Universally Composable Incoercibility*

Dominique Unruh[1] and Jörn Müller-Quade[2]

[1] Saarland University
[2] Karlsruhe Institute of Technology (KIT)

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).

Keywords: Incoercibility, universal composability, voting.

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., [16,1,11,8,9]).

There is, however, a requirement that does not fall into either category: coercion resistance (first noted by [17,2]). 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.

* Partially funded by the Cluster of Excellence "Multimodal Computing and Interaction".

Previous definitions of incoercibility are usually restricted to special domains such as voting (e.g., [2,19,13]). An exception are the models by Canetti and Gennaro [4] and by Moran and Naor [20] 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 [6]. 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 present general feasibility results for two-party protocols. 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 [14]. 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-Gennaro, Moran-Naor). Canetti and Gennaro [4] present a model for defining incoercible secure function evaluation which was subsequently refined by Moran and Naor [20]. The model by Moran and Naor is based on the so-called stand-alone model [5,15, 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.[3] 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.

Everything we said about the work by Moran and Naor also applies to the earlier work by Canetti and Gennaro [4]. 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 [20].

Externalized UC (deniability). Another approach to define properties similar to incoercibility for general protocols is the Externalized UC (EUC) framework proposed by Canetti, Dodis, Pass, and Walfish [7] (also known as Generalized UC, UC with global setup, or, proposed independently by Hofheinz, Müller-Quade, and Unruh [18], 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 [7], 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 the full version [21] we show that under certain conditions, EUC security implies UC/c security.

[3] 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.

2 The Composable Incoercibility Framework (UC/c)

2.1 Review of the UC framework

Our model is based on the Universal Composability (UC) framwork [6]. 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.[4] 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 $\mathcal{Z}$ 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 ρ that specifies the intended protocol behaviour. Then we define what it means that another protocol π (securely) emulates ρ:

Definition 1 (UC [6]). *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

[4] 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.

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 [6].

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). We put no restriction on the communication between machines, $\mathcal{A}, \mathcal{D}, \mathcal{Z}$ may all communicate with each other. Both the adversary and the deceiver may control parties. The exact mechanism of this is the following:

Corruption model. 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 stress that if a party is deceiving, the adversary cannot even observe that party's communication (unless the party communicates over an insecure channel or with a corrupted party).

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.[5]

Security definition. 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). The resulting notion is strictly stronger than UC.

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: $\pi \cup \{\mathcal{A}, \mathcal{D}_S, \mathcal{Z}\}$ and $\rho \cup \{\mathcal{A}_S, \mathcal{D}, \mathcal{Z}\}$.*

Where is the deception strategy? The existence of a deception strategy that honest parties can follow when being coerced is an essential part of any notion of incoercibility. Such a deception strategy also exists in our model: if we consider the deceiver $\tilde{\mathcal{D}}$ that simply obeys any commands (such as "vote for Bob") sent to it by the environment (we call such a deceiver a dummy-deceiver $\tilde{\mathcal{D}}_S$, see Section 2.4), then the corresponding deceiver-simulator describes how a coerced party should behave in any situation. For an example of how to derive a special purpose deception strategy from $\tilde{\mathcal{D}}_S$, see the proof of Theorem 10.

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

[5] A typical example is the key exchange functionality, which returns a random key for both parties [6, full version]. 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.

(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 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.

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 Properties of UC/c security

The proofs in this section are omitted for space reasons. They can be found in the full version [21].

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 [6] 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 3 (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.*

Lemma 4 (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 (i.e., when only considering adversary $\tilde{\mathcal{A}}$ and deceiver $\tilde{\mathcal{D}}$ in Definition 2).*

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 5 (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.*

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

Corollary 6. *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 $\mathcal{G}$.*

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 7 (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* $\text{tally} : (\mathcal{V} \cup \{\bot\})^{\mathcal{P}} \to \mathcal{T}$.

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 $\text{tally}(v_1, \ldots, v_n)$ 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 8 (Voting functionality). *The voting functionality $\mathcal{F}_{\text{vote}} = \mathcal{F}_{\text{vote}}^{\text{tally}}$ expects (at most one) message $v_i \in \mathcal{V}$ from each party $P_i \in \mathcal{P}$. When receiving* `tally` *from T, $\mathcal{F}_{\text{vote}}$ 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 $\mathcal{F}_{\text{vote}}$ computes $t := \text{tally}(v_i, i \in \mathcal{P})$ (the tally) and sends t to the adversary. Then, when $\mathcal{F}_{\text{vote}}$ 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 $\mathcal{F}_{\text{vote}}$ 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 outputs 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 9 (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 9, 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 9 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 10. *Let π be a voting scheme for the tally function* tally*. Assume that π UC/c emulates $\mathcal{F}_{\mathrm{vote}}^{\mathrm{tally}}$ 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 $v_1, \ldots, v_n$ 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 $v_1, \ldots, v_n$ 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 $\mathcal{F}_{\mathrm{vote}}$. (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 $\mathcal{F}_{\mathrm{vote}} := \mathcal{F}_{\mathrm{vote}}^{\mathrm{tally}}$, 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 $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}\}$ are indistinguishable. (We write $\mathcal{F}_{\mathrm{vote}}$ for the protocol containing $\mathcal{F}_{\mathrm{vote}}$ and the dummy parties.)

By construction,

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

(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,

$$\Pr_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B}) = \Pr[\mathrm{EXEC}_{\pi \cup \{\mathcal{A}', \mathcal{D}_S, \mathcal{Z}_{deceive}\}} = 1]. \tag{2}$$

Compare the networks $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$ and $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{obey}\}$. In the first network, $\mathcal{Z}_{deceive}$ instructs the dummy-party $\tilde{P}$ (via the deceiver $\mathcal{D}$) to send the vote v to $\mathcal{F}_{\mathrm{vote}}$. In the second network, $\mathcal{A}'_S$ instructs $\tilde{P}$ to send some other vote v^* to $\mathcal{F}_{\mathrm{vote}}$ (where we write $v^* = \perp$ 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 $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{obey}\}$ is $\mathrm{tally}(\mathcal{B}_{v^*})$, and the distribution of the tally in the network $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$ is $\mathrm{tally}(\mathcal{B}_v)$. Since $\mathcal{Z}_{obey}$ and $\mathcal{Z}_{deceive}$ output the bit b received from $\mathcal{A}'_S$, it follows that

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

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

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

for some negligible function μ. Then with (1) and (2) we get that

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

This shows that the protocol π satisfies the second condition in Definition 9. (Notice that the construction of the deception strategy $\mathfrak{d}$ is independent of $\mathcal{A}$ and $\mathcal{B}$.)

We are left to show that $\mathrm{Tally}_{deceive}(\mathcal{A}, \mathfrak{d}, \mathcal{B})$ and $\mathrm{tally}(\mathcal{B}_v)$ are indistinguishable (first condition of Definition 9).

Let t denote the message received by $\mathcal{Z}_{deceive}$ from the party T (t is the tally). In the network $\mathcal{F}_{\mathrm{vote}} \cup \{\mathcal{A}'_S, \mathcal{D}, \mathcal{Z}_{deceive}\}$, t is the output of $\mathcal{F}_{\mathrm{vote}}$. Thus

the distribution of t is 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 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 tally($\mathcal{B}_v$) are computationally indistinguishable and the first condition of Definition 9 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., [10,12,20,3]), realising UC/c security might be much easier. We therefore particularly propose UC/c as a security definition for that setting.

4 Incoercible two-party protocols

In the previous section, we have seen that UC/c secure protocols are incoercible. We have not, however, shown that such protocols exist at all. Fortunately, the protocols that were presented in [18,7] for general multi-party computation in the externalized UC (EUC) model are also secure in our UC/c model *in the two-party case* and therefore enjoy incoercibility in addition to the properties guaranteed by the EUC model. The proof that their protocols work in our setting is quite technical; we defer it to the full version [21]. We only state the final result here. The protocols from [18,7] can be based on one of the following functionalities:

The *key registeration with knowledge (KRK) functionality* $\mathcal{F}_{\text{krk}}$ 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 *augmented CRS (ACRS) functionality* $\mathcal{F}_{\text{acrs}}$ 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 *signature card functionality* $\mathcal{F}_{\text{sc}}$ 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 $\mathcal{F}_{\text{sc}}$ and receives signatures of m back. The signing key is never revealed.

Theorem 11 (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*[6] *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.*

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?
- *Not knowing who is coerced/corrupted.* In our setting, the deceiver-simulator's strategy may depend on who is corrupted/coerced. If we restrict every party's strategy to be independent of the other parties, can we still construct UC/c secure protocols?

Acknowledgements. We thank Yevgeniy Dodis and Daniel Wichs for extensive discussions. We also thank the anonymous reviewers for helpful comments.

[6] A well-formed functionality is one whose behaviour does not depend on which parties are corrupted or deceiving. We call $\mathcal{G}$ silent if it does not communicate with the adversary or deceiver.

References

1. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC'88. pp. 1–10. ACM (1988)
2. Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: STOC '94. pp. 544–553. ACM (1994)
3. Bohli, J.M., Müller-Quade, J., Röhrich, S.: Bingo voting: Secure and coercion-free voting using a trusted random number generator. In: E-Voting and Identity, VOTE-ID 2007. LNCS, vol. 4896, pp. 111–124. Springer (2007)
4. Canetti, R., Gennaro, R.: Incoercible multiparty computation. In: FOCS'96. p. 504. IEEE (1996)
5. Canetti, R.: Security and composition of multi-party cryptographic protocols. Journal of Cryptology 3(1), 143–202 (2000)
6. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS'01. pp. 136–145. IEEE (2001), full version is IACR ePrint 2000/067
7. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: TCC'07. LNCS, vol. 4392, pp. 61–85. Springer (2007)
8. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: STOC'95. pp. 639–648. ACM Press (1996)
9. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC'02. pp. 494–503. ACM (2002)
10. Chaum, D.: Secret-ballot receipts: True voter-verifiable elections. IEEE Security & Privacy 2(1), 38–47 (2004)
11. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC'88. pp. 11–19. ACM Press (1988)
12. Chaum, D., Ryan, P.Y.A., Schneider, S.A.: A practical voter-verifiable election scheme. In: ESORICS'05. pp. 118–139 (2005)
13. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security 17(4), 435–487 (2009)
14. Forsyth, F.: The Deceiver. Bantam Books (1991), summary available at `http://tinyurl.com/ycvhuod`
15. Goldreich, O.: Foundations of Cryptography – Volume 2 (Basic Applications). Cambridge University Press (2004)
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC'87. pp. 218–229 (1987)
17. Herzberg, A.: Rumpsession, Crypto '91 (1991)
18. Hofheinz, D., Unruh, D., Müller-Quade, J.: Universally composable zero-knowledge arguments and commitments from signature cards. Tatra Mt. Math. Pub. pp. 93–103 (2007)
19. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: 4nd ACM Workshop on Privacy in the Electronic Society (WPES). pp. 61–70. ACM Press (2005)
20. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: CRYPTO'06. LNCS, vol. 4117, pp. 373–392. Springer (2006)
21. Unruh, D., Müller-Quade, J.: Universally composable incoercibility. IACR ePrint 2009/520 (October 2009)