D}^\sigma, \mathcal{A}^\rho)$ such that for all polynomial-time environments $\mathcal{Z}$,

$$\rho \cup \{\mathcal{A}^\rho, \mathcal{D}^\rho_S(\mathcal{D}^\sigma), \mathcal{Z}\} \qquad \text{and} \qquad \sigma \cup \{\mathcal{A}^\sigma_S(\mathcal{D}^\sigma, \mathcal{A}^\rho), \mathcal{D}^\sigma, \mathcal{Z}\} \tag{2}$$

are indistinguishable.

Then, for a given polynomial-time deceiver $\hat{\mathcal{D}}^\sigma$ and a given polynomial-time adversary $\hat{\mathcal{A}}^\pi$, set $\hat{\mathcal{D}}^\pi_S(\mathcal{D}^\sigma) := \mathcal{D}^\pi_S(\mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma))$ and $\hat{\mathcal{A}}^\sigma_S(\hat{\mathcal{D}}^\sigma, \hat{\mathcal{A}}^\pi) := \mathcal{A}^\sigma_S(\hat{\mathcal{D}}^\sigma, \mathcal{A}^\rho_S(\mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma), \hat{\mathcal{A}}^\pi))$. From (1), we have that for all polynomial-time environments $\mathcal{Z}$, $\pi \cup \{\hat{\mathcal{A}}^\pi, \hat{\mathcal{D}}^\pi_S(\hat{\mathcal{D}}^\sigma), \mathcal{Z}\}$ and $\rho \cup \{\mathcal{A}^\rho_S(\mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma), \hat{\mathcal{A}}^\pi), \mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma), \mathcal{Z}\}$ are indistinguishable. And from (2), we have that for all polynomial-time environments $\mathcal{Z}$, $\rho \cup \{\mathcal{A}^\rho_S(\mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma), \hat{\mathcal{A}}^\pi), \mathcal{D}^\rho_S(\hat{\mathcal{D}}^\sigma), \mathcal{Z}\}$ and $\sigma \cup \{\hat{\mathcal{A}}^\sigma_S(\hat{\mathcal{D}}^\sigma, \hat{\mathcal{A}}^\pi), \hat{\mathcal{D}}^\sigma, \mathcal{Z}\}$ are indistinguishable. Since indistinguishability is transitive, $\pi \cup \{\hat{\mathcal{A}}^\pi, \hat{\mathcal{D}}^\pi_S(\hat{\mathcal{D}}^\sigma), \mathcal{Z}\}$ and $\sigma \cup \{\hat{\mathcal{A}}^\sigma_S(\hat{\mathcal{D}}^\sigma, \hat{\mathcal{A}}^\pi), \hat{\mathcal{D}}^\sigma, \mathcal{Z}\}$ are indistinguishable for all polynomial-time $\mathcal{Z}$. Thus, for every polynomial-time deceiver $\mathcal{D}^\sigma$ there exists a polynomial-time deceiver-simulator $\hat{\mathcal{D}}^\pi_S := \hat{\mathcal{D}}^\pi_S(\hat{\mathcal{D}}^\sigma)$ such that for every polynomial-time adversary $\hat{\mathcal{A}}^\pi$ there exists a polynomial-time adversary-simulator $\hat{\mathcal{A}}^\sigma_S := \hat{\mathcal{A}}^\sigma_S(\hat{\mathcal{D}}^\sigma, \hat{\mathcal{A}}^\pi)$ such that for all polynomial-time environments $\mathcal{Z}$, $\pi \cup \{\hat{\mathcal{A}}^\pi, \hat{\mathcal{D}}^\pi_S, \mathcal{Z}\}$ and $\sigma \cup \{\hat{\mathcal{A}}^\sigma_S, \hat{\mathcal{D}}^\sigma, \mathcal{Z}\}$ are indistinguishable. Thus π UC/c emulates ρ. □

Dummy adversary and deceiver. A dummy-adversary is an adversary that just follows the instructions of the environment. More precisely, it forwards all messages it receives to the environment, and sends only the messages the environment instructs it to send. It was shown by Canetti [Can01] in the UC setting that the dummy-adversary is complete, that is, without loss of generality we can consider only the dummy-adversary. Therefore we only have to specify the adversary-simulator for the dummy-adversary instead of having to specify the adversary-simulator for every possible adversary. This simplifies proofs.

In the setting of UC/c, we can additionally consider the dummy-deceiver that just follows the instructions of the environment. Below, we will show that both the dummy-adversary and the dummy-deceiver are complete. Besides strongly simplifying proofs, the completeness of the dummy-deceiver has an additional conceptual advantage. The deceiver-simulator corresponding to the dummy-deceiver encodes a universal deception strategy. That is, for any "ideal deception", it tells us how to perform this deception in the real protocol. The existence of such a universal deception strategy is very important in real life, protocol users need to have an explicit strategy how to lie in which situation; it is not sufficient that such a strategy exists for each situation.

Definition 4 (Dummy-adversary, dummy-deceiver) *The dummy-adversary $\tilde{\mathcal{A}}$ is an adversary that, when receiving a message (id, m) from the environment, sends m to the party with identity id, and that, when receiving m from a party with identity id, sends (id, m) to the environment. The dummy-deceiver $\tilde{\mathcal{D}}$ is defined analogously.*

Definition 5 (UC/c with respect to dummy-adversary/deceiver) *Let π and ρ be protocols. We say that π* UC/c emulates ρ with respect to the dummy-adversary/deceiver *if there exists a polynomial-time deceiver $\tilde{\mathcal{D}}_S$ (the dummy-deceiver-simulator) and a polynomial-time adversary $\tilde{\mathcal{A}}_S$ (the dummy-adversary-simulator) such that for any polynomial-time environment $\mathcal{Z}$ the following networks are indistinguishable:*

$$\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}\} \qquad \textit{and} \qquad \rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}\}$$

Lemma 6 (Completeness of dummy-adversary and dummy-deceiver) *Let π and ρ be protocols. Then π UC/c emulates ρ iff π UC/c emulates ρ with respect to the dummy-adversary/deceiver.*

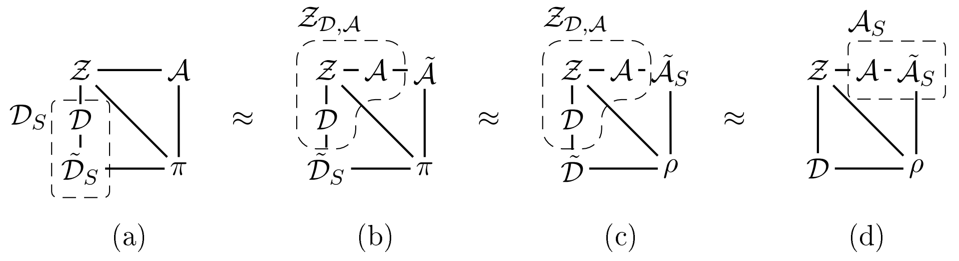

Figure 1: Networks in the proof of Lemma 6. Dashed lines denote machines that internally simulate other machines.

Proof. If π UC/c emulates ρ, then we immediately have that π UC/c emulates ρ with respect to the dummy-adversary/deceiver. For the opposite direction, assume that π UC/c emulates ρ with respect to the dummy-adversary/deceiver.

Let $\tilde{\mathcal{D}}$ and $\tilde{\mathcal{A}}$ be the dummy-deceiver and the dummy-adversary. Let $\tilde{\mathcal{D}}_S$ and $\tilde{\mathcal{A}}_S$ be the dummy-deceiver-simulator and the dummy-adversary-simulator (as in Definition 5).

For an environment $\mathcal{Z}$, a deceiver $\mathcal{D}$, and an adversary $\mathcal{A}$, we define the environment $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$ as follows (see also Figures 1(b) and (c)): $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$ internally simulates $\mathcal{Z}$, $\mathcal{D}$, and $\mathcal{A}$. All communication between the internally simulated $\mathcal{Z}$ and the (external) protocol is forwarded. Messages sent from $\mathcal{Z}$ to the deceiver and adversary are forwarded to the internally simulated $\mathcal{D}$ and $\mathcal{A}$. Incoming communication for $\mathcal{Z}$ from $\mathcal{D}$ and $\mathcal{A}$ is is forwarded to $\mathcal{Z}$. Messages m that $\mathcal{D}$ and $\mathcal{A}$ send to a protocol party with identity id are sent by $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$ as (id, m) to the external deceiver or adversary. That is, assuming the external deceiver or adversary is the dummy-deceiver or dummy-adversary, $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$ instructs the dummy-deceiver/adversary to deliver the message sent by the internally simulated $\mathcal{D}$ or $\mathcal{A}$. Incoming messages (id, m) from the external adversary/deceiver are passed to the internally simulated $\mathcal{D}$ and $\mathcal{A}$ as m.

For an adversary $\mathcal{A}$, we define the adversary-simulator $\mathcal{A}_S = \mathcal{A}_S(\mathcal{A})$ as follows (see also Figure 1(d)): $\mathcal{A}_S$ internally simulates $\mathcal{A}$ and $\tilde{\mathcal{A}}_S$. Communication between the external environment and the internally simulated $\mathcal{A}$ is forwarded. Communication between the external protocol and the internally simulated $\tilde{\mathcal{A}}_S$ is forwarded. When $\mathcal{A}$ sends a message m to a protocol machine with identity id, (id, m) is instead passed to $\tilde{\mathcal{A}}_S$ as coming from the environment. Messages (id, m) from the internally simulated $\tilde{\mathcal{A}}_S$ to the environment are passed to $\mathcal{A}$ as m. (This is analogous to how $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$ reroutes between $\mathcal{A}$ and the protocol.)

For a deceiver $\mathcal{D}$, $\mathcal{D}_S(\mathcal{D})$ is defined analogously to $\mathcal{A}_S(\mathcal{A})$. See Figure 1(a).

We then have that the following networks are perfectly indistinguishable for all environments $\mathcal{Z}$: $\pi \cup \{\mathcal{A}, \mathcal{D}_S, \mathcal{Z}\}$ and $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_{\mathcal{D},\mathcal{A}}\}$ (Figures 1(a) and (b)). This is due to the fact that the dummy-adversary $\tilde{\mathcal{A}}$ just forwards the messages between π and the adversary $\mathcal{A}$ simulated by $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$, and that $\mathcal{D}_S$ by definition consists of $\mathcal{D}$ (simulated by $\mathcal{Z}$ in the second network) and $\tilde{\mathcal{D}}_S$.

Analogously, the following networks are perfectly indistinguishable for all environments $\mathcal{Z}$: $\rho \cup \{\mathcal{A}_S, \mathcal{D}, \mathcal{Z}\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_{\mathcal{D},\mathcal{A}}\}$. Cf. Figures 1(d) and (c).

Furthermore, since π UC/c emulates ρ with respect to the dummy-adversary/deceiver, for all polynomial-time environments $\mathcal{Z}_{\mathcal{D},\mathcal{A}}$, $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_{\mathcal{D},\mathcal{A}}\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_{\mathcal{D},\mathcal{A}}\}$ are indistinguishable. Cf. Figures 1(b) and (c).

Since indistinguishability is transitive, $\pi \cup \{\mathcal{A}, \mathcal{D}_S, \mathcal{Z}\}$ and $\rho \cup \{\mathcal{A}_S, \mathcal{D}, \mathcal{Z}\}$ are indistinguishable for all polynomial-time environments $\mathcal{Z}$. Furthermore, $\mathcal{D}_S$ and $\mathcal{A}_S$ are polynomial-time, and their construction does not depend on $\mathcal{Z}$, and the construction of $\mathcal{D}_S$ does not depend on $\mathcal{D}$. Thus π UC/c emulates ρ. □

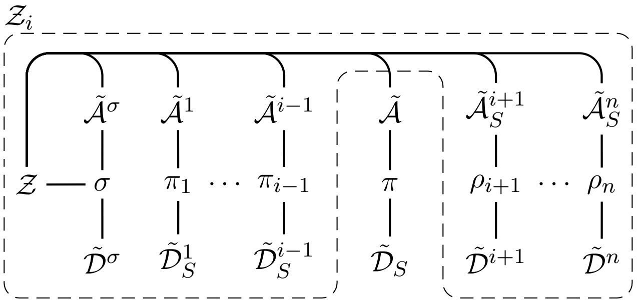
$\mathcal{Z}_i$
$\tilde{\mathcal{A}}^\sigma$
$\tilde{\mathcal{A}}^1$
$\tilde{\mathcal{A}}^{i-1}$
$\tilde{\mathcal{A}}$
$\tilde{\mathcal{A}}_S^{i+1}$
$\tilde{\mathcal{A}}_S^n$
$\mathcal{Z}$
σ
$\pi_1 \cdots \pi_{i-1}$
π
$\rho_{i+1} \cdots \rho_n$
$\tilde{\mathcal{D}}^\sigma$
$\tilde{\mathcal{D}}_S^1$
$\tilde{\mathcal{D}}_S^{i-1}$
$\tilde{\mathcal{D}}_S$
$\tilde{\mathcal{D}}^{i+1}$
$\tilde{\mathcal{D}}^n$

Figure 2: Hybrid network $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_i\}$ in the proof of Theorem 7. The hybrid environment $\mathcal{Z}_i$ is depicted by the dashed line. Connections between $\mathcal{Z}$ and the deceivers, as well as connections between σ and the instances of π are omitted. The connections between $\mathcal{Z}$ and the deceivers are analogous to those between $\mathcal{Z}$ and the adversaries.

2.6 Universal composition

One of the main advantages of the UC framework is the universal composition theorem. This theorem guarantees that a UC secure protocol π can be securely used as a subprotocol of arbitrary other protocols σ, even when σ and polynomially many instances of π run concurrently. The same compositionality result also holds for the UC/c security notion.

To formulate the composition theorem, we introduce some notation. Let π and σ be protocols. Then let σ^π denote the protocol where σ invokes a polynomial number of instances of the subprotocol π. That is, machines in σ may give inputs to machines in π, these inputs are treated by π as coming from the environment. When the machines in π give output back to the environment, these are sent to the invoking machines in σ. Thus, in a sense, in σ^π, the protocol σ plays the role of the environment for the instances of π. For example, if $\sigma^{\mathcal{F}}$ is a protocol using a commitment functionality $\mathcal{F}$ (i.e., $\sigma^{\mathcal{F}}$ is a protocol in the $\mathcal{F}$-hybrid model), then σ^π would be the protocol that uses the subprotocol π instead of using the commitment functionality $\mathcal{F}$. The following theorem guarantees that, if π UC/c emulates some other protocol ρ (e.g., $\rho = \mathcal{F}$), we do not loose security if we replace subprotocol invocations of ρ by subprotocol invocations of π.

Theorem 7 (Universal composition) *Let π, ρ, and σ be polynomial-time protocols. Assume that π UC/c emulates ρ. Then σ^π UC/c emulates σ^ρ.*

Proof. Let $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{D}}$ denote the dummy-adversary and the dummy-deceiver. Due to Lemma 6, it is sufficient to construct a polynomial-time deceiver-simulator $\tilde{\mathcal{D}}_S^*$ and a polynomial-time adversary-simulator $\tilde{\mathcal{A}}_S^*$ such that for all polynomial-time environments $\mathcal{Z}$, the networks $\sigma^\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S^*, \mathcal{Z}\}$ and $\sigma^\rho \cup \{\tilde{\mathcal{A}}_S^*, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are indistinguishable. (We write $\tilde{\mathcal{A}}_S^*$ and $\tilde{\mathcal{D}}_S^*$ to distinguish from the simulators $\tilde{\mathcal{A}}_S$ and $\tilde{\mathcal{D}}_S$ defined below.)

By assumption, there are a polynomial-time deceiver-simulator $\tilde{\mathcal{D}}_S$ and a polynomial-time adversary-simulator $\tilde{\mathcal{A}}_S$ such that for all polynomial-time environments $\mathcal{Z}$, the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are indistinguishable.

Let n be a polynomial upper bound on the number of instances of π or ρ that are invoked by σ.

Given an environment $\mathcal{Z}$, we define the hybrid environment $\mathcal{Z}_i$. $\mathcal{Z}_i$ is depicted in Figure 1 by the area enclosed by a dashed line. $\mathcal{Z}_i$ internally simulates the following machines: The original environment $\mathcal{Z}$; an instance of σ; $i-1$ instances of π, denoted $\pi_1, \dots, \pi_{i-1}$; $n-i-1$ instances of ρ, denoted $\rho_{i+1}, \dots, \rho_n$; an instance of $\tilde{\mathcal{A}}$, denoted $\tilde{\mathcal{A}}^\sigma$; $i-1$ instances of $\tilde{\mathcal{A}}$, denoted $\tilde{\mathcal{A}}^1, \dots, \tilde{\mathcal{A}}^{i-1}$; $n-i-1$ instances of $\tilde{\mathcal{A}}_S$, denoted $\tilde{\mathcal{A}}_S^{i+1}, \dots, \tilde{\mathcal{A}}_S^n$; an instance of $\tilde{\mathcal{D}}$, denoted $\tilde{\mathcal{D}}^\sigma$; $i-1$ instances of $\tilde{\mathcal{D}}_S$, denoted $\tilde{\mathcal{D}}_S^1, \dots, \tilde{\mathcal{D}}_S^{i-1}$; $n-i-1$ instances of $\tilde{\mathcal{D}}$, denoted $\tilde{\mathcal{D}}^{i+1}, \dots, \tilde{\mathcal{D}}^n$.

For the sake of concreteness in the following specification of the behaviour of $\mathcal{Z}$, assume that $\mathcal{Z}$ is executed in a network with the protocol π, the dummy-adversary $\tilde{\mathcal{A}}$, and the dummy-deceiver-simulator $\tilde{\mathcal{D}}_S$.

Messages of the form (id, m) that $\mathcal{Z}$ sends to the adversary (meaning that $\mathcal{Z}$ instructs the adversary to deliver m to the machine with identity id) are routed to one of the instances of $\tilde{\mathcal{A}}$ or $\tilde{\mathcal{A}}_S$: If id is a machine in σ, the message (id, m) is passed to the internally simulated $\tilde{\mathcal{A}}^\sigma$. If id is a machine in the j-th instance of π, we distinguish three cases: If $j < i$, then (id, m) is passed to the internally simulated $\tilde{\mathcal{A}}^j$. If $j = i$, then (id, m) is passed to the external adversary $\mathcal{A}$. If $j > i$, then (id, m) is passed to the internally simulated $\tilde{\mathcal{A}}_S^j$. Similarly, we proceed for messages from $\mathcal{Z}$ to the deceiver: These messages are forwarded to $\tilde{\mathcal{D}}^\sigma$, $\tilde{\mathcal{D}}_S^j$ $(j < i)$, the external $\tilde{\mathcal{D}}_S$ $(j = i)$, or $\tilde{\mathcal{D}}^j$ $(j > i)$. Messages in the opposite direction are handled analogously. Communication between σ and $\mathcal{Z}$ is forwarded normally. Communication between σ and some instance of π or ρ is forwarded to the corresponding instance of π. That is, if σ sends a message to a machine in the j-th instance of π or ρ, this message is forwarded to the corresponding machine in the internally simulated π_j if $j < i$, in the external protocol π if $j = i$, and in the internally simulated ρ_j if $j > i$.

We observe the following facts about $\mathcal{Z}_i$.

First, by construction, the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_i\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_{i+1}\}$ are perfectly indistinguishable.

Furthermore, since the dummy-adversary only forwards messages between $\mathcal{Z}$ and the protocol, we can replace a single instance of the dummy-adversary that handles all instances of π and σ (as in the network $\sigma^\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S^*, \mathcal{Z}\}$) by individual instances of the dummy-adversary for each protocol instance (as in the network $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}$). More precisely, the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}$ and $\sigma^\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S^*, \mathcal{Z}\}$ are perfectly indistinguishable if we define $\tilde{\mathcal{D}}_S^*$ to be the (polynomial-time) deceiver-simulator that internally simulates $\tilde{\mathcal{D}}^\sigma, \tilde{\mathcal{D}}_S^1, \dots, \tilde{\mathcal{D}}_S^{n-1}, \tilde{\mathcal{D}}_S$.

Similarly, we have that networks $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_1\}$ and $\sigma^\rho \cup \{\tilde{\mathcal{A}}_S^*, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are perfectly indistinguishable if we define $\tilde{\mathcal{A}}_S^*$ to be the (polynomial-time) adversary-simulator that internally simulates $\tilde{\mathcal{A}}^\sigma, \tilde{\mathcal{A}}_S, \tilde{\mathcal{A}}_S^2, \dots, \tilde{\mathcal{A}}_S^n$.

Finally, we claim that the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_1\}$ are indistinguishable (we show this below). If we have shown this claim, Theorem 7 follows since then $\sigma^\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S^*, \mathcal{Z}\}$ and $\sigma^\rho \cup \{\tilde{\mathcal{A}}_S^*, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are indistinguishable, and $\tilde{\mathcal{A}}_S^*$ and $\tilde{\mathcal{D}}_S^*$ are constructed independently of $\mathcal{Z}$ and polynomial-time.

We proceed to show that $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_1\}$ are indistinguishable.

Let $\mathcal{Z}^*$ be the environment which on input (i, z) simulates $\mathcal{Z}_i$ with input z. By definition of $\tilde{\mathcal{D}}_S$ and $\tilde{\mathcal{A}}_S$, the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}^*\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}^*\}$ are indistinguishable. Thus, there is a negligible function μ such that

$$|\Pr[\mathrm{EXEC}_{\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}^*\}}(k, (i, z)) = 1] - \Pr[\mathrm{EXEC}_{\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}^*\}}(k, (i, z)) = 1]| \leq \mu(k)$$

for all $k, i \in \mathbb{N}$, $z \in \{0, 1\}^*$. By construction of $\mathcal{Z}^*$, this implies that

$$|\Pr[\mathrm{EXEC}_{\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_i\}}(k, z) = 1] - \Pr[\mathrm{EXEC}_{\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_i\}}(k, z) = 1]| \leq \mu(k)$$

for all $k, i \in \mathbb{N}$, $z \in \{0, 1\}^*$. Using the triangle inequality and the fact that $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_i\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_{i+1}\}$ are perfectly indistinguishable, we get

$$|\Pr[\mathrm{EXEC}_{\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}}(k, z) = 1] - \Pr[\mathrm{EXEC}_{\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_1\}}(k, z) = 1]| \leq n\mu(k).$$

Since n is a polynomial, $n\mu$ is negligible. Thus the networks $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}_n\}$ and $\rho \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}_1\}$ are indistinguishable. □

The most common use case of the composition theorem is given by the following corollary:

Corollary 8 *Let π and σ be polynomial-time protocols, and $\mathcal{F}$ and $\mathcal{G}$ be polynomial-time functionalities. Assume that π UC/c emulates $\mathcal{F}$ and that $\sigma^{\mathcal{F}}$ UC/c emulates $\mathcal{G}$. Then σ^{π} UC/c emulates $\mathcal{G}$.*

Proof. Immediate from the universal composition theorem (Theorem 7), and the transitivity of UC/c emulation (Lemma 3). □

3 Voting schemes

In this section we illustrate the UC/c security notion by applying it to the special case of voting schemes. We give a definition of incoercibility that is tailored to the specific case of voting protocols and show that this definition is implied by the UC/c security notion.

Definition 9 (Voting scheme) *Fix sets $\mathcal{V}$ (the set of votes), $\mathcal{T}$ (the set of tallies), $\mathcal{P}$ (the set of voters). A tally function is an efficiently computable function* $\mathrm{tally} : (\mathcal{V} \cup \{\bot\})^{\mathcal{P}} \to \mathcal{T}$.

A voting scheme for tally *is a two-stage protocol. We call the stages voting phase and tallying phase. In such a protocol, each party $P_i \in \mathcal{P}$ gets an input $v_i \in \mathcal{V} \cup \{\bot\}$ (the vote of P_i). $v_i = \bot$ means that the P_i does not participate in the protocol (abstention). In the end of the tallying phase a distinguished party T outputs a value $t \in \mathcal{T}$.*

Typically, $\mathcal{V}$ would be the set of all candidates. In more complex schemes, elements of $\mathcal{V}$ might be, e.g., ordered lists of candidates in order of decreasing precedence. The set of tallies $\mathcal{T}$ usually is the set of all functions $\mathcal{V} \to \mathbb{N}_0$. Alternatively, in a voting scheme which only announces the winner, we would have have $\mathcal{T} = \mathcal{V}$. The tally function $\mathrm{tally}(v_1, \dots, v_n)$ specifies what the correct tally is for the votes $v_i \in \mathcal{V} \cup \{\bot\}$ where $v_i = \bot$ denotes abstention.

Note that we do not require that the parties $P_i \neq T$ are aware whether they are in the tallying or the voting phase. Such a requirement might be difficult to ensure in an asynchronous environment. In particular, votes cast during the tallying phase (but before the tally is announced) might or might not be counted.

An ideal voting scheme is given by the following functionality:

Definition 10 (Voting functionality) *The voting functionality $\mathcal{F}_{\mathrm{vote}} = \mathcal{F}_{\mathrm{vote}}^{\mathrm{tally}}$ expects (at most one) message $v_i \in \mathcal{V}$ from each party $P_i \in \mathcal{P}$. When receiving* `tally` *from T, $\mathcal{F}_{\mathrm{vote}}$ sets $v_i := \bot$ for all $P_i \in \mathcal{P}$ from which it did not receive a message $v_i \in \mathcal{V}$ yet. Then $\mathcal{F}_{\mathrm{vote}}$ computes $t :=$ $\mathrm{tally}(v_i, i \in \mathcal{P})$ (the tally) and sends t to the adversary. Then, when $\mathcal{F}_{\mathrm{vote}}$ receives* `deliver` *from the adversary, it sends t to the party T.*

This functionality models that the tally output by T is correctly computed using the tally function (as long as T itself is not corrupted) and that the individual votes are secret (even if T is corrupted).

Natural properties of voting schemes are, e.g., correctness (the tally is correct even in the presence of an adversary) and anonymity (the adversary cannot tell who voted for whom, except as deducible from the tally itself). We will not formalise these properties here, but it is easy to see that a voting scheme that UC emulates the voting functionality $\mathcal{F}_{\mathrm{vote}}$ satisfies reasonable formalisations of these properties. Since the UC/c security notion is stronger than UC, this implies that these elementary properties are satisfied by UC/c secure voting scheme, too.

In our context, the most interesting property of a voting scheme is incoercibility. We will first formalise what incoercibility means for voting schemes (independently of our framework). Then we will show that incoercibility of voting schemes is implied by security in the UC/c framework. Assume some party P that wants to cast a vote v. In an incoercible voting scheme, we expect that if the adversary $\mathcal{A}$ forces a party P to deviate from the protocol, $\mathcal{A}$ should not be able to tell the difference between P obeying the adversary $\mathcal{A}$, or the party P casting the vote v anyway (we say P deceives the adversary). Of course, since the adversary learns the tally, this goal is unachievable – the tally always leaks a non-negligible amount of information about the vote of P (at least if the number of voters is polynomial). We can only achieve the following: The adversary's advantage in distinguishing between P obeying and P deceiving is not greater than the advantage with which the adversary could distinguish these two cases given only the tally. To formulate this definition, we first introduce some notation:

Fix a voter $P \in \mathcal{P}$ and a vote $v \in \mathcal{V} \cup \{\bot\}$. Fix a distribution $\mathcal{B}$ on $(\mathcal{V} \cup \{\bot\})^{\mathcal{P}\setminus\{P\}}$. ($\mathcal{B}$ represents the distribution of the votes of the other voters.) Given a vote v, let $\mathcal{B}_v$ denote the distribution over $(\mathcal{V} \cup \{\bot\})^{\mathcal{P}}$ that chooses the votes for all $P_i \in \mathcal{P} \setminus \{P\}$ according to $\mathcal{B}$ and uses the vote v for P. Accordingly, $\mathrm{tally}(\mathcal{B}_v)$ denotes the tally resulting from votes chosen according to $\mathcal{B}_v$. Let $\mathrm{Adv}_{ideal}(\mathcal{B}, v) := \max_{v^*} \Delta(\mathcal{B}_v, \mathcal{B}_{v^*})$ where v^* ranges over $\mathcal{V} \cup \{\bot\}$ and Δ denotes the statistical distance. (Adv_{ideal} describes how well an adversary can distinguish between being obeyed and being deceived using only the tally.)

A voting adversary is an adversary that controls a party P (however, depending on the setting, P may choose to ignore the instructions given by the adversary) and that may decide when the tallying phase starts. We require that a voting adversary eventually starts the tallying phase. Furthermore, when the party T outputs the tally, the tally is given to the voting adversary. In the end, the voting adversary output a bit b.

Given a voting adversary $\mathcal{A}$, let $\Pr_{obey}(\mathcal{A}, \mathcal{B})$ be the probability that $\mathcal{A}$ outputs 1 in the case that the party P follows the instructions of the adversary (i.e., P is corrupted) and all other parties honestly follow the protocol (with inputs chosen according to $\mathcal{B}$).

Given some program code $\mathfrak{d}$ (the deception strategy for P), let $\Pr_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ denote the probability that the adversary $\mathcal{A}$ outputs 1 if P follows the instructions in $\mathfrak{d}$ and all other parties honestly follow the protocol (with inputs chosen according to $\mathcal{B}$). (Intuitively, $\mathfrak{d}$ is a strategy that tells P how to vote for v and simultaneously make the adversary believe that P obeys the adversary.) We assume that $\mathfrak{d}$ gets v and the identity of P as input. In the same setting, let $\mathrm{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ denote the tally output by T.

Definition 11 (Incoercible voting schemes) *A voting scheme is incoercible if there is a deception strategy $\mathfrak{d}$ such that for every polynomial-time voting adversary, every voter $P \in \mathcal{P}$, every vote $v \in \mathcal{V}$, and every efficiently sampleable distribution $\mathcal{B}$ the following holds:*

- The deception strategy casts the right vote: *The random variables $\mathrm{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ and $\mathrm{tally}(\mathcal{B}_v)$ are computationally indistinguishable.*

- The adversary cannot distinguish between being obeyed and being deceived: *For some negligible function μ we have that*

$$\left|\Pr_{obey}(\mathcal{A}, \mathcal{B}) - \Pr_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})\right| \leq \mathrm{Adv}_{ideal}(\mathcal{B}, v) + \mu.$$

Many variants of this definition are possible. For example, one could allow the voting adversary to corrupt additional parties from $\mathcal{P} \setminus \{P\}$. (In this case, one would have to adapt the definition of Adv_{ideal}.) For the sake of simplicity, we do not strive to find the most general formulation of Definition 11, especially in view of the fact that the UC/c framework already provides us with a very general definition of incoercibility.

We will now show that incoercibility in the sense of Definition 11 is already implied by UC/c security. We find that the proof of the following theorem is very instructive because it gives some intuition for the UC/c framework, and because it illustrates how application-specific incoercibility definitions (not restricted to the application of voting) can be proven to be implied by UC/c security.

Theorem 12 *Let π be a voting scheme for the tally function* tally. *Assume that π UC/c emulates $\mathcal{F}_{\text{vote}}^{\text{tally}}$ with static corruption/deception. Then π is an incoercible voting scheme.*

Proof. Fix a voting adversary $\mathcal{A}$. We define the UC/c adversary $\mathcal{A}'$ to behave like $\mathcal{A}$, except that when $\mathcal{A}$ starts the tallying phase, $\mathcal{A}'$ instead sends `tally` to the environment. When $\mathcal{A}$ would give an output b, $\mathcal{A}'$ sends b to the environment.

We define an environment $\mathcal{Z}_{obey} := \mathcal{Z}_{obey}^{P,v,\mathcal{B}}$ as follows: Initially, $\mathcal{Z}_{obey}$ sends a corruption request to the party P. Then $\mathcal{Z}_{obey}$ chooses votes $v_1, \dots, v_n$ according to the distribution $\mathcal{B}$ and gives these votes as input to the parties $P_i \in \mathcal{P} \setminus \{P\}$ (or, if $v_i = \bot$, sends no input to P_i). When the adversary sends `tally` to $\mathcal{Z}_{obey}$, $\mathcal{Z}_{obey}$ sends `tally` to the party T. When the adversary sends b to $\mathcal{Z}_{obey}$, $\mathcal{Z}_{obey}$ terminates with output b.

Furthermore, we define $\mathcal{Z}_{deceive} := \mathcal{Z}_{deceive}^{P,v,\mathcal{B}}$ as follows: Initially, $\mathcal{Z}_{deceive}$ sends a deception request to the party P. Then $\mathcal{Z}_{deceive}$ chooses votes $v_1, \dots, v_n$ according to the distribution $\mathcal{B}$ and gives these votes as input to the parties $P_i \in \mathcal{P} \setminus \{P\}$ (or, if $v_i = \bot$, sends no input to P_i). Then it sends v to the deceiver. (This will make the deceiver $\mathcal{D}$ defined below instruct P to cast vote v.) When the adversary sends `tally` to $\mathcal{Z}_{deceive}$, $\mathcal{Z}_{deceive}$ sends `tally` to the party T. When the adversary sends b to $\mathcal{Z}_{deceive}$, $\mathcal{Z}_{deceive}$ terminates with output b.

We define the deceiver $\mathcal{D}$ as follows: When receiving a state from party P, $\mathcal{D}$ instructs P to send this state to the adversary. (This is necessary only for formal reasons: since the adversary should believe that P is corrupted, he expects a state from P. Since we are in the case of static corruptions/deceptions, the state is only sent before the start of the protocol and is thus empty.) When $\mathcal{D}$ receives v from the environment, $\mathcal{D}$ instructs P to send v to the functionality $\mathcal{F}_{\text{vote}}$. (I.e., P should cast the vote v.) Messages coming from the adversary are ignored. In particular, when the adversary instructs P to cast some other vote, this is ignored.

Since π UC/c emulates $\mathcal{F}_{\text{vote}} := \mathcal{F}_{\text{vote}}^{\text{tally}}$, there exist a polynomial-time deceiver-simulator $\mathcal{D}_S$ and a polynomial-time adversary-simulator $\mathcal{A}'_S$ such that for all polynomial-time environments $\mathcal{Z}$, the networks $\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}\}$ and $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}\}$ are indistinguishable. (We write $\mathcal{F}_{\text{vote}}$ for the protocol containing $\mathcal{F}_{\text{vote}}$ and the dummy parties.)

By construction,

$$\Pr_{obey}(\mathcal{A}, \mathcal{B}) = \Pr[\text{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{obey}\}} = 1]. \tag{3}$$

(We omit the arguments k, z from EXEC for brevity.) Note that since no party is deceiving, the deceiver-simulator $\mathcal{D}_S$ does nothing.

We define the deception strategy $\mathfrak{d}$ as follows: A party P following $\mathfrak{d}$ and wishing to cast the vote v internally simulates $\mathcal{D}_S$. Then P sends the empty state to $\mathcal{D}_S$. (This is done for formal reasons: in the UC/c framework, $\mathcal{D}_S$ would get such an empty state when P is deceiving from the start. Hence this message informs $\mathcal{D}_S$ that P is deceiving.) Then P sends v to the internally simulated $\mathcal{D}_S$ as coming from the environment. Then P follows the instructions that $\mathcal{D}_S$ gives to it. In the case that only P is deceiving, $\mathcal{D}_S$ only sends instructions to P. Thus it is not necessary that P simulates any other machines communicating with $\mathcal{D}_S$.

Then, by construction,

$$\Pr_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B}) = \Pr[\text{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{deceive}\}} = 1]. \tag{4}$$

Compare the networks $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$ and $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{obey}\}$. In the first network, $\mathcal{Z}_{deceive}$ instructs the dummy-party $\tilde{P}$ (via the deceiver $\mathcal{D}$) to send the vote v to $\mathcal{F}_{\text{vote}}$. In the second network, $\mathcal{A}'_S$ instructs $\tilde{P}$ to send some other vote v^* to $\mathcal{F}_{\text{vote}}$ (where we write $v^* = \bot$ to indicate that $\mathcal{A}'_S$ does not instruct $\tilde{P}$ to vote before $\mathcal{A}'_S$ sends `tally` to the environment). In the ideal model, $\tilde{P}$ does not receive any incoming messages from other parties. Thus, in both networks, $\mathcal{A}'_S$ does not get any messages from $\tilde{P}$. Thus, $\mathcal{A}'_S$ can only use the tally to distinguish the networks. The distribution of the tally in the network $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{obey}\}$ is $\text{tally}(\mathcal{B}_{v^*})$, and the distribution of the tally in the network $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$ is $\text{tally}(\mathcal{B}_v)$. Since $\mathcal{Z}_{obey}$ and $\mathcal{Z}_{deceive}$ output the bit b received from $\mathcal{A}'_S$, it follows that

$$\begin{aligned}\left|\Pr[\text{EXEC}_{\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{obey}\}} = 1] - \Pr[\text{EXEC}_{\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}} = 1]\right| \\ \leq \max_{v^* \in \mathcal{V} \cup \{\perp\}} \Delta(\mathcal{B}_v, \mathcal{B}_{v^*}) = \text{Adv}_{ideal}(\mathcal{B}, v).\end{aligned}$$

Since for all polynomial-time $\mathcal{Z}$, the networks $\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}\}$ and $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}\}$ are indistinguishable, it follows that

$$\left|\Pr[\text{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{obey}\}} = 1] - \Pr[\text{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{deceive}\}} = 1]\right| \leq \text{Adv}_{ideal}(\mathcal{B}, v) + \mu$$

for some negligible function μ. Then with (3) and (4) we get that

$$\left|\Pr_{obey}(\mathcal{A}, \mathcal{B}) - \Pr_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})\right| \leq \text{Adv}_{ideal}(\mathcal{B}, v) + \mu.$$

This shows that the protocol π satisfies the second condition in Definition 11. (Notice that the construction of the deception strategy $\mathfrak{d}$ is independent of $\mathcal{A}$ and $\mathcal{B}$.)

We are left to show that $\text{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ and $\text{tally}(\mathcal{B}_v)$ are indistinguishable (first condition of Definition 11).

Let t denote the message received by $\mathcal{Z}_{deceive}$ from the party T (t is the tally). In the network $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$, t is the output of $\mathcal{F}_{\text{vote}}$. Thus the distribution of t is $\text{tally}(\mathcal{B}_v)$: The party P is instructed by $\mathcal{D}$ to send the vote v, all other parties cast votes chosen according to the distribution $\mathcal{B}$.

In the network $\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{deceive}\}$, by construction of $\mathcal{Z}_{deceive}$ and of $\mathfrak{d}$, the distribution of t is $\text{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$.

For contradiction, assume that $\text{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ and $\text{tally}(\mathcal{B}_v)$ were not computationally indistinguishable. Then there is an efficiently computable function $f : \{0,1\}^* \to \{0,1\}$ such that $|\Pr[f(\text{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})) = 1] - \Pr[f(\text{tally}(\mathcal{B}_v)) = 1]|$ is not negligible. Then we define $\mathcal{Z}^*_{deceive}$ like $\mathcal{Z}_{deceive}$, except that $\mathcal{Z}^*_{deceive}$ outputs $f(t)$. Then $|\Pr[\text{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}^*_{deceive}\}} = 1] - \Pr[\text{EXEC}_{\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}^*_{deceive}\}} = 1]|$ is not negligible. This is a contradiction to the fact that for all polynomial-time $\mathcal{Z}$, the networks $\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}\}$ and $\mathcal{F}_{\text{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}\}$ are indistinguishable. Thus $\text{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ and $\text{tally}(\mathcal{B}_v)$ are computationally indistinguishable and the first condition of Definition 11 is satisfied by π. □

The design of voting protocols that are UC/c secure is, of course, an open problem. We believe designing UC/c secure remote voting schemes to be a challenging problem that may involve novel cryptographic techniques. In the case of non-remote voting (i.e., involving voting booths and other partially trusted setup such as in, e.g., [Cha04, CRS05, MN06, BMQR07]), realising UC/c security might be much easier. We therefore particularly propose UC/c as a security definition for that setting.

Forced-abstention attacks. A protocol that UC/c emulates the functionality $\mathcal{F}_{\text{vote}}$ is also secure against forced-abstention attacks: In the ideal model, a deceiving party can cast a vote without the adversary noticing. Thus in the real model, a party can also vote without the adversary noticing. In some settings, security against forced-abstention attacks is impossible to achieve: If the adversary controls the network and can observe all communication of a party P, the adversary will always notice when a party participates in the protocol. To model a weaker form of incoercibility that does not imply security against forced-abstention attacks, one can change the definition of $\mathcal{F}_{\text{vote}}$ such that $\mathcal{F}_{\text{vote}}$ informs the adversary whenever a vote has been cast (revealing the identity of the voter, but note the vote itself).

4 Incoercible two-party protocols

In this section, we show that at least with respect to static corruptions/deceptions, UC/c secure two-party computation is possible using natural setup assumptions (such as, e.g., a public key infrastructure). We show this by proving that under certain conditions, protocols secure in the so-called Externalized UC framework are also UC/c secure. This allows us to reuse existing results in that framework.

4.1 Externalized UC framework

We first give a short overview over the Externalized UC (EUC) framework as proposed by Canetti, Dodis, Pass, and Walfish [CDPW07] (also known as Generalized UC, UC with global setup, or, proposed independently by Hofheinz, Müller-Quade, and Unruh [HUMQ07], UC with catalysts).

First, consider the UC framework and assume that some real protocol π uses a functionality $\mathcal{F}$, say a CRS functionality. Then only the adversary and the protocol parties have direct access to the CRS. The environment learns the CRS only through the adversary. In the real model, this is as good as having direct access because without loss of generality, the adversary will not lie to the environment. In the ideal model, however, the simulator can choose an arbitrary (fake) value for the CRS instead (containing a trapdoor); the environment will not be able to notice that the CRS was chosen differently. The security proof of most UC secure protocols in the CRS-hybrid model are based on a simulator that chooses such a fake CRS. However, there are two disadvantages in letting the simulator choose the value of the CRS. First, when composing different protocols that all use a CRS, each of them needs its own CRS. Second, as pointed out by Pass [Pas03], a security definition where the simulator may choose the value of the CRS does not guarantee deniability.

The EUC framework removes these two restrictions by extending the UC framework. In the EUC framework, the environment is allowed to directly query the functionality $\mathcal{F}$, both in the real and in the ideal model. For example, in the case of the CRS functionality, the environment will know the true value of the CRS (as chosen by the functionality), and the simulator will not be able to make up a fake value.

To make this more formal, we first introduce the notion of a *shared functionality.* Such a functionality is derived from a normal functionality but additionally honours requests from the environment. The environment can make requests in the name of any party and in the name of the adversary.

Definition 13 (Shared functionality) *Let $\mathcal{F}$ be a functionality. The shared functionality $\bar{\mathcal{F}}$ behaves like $\mathcal{F}$, with the following extension: When $\bar{\mathcal{F}}$ gets a message from some protocol party or the adversary, the request is forwarded to an internally simulated $\mathcal{F}$, and the answer m' of $\mathcal{F}$ is forwarded back to the party or adversary. When $\bar{\mathcal{F}}$ gets a message (P, m) from the environment where P is the identity of some party, the message m is given to the internally simulated $\mathcal{F}$ as coming from P. The answer m' of $\mathcal{F}$ is forwarded back to $\mathcal{Z}$.*

Given this notion of a shared functionality, it is easy to define EUC security. In the EUC framework, the environment has access to the shared functionality both in the real and in the ideal model.

Definition 14 (EUC security) *Let π be a protocol using a shared functionality $\bar{\mathcal{F}}$. Let ρ be a protocol. We say that π $\bar{\mathcal{F}}$-*EUC *emulates ρ if for any polynomial-time adversary $\mathcal{A}$ there exists a polynomial-time adversary $\mathcal{S}$ (the adversary-simulator) such that for any polynomial-time environment $\mathcal{Z}$ the networks $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ and $\rho \cup \{\bar{\mathcal{F}}, \mathcal{S}, \mathcal{Z}\}$ are indistinguishable.*

Since π already contains $\bar{\mathcal{F}}$, we have that $\bar{\mathcal{F}}$ is present both in the real and in the ideal model.

Since the simulator is not allowed to simulate the functionality $\bar{\mathcal{F}}$ any more, EUC security is strictly stronger than UC security. In particular, it was shown by Canetti et al. [CDPW07] that in the EUC framework, it is not even possible to construct secure commitment protocols using a CRS. There are, however, alternative functionalities that allow to design EUC secure protocols:

The *key registeration with knowledge (KRK) functionality* $\mathcal{F}_{\mathrm{krk}}$ is a functionality where each party may register a public key/secret key pair and every party may request the public keys of all parties and the secret key of itself. The *restricted KRK functionality* $\mathcal{F}^*_{\mathrm{krk}}$ is defined like $\mathcal{F}_{\mathrm{krk}}$ except that *uncorrupted* parties are not allowed to retrieve their secret key.[4]

The *augmented CRS (ACRS) functionality* $\mathcal{F}_{\mathrm{acrs}}$ chooses a public key and a corresponding master secret key, and derives for each party a corresponding individual secret key. The public key is given to all parties, the secret key of each party is only given to that party. The *restricted ACRS functionality* $\mathcal{F}^*_{\mathrm{acrs}}$ is defined like $\mathcal{F}_{\mathrm{acrs}}$, except that uncorrupted parties are not allowed to retrieve their secret key.

For details on the restricted KRK and ACRS functionalities, see [CDPW07]. (They are simply called KRK and ACRS functionalities ($\mathcal{G}_{\mathrm{krk}}$ and $\mathcal{G}_{\mathrm{acrs}}$) there, we added the qualifier "restricted" for disambiguation.)

The *signature card functionality* $\mathcal{F}_{\mathrm{sc}}$ with owner P picks a signing/verification key pair and reveals the verification key to all parties. The party P (the owner) may send arbitrary messages m to $\mathcal{F}_{\mathrm{sc}}$ and receives signatures of m back. The signing key is never revealed. The *restricted signature card functionality* $\mathcal{F}^*_{\mathrm{sc}}$ additionally allows one protocol session to lock the signature card. While the signature card is locked by a given protocol session, in all other protocol sessions, even the owner P may not sign messages.[5]

For details on the restricted signature card functionality, see [HUMQ07] (simply called the signature card functionality $\mathcal{F}_{\mathrm{sc}}$ there).

Theorem 15 (EUC multi-party computation [HUMQ07, CDPW07]) *Let $\bar{\mathcal{F}} \in \{\bar{\mathcal{F}}^*_{\mathrm{krk}}, \bar{\mathcal{F}}^*_{\mathrm{acrs}}, \bar{\mathcal{F}}^*_{\mathrm{sc}}\}$. Let $\mathcal{G}$ be a well-formed[6] functionality. Then there is a protocol π in the $\bar{\mathcal{F}}$-hybrid model such that π $\bar{\mathcal{F}}$-EUC emulates $\mathcal{G}$ with static corruptions.*

Proof. Canetti, Dodis, Pass, and Walfish [CDPW07] show that for $\bar{\mathcal{F}} \in \{\bar{\mathcal{F}}^*_{\mathrm{krk}}, \bar{\mathcal{F}}^*_{\mathrm{acrs}}\}$, there is a protocol π_{com} in the $\bar{\mathcal{F}}$-hybrid model such that $\bar{\mathcal{F}}$-EUC emulates the commitment functionality

[4]The definition of $\mathcal{F}_{\mathrm{krk}}$ in [CDPW07] lets parties choose the randomness used to generate their key pair when registering. Thus every party knows its own secret key and the restriction that uncorrupted parties are not allowed to retrieve their own secret keys is meaningless. We therefore assume that the intended definition in [CDPW07] is that only corrupted parties may chose the randomness while for uncorrupted ones the randomness is chosen by the functionality.

[5]Strictly speaking, a functionality with such a locking mechanism does not fit our definition of shared functionalities: Such a locking functionality will have to distinguish different protocol sessions. In particular, it might answer differently to a query sent by the owner, and the same query sent by the environment in the name of the owner, contradicting Definition 13. This can be remedied by a slight change in the definition of the locking mechanism: Instead of locking with respect to a given session, locking requests are accompanied by a secret random nonce N. Then only unlocking and signing requests containing N will be honoured. As long as the uncorrupted parties do not divulge N, this has the same effect as session-wise locking, and the functionality will then not have to distinguish between the environment and protocol parties.

[6]A well-formed functionality is one whose behaviour does not depend on which parties are corrupted or deceiving.

$\mathcal{F}_{\mathrm{com}}$. Hofheinz, Müller-Quade, and Unruh [HUMQ07] show that for $\bar{\mathcal{F}} = \bar{\mathcal{F}}^*_{\mathrm{sc}}$, there is a protocol π_{com} in the $\bar{\mathcal{F}}$-hybrid model such that $\bar{\mathcal{F}}$-EUC emulates $\mathcal{F}_{\mathrm{com}}$. [CDPW07, after Thm. 5] show that for any shared functionality $\bar{\mathcal{F}}$ and any well-formed functionality $\mathcal{G}$, given a protocol π_{com} $\bar{\mathcal{F}}$-EUC emulating $\mathcal{F}_{\mathrm{com}}$, we can construct a protocol π that $\bar{\mathcal{F}}$-EUC emulates $\mathcal{G}$ with static corruptions. (Indeed, if $\mathcal{G}$ is not only well-formed, but even adaptively well-formed as defined by Canetti, Lindell, Ostrovsky, and Sahai [CLOS02], then π even $\bar{\mathcal{F}}$-EUC emulates $\mathcal{G}$ with *adaptive* corruptions. However, we do not need this fact in the following.) Note that this result implicitly uses our convention that we use secure channels that do not leak anything to the adversary. Otherwise, we could not realise all functionalities $\mathcal{G}$; only functionalities that notify the adversary when invoked would be possible. □

4.2 EUC security implies UC/c security

In this section, we show that under certain conditions, an EUC secure protocol is already UC/c secure with static corruptions/deceptions. To state our result, we first introduce some additional notation.

First, to capture the relation between $\bar{\mathcal{F}}^*_{\mathrm{krk}}, \bar{\mathcal{F}}^*_{\mathrm{acrs}}, \bar{\mathcal{F}}^*_{\mathrm{sc}}$ and $\mathcal{F}_{\mathrm{krk}}, \mathcal{F}_{\mathrm{acrs}}, \mathcal{F}_{\mathrm{sc}}$, we introduce the notion of a restriction:

Definition 16 (Restrictions) *Let $\mathcal{F}$ and $\mathcal{F}^*$ be functionalities. We say $\mathcal{F}^*$ is a restriction of $\mathcal{F}$ if $\mathcal{F}^*$ behaves like $\mathcal{F}$, except that for each party P there is an efficiently recognisable set C_P of messages such that $\mathcal{F}^*$ ignores any message $m \in C_P$ from P. Here C_P may depend on the messages exchanged between $\mathcal{F}^*$ and P so-far.*

We say $\bar{\mathcal{F}}^*$ is a shared restriction of $\mathcal{F}$ if there exists a restriction $\mathcal{F}^*$ of $\mathcal{F}$ such that $\bar{\mathcal{F}}^*$ is the shared functionality corresponding to $\mathcal{F}^*$.

We call a protocol $\pi \ni \bar{\mathcal{F}}^*$ restriction-compatible to $\bar{\mathcal{F}}^*$ if no honest party P in π ever sends a message $m \in C_P$ to $\bar{\mathcal{F}}^*$.

For example, in the case of $\bar{\mathcal{F}}^* = \bar{\mathcal{F}}^*_{\mathrm{krk}}$, C_P would be the set of messages requesting a secret key. Hence $\bar{\mathcal{F}}^*_{\mathrm{krk}}$ is a shared restriction of $\mathcal{F}_{\mathrm{krk}}$. Similarly, $\bar{\mathcal{F}}^*_{\mathrm{acrs}}$ is a shared restriction of $\mathcal{F}_{\mathrm{acrs}}$. In the case of $\bar{\mathcal{F}}^* = \bar{\mathcal{F}}^*_{\mathrm{sc}}$, when the signature card is locked for the session with nonce N (see footnote 5), C_P would be the set of unlocking and signing requests coming from the owner P of the card but not tagged with N. When the signature card is not locked, C_P is empty. Hence $\bar{\mathcal{F}}^*_{\mathrm{sc}}$ is a shared restriction of $\mathcal{F}_{\mathrm{sc}}$.

We additionally need to refine the notion of EUC security to capture certain technical requirements on the simulator:

Definition 17 (Special simulator) *We say π $\bar{\mathcal{F}}$-EUC emulates ρ with a special simulator if the adversary-simulator $\tilde{\mathcal{A}}_S$ corresponding to the dummy-adversary $\tilde{\mathcal{A}}$ has the following property:*

When the environment sends a message $(\mathcal{F}, m)$ to the $\tilde{\mathcal{A}}_S$, $\tilde{\mathcal{A}}_S$ sends m to $\bar{\mathcal{F}}$. When $\bar{\mathcal{F}}$ sends m to $\tilde{\mathcal{A}}_S$, $\tilde{\mathcal{A}}_S$ sends $(\mathcal{F}, m)$ to the environment. These messages are not recorded in the state of $\tilde{\mathcal{A}}_S$. $\tilde{\mathcal{A}}_S$ never sends a message m to $\bar{\mathcal{F}}$ unless he got $(\mathcal{F}, m)$ from the environment, and never sends a message $(\mathcal{F}, m')$ to the environment unless he got m' from $\bar{\mathcal{F}}$.

In other words, $\tilde{\mathcal{A}}_S$ provides a direct connection between $\bar{\mathcal{F}}$ and $\mathcal{Z}$ which he does not even listen too. In the case that no party is corrupted, the dummy-adversary $\tilde{\mathcal{A}}$ in the real model only has access to $\bar{\mathcal{F}}$. Thus in this case, the behaviour of $\tilde{\mathcal{A}}_S$ is fully specified by Definition 17, namely $\tilde{\mathcal{A}}_S$ forwards messages between $\bar{\mathcal{F}}$ and the environment and does nothing else. In most protocols, this is the natural behaviour of $\tilde{\mathcal{A}}_S$ in the uncorrupted case anyway.

If some party is corrupted, Definition 17 implies that $\tilde{\mathcal{A}}_S$ cannot query $\bar{\mathcal{F}}$ (except when forwarding messages from the environment). For example, if $\bar{\mathcal{F}} = \bar{\mathcal{F}}^*_{\mathrm{krk}}$, then $\tilde{\mathcal{A}}_S$ could not even query $\bar{\mathcal{F}}$ to get the public keys of the parties. This seems to be a strong restriction. The adversary-simulator $\tilde{A}_S$ can, however, instruct the corrupted party to request the public keys from the functionality. Thus everything the adversary could do by directly contacting $\bar{\mathcal{F}}$ can also be done by giving suitable instructions to the corrupted party. Analogous reasoning holds for $\bar{\mathcal{F}} = \bar{\mathcal{F}}^*_{\text{acrs}}$ and $\bar{\mathcal{F}} = \bar{\mathcal{F}}^*_{\text{sc}}$. Thus at least for these functionalities, Definition 17 does not pose a restriction in the case of a corrupted party.

Corollary 18 (EUC multi-party computation with special simulator) *Let $\bar{\mathcal{F}}^* \in \{\bar{\mathcal{F}}^*_{\text{krk}}, \bar{\mathcal{F}}^*_{\text{acrs}}, \bar{\mathcal{F}}^*_{\text{sc}}\}$. Let $\mathcal{G}$ be a well-formed functionality. Then there is a protocol π in the $\bar{\mathcal{F}}^*$-hybrid model such that π $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ with static corruptions and special simulator. Furthermore, π is restriction-compatible to $\bar{\mathcal{F}}^*$.*

Proof. The simulators in the constructions from [HUMQ07, CDPW07] already construct dummy-adversary-simulators that fulfil Definition 17 in the uncorrupted case. In the corrupted case, their simulators can be made to fulfil Definition 17 by replacing all direct requests to $\bar{\mathcal{F}}^*$ by indirect calls through the corrupted party. The protocols constructed in [HUMQ07, CDPW07] are restriction-compatible to $\bar{\mathcal{F}}^*$. Then the proof is as for Theorem 15. $\square$

Finally, we will consider a restricted class of functionalities:

Definition 19 (Silent functionalities) *A functionality $\mathcal{G}$ is* silent *if it ignores all messages from the adversary or the deceiver and never sends messages to the adversary or the deceiver.*

In other words, silent functionalities are those that leak no information. Note that it is not excluded that the adversary or deceiver indirectly gets access to $\mathcal{G}$ through a corrupted or deceiving party.

The main result of this section (Theorem 23) will be to show that, under certain conditions, a protocol π UC/c emulates a functionality $\mathcal{G}$ with static corruptions/deceptions if π EUC emulates $\mathcal{G}$ with static corruptions.

The most important case is covered by the following lemma:

Lemma 20 *Let $\mathcal{G}$ be a silent polynomial-time functionality. Let $\mathcal{F}$ be a polynomial-time functionality and let $\bar{\mathcal{F}}^*$ be a shared restriction of $\mathcal{F}$. Let π be a two-party protocol in the $\mathcal{F}$-hybrid model with parties P and Q. Let $\pi_0 := \pi \setminus \mathcal{F} \cup \{\bar{\mathcal{F}}^*\}$. Assume that π_0 is restriction-compatible to $\bar{\mathcal{F}}^*$ and that π_0 $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ for for static corruptions of Q.*[7]

Then π UC/c emulates $\mathcal{G}$ for statically corrupted Q and for statically deceiving Q.[8]

Proof. To simplify the proof, we first introduce some alternative notation. In the definition of the network model in the UC framework, machines specify the recipient of a message m by attaching the identity of the recipient to the message. Although this is convenient for defining protocols, it makes certain proofs relatively difficult to formulate: In intermediate proof steps, we often consider changed machines that send their messages to different recipients than their original program would prescribe (the machines are "rewired"). Furthermore, machines like the dummy-adversary that simply forward messages expect headers with the recipients of the messages. Explaining the constructions below in such a setting leads to complicated and hard-to-read textual descriptions even in the case of relatively simple "rewiring".

To make presentation simpler, we instead assume that each machine has a number of named ports. Between two ports a and b we can have a connection which means that messages send on

[7] That is, we assume a corruption schedule in which P is never corrupted and Q may be corrupted, but only before the protocol starts.

[8] That is, we assume a corruption schedule in which the environment always makes Q corrupted or deceiving before the protocol start and never leaves Q uncontrolled and always leaves P uncontrolled.

a are received on b. Any network using this formalism can easily be converted into a network using the original formalism with message-headers. The formalism using ports, however, has the advantage that we can easily describe the configuration of a network by giving a picture with lines indicating the connections between ports. In the pictures, we follow the convention that in all occurrences of a given machine, the relative position of its ports is the same. This allows to compare different networks without having to pay attention to the (somewhat hard to read) port names.

EUC security, uncorrupted case. By assumption, π_0 $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ with a special simulator. Consider an environment $\mathcal{Z}^*$ that does not query the functionality $\bar{\mathcal{F}}^*$ in the name of P (but $\mathcal{Z}^*$ may query $\bar{\mathcal{F}}^*$ in the name of Q). Let $\hat{\mathcal{A}}^0$ denote the dummy-adversary in this case. (We write $\hat{\mathcal{A}}$ instead of $\tilde{\mathcal{A}}$ to distinguish this dummy-adversary for the UC/c setting, and we write the superscript 0 to indicate that it is the dummy-adversary for the uncorrupted case.) The resulting real model $\pi_0 \cup \{\hat{\mathcal{A}}^0, \mathcal{Z}^*\}$ is depicted in Figure 3, network A. (Here and in the following pictures, we draw several boxes marked $\mathcal{Z}^*$ or $\mathcal{Z}$. These a supposed to denote a *single* machine, the separation into several boxes is only for graphical reasons.)

Note the following particularities:

Instead of $\bar{\mathcal{F}}^*$, we have written $\mathcal{F}$ in network A because the shared functionality $\bar{\mathcal{F}}^*$ and the functionality $\mathcal{F}$ behave identically, except that we allow $\mathcal{Z}^*$ to access $\bar{\mathcal{F}}^*$ in the name of Q and that certain messages from uncorrupted parties are ignored by $\bar{\mathcal{F}}^*$. The additional access for $\mathcal{Z}^*$, however, is already expressed by the connections in network A. And the messages that would be ignored by $\bar{\mathcal{F}}^*$ are not sent by honest parties anyway since π_0 is restriction-compatible to $\bar{\mathcal{F}}^*$.

Furthermore, there are two connections ending in the port *fq* of $\mathcal{F}$. This is due to the fact that both $\mathcal{Z}^*$ and Q may access $\mathcal{F}$ in the name of Q. Responses to messages from $\mathcal{Z}^*$ and Q arriving at that port are sent back to the $\mathcal{Z}^*$ and Q, respectively. The dummy-adversary has no connections to the protocol parties P and Q because they are uncorrupted and we assume secure channels.

The corresponding ideal model $\mathcal{G} \cup \{\bar{\mathcal{F}}^*, \hat{\mathcal{A}}^0_S, \mathcal{Z}^*\}$ is depicted in Figure 3, network B. Here the dummy-adversary-simulator $\hat{\mathcal{A}}^0_S$ is a machine that simply forwards all messages between its ports *fa* and *fa*$'$. This is due to the fact that we assumed EUC-emulation with a special simulator. The dummy-parties are denoted by $\tilde{P}$ and $\tilde{Q}$.

Since π_0 $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ with a special simulator, we have that the networks A and B are indistinguishable for all polynomial-time environments $\mathcal{Z}^*$.

EUC security, Q corrupted. By assumption, π_0 $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ in the case of corrupted Q. Consider an environment $\mathcal{Z}$ that does not query the functionality $\bar{\mathcal{F}}^*$ (neither in the name of P nor Q). Let $\hat{\mathcal{A}}$ denote the dummy-adversary in this case. The resulting real model $\pi_0 \cup \{\hat{\mathcal{A}}, \mathcal{Z}\}$ is depicted in Figure 3, network C. In network C, we write Q^* for the corrupted party Q for clarity (it will be important later to distinguish Q^* from the uncorrupted Q). The ideal model $\mathcal{G} \cup \{\bar{\mathcal{F}}^*, \hat{\mathcal{A}}_S, \mathcal{Z}\}$ is depicted in Figure 3, network D. We write $\tilde{Q}^*$ for the corrupted dummy-party $\tilde{Q}$. We can use $\mathcal{F}$ instead of $\bar{\mathcal{F}}^*$ for the same reasons as in the case of an honest Q. Note that the dummy-adversary-simulator $\hat{\mathcal{A}}_S$ routes all queries from $\mathcal{Z}$ on the port *fa*$'$ directly to the functionality $\mathcal{F}$. This is because we assumed EUC emulation with special simulator. We then have that for all polynomial-time $\mathcal{Z}$, the networks C and D are indistinguishable.

UC/c security, Q corrupted. We now proceed to show that π UC/c emulates $\mathcal{G}$ for statically corrupted Q. By Lemma 6, it is sufficient to construct an adversary-simulator $\tilde{\mathcal{A}}_S$ and a deceiver-simulator $\tilde{\mathcal{D}}^0_S$ that for every environment $\mathcal{Z}$ that corrupts Q, the real model $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}^0_S, \mathcal{Z}\}$ and the ideal model $\mathcal{G} \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}^0, \mathcal{Z}\}$ are indistinguishable. Here $\tilde{\mathcal{A}}$ denotes the dummy-adversary, and $\tilde{\mathcal{D}}^0$ the dummy-deceiver. (We use the superscript 0 to stress the fact that we are considering the case that no party is corrupted, and thus $\tilde{\mathcal{D}}^0$ and $\tilde{\mathcal{D}}^0_S$ are trivial.) The real model is depicted

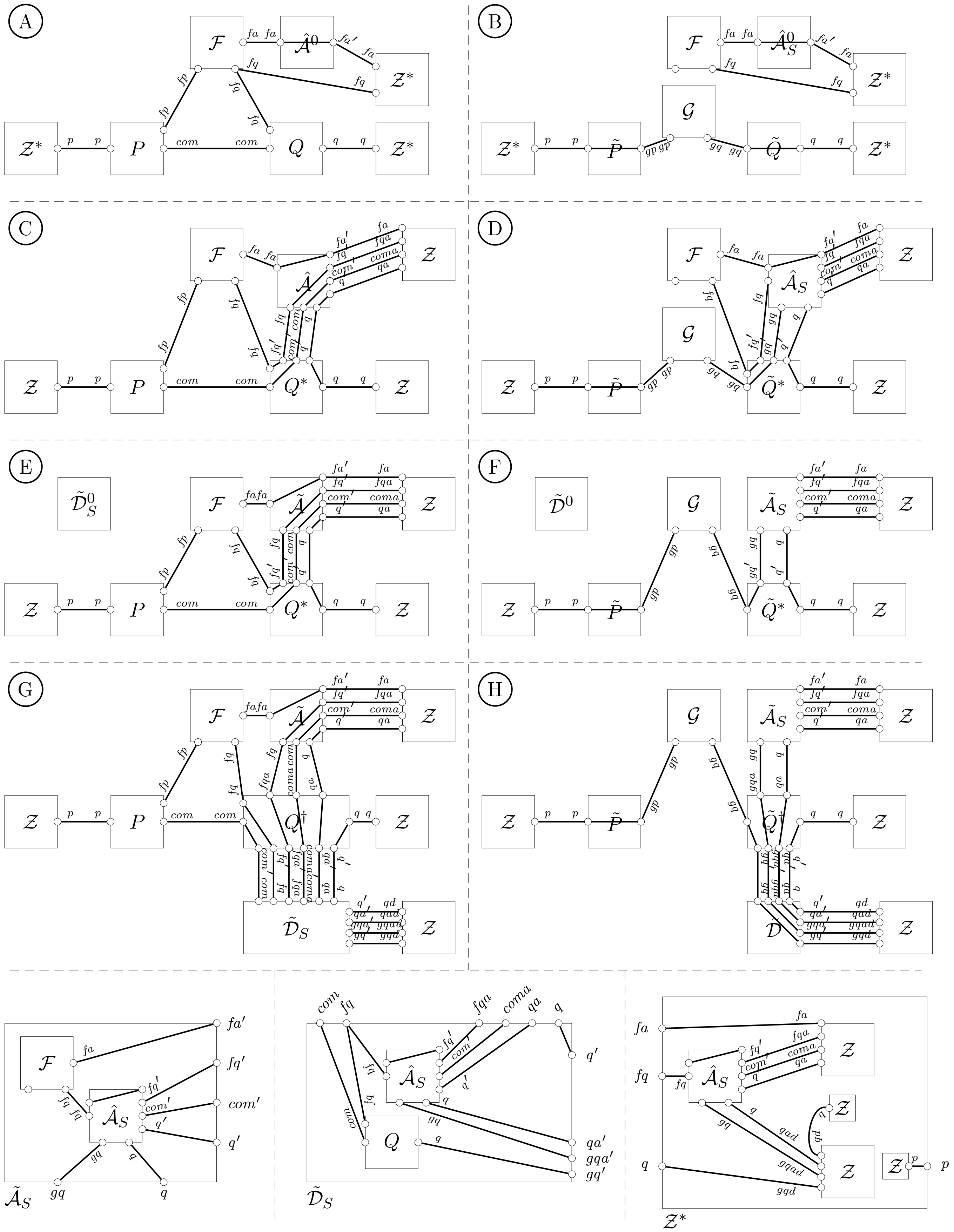

Figure 3: Networks A–H and machines $\tilde{\mathcal{A}}_S$, $\tilde{\mathcal{D}}_S$, and $\mathcal{Z}^*$ from the proof of Lemma 20.

in Figure 3, network E. The ideal model is depicted in network F. Note that the dummy-deceiver $\tilde{\mathcal{D}}^0$ has no connections to the protocol or the functionality $\mathcal{G}$ (the latter is silent by assumption), and therefore also no connections to the environment. The same holds for $\tilde{\mathcal{D}}_S^0$. We can thus fix $\tilde{\mathcal{D}}_S^0$ to be the machine that does nothing. We construct the adversary-simulator $\tilde{\mathcal{A}}_S$ as follows: It internally simulates an instance of the functionality $\mathcal{F}$, and an instance of the adversary-simulator $\hat{\mathcal{A}}_S$ (from the EUC setting). $\tilde{\mathcal{A}}_S$ connects these machines between each other and to the ports of $\tilde{\mathcal{A}}_S$ as depicted in Figure 3, bottom.

With this definition of $\tilde{\mathcal{A}}_S$, we have that the networks C and E are perfectly indistinguishable, and the networks D and F are perfectly indistinguishable. (To see that two networks are perfectly indistinguishable, check for each port of the machines $\mathcal{Z}$, $\mathcal{F}$, $\mathcal{G}$, P, $\hat{\mathcal{A}}_S$ that the connection, when following all forwardings, leads to the same port in both networks. E.g., in network D the port q of $\mathcal{Z}$ is connected to q of $\tilde{Q}^*$, from there to q' of $\tilde{Q}^*$, to q of $\hat{\mathcal{A}}_S$; in network F, the port q of $\mathcal{Z}$ is connected to q of $\tilde{Q}^*$, to q' of $\tilde{Q}^*$, to q of $\tilde{\mathcal{A}}_S$, and finally to q of the internally simulated $\hat{\mathcal{A}}_S$; thus in both networks, the port q of $\mathcal{Z}$ is connected to the port q of $\hat{\mathcal{A}}_S$.) Note that in network D, the connection between $\mathcal{Z}$'s and $\mathcal{F}$'s ports *fa* is routed through $\hat{\mathcal{A}}_S$, while in network F it does not reach $\hat{\mathcal{A}}_S$. This is permissible because by assumption, $\hat{\mathcal{A}}_S$ only forwards this connection without accessing the transferred data in any way. Since we already know that networks C and D are indistinguishable, it follows that networks E and F are indistinguishable, thus we have that π UC/c emulates $\mathcal{G}$ for statically corrupted Q.

UC/c security, Q deceiving. We now proceed to show that π UC/c emulates $\mathcal{G}$ for statically deceiving Q. By Lemma 6, it is sufficient to construct an adversary-simulator $\tilde{\mathcal{A}}_S$ and a deceiver-simulator $\tilde{\mathcal{D}}_S$ such that for every environment $\mathcal{Z}$ that makes Q deceiving, the real model $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}\}$ and the ideal model $\mathcal{G} \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are indistinguishable. Here $\tilde{\mathcal{A}}$ denotes the dummy-adversary, and $\tilde{\mathcal{D}}$ the dummy-deceiver. The real model is depicted in Figure 3, network G. The ideal model is depicted in network H. We write $Q^\dagger$ for the deceiving party Q. $Q^\dagger$ has ports *com*, *fq*, and q to P, $\mathcal{F}$, and $\mathcal{Z}$. Furthermore, since from the point of view of the adversary, $Q^\dagger$ is supposed to look like the corrupted Q^*, it has ports *coma*, *fqa*, and *qa* that are connected to the adversary (and that would, if $Q^\dagger$ was corrupted, be connected to the ports *com*, *fq*, and q). Finally, since $Q^\dagger$ is controlled by the deceiver-simulator, for each port x, it has a port x' connected to the deceiver-simulator. Analogously for the deceiving dummy-party $\tilde{Q}^\dagger$.

We use the same adversary-simulator $\tilde{\mathcal{A}}_S$ as above (see Figure 3, bottom). The deceiver-simulator $\tilde{\mathcal{D}}_S$ internally simulates the machines $\hat{\mathcal{A}}_S$ (the adversary-simulator from the EUC setting) and the uncorrupted Q.[9] The ports are connected as shown in Figure 3, bottom. Note that both $\hat{\mathcal{A}}_S$ and Q are connected to $\tilde{\mathcal{D}}_S$'s port *fq*. This means that messages from both $\hat{\mathcal{A}}_S$ and Q are forwarded to that port, and answers are sent back to the corresponding sender. (As was the case with the port *fq* of $\mathcal{F}$ in network A.)

To show that for any environment $\mathcal{Z}$, the networks G and H are indistinguishable, we additionally construct an environment $\mathcal{Z}^*$ internally simulating $\mathcal{Z}$ and $\hat{\mathcal{A}}_S$ with connections as shown in Figure 3, bottom.

With these definitions of $\tilde{\mathcal{D}}_S$, and $\mathcal{Z}^*$, the networks A and G are perfectly indistinguishable. And with the definitions of $\tilde{\mathcal{A}}_S$ and $\mathcal{Z}^*$, the networks B and H are perfectly indistinguishable. Since we know that for any polynomial-time $\mathcal{Z}^*$, the networks A and B are indistinguishable, it follows that the networks G and H are indistinguishable, thus π UC/c emulates $\mathcal{G}$ for statically deceiving Q.

Summing up. We have shown that π UC/c emulates $\mathcal{G}$ for statically corrupted Q and that π UC/c emulates $\mathcal{G}$ for statically deceiving Q. Thus π UC/c emulates $\mathcal{G}$ for static corrup-

[9]It is at this point that we use that fact that $\mathcal{G}$ is silent. If there was a connection between $\mathcal{G}$ and $\hat{\mathcal{A}}_S$, $\tilde{\mathcal{D}}_S$ would have to connect to $\mathcal{G}$ in the name of the adversary. But $\tilde{\mathcal{D}}_S$ cannot do this.

tions/deceptions of Q: The deceiver-simulator behaves like $\tilde{\mathcal{D}}_S^0$ above when Q is corrupted, and like $\tilde{\mathcal{D}}_S$ above when Q is deceiving. The adversary-simulator is $\tilde{\mathcal{A}}_S$ in both cases (this is important, since the adversary-simulator cannot distinguish whether a party is corrupted or deceiving). □

Lemma 21 *Let π be a protocol, let $\mathcal{F}, \mathcal{G}$ be functionalities, and let $\bar{\mathcal{F}}^*$ be a shared restriction of $\mathcal{F}$. Let $\pi_0 := \pi \setminus \{\mathcal{F}\} \cup \{\bar{\mathcal{F}}^*\}$. If π_0 is restriction-compatible to $\bar{\mathcal{F}}^*$ and π_0 $\bar{\mathcal{F}}^*$-EUC emulates $\mathcal{G}$ without corruptions, then π UC/c emulates $\mathcal{G}$ without corruptions/deceptions.*

Proof. EUC emulation implies UC emulation. Hence π_0 UC emulates $\mathcal{G}$ without corruptions. Since π_0 is restriction-compatible, π never sends a query to $\mathcal{F}$ that $\bar{\mathcal{F}}^*$ would ignore. Thus π UC emulates $\mathcal{G}$ without corruptions. In the case without corruptions/deceptions, UC/c emulation is equivalent to UC emulation. Hence π UC/c emulates $\mathcal{G}$ without corruptions/deceptions. □

Lemma 22 *Let π be a two-party protocol with parties P and Q and using a polynomial-time functionality $\mathcal{F}$, and let $\mathcal{G}$ be a polynomial-time functionality. Then π UC/c emulates $\mathcal{G}$ for statically corrupted or deceiving P and Q.*[10]

Proof. Let $\tilde{\mathcal{A}}$ denote the dummy-adversary and $\tilde{\mathcal{D}}$ the dummy-deceiver. By Lemma 6, we need to show that there is a dummy-adversary-simulator $\tilde{\mathcal{A}}_S$ and a dummy-deceiver-simulator $\tilde{\mathcal{D}}_S$ such that the real model $\pi \cup \{\tilde{\mathcal{A}}, \tilde{\mathcal{D}}_S, \mathcal{Z}\}$ and the real model $\mathcal{G} \cup \{\tilde{\mathcal{A}}_S, \tilde{\mathcal{D}}, \mathcal{Z}\}$ are indistinguishable for polynomial-time environments $\mathcal{Z}$ where $\mathcal{Z}$ does one of the following at the beginning of the execution: (i) $\mathcal{Z}$ corrupts P and Q, (ii) $\mathcal{Z}$ corrupts P and makes Q deceiving, (iii) $\mathcal{Z}$ makes P deceiving and corrupts Q, (iv) $\mathcal{Z}$ makes P and Q deceiving.

We use the notation using ports described in the proof of Lemma 20. We define $\tilde{\mathcal{A}}_S$ to internally simulate $\mathcal{F}$ and to forward messages between its ports as depicted in Figure 4, network B. Then, in case (i), the real model is network A, and the ideal model is network B. Networks A and B are perfectly indistinguishable. (This can be seen by following the connections in Figure 4.)

In case (ii), we define $\tilde{\mathcal{D}}_S$ to internally simulate $\mathcal{G}$ and to forward messages between its ports as depicted in Figure 4, network C. The real model is network C, and the ideal model is network D. Note that in network D, we use the same definition for $\tilde{\mathcal{A}}_S$ as in network B since the adversary-simulator cannot distinguish whether Q is corrupted or deceiving. Networks C and D are perfectly indistinguishable.

Case (iii) is handled analogously to case (ii).

In case (iv), we define $\tilde{\mathcal{D}}_S$ to internally simulate $\mathcal{G}$ and to forward messages between its ports as depicted in Figure 4, network E. (Note that $\tilde{\mathcal{D}}_S$ is defined differently than in the previous cases. This is possible because the deceiver-simulator can distinguish whether parties are corrupted or deceiving.) The real model is network E, and the ideal model is network F. Networks E and F are perfectly indistinguishable.

Thus, in all cases, the real and the ideal model are perfectly indistinguishable. Hence π UC/c emulates $\mathcal{G}$ for statically corrupted or deceiving P and Q. □

Theorem 23 *Let $\mathcal{G}$ be a silent polynomial-time functionality. Let $\mathcal{F}$ be a polynomial-time functionality and let $\bar{\mathcal{F}}^*$ be a shared restriction of $\mathcal{F}$. Let π be a two-party protocol in the $\mathcal{F}$-hybrid model with parties P and Q. Let $\pi_0 := \pi \setminus \mathcal{F} \cup \{\bar{\mathcal{F}}^*\}$. Assume that π_0 is restriction-compatible to $\bar{\mathcal{F}}^*$ and that π_0 $\bar{\mathcal{F}}$-EUC emulates $\mathcal{G}$ with static corruptions and special simulator.*

Then π UC/c emulates $\mathcal{G}$ with static corruptions/deceptions.

[10]That is, we consider environments that statically make P and Q corrupted or deceiving, and that do not leave any of the machines uncontrolled. We do allow that one party is corrupted and the other deceiving.

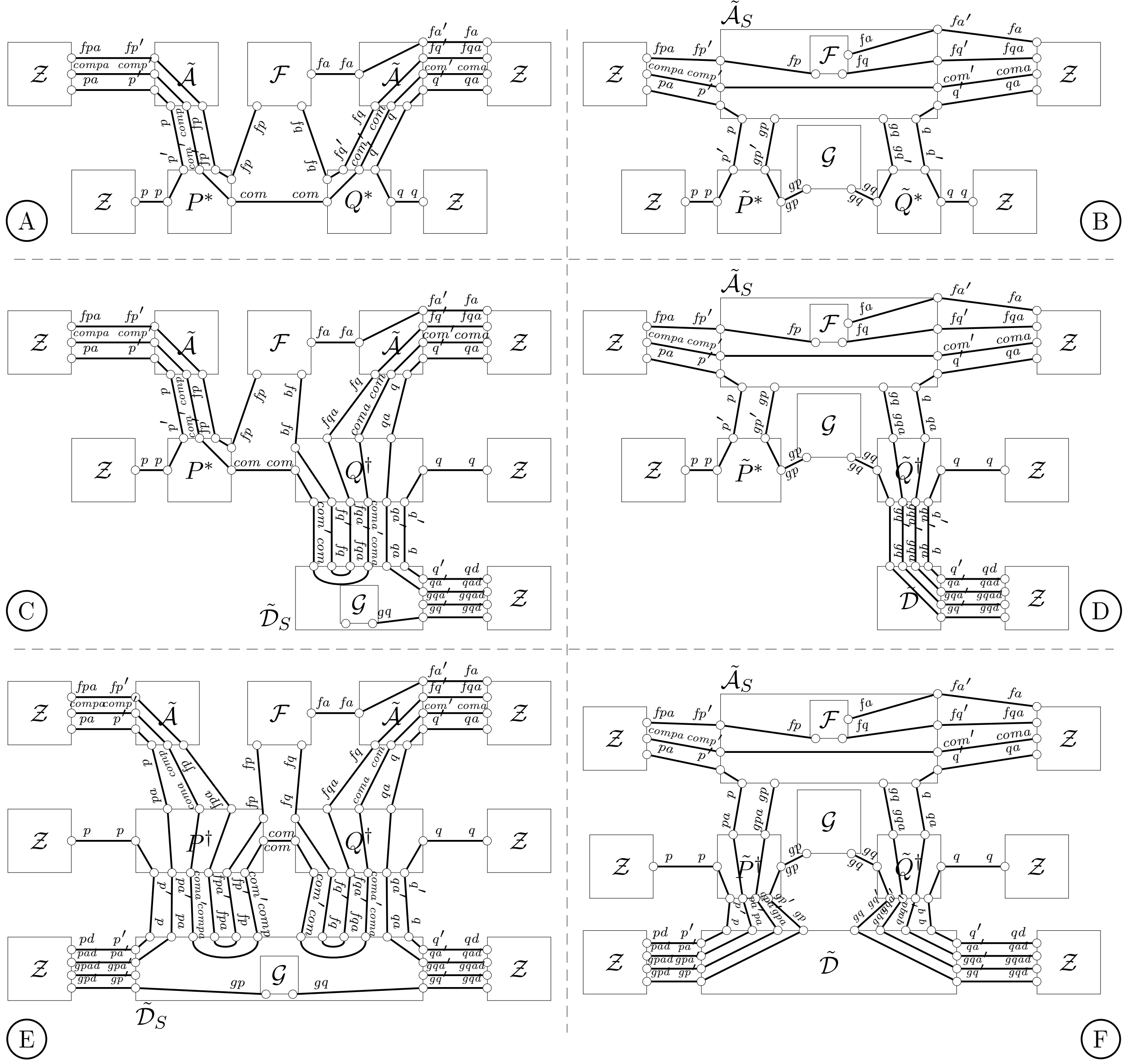

Figure 4: Networks A–F from the proof of Lemma 22.

Proof. From Lemmas 20, 21, and 22. □

Corollary 24 (UC/c two-party computation) *Let $\mathcal{F} \in \{\mathcal{F}_{\mathrm{krk}}, \mathcal{F}_{\mathrm{acrs}}, \mathcal{F}_{\mathrm{sc}}\}$. Let $\mathcal{G}$ be a well-formed silent functionality. Then there is a protocol π in the $\mathcal{F}$-hybrid model such that π UC/c emulates $\mathcal{G}$ with static corruptions/deceptions.*

Proof. From Corollary 18 and Theorem 23 and the fact that $\bar{\mathcal{F}}^{*}_{\mathrm{krk}}, \bar{\mathcal{F}}^{*}_{\mathrm{acrs}}, \bar{\mathcal{F}}^{*}_{\mathrm{sc}}$ are shared restrictions of $\mathcal{F}_{\mathrm{krk}}, \mathcal{F}_{\mathrm{acrs}}, \mathcal{F}_{\mathrm{sc}}$. □

5 Conclusions and open problems

We have presented the UC/c framework. This framework enables us to model the incoercibility of general multi-party protocols. The UC/c framework comes with a strong composition theorem (universal composition). We have shown that with respect to static coercions/deceptions, arbitrary two-party protocol tasks can be realised in the framework.

Directions for future work include:

- *Good-guy/bad-guy coercions.* Our feasibility results only hold for static coercions/deceptions. We believe that feasibility results similar to those presented in Section 4 can be shown for good-guy coercions. To achieve protocols that are secure with respect to bad-guy coercions, we believe that new cryptographic techniques will have to be developed.
- *Insecure channels.* We assumed perfectly secure channels, i.e., channels where the adversary does not even notice that a message is sent. Can the results from Section 4 be generalised to a setting with weaker assumptions on the channels?
- *Multi-party protocols.* Our feasibility results are restricted to two-party protocols. To capture important cases like voting protocols we need to extend this to multi-party protocols.
- *Impossibility results.* Since incoercibility is a strong requirement, we also expect that many protocol tasks cannot be fulfilled. For example, is it possible to realise a non-trivial protocol task using only a common reference string?

Acknowledgements. We thank Yevgeniy Dodis and Daniel Wichs for extensive discussions.

References

[BMQR07] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In *E-Voting and Identity, VOTE-ID 2007*, volume 4896 of *LNCS*, pages 111–124, Berlin/Heidelberg, 2007. Springer.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Twentieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1988*, pages 1–10. ACM Press, 1988.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC '94*, pages 544–553. ACM, 1994.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`, a strongly revised full version appeared as [Can05].

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, December 2005. Full and revised version of [Can01], online available at `http://eprint.iacr.org/2000/067`.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Twentieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1988*, pages 11–19. ACM Press, 1988. Online available at `http://www.cs.mcgill.ca/~crepeau/GZIP/CCD88.ps.gz`.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography, Proceedings of TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer-Verlag, March 2007. Preprint on IACR ePrint 2006/432.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multiparty computation. In *Twenty-Eighth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1995*, pages 639–648. ACM Press, 1996. Extended version online available at `http://www.wisdom.weizmann.ac.il/~oded/PS/tr682.ps`.

[CG96] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 504, Washington, DC, USA, 1996. IEEE Computer Society. Long version available at `http://eprint.iacr.org/1996/001`.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *ESORICS*, pages 118–139, 2005.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[For91] Frederick Forsyth. *The Deceiver*. Bantam Books, 1991. Summary available at `http://tinyurl.com/ycvhuod`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications).* Cambridge University Press, May 2004. Preliminary version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[Her91] Amir Herzberg. Rumpsession, Crypto '91, 1991.

[HUMQ07] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Universally composable zero-knowledge arguments and commitments from signature cards. *Tatra Mt. Math. Pub.*, pages 93–103, 2007.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proc. 4nd ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 61–70. ACM Press, 2005.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *Advances in Cryptology, Proceedings of CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 316–337. Springer-Verlag, 2003. Online available at `http://www.nada.kth.se/~rafael/papers/denzk.ps`.

[Tru07] Trusted Computing Group. TPM main specification level 2 version 1.2, 2007. online available at `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, adopted as ISO/IEC standard 11889.

Index