Towards Multiparty Computation Withstanding Coercion of All Parties

Ran Canetti[1](✉) and Oxana Poburinnaya[2,3]

[1] Boston University, Boston, USA
canetti@bu.edu
[2] University of Rochester, Rochester, USA
[3] Ligero, Inc., Rochester, USA
oxanapob@bu.edu

Abstract. Incoercible multi-party computation [Canetti-Gennaro'96] allows parties to engage in secure computation with the additional guarantee that the public transcript of the computation cannot be used by a coercive external entity to verify representations made by the parties regarding their inputs to and outputs from the computation. That is, any deductions regarding the truthfulness of such representations made by the parties could be made even without access to the public transcript. To date, all incoercible secure computation protocols withstand coercion of only a fraction of the parties, or else assume that all parties use an execution environment that makes some crucial parts of their local states physically inaccessible even to themselves.

We consider, for the first time, the setting where *all parties* are coerced, and the coercer expects to see *the entire history of the computation*. In this setting we construct:

- A general multi-party computation protocol that withstands coercion of all parties, as long as none of the coerced parties cooperates with the coercer, namely they all use the prescribed "faking algorithm" upon coercion. We refer to this case as *cooperative incoercibility*. The protocol uses deniable encryption and indistiguishability obfuscation, and takes 4 rounds of communication.
- A general two-party computation protocol that withstands even the "mixed" case where some of the coerced parties cooperate with the coercer and disclose their true local states. This protocol is limited to computing functions where the input of one of the parties is taken from a small (poly-size) domain. This protocol uses deniable encryption with public deniability for one of the parties; when instantiated using the deniable encryption of Canetti, Park, and Poburinnaya [Crypto'20], it takes 3 rounds of communication.

Finally, we show that protocols with certain communication pattern cannot be incoercible, even in a weaker setting where only some parties are coerced.

R. Canetti—Boston University. Member of the CPIIS. Supported by NSF Awards 1931714, 1801564, 1414119, and the DARPA DEVE program.

O. Poburinnaya—University of Rochester and Ligero, Inc. The work was done in part while in Boston University.

R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12551, pp. 410–438, 2020.
https://doi.org/10.1007/978-3-030-64378-2_15

1 Introduction

Consider a tight-knit society whose members regularly meet behind closed doors and run their society's business with complete privacy. An external entity might be able to deduce information on the nature of the interactions that take place in the society's meetings from the external behavior of the society members, but no direct information on what really takes place at the meetings can be obtained. As long as the meetings are not directly monitored by the external entity, this continues to be the case even if the external entity has coercive power over the society's members and demand that they fully disclose the contents of the meetings: All that the coercive entity can obtain is the word of the members, which may or may not be truthful.

Can we reproduce this situation online, where the society members communicate over public channels that are accessible to the external entity? That is, can the society members engage in a multiparty computation that allows them to limit the power of the external coercive entity to the power that it had when they met behind closed doors? Furthermore, can they do so even in the case where *all* members are coerced, and the coercive entity now expects to have the complete history of the interaction and the local states of all parties, including all the local randomness used? Indeed, doing so essentially results in rewriting the entire history of a system, in a way that's undetectable to anyone that did not directly witness the events at the time and location where they took place.

This is a special case of the *incoercible multiparty computation* problem, first studied in [CG96]. In a nutshell, a multiparty protocol is *incoercible* if it enables the participants to preserve the privacy of their inputs and outputs even against coercive adversaries who demand to see the entire internal state of coerced parties[1]. Towards this end, each party is equipped with a "faking procedure" that enables it to run the protocol as prescribed on the given input x, obtain an output y, and then, given arbitrary values x', y', generate a "fake internal state" (or, equivalently, fake local randomness) r' such that the public communication transcript of the party is consistent with input x', output y' and randomness r'. Moreover, we would like to guarantee more global incoercibility properties. Specifically: (a) As long as the inputs and outputs claimed by the coerced parties are consistent with the evaluated function, the entire information reported by the parties should look like an honest execution of the protocol with these inputs and outputs; this should hold regardless of whether the inputs and outputs are true or fake or partially true and partially fake. (b) If the claimed inputs and outputs are not globally consistent with the evaluated function, the coercer should not be able to deduce any information which it cannot deduce given the inputs and outputs alone - such as, e.g., the identities of parties which reported fake values.

Incoercibility might indeed appear unobtainable at first. Still, [CG96] construct an incoercible general multi-party function evaluation protocol, for the case where only a minority of the parties are coerced, and furthermore the coercions takes place at the end of the interaction. The [CG96] protocol assumes sender-deniable encryption [CDNO96, SW14]. The works of [DKR14] and [CGP15] extend these results to the case where all but one of the parties are coerced. The works of [MN06, AOZZ15] consider the case where all parties are coerced - in fact they consider an even more adversarial setting of *active coercions*, where the coercer may force parties to deviate from the protocol, to

[1] In [AOZZ15] this is referred to as *semi-honest incoercibility*.

make it harder for them to deceive the coercer. However, they assume that the parties have access to secure hardware whose internals are not available *even to the parties themselves.*

Still, whether incoercibility is at all possible in a setting where all parties are coerced, the communication is public, and the parties have full access to the transcripts of their own internal computations, has remains open. Indeed, in this case the adversary obtains an entire transcript of a computation, which can be verified step by step. Still, it should be unable to tell a fake transcript from a real one.

Our Results. We consider the case where the parties have full access to the computing devices they use, all communication is public, and all parties are eventually coerced. Still, we concentrate on the case where coercions take place only at the end of the protocol. We consider two main settings, or levels, of incoercibility:

Cooperative Incoercibility. In the cooperative incoercibility setting, it is guaranteed that all parties "cooperate" with each other, in a sense that either they all present their real randomness (and real inputs and outputs), or they all present randomness computed via their faking algorithms, along with the corresponding input and output values (which may be either fake or real). This scenario corresponds to a standard setting where a group of participants wants to protect itself against an external coercer. (We stress that we assume no additional coordination between parties: each party runs its faking algorithm locally, only based on the information available to that party. Further, the inputs and outputs claimed by the parties need not necessarily be globally consistent with each other.)

Full Incoercibility. In this setting, there are no guarantees of behavior of the parties. In particular, upon coercion, some parties may decide to present their real randomness (and real inputs and outputs), and some parties may decide to present their fake randomness (and real or fake inputs and outputs). Further, the claimed inputs and outputs could even be globally inconsistent - and still the protocol has to hide everything which is not revealed by inputs and outputs alone (e.g., the identities of the liars could still remain hidden). This definition additionally gives protection in the setting where parties have conflicting insentives and might act against each other; we will refer to the case where parties present mixed (real and fake) randomness as *off-the-record* case. Thus, full incoercibility incorporates both cooperative incoerciblity and off-the-record incoercibility.

Moreover, in full incoercibility we even allow the environment to make standard (adaptive) corruption requests, in addition to coercion requests.[2]

We show:

– A cooperatively incoercible protocol for general secure multi-party function evaluation. Our protocol works in the common reference string (CRS) model and requires 4 rounds of communication.

[2] Note that the adversary receives the party's true internal state both in case of a corruption and in case of a coercion, if that party decides to tell the truth. However, in the former case the adversary knows that the given internal state is authentic, and in the latter it doesn't.

- A fully incoercible protocol for secure two-party function evaluation, for functions with poly-size input domains. For this protocol, we build an incoercible oblivious transfer (OT) from any deniable encryption with certain properties. In particular, our construction, instantiated with deniable encryption of [CPP20], yields a 3-message protocol in the CRS model.
- For $n \geq 3$, no n-party protocols with a certain communication pattern can be secure even against coercion of 2 parties, except for trivial functions.

On the Applicability of Off-the-Record/Full incoercibility. First, with cooperative incoercibility only, the adversarial parties may be able to provide an unequivocal proof that other parties are lying; thus, this type of incoercibility doesn't protect the participants against each other. In contrast, incoercibility in the off-the-record case guarantees that the coercer will not be able to use the protocol transcript to verify claims of parties—any deduction made by the coercer will be exclusively based on "taking the word" of the coerced parties.

Furthermore, cooperatively incoercible protocols may drop their security guarantees if some parties give real coins to the coercer, and other parties give fake coins. Because of that, cooperatively incoercible protocols impose a classical prisoner's dilemma onto the participants, due to the fact that the identities of liars could be revealed by such a protocol. Indeed, upon coercion, each party has to make a decision - whether to lie or tell the truth. On one hand, for each party it is better to tell the truth - otherwise it may get caught lying, if some other party tells the truth. On the other hand, all parties jointly are better off if they all present fake randomness - no one gets caught, and their inputs remain protected. In contrast, if a protocol remains incoercible even in the off-the-record setting, once parties have already decided on which inputs and outputs to disclose, for each party it is strictly better to disclose fake randomness (even if it still reports the true inputs and outputs). Indeed, no matter how other parties act, the protocol guarantees that nothing is revealed beyond inputs and outputs.

1.1 Related Work

Prior Work on Generic Incoercible MPC. The prior work on generic incoercible MPC can be split into two parts, depending on whether it focuses on *semi-honest* or *active* coercion (in the language of [AOZZ15]). Intuitively, a coercer is semi-honest if it lets the party participate in the protocol as prescribed (by following the instructions of the protocol), but after that demands to see the entire view of that party and checks whether it matches the claimed input of that party. In contrast, an *active* coercer assumes full control over the party and in particular may instruct the party to deviate from the protocol, in order to make it harder for the party to deceive the coercer.

As already noted in [BT94] in the context of secure voting, active coercion is clearly unachievable with cryptography alone: coerced parties have no hope of lying about their inputs if the adversary watches over their shoulder during the computation. As a result, security against active coercion requires some form of physical unaccessibility assumption. Indeed, to come up with the protocol secure against active coercion, [AOZZ15] makes use of a stateful hardware token which can generate keys, distribute them to all parties, and encrypt.

In contrast, semi-honest incoercibility is well within the reach of "digital cryptography", without the need to assume inaccessible hardware: sender-deniable encryption and encryption deniable for both parties was constructed by [SW14] and [CPP20], respectively, from indistinguishability obfuscation and one-way functions, and it was shown back in 1996 how to transform any sender-deniable encryption into incoercible MPC which withstands coercion of up to half participants [CG96]. The protocols of [CGP15, DKR14], originally devised as adaptively secure protocols withstanding corruption of all parties, can withstand coercion of a single party.

Note that, although from a practical standpoint active incoercibility is stronger and more desirable than semi-honest one, from theoretical perspective semi-honest and active incoercibility are two completely different and incomparable problems. Indeed, achieving semi-honest incoercibility requires solving the problem of "inverting the computation" - i.e. finding randomness which makes some computation appear to stem from a different input (note that this problem is also interesting on its own, without its connection to incoercibility). Active incoercibility, as discussed above, inherently requires inaccessible hardware to hide parts of the computation and thus avoids the inverting problem altogether; instead, the goal there is to ensure that the active coercer cannot force parties to output something committing, while making the underlying physical assumptions as realistic as possible.

Impossibility Results. [CG96] shows that semi-honest incoercible computation is not achievable against unbounded adversaries; this impossibility holds even in the presence of private channels. To the best of our knowledge, in the computational setting no impossibility results specific to incoercible MPC were known. However, the impossibility of non-interactive (i.e. 2-message) receiver-deniable encryption [BNNO11] immediately implies that 2-round incoercible MPC is impossible, even against coercion of a single party which receives the output[3] (in particular, the 2-round protocol of [CGP15] only withstands coercion of a party which doesn't receive the output); this impossibility holds for all functions which imply a bit transmission.

On the Difference Between the Definitions of Incoercible MPC in [CG96] *and* [CGP15]. In this work we use the definition of incoercible computation from [CGP15]. We briefly explain how it differs from the one in [CG96]. The definitions of [CG96] and [CGP15] are conceptually similar but differ in case when an environment instructs a party to fake, but sets its fake input and output to be exactly the same as its real input and output. In this case the definition in [CG96] instructs the party to output its true randomness, while the definition in [CGP15] instructs the party to run the fake algorithm anyways and output the resulting fake randomness.

[3] Indeed, any incoercible protocol for a message transmission functionality can be turned into a 2-message receiver-deniable encryption, by letting the party R which receives the output be a receiver of deniable encryption, and letting the sender run the MPC protocol on behalf of all other parties. In particular, the first message (sent by the receiver) will consist of all messages sent by R in the first round of the protocol, and the second message (sent by the sender) will consist of all messages sent to R in rounds 1 and 2. Messages sent by R in round 2 of MPC protocol do not have to be sent, since S doesn't receive the output, nor does S have to deny later.

This difference may appear minor - indeed, if a party is not going to lie about its inputs nor outputs, why fake the randomness? Nevertheless, there are situations when a party may want to fake its randomness anyways. Indeed, as we discuss in the technical overview, our incoercible MPC protocol only retains its incoercibility properties as long as *all* parties disclose their fake coins to the coercer. In particular, there may be a party which has no interest in lying about its own input, but which anticipates that other participants may need to lie about theirs, and which thus decides to give out its fake randomness to make sure its true randomness doesn't compromise other parties' security.

Deniable Encryption. Perhaps the most relevant prior work for us is the interactive deniable encryption of [CPP20], which we use as a building block in all our protocols. It is an encryption scheme which withstands coercion of both the sender and the receiver even in the off-the-record setting. The protocol requires a common reference string, takes 3 rounds, and assumes subexponential indistinguishability obfuscation and one way functions.

1.2 Technical Overview

On the Definition of Incoercible Computation. We use the definition of incoercible computation from [CGP15], which can be seen as the "UC equivalent" of the definition of [CG96], with one critical difference. (See Sect. 1.1 for the discussion of the difference between the two.) Specifically, this definition models coercion as the following special form of corruption: When a party is notified that it is coerced, it first contacts the its caller to ask whether to disclose true or fake randomness, and if fake, what value (input and output) to report to the coercer. The response can be either "fake" together with an input and an output, or "tell the truth". In the former case, the coerced party runs the faking algorithm with the prescribed value; in the latter case it reveals its actual internal state.

In the ideal process, when the simulator asks to coerce a party, the ideal functionality obtains from the environment either the value v to be presented, or the "tell the truth" directive. If the response was a value v, then the functionality forwards v to the simulator. If the response was "tell the truth", then the ideal functionality provides the actual input and output values of the coerced party to the simulator. *Crucially, the simulator isn't told if this value is true or fake.* Intuitively, the fact that the simulator can simulate the protocol without learning whether the inputs were real or fake, means that in the real world the adversary doesn't learn this information either.

This definition in particular means that the protocol must maintain the best possible incoercibility even when *claimed inputs and outputs are inconsistent*. For instance, even in case of clearly inconsistent inputs and outputs, the total number of liars or their identities may be still hidden; thus the real-world protocol is required to provide the same guarantee.

We note that we still allow standard adaptive corruption requests, in addition to coercion requests.

We refer to this setting as *full incoercibility*. The case of *cooperative* incoercibility is obtained from the above definition by restricting the environment in two ways: first, it must either provide “tell the truth” to *all* parties, or else provide it to *no* participant; second, we prohibit corruption operation.

As noted in [CG96], this definition of incoercibility immediately implies semi-honest adaptive security.

Obstacles to Incoercibility: Inversion and Coordination. We start by giving some intuition for why it is hard to build incoercible protocols. For instance, consider a two-party computation protocol based on Yao garbled circuits [Yao86], where the sender sends a garbled circuit, together with labels for the inputs of the sender and the receiver; the latter is sent via oblivious transfer (OT). If both the sender and the receiver become coerced and decide to lie about their inputs and outputs, then:

- the receiver should demonstrate the adversary how it receives (potentially incorrect) output of the OT corresponding to a different receiver bit. At the same time, the receiver shouldn’t be able to obtain the *true* OT output for that bit - indeed, this would violate sender privacy.
- the sender should explain how the garbled circuit was generated, i.e. provide its generation randomness. The problem is, the sender has already “committed” to its own labels (by sending them over the public channel), and now it has to come up with different generation randomness such that those labels, initially corresponding to its true input, now represent a fake input.
- further, this generation randomness also has to be consistent with the labels of the receiver and fake input of the receiver, which the sender doesn’t know.

This example already demonstrates two difficulties with designing incoercible protocols. One is the problem of *inversion*, where some or all parties have to invert some randomized function $f(x; r)$ with respect to a different x' (like the generation of a garbled circuit)[4]. The other is a problem of *coordination*, where parties have to lie about their intermediate states *in a consistent way*, even though parties do not know fake inputs and outputs of each other.

These problem are reminiscent of the problems arising in the context of adaptive security. However, incoercibility is much stronger than adaptive security. Indeed, in the setting of adaptive security fake randomness is only created by a simulator in the proof, as part of a mental experiment, and not by parties in the real protocol. In particular, the simulator may keep secret trapdoors to help with generating fake randomness (thus simplifying the problem of inversion), and the fact that all fake randomness is generated by the same entity eliminates the problem of coordination. In contrast, in incoercible protocols, the parties themselves should be able to fake their randomness, and they must do so independently of each other (after the protocol finishes).

These issues manifest themselves even in a simpler task of a message transmission function. To date, despite having a number of clean and modular constructions of adaptively secure (or, non-committing) encryption schemes [DN00,CDMW09,BH92, HOR15,HORR16,YKT19] we have only one construction of a deniable encryption

[4] Note that in the model where not everybody is coerced, it is easy to avoid the inversion problem altogether by, e.g., secret-sharing r across all parties, thus guaranteeing that the coercer never gets to see r.

scheme (withstanding coercion of both parties), and this construction is very non-modular: it is built from the ground up using obfuscation, and both the construction and its security proof are quite heavy [CPP20].

Thus, we have two potential approaches for designing incoercible MPC. One is to build the whole protocol from scratch, perhaps using obfuscation, similar to the construction of deniable encryption; needless to say, such a construction is likely to be even more complicated. The other approach is to use deniable encryption as a primitive and explore how much incoercibility can we obtain by composing it with other primitives.

In this work we take the latter approach. We show how to combine deniable encryption with adaptive security to obtain an incoercible protocol, and how to turn certain deniable encryption schemes into incoercible OT, thus yielding incoercible 2PC with short inputs.

Our Setting. We allow parties (and adversaries) to have an access to a common reference string (CRS), which has to be generated only once and is good for unboundedly many executions. However, we require that protocols should only rely on cryptographic assumptions (as opposed to inaccessible hardware assumptions). We consider the case where coercions and corruptions happen only after the execution has finished. Our two main settings are the setting of *cooperative incoercibility*, where security is guaranteed to hold only as long as partitipants lie or tell the truth simultaneously, and the setting of *full incoercibility*, which doesn't have such a restriction.

We present semi-honestly incoercible protocols, which withstand coercion of all participants. Faking procedure of each party is local: that is, each party fakes only based on its own real and fake inputs and outputs and the information made available by the protocol. In particular, neither party knows fake inputs of other parties, nor does it know whether other parties are corrupted or coerced, and if coerced, whether they tell the truth or li.e.

Deniable Encryption. A common building block in all our protocols is a deniable encryption scheme [CDNO96], which can be thought of as an incoercible protocol for the message transmission functionality. We require deniable encryption which remains deniable even when both parties are coerced, and even in the off-the-record setting; as of September 2020, the only such protocol is given by [CPP20]. Roughly, deniable encryption give the following security guarantee:

1. the adversary cannot distinguish whether it sees
   - real randomness s of the sender, real randomness r of the receiver, and the communication transcript for plaintext m, or
   - fake randomness of the sender s' consistent with fake m, fake randomness of the receiver r' consistent with fake m, and the communication transcript for plaintext m'.
2. (off-the-record setting) the adversary cannot distinguish whether it sees
   - real randomness s of the sender, fake randomness r' of the receiver consistent with m', and the communication transcript for plaintext m,
   - fake randomness of the sender s' consistent with m, real randomness of the receiver r consistent with m', and the communication transcript for plaintext m',

- fake randomness s' of the sender consistent with m, fake randomness r' of the receiver consistent with m', and the communication transcript for plaintext m''.

This should hold for any, potentially equal, m, m', m''.

Incoercible Oblivious Transfer and 2PC with Short Inputs. Incoercible oblivious transfer has the functionality of a standard oblivious transfer - i.e. it allows the receiver to obtain exactly one value x_b (corresponding to its own input b), out of two values x_0, x_1 held by the sender. However, it additionally provides security guarantees against a coercer: that is, even if the coercer demands to see all randomness used by both parties in the protocol, parties can successfully lie about their inputs. That is, the sender can claim that it used any, possibly different inputs x'_0, x'_1 (and provide convincing randomness supporting this claim). Similarly, the receiver can claim it used a possibly different input bit b', and received a different output x' of its choice.

This primitive can be constructed from any receiver-oblivious deniable encryption (DE) with public receiver deniability. Here "public receiver deniability" means that the faking algorithm of the receiver doesn't take true receiver coins as input (thus anyone can fake on behalf of the receiver). "Receiver-oblivious" DE means that the adversary cannot tell if the receiver messages were generated honestly (following the algorithm of DE), or instead chosen at random (in this case, we say that these messages were generated obliviously); further, this indistinguishability should hold even given fake random coins of the sender. We note that deniable encryption of [CPP20] has public receiver deniability, and in the full version we show that it is also receiver-oblivious.

Theorem 1. *Any receiver-oblivious deniable encryption, which remains deniable even in the off-the-record setting and has public receiver deniability, can be converted into fully incoercible* 1*-out-of-*m *oblivious transfer, for any polynomial* m*, in a round-preserving way.*

The construction of incoercible OT is inspired by the construction of adaptively secure OT from non-committing (adaptively secure) encryption [CLOS02]. Namely, let x_0, x_1 be the inputs of the sender, and b be the input of the receiver. The parties should run in parallel two instances of DE: DE_0 and DE_1. The sender's input to each DE_i is x_i, for both $i = 0, 1$. The receiver should pick random r and generate messages of DE_b honestly (using r as randomness of the receiver in the protocol), while messages of DE_{1-b} should be generated by the receiver obliviously.

It is easy to see that the receiver can learn only x_b but not x_{1-b}, since the receiver knows r, which allows it to decrypt DE_b, but doesn't know randomness for DE_{1-b} and therefore cannot decrypt it. The sender, in turn, doesn't learn the receiver bit b, since it doesn't know which execution was generated obliviously by the receiver. Further, this OT is indeed incoercible: the sender can directly use deniability of DE to claim that different inputs x'_0, x'_1 were sent. The receiver can lie about its input b by claiming that DE_b was generated obliviously, and by presenting fake r' as randomness for DE_{1-b}. This fake r' can be generated by using the faking algorithm on DE_{1-b} and y', where y' is the desired fake output of the oblivious transfer. Note that the receiver doesn't know true coins for obliviously generated DE_{1-b}, but it can generate fake r' anyway due to the fact that receiver deniability is public.

This construction can be extended to 1-out-of-m incoercible OT in a straightforward way.

Incoercible 2PC for Short Inputs from incoercible OT. Recall that, when the number m of possible inputs of some party is polynomial, standard 1-out-of-m OT immediately implies general 2PC [GMW87]: The OT sender should input to the OT m possible values of $f(x, y)$, corresponding to m possible values of the receiver's input y, and a single sender's input x. Using incoercible 1-out-of-m OT in this protocol immediately makes the resulting 2PC protocol incoercible.

Incoercible MPC from OT? Despite the fact that standard OT implies general secure multi-party computation [GMW87], it is not clear whether *incoercible* OT implies *incoercible* MPC as well. In particular, simply plugging (even ideal) incoercible OT into the protocol of [GMW87] doesn't seem to result in an incoercible protocol, even just for two parties. The problem here is the following: recall that this protocol works by letting the parties compute additive secret shares of each wire of the circuit of $f(x_1, x_2)$. On one hand, since in the normal execution two shares add up to the value of the wire of $f(x_1, x_2)$, the same should hold in the fake case: fake secret shares should add up to the value of the wire of $f(x_1', x_2')$. However, it is not clear how, upon coercion, parties can compute these fake shares *locally*, without the knowledge of the other party's input.

Incoercible MPC. A natural starting point for building an incoercible MPC is to make parties run any secure MPC protocol, where each message is encrypted under a separate instance of deniable encryption. If in addition the parties are allowed to communicate outside of the view of the adversary - e.g. by meeting physically - and if they are comfortable sharing their fake inputs with each other, this method immediately gives incoercible MPC. Indeed, upon coercion parties can use their out-of-band channel to all agree on some transcript $\mathsf{tr}' = \mathsf{tr}(\{x_i', r_i'\})$ of an underlying MPC executed on their fake inputs. When coerced, each party can use deniability of encryption to lie (by presenting consistent randomness and keys of deniable encryption) that it sent and received messages of tr'. In addition, each party should claim that x_i', r_i' are the true input and randomness which it used to compute the messages of tr'.

However, this protocol fails when no out-of-band interaction is possible, since parties do not have means to agree on tr'. To fix this problem, we combine deniability with adaptive security. That is, we use MPC which is adaptively secure and has a special property called *corruption oblivious simulation* (defined in [BCH12] in a setting of leakage tolerance). Roughly, it means that there is a "main" simulator which simulates the transcript, and in addition each party has its own, "local" simulator which simulates the coins of that party, using that party's inputs only and the state of the "main" simulator (but not the inputs of other parties). If parties had a way to agree on the same simulation randomness r_{Sim}, then upon coercion, they could do the following: First they should run the main simulator on r_{Sim} to generate (the same) simulated transcript tr' of an underlying adaptive MPC, and then each party should use its own local simulator to locally compute fake coins consistent with this simulated transcript and its own input. Finally, as before, each party can use deniability of encryption to claim that the messages of tr' were indeed sent.

It remains to determine how the parties agree on the random coins r_{Sim} of the main simulator. A natural approach to do this is to let one of the parties (say, the first) choose r_{Sim} at random and send it, encrypted under deniable encryption, to each other party at the beginning of the protocol, for case that they need to fake later. However, this introduces another difficulty: now the adversary can demand to see r_{Sim}, and revealing it would allow the adversary to check that the transcript was simulated and thus detect a li.e. Therefore, instead of sending r_{Sim}, the first party should send randomly chosen seed s to all other parties. This seed is not used by parties in the execution of the protocol. However, upon coercion each party can use a pseudorandom generator to expand s into a string $r_{\mathsf{Sim}}||s'$, where r_{Sim}, as before, is used to produce the same simulated transcript of an adaptive MPC, and s' is what parties will claim as their fake seed (instead of a true seed s). Note that it is safe to reveal s' to the adversary, since $s'||r_{\mathsf{Sim}}$ is pseudorandom, and therefore s' cannot help the adversary to indicate in any way that tr' was simulated.

We underline that security of this protocol is only maintains in the cooperative setting. As a result, this protocol is useful in scenarios where parties "work together" and are interested in keeping all their inputs secret, rather than turn against each other trying to make sure others get caught cheating. We note however that the protocol remains secure even if *inputs* of some parties are real and inputs of some other parties are fake - as long as *randomness* of all parties is fake. Indeed, it might happen so that a certain party is not interested in lying about its input, but still wishes the whole group of people to succeed in deceiving; then this party may provide fake randomness for its real input, thus not ruining the joint attempt to deceive, while achieving its own goals[5]. Further, this protocol maintains the best possible security even in the case when the claimed inputs and outputs are clearly inconsistent.

4-*Round Protocol for Incoercible MPC.* We now describe the same protocol more formally and in particular show how to achieve 4 rounds of communication:

Theorem 2. *It is possible to build cooperatively incoercible secure function evaluation protocol from deniable encryption and adaptively secure MPC protocol with a global CRS and corruption-oblivious simulator.*

We need the following ingredients for our protocol:

– 2-round adaptively secure MPC aMPC with global CRS[6] and corruption-oblivious simulator, e.g. that of [CPV17].
– 3-round delayed-input[7] deniable encryption DE, e.g. that of [CPP20]. While that construction is not delayed-input, we observe that it is easy to turn any deniable encryption into its delayed-input version. This can be done by letting the sender send a randomly chosen key k using deniable encryption, and also send $m \oplus k$ in the clear at the last round.

[5] Note that this scenario highlights a subtle but important difference between the modelling of coercion in [CG96] and [CGP15]. Indeed, in [CG96], if the party is given a real input, it has to provide its true randomness.

[6] The CRS of the protocol is said to be global, if the simulator can simulate the execution, *given* the CRS (as opposed to generating the CRS on its own, possibly from a different distribution, or with underlying trapdoors).

[7] That is, only the third message of the sender depends on the plaintext.

Then our protocol proceeds as follows:

1. In rounds $1-3$ parties exchange the messages of the *first* round of aMPC, encrypted under point-to-point deniable encryption.
2. In rounds $2 - 4$ parties exchange the messages of the *second* round of aMPC, encrypted under point-to-point deniable encryption. It is important that deniable encryption requires its input only by the last round, since parties receive the messages of the first round of aMPC only after round 3.
3. In rounds $2 - 4$ party 1 sends to each party randomly chosen seed, encrypted under point-to-point deniable encryption. Note that each party receives the same value of seed.

After round 4, parties learn all messages of aMPC and therefore can compute the output. Note that our protocol is delayed input, since inputs are required only by round 3. Upon coercion, each party first computes fake transcript tr' of aMPC. tr' is computed by running the "main" simulator of aMPC on r_{Sim}, where r_{Sim} is obtained by expanding seed into $\mathsf{seed}'||r_{\mathsf{Sim}}$ using a prg. (Note that parties use the same r_{Sim} and therefore obtain the same tr' upon coercion). Next, each party can use its local simulator to produce fake coins consistent with tr' and fake input x'. Therefore, each party can claim that the transcript of the underlying protocol was tr', and this claim will be consistent with party's own fake input, and across different parties. Finally, each party should claim that the seed value sent by party 1 was in fact seed'.

Note that our construction crucially uses the fact that underlying adaptive MPC has *global* CRS. Indeed, this allows to put this CRS as part of the final CRS of the protocol, and lets parties simulate the transcript of underlying adaptive MPC with respect to that CRS. Had the CRS been local, parties would have to generate it during the protocol and thus eventually provide the adversary with the generation coins; yet, security of protocols with local CRS usually holds only as long as the generation randomness of this CRS remains private.

Impossibility of Incoercible MPC with Lazy Parties

Impossibility of Incoercible MPC with Lazy Parties. We show that unlike 2-party protocols, multiparty protocols with some communication structure cannot be incoercible (this holds even against coercion of only 2 parties). Concretely, let us say that a party is *lazy*, if it only sends its messages in the first and the last round of a protocol, but doesn't send anything in intermediate rounds (if any). In particular, in all 2 round protocols all parties are lazy by definition. We show that coercing a lazy party and some output-receiving party allows to learn information about inputs of other parties, therefore rendering the protocol insecure for most functions:

Theorem 3. *Assume there exists an n-party protocol withstanding 2 corruptions and 1 coercion for computing function f with a lazy party, where $n \geq 3$. Then the function f is such that for any inputs $x_1, \ldots, x_n$ it is possible, given x_1, x_n, and $f(x_1, \ldots, x_n)$, to compute $f(x, x_2, \ldots, x_n)$ for any x.*

We consider this negative result to be especially important in light of the fact that building fully incoercible protocols may require complicated obfuscation-based constructions. For instance, consider the following natural attempt to build a 3-round fully incoercible protocol. Take deniable encryption of [CPP20] which essentially lets the sender send an encryption of a plaintext together with some auxiliary information, which the receiver can decrypt using an obfuscated decryption program. This protocol features a "ping-pong" communication pattern, with a total of 3 messages sent between a sender and a receiver. One could attempt to turn it into MPC with a similar "ping-pong" communication pattern by letting $n-1$ senders $P_1, \ldots, P_{n-1}$ send its input to a single receiver P_n in a similar manner, and let the obfuscated evaluation program of the receiver decrypt the messages and evaluate the result. While this approach sounds very plausible and appealing in a sense that it potentially requires only minor modifications of the construction of deniable encryption, our impossibility result implies that such protocol cannot be incoercible.

Finally, it is interesting to note that this impossibility result is "tight" both with respect to the number of participants n, and with respect to coercion operation (as opposed to adaptive corruption). Indeed, there exists a 3-round *two-party* incoercible protocol (e.g. our OT-based protocol), and a 3-round multi-party *adaptively secure* protocol [DKR14], which features such a "ping-pong" communication pattern.[8]

To get an idea of why impossibility holds, consider standard MPC with a super-lazy party who only sends its messages in the very last round; clearly, such a protocol is insecure, since the adversary who corrupts this party together with some output-receiving party can rerun the protocol on many inputs of the lazy party and therefore infer some information about the inputs of uncorrupted parties.

Such an attack in the standard MPC case doesn't work when a lazy party sends messages in two rounds of the protocol. However, we show that in case of incoercible protocols there is a way for a lazy party to modify its last message such that the protocol now thinks that a different input is used - despite the fact that its first message still corresponds to the original input. With this technique in place we can mount the same attack as described before. This technique is based on the observation in [CPP20] that sender-deniability in any deniable encryption implies that a party can "fool" its own protocol execution into thinking that a different input is being used. We refer the reader to Sect. 5 for details.

Discussion, Open Problems, and Future Work. Our results naturally lead to the following open problems:

- *Round Complexity:* is it possible to build an incoercible protocol, withstanding coercion of all parties, for general functions in 3 rounds?
- *Full Incoercibility:* Is it possible to obtain a protocol which withstands coercion of all parties and remains incoercible even in the off-the-record setting - with any number of rounds?

[8] Note that formally speaking, the protocol of [DKR14] takes 4 rounds; however, the receiver learns the output already after round 3. The 4-th round is only required to send this output back to everyone.

The protocols in this paper follow a blueprint of composing deniable encryption with non-deniable primitives, resulting in a simple and clean protocol design. However, it could be problematic to use this approach for answering the questions listed above. The reason is the following. Since incoercible MPC implies deniable encryption, any construction of incoercible MPC:

- either has to use some construction of deniable encryption,
- or has to build deniable encryption from scratch, at least implicitly.

As we explain in more detail next, improving on our results would likely require the latter. This is a problem because the only known construction of encryption which is deniable for both parties [CPP20] is fairly complex and has lengthy proofs (the paper is more than 250 pages), and moreover, complex constructions could be inherent for deniable encrypion, because of a certain attack which can be done by the adversary (see the technical overview of [CPP20] for more details).

We now give more details about each open question separately.

Round Complexity. We show the existence of a 4-round deniable protocol, whereas 2-round incoercible protocols are ruled out by the impossibility of receiver-deniable encryption in 2 rounds [BNNO11]. This leads to a natural question of whether deniable computation can be done in 3 rounds generically.

It could be hard to achieve this by using deniable encryption as a building block. Since deniable encryption itself provably takes 3 rounds of communication, this means that only the last message in the protocol can be "protected" by deniability of encryption; yet, previous messages have to depend on the inputs as well and somehow have to be deniable. We leave it to future work to either extend this argument towards a lower bound, or to come up with a protocol which avoids this issue.

Off-the-Record Incoercibility. A natural attempt to build an off-the-record incoercible protocol is to combine deniable encryption (secure even in the off-the-record setting) with other, weaker-than-incoercible primitives (e.g. standard MPC). Unfortunately, this is unlikely to help. Indeed, a very simple argument made by [AOZZ15] shows that in any construction of off-the-record incoercible MPC with the help of secure channels, parties have to use these (perfectly deniable!) channels in an inherently non-deniable way: that is, if a party sends (receives) a message M via secure channel during the protocol, then its faking algorithm cannot instruct this party to lie about M[9]. This can be informally interpreted as follows: in any incoercible protocol which uses deniable encryption, deniable encryption can be replaced with standard encryption such that the protocol still remains incoercible[10]. This in turn indicates that such a protocol would have to be incoercible to begin with.

[9] Roughly, this is because said party doesn't know whether its peer is lying or telling the truth; it could be telling the truth, thus revealing true M, and from definition of off-the-record deniability, their joint state should look valid even in the case when the party is lying and its peer is telling the truth - as long as their inputs and outputs are consistent.

[10] We underline again that this is an informal statement - indeed, such a statement is tricky to even formalize, let al.one prove.

2 Preliminaries

2.1 Incoercible Computation

We use the definition of incoercible computation from [CGP15], which can be regarded as a re-formulation of the definition of [CG96] within the UC framework. (We note that the formulations of [MN06, AOZZ15] are similar to and consistent with the one we use, with the exception that they allow also Byzantine corruptions and incorporate modeling of ideally opaque hardware.) Specifically, we let the adversary send a special coercion message (in addition to standard corruption messages) to parties; upon receiving this message a party notifies a predetermined external entity (say, its "caller" via subroutine output) that it was coerced and expects an instruction to either "tell the truth", in which case it reveals its entire local state to the adversary, or "fake to input x and output y", in which case the party runs the faking algorithm provided as part of the protocol, on x, y and the current local state, and uses the output of the algorithm as the fake internal state reported to the adversary. We also restrict the parties to accept coercion/corruption messages from the environment only once the protocol execution ended. We refer to this setting as full incoercibility.

Cooperative Environments. If an environment is guaranteed to either instruct all coerced parties to "tell the truth", or else neither of the coerced/corrupted parties are instructed to "tell the truth" (in which case, each party is instructed to fake to some input x and output y, of the environment's choice), and in addition if standard corruptions are prohibited, then we say that it is cooperative.

Incoercible Ideal Functionalities. An ideal functionality can now guarantee incoercibility via the following mechanism: When asked by the adversary (or, simulator) to coerce a party P, the ideal functionality outputs a request to coerce P to the said external entity, in the same way as done by the protocol. If the response is "fake to input x and output y, then the pair x, y is returned to the adversary. If the response is "tell the truth" then the actual input x and output y are returned to the adversary. *Crucially, the simulator is not told whether the values received are real or fake.*

This behavior is intended to mimic the situation where the computation is done "behind closed doors" and no information about it is ever exposed, other than the inputs and outputs of the parties. In particular, such an ideal functionality does not prevent situations where the outputs of the parties are globally inconsistent with their inputs, or where a certain set of inputs of the parties are inconsistent with auxiliary information that's known outside the protocol execution. Indeed, the only goal here is to guarantee that any determination made by an external coercer (modeled by the environment) after interacting with the protocol, could have been done in the ideal model, given only the claimed inputs and outputs.

Figures 1, 2 and 3 depict incoercible variants of the standard ideal functionalities for secure message transmission, oblivious transfer, and multiparty function evaluation, respectively.

We say that π is a fully incoercible message transmission protocol if π UC-realizes $\mathcal{F}_{imt}$. If π UC-realizes $\mathcal{F}_{imt}$ only with respect to cooperative environments then π is a

Functionality $\mathcal{F}_{\text{IMT}}$

- Upon receiving input $(\texttt{Send}, sid, R, m)$ from party S, where R is an identity for the intended receiver, send $(sid, S, R, |m|)$ to the adversary. When receiving $\texttt{ok}$ from the adversary, output $(\texttt{Receive}, sid, S, m)$ to R.
- Upon receiving $(\texttt{Coerce}, sid, P)$ from the adversary, where $P \in \{S, R\}$, output $(\texttt{Coerce}, sid)$ to P. Upon receiving V from P do: If $V = (\texttt{tell-truth})$ then send m to the adversary. If $V = (\texttt{fake-to}, m')$ then send m' to the adversary.
- Upon receiving $(\texttt{Corrupt}, sid, P)$ from the adversary, where $P \in \{S, R\}$, output $(\texttt{Corrupt}, sid)$ to P, and send m to the adversary.

Fig. 1. The incoercible message transmission functionality $\mathcal{F}_{imt}$.

Functionality $\mathcal{F}_{\text{IOT}}$

- Upon receiving input $(\texttt{OT-Sender}, sid, R, (m_0, m_1))$ from party S, where R is an identity for the intended receiver, send (sid, S, R) to the adversary. When receiving $\texttt{ok}$ from the adversary, output $(\texttt{OT-Receiver}, sid, S)$ to R.
- Upon receiving input $(\texttt{OT-Receiver}, sid, b)$ from R, send sid to the adversary. When receiving $\texttt{ok}$ from the adversary, output $(\texttt{OT-Receiver}, sid, m_b)$ to R.
- Upon receiving $(\texttt{Coerce}, sid, P)$ from the adversary, where $P \in \{S, R\}$, output $(\texttt{Coerce}, sid)$ to P. Upon receiving V from P do: If $V = (\texttt{tell-truth})$ then send P's input and output to the adversary. If $V = (\texttt{fake-to}, v)$ then send v to the adversary.
- Upon receiving $(\texttt{Corrupt}, sid, P)$ from the adversary, where $P \in \{S, R\}$, output $(\texttt{Corrupt}, sid)$ to P, and send P's input and output to the adversary.

Fig. 2. The incoercible oblivious transfer functionality $\mathcal{F}_{iot}$.

Functionality $\mathcal{F}_{\text{IFE}}$

- Upon receiving input $(\texttt{Init}, sid, P_1, ..., P_n, f$ from party P_i, send $(sid, P_1, ..., P_n, f)$ to the adversary. When receiving $(\texttt{ok}, P_i)$ from the adversary, output $(\texttt{Init}, sid, P_1, ..., P_n, f)$ to P_i.
- Upon receiving input $(\texttt{Init}, sid, x_i)$ from P_i, record (P_i, x_i). Once (P_i, x_i) are recorded for all $i = 1..n$, compute $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$ and send $(\texttt{Output}, sid)$ to the adversary.
- When receiving $\texttt{output}$ from P_i, output y_i to P_i.
- Upon receiving $(\texttt{Coerce}, sid, P_i)$ from the adversary output $(\texttt{Coerce}, sid)$ to P_i. Upon receiving V from P_i do: If $V = (\texttt{tell-truth})$ then send P_i's input and output to the adversary. If $V = (\texttt{fake-to}, v)$ then send v to the adversary.
- Upon receiving $(\texttt{Corrupt}, sid, P_i)$ from the adversary output $(\texttt{Corrupt}, sid)$ to P_i, and send P_i's input and output to the adversary.

Fig. 3. The incoercible function evaluation functionality $\mathcal{F}_{ife}$.

cooperatively incoercible message transmission protocol. Incoercible oblivious transfer and function evaluation are defined analogously.

2.2 Other Preliminaries

Our protocols require deniable encryption with special properties, and adaptively secure MPC with corruption-oblivious simulator. An informal description of these primitives can be found in the introduction. We refer the reader to the full version for rigorous definitions.

3 Incoercible Oblivious Transfer

In this section we describe our construction of incoercible oblivious transfer. As noted in the introduction, such a protocol immediately implies incoercible 2PC for the case where one of the parties has polynomial input space.

3.1 Protocol Description

For simplicity, we consider 1-out-of-2 OT (the construction can be generalized to 1-out-of-n OT in a straightforward way), and we also assume that all inputs are bits. Our protocol is described on Fig. 4. It requires a special deniable encryption (DE) scheme, where deniability of the receiver is public (i.e. the faking algorithm of the receiver doesn't take receiver's true coins as input), and which satisfies receiver-obliviousness, i.e. the real transcript is indistinguishable from a transcript where receiver simply generated all its messages at random. As noted in [CPP20], their DE protocol satisfies public receiver deniability. In the full version we note that this protocol is also receiver-oblivious.

Before stating the theorem, we remind that we consider the model of semi-honest coercions of potentially all parties, and we assume that all coercions happen after the protocol finishes. We refer the reader to Sect. 2 for a description of our coercion model.

Theorem 4. *Assume* DE *is an interactive deniable encryption scheme which satisfies public receiver deniability and receiver obliviousness, and remains deniable even in the off-the-record scenario. Then the protocol on Fig. 2 is a semi-honest, fully incoercible oblivious transfer protocol.*

3.2 Proof of the Theorem

Correctness. Correctness immediately follows from correctness of deniable encryption.

Incoercibility. Consider the simulator depicted on Fig. 5, which essentially generates two transcripts of deniable encryption, each encrypting plaintext $m = 0$, and then uses faking algorithm of deniable encryption to simulate the coins. Note that the simulator generates the simulated coins in the same way (by using faking algorithm), no matter whether the party is corrupted or coerced.

Incoercible Oblivious Transfer

The CRS: $\mathsf{CRS} = \mathsf{CRS}_{\mathsf{DE}}$, where $\mathsf{CRS}_{\mathsf{DE}}$ is a CRS of deniable encryption with receiver-obliviousness, and public receiver deniability.
Inputs: inputs x_0, x_1 of the sender S; input bit b of the receiver R.
The protocol:
The sender chooses random coins s_0, s_1 for two executions of deniable encryption, where S acts as a sender. The receiver chooses randomness r for a single execution of deniable encryption where it acts as a receiver. The sender and the receiver run two instances of deniable encryption, DE_0 and DE_1, in parallel. Here:

- In each execution i, for $i = 0, 1$, the sender computes its messages by honestly running the code of deniable encryption on its input x_i, randomness s_i, and the transcript so far;
- In the execution b the receiver computes its messages by honestly running the code of deniable encryption on its randomness r and the transcript so far. In the execution $1 - b$ the receiver instead generates all its messages at random, using randomly chosen $\tilde{r}$.

At the end of both executions, the receiver sets its output in the protocol to be $\mathsf{DE.Dec}(r; \mathsf{DE}_b)$.

Faking procedure of the sender S

Inputs: fake inputs x_0', x_1' of the sender, true inputs and randomness x_0, s_0, x_1, s_1 of the sender, the protocol transcript $(\mathsf{DE}_0, \mathsf{DE}_1)$, and the CRS.
In order to fake, the sender runs the faking algorithm of deniable encryption for each execution, i.e. computes $s_i' \leftarrow \mathsf{DE.SFake}(s_i, x_i, x_i', \mathsf{DE}_i; \cdot)$ for both $i = 0, 1$. It gives s_0', s_1' to the adversary.

Faking procedure of the receiver R

Inputs: fake input b' and fake output x' of the receiver, true inputs and randomness $b, r, \tilde{r}$ of the receiver, the protocol transcript $(\mathsf{DE}_0, \mathsf{DE}_1)$, and the CRS.
In order to fake, the receiver claims that messages of the receiver in execution $1-b'$ were generated at random, and sets fake $\tilde{r}'$ to be the concatenation of these receiver messages. Next, it uses public deniability of the receiver to compute $r' \leftarrow \mathsf{DE.RFake}(x', \mathsf{DE}_{b'}; \cdot)$. It gives $r', \tilde{r}'$ to the adversary.

Fig. 4. Incoercible oblivious transfer.

We need to show that for every pattern of corruptions and coercions, and every set of real and fake inputs and outputs, the real execution is indistingusihable from a simulated one. This boils down to showing indistinguishability in the following cases:

1. If claimed inputs and outputs are consistent, we should prove indistinguishability between the case where both the sender and the receiver show their true coins, the case where both the sender and the receiver show their fake coins, the case where the sender shows true coins and the receiver shows fake coins, and the case where the sender shows fake coins and the receiver shows true coins.

Simulation of communication

Inputs given to simulate the communication: CRS.
The simulator chooses random s_0, s_1, r_0, r_1, and computes $\mathsf{DE}_i \leftarrow \mathsf{DE}(s_i, r_i, 0)$ for both $i = 0, 1$, i.e. sets DE_i to be the transcript of the protocol for deniable encryption, computed with the sender input 0, sender randomness s_i, and receiver randomness r_i. $(\mathsf{DE}_0, \mathsf{DE}_1)$ is a simulated transcript of the protocol.

Simulation of corruption and coercion of the sender S

Inputs additionally given to simulate the coercion of S: claimed inputs x'_0, x'_1 of S.
The simulator computes $s'_i \leftarrow \mathsf{DE.SFake}(s_i, 0, x'_i, \mathsf{DE}_i; \cdot)$ for both $i = 0, 1$. It gives s'_0, s'_1 to the adversary.

Simulation of corruption and coercion of the receiver R

Inputs additionally given to simulate the coercion of R: claimed input b', claimed output x'.
The simulator claims that messages of the receiver in execution $1 - b'$ were generated at random, and sets fake $\tilde{r}'$ to be the concatenation of these receiver messages. Next, it computes $r'_{b'} \leftarrow \mathsf{DE.RFake}(x', \mathsf{DE}_{b'}; \cdot)$. It gives $r'_{b'}, \tilde{r}'$ to the adversary.

Fig. 5. Simulation

2. If claimed inputs and outputs are inconsistent, we should prove indistinguishability between the case where the sender shows true coins and the receiver shows fake coins, the case where the sender shows fake coins and the receiver shows true coins, and the case where they both show fake coins.

The proof is very straighforward and uses two main steps - (a) switching between normally and obliviously generated execution of DE, using obliviousness and public receiver deniability of DE, and (b) switching randomness of DE of the sender between real and fake, using sender-deniability of DE.

Below we formally prove indistinguishability between the simulated execution $(\mathsf{Hyb}_{\mathsf{Sim}})$ and the real execution with consistent inputs x'_0, x'_1, b' and output $x'_{b'}$, where both parties tell the truth (i.e. disclose their true coins) $(\mathsf{Hyb}_{\mathsf{Real}})$. Indistinguishability between other distributions can be shown in a very similar manner.

- $\mathsf{Hyb}_{\mathsf{Sim}}$. This is the execution from Fig. 5, where both the sender and the receiver are either corrupted or coerced, and the values reported to the simulator are the following: inputs x'_0, x'_1 of the sender, input b' of the receiver, output $x' = x'_{b'}$ of the receiver. The simulator gives the adversary $(\mathsf{DE}_0, \mathsf{DE}_1, s'_0, s'_1, r'_{b'}, \tilde{r}')$.
- Hyb_1. In this hybrid the receiver generates messages in $\mathsf{DE}_{1-b'}$ obliviously (instead of generating them honestly, using $r_{1-b'}$). Indistinguishability between this and the previous hybrid follows from obliviousness of the receiver of deniable encryption. Note that it is important for the reduction that the receiver deniability is public, since the reduction needs to compute fake randomness of execution $1-b'$, $r'_{1-b'}$, for which it doesn't know the true coins $r_{1-b'}$.

- Hyb_2. In this hybrid the sender encrypts x'_0 (instead of 0) in the execution $i = 0$. It also gives the adversary its true randomness s_0 instead of fake s'_0. Indistinguishability follows from bideniability of the encryption scheme DE_0.
- $\mathsf{Hyb}_{\mathsf{Real}}$. In this hybrid the sender encrypts x'_1 (instead of 0) in the execution $i = 1$. It also gives its true randomness s_1 instead of fake s'_1. Indistinguishability follows from sender deniability of the encryption scheme DE_1.
  Note that this distribution corresponds to the real world where parties use x'_0, x'_1, b' as inputs.

4 4-Round Incoercible MPC

4.1 Description of the Protocol

In this section we describe our protocol achieving incoercibility even when all parties are coerced, but only in cooperative scenario. That is, as discussed in the introduction, the deception remain undetectable only as long as *all* parties lie about their randomness (however, then can still tell the truth about their inputs, if they choose so). We remind that in this work we only focus on coercions and corruptions which happen after the protocol execution.

Our protocol is presented on Fig. 6. As discussed more in detail in the introduction, the protocol essentially instructs parties to run the underlying adaptively secure protocol, where each message is encrypted under a separate instance of deniable encryption. In addition, party P_1 sends to everyone the same seed seed of the prg, to be used in the faking procedure. Parties' faking algorithm instructs parties to use seed to derive (the same for all parties) coins r_{Sim}, which are used to generate (the same for all parties) simulated transcript σ' of the underlying MPC. Next each party uses the local simulator of that MPC (recall that we need that MPC to have corruption-oblivious simulator) to simulate its own fake coins of the underlying MPC. Finally parties claim that they indeed exchanged messages of σ', using deniability of encryption.

Faking the Inputs vs Faking the Inputs and the Outputs. We note that it is enough for parties to be able to fake their inputs (as opposed to inputs and outputs), due to the standard transformation allowing parties to mask their output with a one time pad k: $f'((x_1, k_1), (x_2, k_2)) = f(x_1, x_2) \oplus k_1 || f(x_1, x_2) \oplus k_2$. Indeed, here faking the output can be achieved by faking inputs k_i instead. Thus, in the protocol, we only describe an input-faking mechanism.

Theorem 5. *Assume the existence of the following primitives:*

- $\mathsf{aMPC} = (\mathsf{aMPC.msg1}, \mathsf{aMPC.msg2}, \mathsf{aMPC.Eval}, \mathsf{aMPC.Sim}, \mathsf{aMPC.Sim}_i)$ *is a* 2-*round adaptively secure MPC with corruption-oblivious simulation, in a global CRS model;*
- $\mathsf{DE} = (\mathsf{DE.msg1}, \mathsf{DE.msg2}, \mathsf{DE.msg3}, \mathsf{DE.Dec}, \mathsf{DE.SFake}, \mathsf{DE.RFake})$ *is a 3-message, delayed-input deniable encryption protocol, in a CRS model;*
- prg *is a pseudorandom generator.*

Then the protocol iMPC *on Figs. 6, 7 is a* 4*-round semi-honest MPC protocol in a CRS model*[11]*, which is cooperatively incoercible.*

We note that all required primitives can be built using subexponentially-secure indistinguishability obfuscation and one-way functions [CPP20, CPV17]. Therefore we obtain the following corollary:

Corollary 1. *Assume the existence of subexponentially secure indistinguishability obfuscation and subexponentially secure one-way functions. Then in a CRS model there exists a* 4*-round semi-honest MPC, which is cooperatively incoercible.*

Notation and Indexing. Subscript i, j on the message of the protocol means that the message is sent from P_i to P_j. Subscript i, j of the randomness means that this randomness is used as sender or receiver randomness in the protocol where i is the sender and j is the receiver.

For example, $M1_{i,j}$ is the first message of aMPC, sent from P_i to P_j. Our protocol transmits this message inside deniable encryption, which in turn consists of messages $a1_{i,j}$, $a2_{j,i}$, and $a3_{i,j}$. To compute these messages, party P_i uses its sender randomness $s_{i,j,1}$, and party P_j uses its receiver randomness $r_{i,j,1}$.

4.2 Proof of the Theorem

Correctness. Correctness of the protocol immediately follows from correctness of the underlying aMPC protocol and correctness of deniable encryption DE.

Incoercibility. We define a simulator which can simulate communication and internal states of all parties, given inputs and outputs only, but without knowing whether these inputs are real or fake.

We can assume that the simulator knows the output y before the protocol starts, due to the following standard transformation, where parties additionally choose OTP keys k_i and use it to mask the output: $f'((x_1, k_1), x_2, k_2) = f(x_1, x_2) \oplus k_1 || f(x_1, x_2) \oplus k_2$. Due to this transformation, the simulator can always choose output z of parties uniformly at random, and once the first coercion occurs and the true output y becomes known, set the corresponding k_i to be $z \oplus (y||y)$. From now on we assume that the simulator knows the output y ahead of time.

Simulation. The simulator is formally described on Fig. 8. Informally, the simulator uses the underlying simulator of aMPC to simulate communication between parties, σ'. It then encrypts messages of σ' under deniable encryption. It encrypts randomly chosen seed$'$ under deniable encryption as well. This concludes the description of simulation of communication.

Upon coercion of a party, given an input x'_i (without knowing whether x_i is real or fake), the simulator computes fake random coins of aMPC by running the local simulator aMPC.Sim$_i$ on input x'_i. These are the only coins which are faked by the simulator; the simulator reveals true values of seed$'$ and all randomness of DE.

[11] Note that our CRS is global (recall that the notion of deniability or incoercibility only makes sense in the global CRS model).

4-round incoercible MPC protocol iMPC:

The CRS: $\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{DE}}, \mathsf{CRS}_{\mathsf{aMPC}})$, where $\mathsf{CRS}_{\mathsf{DE}}$ is a CRS of deniable encryption, and $\mathsf{CRS}_{\mathsf{aMPC}}$ is a CRS of adaptively secure MPC protocol.
Inputs: inputs $x_1, \ldots, x_n$ of parties $P_1, \ldots, P_n$, respectively;
Randomness: each party P_i generates the following random values:

1. $s_{i,j,1}$, $r_{i,j,1}$, $j \neq i$, which is sender and receiver randomness of DE, used to send and receive aMPC messages of round 1;
2. $s_{i,j,2}$, $r_{i,j,2}$, $j \neq i$, which is sender and receiver randomness of DE, used to send and receive aMPC messages of round 2;
3. $s_{\mathsf{aMPC},i}$, which is randomness of party P_i in the underlying aMPC protocol.

In addition, party P_1 chooses at random:

1. seed, which will be used by parties to generate coins of the simulator r_{Sim} and fake seed';
2. $s_{1,j,3}$, $j \neq 1$, which is sender randomness of DE used to send seed;

Finally, parties P_i, $i \neq 1$ generate $r_{1,i,3}$, which is receiver randomness of DE, used to receive seed.
We denote all randomness generated by each party P_i by s_i.
The protocol:

1. **Round 1:** Each party P_i sends to each other party P_j, $j \neq i$, the following:
   $a1_{i,j} = \mathsf{DE.msg1}(\mathsf{CRS}_{\mathsf{DE}}; s_{i,j,1})$.
2. **Round 2:** Each party P_i sends to each other party P_j, $j \neq i$, the following:
   - $a2_{i,j} = \mathsf{DE.msg2}(\mathsf{CRS}_{\mathsf{DE}}; r_{i,j,1}, a1_{j,i})$.
   - $b1_{i,j} = \mathsf{DE.msg1}(\mathsf{CRS}_{\mathsf{DE}}; s_{i,j,2})$.

   In addition, P_1 sends to each other party P_j, $j \neq 1$, the following:
   - $c1_{1,j} = \mathsf{DE.msg1}(\mathsf{CRS}_{\mathsf{DE}}; s_{1,j,3})$.
3. **Round 3:** Each party P_i for each $j \neq i$ computes $\{M1_{i,1}, \ldots, M1_{i,n}\} \leftarrow \mathsf{aMPC.msg1}(\mathsf{CRS}_{\mathsf{aMPC}}; x_i; s_{\mathsf{aMPC},i})$, and sends the following:
   - $a3_{i,j} = \mathsf{DE.msg3}(\mathsf{CRS}_{\mathsf{DE}}; s_{i,j,1}, M1_{i,j}, a1_{i,j}, a2_{j,i})$.
   - $b2_{i,j} = \mathsf{DE.msg2}(\mathsf{CRS}_{\mathsf{DE}}; r_{i,j,2}, b1_{j,i})$.

   In addition, each party P_i except P_1 sends to P_1 the following:
   - $c2_{i,1} = \mathsf{DE.msg2}(\mathsf{CRS}_{\mathsf{DE}}; r_{1,i,3}, c1_{1,i})$.
4. **Round 4:** Each party P_i, for each $j \neq i$, computes $M1_{j,i} \leftarrow \mathsf{DE.Dec}(\mathsf{CRS}_{\mathsf{DE}}; r_{j,i,1}, a1_{j,i}, a2_{i,j}, a3_{j,i})$. Next for each $j \neq i$ it computes $\{M2_{i,1}, \ldots, M2_{i,n}\} \leftarrow \mathsf{aMPC.msg2}(\mathsf{CRS}_{\mathsf{aMPC}}; x_i, M1_{1,i}, \ldots, M1_{n,i}; s_{\mathsf{aMPC},i})$, and sends the following:
   - $b3_{i,j} = \mathsf{DE.msg3}(\mathsf{CRS}_{\mathsf{DE}}; s_{i,j,2}, M2_{i,j}, b1_{i,j}, b2_{j,i})$.

   In addition, P_1 sends to each other party P_j, $j \neq 1$, the following:
   - $c3_{1,j} = \mathsf{DE.msg3}(\mathsf{CRS}_{\mathsf{DE}}; s_{1,j,3}, \mathsf{seed}, c1_{1,j}, c2_{j,1})$.
5. **Evaluation:** Each party P_i, for each $j \neq i$, computes $M2_{j,i} \leftarrow \mathsf{DE.Dec}(\mathsf{CRS}_{\mathsf{DE}}; r_{j,i,2}, b1_{j,i}, b2_{i,j}, b3_{j,i})$. Next for each $j \neq i$ it computes $y \leftarrow \mathsf{aMPC.Eval}(\mathsf{CRS}_{\mathsf{aMPC}}; x_i, M1_{1,i}, \ldots, M1_{n,i}, M2_{1,i}, \ldots, M2_{n,i}; s_{\mathsf{aMPC},i})$. It sets y to be its output in the protocol.

By $\pi = \mathsf{iMPC}(\mathsf{CRS}, (x_1, s_1), \ldots, (x_n, s_n)) = (\{a1_{i,j}, a2_{i,j}, a3_{i,j}\}_{i \neq j}, \{b1_{i,j}, b2_{i,j}, b3_{i,j}\}_{i \neq j}, \{c1_{1,j}, c2_{j,1}, c3_{1,j}\}_{j \neq 1})$ we denote the transcript of our protocol.
By $\sigma = \mathsf{aMPC}(\mathsf{CRS}_{\mathsf{aMPC}}, (x_1, s_{\mathsf{aMPC},1}), \ldots, (x_n, s_{\mathsf{aMPC},n})) = (\{M1_{i,j}, M2_{i,j}\}_{i \neq j})$ we denote the transcript of underlying adaptive MPC protocol aMPC.

Fig. 6. 4-round incoercible MPC protocol.

Faking procedure of party P_i, $i = 1, \ldots, n$

Inputs: P_i's true input x_i, fake input x'_i, true output y, real random coins s_i, and the protocol transcript π.

1. **learning the seed:** P_1 knows the seed seed (which it generated). For $i \neq 1$, P_i computes $\mathsf{seed} \leftarrow \mathsf{DE.Dec}(\mathsf{CRS_{DE}}; r_{1,i,3}, c1_{1,i}, c2_{i,1}, c3_{1,i})$.
2. **expanding the seed:** P_i computes $\mathsf{prg}(\mathsf{seed})$ and parses the result as $r_{\mathsf{Sim}}||\mathsf{seed}'$, where $|\mathsf{seed}| = |\mathsf{seed}'|$.
3. **computing fake transcript:** P_i computes the fake transcript and state $(\sigma', \mathsf{state}) \leftarrow \mathsf{aMPC.Sim}(\mathsf{CRS_{aMPC}}, y, r_{\mathsf{Sim}})$ of the underlying 2-round MPC protocol. Let $\sigma' = (\{M1'_{i,j}, M2'_{i,j}\}_{i \neq j})$.
4. **computing fake coins of the underlying MPC:** P_i computes the fake coins $s'_{\mathsf{aMPC},i} \leftarrow \mathsf{aMPC.Sim}_i(\mathsf{CRS_{aMPC}}, \mathsf{state}, x'_i, y)$ of the underlying MPC protocol, using the local simulator.
5. **computing fake coins of deniable encryption:** P_i computes the fake coins for each instance of deniable encryption as follows:
   $s'_{i,j,1} \leftarrow \mathsf{DE.SFake}(\mathsf{CRS_{DE}}, s_{i,j,1}, M1_{i,j}, M1'_{i,j}, a1_{i,j}, a2_{j,i}, a3_{i,j}; \cdot)$, to claim that it sent $M1'_{i,j}$ instead of $M1_{i,j}$;
   $s'_{i,j,2} \leftarrow \mathsf{DE.SFake}(\mathsf{CRS_{DE}}, s_{i,j,2}, M2_{i,j}, M2'_{i,j}, b1_{i,j}, b2_{j,i}, b3_{i,j}; \cdot)$, to claim that it sent $M2'_{i,j}$ instead of $M2_{i,j}$;
   $r'_{i,j,1} \leftarrow \mathsf{DE.RFake}(\mathsf{CRS_{DE}}, r_{i,j,1}, M1_{j,i}, M1'_{j,i}, a1_{j,i}, a2_{i,j}, a3_{j,i}; \cdot)$, to claim that it received $M1'_{j,i}$ instead of $M1_{j,i}$;
   $r'_{i,j,2} \leftarrow \mathsf{DE.RFake}(\mathsf{CRS_{DE}}, r_{i,j,2}, M2_{j,i}, M2'_{j,i}, b1_{j,i}, b2_{i,j}, b3_{j,i}; \cdot)$, to claim that it received $M2'_{j,i}$ instead of $M2_{j,i}$.
   Further, if $i = 1$, then for each $j \neq 1$ the party computes:
   $s'_{1,j,3} \leftarrow \mathsf{DE.SFake}(\mathsf{CRS_{DE}}, s_{1,j,3}, \mathsf{seed}, \mathsf{seed}', c1_{1,j}, c2_{j,1}, c3_{1,j}; \cdot)$, to claim that it sent seed' instead of seed.
   If $i \neq 1$, then P_i computes
   $r'_{1,i,3} \leftarrow \mathsf{DE.RFake}(\mathsf{CRS_{DE}}, r_{1,i,3}, \mathsf{seed}, \mathsf{seed}', c1_{1,i}, c2_{i,1}, c3_{1,i}; \cdot)$, to claim that it received seed' instead of seed.

The output of the faking procedure: Finally, P_i gives the adversary its fake internal state s'_i, where:

- If $i \neq 1$, $s'_i = \{s'_{i,j,1}\}_{j \neq i}, \{r'_{i,j,1}\}_{j \neq i}, \{s'_{i,j,2}\}_{j \neq i}, \{r'_{i,j,2}\}_{j \neq i}, \{r'_{1,i,3}\}, s'_{\mathsf{aMPC},i}$.
- If $i = 1$, $s'_i = \{s'_{i,j,1}\}_{j \neq i}, \{r'_{i,j,1}\}_{j \neq i}, \{s'_{i,j,2}\}_{j \neq i}, \{r'_{i,j,2}\}_{j \neq i}, \{s'_{1,j,3}\}_{j \neq 1}$, $s'_{\mathsf{aMPC},i}, \mathsf{seed}'$.

(Note that all other information which P_i should know in the honest execution, e.g. seed' or $M1'_{i,j}$, can be derived by the adversary using random coins s'_i, input x'_i, the transcript π, and the CRS.)

Fig. 7. Faking procedure of party P_i, $i = 1, \ldots, n$

Let $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$ be some inputs to the protocol, and let y be some output. Consider the following distributions:

- $\mathsf{Hyb}_{\mathsf{Real}}$: this is the distribution corresponding to the real execution of the protocol with inputs $x'_1, \ldots, x'_n$, where parties disclose their *true* inputs and randomness.

Simulation of communication

Inputs given to simulate the communication: CRS; output of the protocol y

1. **computing simulated transcript:** the simulator chooses r_{Sim} at random and computes the simulated transcript and state $(\sigma', \mathsf{state}) \leftarrow \mathsf{aMPC.Sim}(\mathsf{CRS}_{\mathsf{aMPC}}, y, r_{\mathsf{Sim}})$ of the underlying MPC protocol. Let $\sigma' = (\{M1'_{i,j}, M2'_{i,j}\}_{i \neq j})$.
2. **computing messages of π:** the simulator chooses seed' at random. It also chooses $\{s_{i,j,1}\}_{j \neq i}, \{r_{i,j,1}\}_{j \neq i}, \{s_{i,j,2}\}_{j \neq i}, \{r_{i,j,2}\}_{j \neq i}, \{r_{1,j,3}\}_{j \neq 1}, \{s_{1,j,3}\}_{j \neq 1}$ uniformly at random, and uses these randomness to compute messages of deniable encryption, $(\{a1_{i,j}, a2_{i,j}, a3_{i,j}\}_{i \neq j}, \{b1_{i,j}, b2_{i,j}, b3_{i,j}\}_{i \neq j}, \{c1_{1,j}, c2_{j,1}, c3_{1,j}\}_{j \neq 1})$, encrypting $M1'_{i,j}$, $M2'_{i,j}$, seed', respectively.
3. **the output of the simulator:** The simulator outputs the simulated communication $\pi' = (\{a1_{i,j}, a2_{i,j}, a3_{i,j}\}_{i \neq j}, \{b1_{i,j}, b2_{i,j}, b3_{i,j}\}_{i \neq j}, \{c1_{1,j}, c2_{j,1}, c3_{1,j}\}_{j \neq 1})$.

Simulation of coercion of P_i

Inputs additionally given to simulate the coercion of the party P_i: P_i's input x'_i (without the information whether this input is real or fake)

1. **computing fake coins of the underlying MPC:** the simulator computes the fake coins $s'_{\mathsf{aMPC},i} \leftarrow \mathsf{aMPC.Sim}_i(\mathsf{CRS}_{\mathsf{aMPC}}, \mathsf{state}, x'_i, y; \cdot)$ of the underlying MPC protocol, using the local simulator.
2. **The output of the simulator:** The simulator gives the adversary simulated internal state s'_i of P_i, where:
   - If $i \neq 1$, $s'_i = \{s_{i,j,1}\}_{j \neq i}, \{r_{i,j,1}\}_{j \neq i}, \{s_{i,j,2}\}_{j \neq i}, \{r_{i,j,2}\}_{j \neq i}, \{r_{1,i,3}\}$, $s'_{\mathsf{aMPC},i}$.
   - If $i = 1$, $s'_i = \{s_{i,j,1}\}_{j \neq i}, \{r_{i,j,1}\}_{j \neq i}, \{s_{i,j,2}\}_{j \neq i}, \{r_{i,j,2}\}_{j \neq i}, \{s_{1,j,3}\}_{j \neq 1}$, $s'_{\mathsf{aMPC},i}$, seed'.

   (Note that all other information which P_i should know in the honest execution, e.g. seed' or $M1'_{i,j}$, can be derived by the adversary using random coins s'_i, input x'_i, the transcript π', and the CRS.)

Fig. 8. Simulation

- $\mathsf{Hyb}_{\mathsf{Fake}}$: this is the distribution corresponding to the real execution of the protocol with inputs $x_1, \ldots, x_n$, where parties disclose *fake* inputs $x'_1, \ldots, x'_n$, output y, and fake randomness.
- $\mathsf{Hyb}_{\mathsf{Sim}}$: this is the distribution corresponding to the simulation from Fig. 8, where the simulator is given output y and claimed inputs $x'_1, \ldots, x'_n$.

We need to show the following:

1. If $x'_1, \ldots, x'_n$ and y are consistent (i.e. $f(x'_1, \ldots, x'_n) = y$), then we need to show that $\mathsf{Hyb}_{\mathsf{Sim}} \approx \mathsf{Hyb}_{\mathsf{Real}}$ and $\mathsf{Hyb}_{\mathsf{Sim}} \approx \mathsf{Hyb}_{\mathsf{Fake}}$.
2. If $x'_1, \ldots, x'_n$ and y are not consistent, then we need to show that $\mathsf{Hyb}_{\mathsf{Sim}} \approx \mathsf{Hyb}_{\mathsf{Fake}}$.

We show this below. First, we show indistinguishability between $\mathsf{Hyb}_{\mathsf{Sim}} \approx \mathsf{Hyb}_{\mathsf{Fake}}$, for any values $x_1, \ldots, x_n$, $x'_1, \ldots, x'_n$, and y:

- $\mathsf{Hyb}_{\mathsf{Fake}}$. We start with the distribution corresponding to the real-world execution of the protocol, where parties fake their random coins upon coercion. In other words, the adversary sees CRS, π, and x'_i, s'_i for each i, generated as in Figs. 6, 7. In particular, the truly sent transcript σ of the underlying MPC is a transcript on inputs x_i; however, parties claim that they instead sent (simulated) transcript σ', which appears consistent with fake inputs x'_i.
- Hyb_1. In this hybrid P_1 sends seed' instead of seed inside $\{c1_{1,j}, c2_{j,1}, c3_{1,j}\}_{j\neq 1}$, and parties (both senders and receivers) give the adversary true randomness for this deniable encryption (instead of faking it to seed'). Indistinguishability between this and the previous distribution holds by $n-1$ invocations of bideniability of encryption for plaintexts seed and seed'.
- Hyb_2. In this hybrid we switch $r_{\mathsf{Sim}}||\mathsf{seed}'$ from $\mathsf{prg}(\mathsf{seed})$ to uniformly random. Indistinguishability holds by security of a prg. Note that seed is not used anywhere else in the distribution, thus the reduction is possible.
- $\mathsf{Hyb}_{\mathsf{Sim}}$. In this hybrid we set $\{a1_{i,j}, a2_{i,j}, a3_{i,j}\}_{i\neq j}$ to encrypt 1-round messages of simulated σ' (consistent with fake x'_i), instead of encrypting 1-round messages of real transcript σ (consistent with x_i). Also, all parties give true randomness $\{s_{i,j,1}\}_{j\neq i}, \{r_{i,j,1}\}_{j\neq i}$, instead of giving fake randomness consistent with σ'.
  Similarly, we change $\{b1_{i,j}, b2_{i,j}, b3_{i,j}\}_{i\neq j}$ to encrypt 1-round messages of simulated σ' (consistent with fake x'_i), instead of encrypting 1-round messages of real transcript σ (consistent with x_i). Also, all parties give true randomness $\{s_{i,j,2}\}_{j\neq i}, \{r_{i,j,2}\}_{j\neq i}$, instead of giving fake randomness consistent with σ'.
  Indistinguishability between this and the previous distribution holds by $2n(n-1)$ invocations of bideniability of encryption, where plaintexts are messages of σ and σ'.
  Note that this is the simulated distribution.

Further, for the case when $f(x'_1, \ldots, x'_n) = y$, in one last step we show that $\mathsf{Hyb}_{\mathsf{Sim}} \approx \mathsf{Hyb}_{\mathsf{Real}}$:

- $\mathsf{Hyb}_{\mathsf{Real}}$. Compared to $\mathsf{Hyb}_{\mathsf{Sim}}$, we switch the messages of aMPC, encrypted inside deniable encryption, from simulated σ' to real σ, which is the true transcript of aMPC on inputs x'_i. In addition, parties reveal their true randomness $s_{\mathsf{aMPC},i}$ instead of computing simulated $s'_{\mathsf{aMPC},i}$ consistent with x'_i using the local simulator $\mathsf{aMPC.Sim}_i$.
  Indistinguishability between this and the simulation follows from adaptive security of aMPC. Note that indeed r_{Sim}, randomness of the simulator, is not used anywhere else in the distribution.
  This distribution corresponds to the real execution of the protocol on inputs x'_i, where parties disclose their true randomness upon being coerced.

This concludes the security proof.

5 Incoercible MPC with Lazy Parties is Impossible

In this section we describe our impossibility result for incoercible MPC protocols with a certain communication pattern. We consider the synchronous model of communication, where parties send their messages in rounds. We call a party *lazy*, if it sends its messages only in the first and in the last round of the protocol, but not in any other round[12]. We show that a protocol for 3 or more parties cannot be incoercible, as long as there is at least one lazy party Z, and there is another party (different from Z) which receives the output.

In particular, this impossibility rules out protocols with the following communication structure, which is a natural extention of a "ping-pong" communication of 3-message 2PC to a multiparty setting: assume just one party receives the output; we call this party the receiver, and call all other parties the senders. Then the communication proceeds as follows:

- In round 1 the senders send out their messages to everybody;
- In round 2 the receiver sends its messages to the senders;
- In round 3 the senders send out their messages to everybody[13].

Our impossibility is based on the fact that in an incoercible protocol with lazy party Z it is possible to do a variation of a residual function attack, similar to impossibility of standard (non-incoercible) non-interactive MPC. Concretely, we show that lazy party Z can always pick an input x' different from its real input x and generate a different last message of the protocol corresponding to new input x', such that the resulting transcript will be a valid transcript for this new input x', as if Z used x' even in the first message (despite the fact that in reality its first message was generated using x). As a result, the adversary may coerce (or even corrupt) Z together with some output-receiving party and evaluate the function on any possible input of Z, thus compromising security of other parties.

Theorem 6. *Let $n \geq 3$, and assume there exists an n-party protocol for evaluating function $f(x_1, \ldots, x_n)$, such that P_1 is lazy and P_n receives the output. Further, assume it is secure against up to one coercion and up to two corruptions. Then the function f is such that for any inputs $x_1, \ldots, x_n$ it is possible, given x_1, x_n, and $f(x_1, \ldots, x_n)$, to compute $f(x, x_2, \ldots, x_n)$ in polynomial time for any x of the same length as x_1.*

Note that, while the theorem statement also holds for the case of 2 parties, it doesn't imply any impossibility since for any 2-input function f it is always possible to compute $f(\cdot, x_2)$ given x_1, x_2, and thus the theorem doesn't impose any restrictions on functions f which can be computed incoercibly using 2-party protocols.

Proof of Theorem 6. Without loss of generality we assume that the lazy party is P_1, and party which receives the output is P_n. Further, we assume that P_1 is the first to send its messages in round 1, and the last to send its messages in round N.

Let us denote the randomness of P_1 by r_1, the concatenated randomness of all other parties by $R = r_2|| \ldots ||r_n$, the input of P_1 by x_1, the concatenated input of all other parties by $X = x_2|| \ldots ||x_n$. In addition, let X^0 denote some fixed set of inputs such

[12] In particular, when the protocol requires only 2 rounds, each party is lazy by definition.

[13] Note that in standard, non-deniable MPC the last message doesn't need to be sent to parties who don't receive the output. However, in deniable MPC parties who don't get the output may still need the last message in order to fake.

that $|X| = |X^0|$, e.g. all-zero inputs $0^{|X|}$. Let NMF_i denote the next message function of the protocol for party 1 in round i. Let $\mathsf{Eval}(x_n; transcript; r_n)$ denote the output evaluation function of party P_n which takes as input its randomness r_n, input x_n, and all communication in the protocol. Let $\alpha = \mathsf{NMF}_1(x_1; r_1)$ denote the concatenated messages sent by P_1 to all other parties in round 1, $, =, (\alpha; X; R)$ denote the concatenated messages sent by parties $P_2, \ldots, P_n$ in all rounds, $,_{N-1}$ denote , except for messages of the last round, and $\beta = \mathsf{NMF}_N(x_1, ,_{N-1}; r_1)$ denote the concatenated messages sent by P_1 to all other parties in round N. Finally, let $\mathsf{Fake}_1(r_1, x_1, x_1', , ; \rho)$ denote the faking algorithm of party P_1, which takes as input its true coins and input r_1, x_1, desired fake input x_1', and ,, all communication sent to P_1. Fake_1 could be deterministic or randomized; without loss of generality we assume that it is randomized using its own random coins ρ.

Consider the following algorithm NewMessage (Fig. 9) which for any x_1' allows P_1 to generate a different β' such that $(\alpha, , , \beta')$ is a valid transcript resulting in the output $f(x_1', X)$. The intuition behind this procedure is as follows: First, P_1 computes a transcript which starts with the same α but continues with a different $\tilde{,}$ (computed under freshly chosen randomness of other parties and fixed inputs X^0). Next, it runs its faking algorithm to generate fake coins r_1' which make this transcript look consistent with x_1' (in particular, this makes r_1', x_1' look like valid coins and input for α, even though α was generated under x_1). Finally, it uses fake r_1' to generate its last message β' using the original communication , and new input x_1'. In the following Lemma 1 we claim that β', together with the original communication $(\alpha, ,)$, forms a valid transcript for inputs x_1, X which will be evaluated correctly by the output-receiving party:

Algorithm NewMessage

$\mathsf{NewMessage}(x_1, r_1, x_1', \alpha, \mathsf{com}; \rho)$
Inputs: input x_1 and randomness r_1 of P_1 in the MPC protocol; new desired input x_1'; communication of P_1 in round 1 α, communication of all other parties com; local random coins $\rho = \rho_1 || \rho_2$.
Constants: arbitrary fixed input X^0 of length $|X|$, e.g. all-zero input $X^0 = 0^{|X|}$.

- Compute $\widetilde{\mathsf{com}} = \mathsf{com}(\alpha; X^0; \rho_1)$.
- Compute $r_1' \leftarrow \mathsf{Fake}_1(r_1, x_1, x_1', \widetilde{\mathsf{com}}; \rho_2)$.
- Output $\beta' = \mathsf{NMF}_N(x_1', \mathsf{com}_{N-1}; r_1')$.

Fig. 9. Algorithm NewMessage to generate the last message consistent with a different x_1'.

Lemma 1. *Let $\alpha, , , \beta'$ be generated as described above, and let the protocol be secure against the coercion of P_1. Then for any f, x_1, X, x_1', with overwhelming probability over the choice of r_1, R, ρ it holds that* $\mathsf{Eval}(x_n; \alpha, , , \beta'; r_n) = f(x_1', X)$.

We defer the proof of Lemma 1 to the full version.

Now we finish the proof of the Theorem 6. We claim that the adversary who corrupts P_1 and P_n in the real world can compute $f(x, x_2, \ldots, x_n)$ for any input x (of the same length as x_1), where $x_1, \ldots, x_n$ are inputs of the parties in the protocol. Indeed, the adversary can do so in two steps: first it corrupts P_1 to learn r_1 and x_1 and runs $\beta' \leftarrow \mathsf{NewMessage}(x_1, r_1, x, \alpha, , ; \rho)$ for any desired input x and random ρ (as before, $\alpha, ,$ is the communication of P_1 in round 1 and of all other parties). Next it corrupts P_n to learn r_n and computes $\mathsf{Eval}(x_n; \alpha, , , \beta'; r_n)$, which is with overwhelming probability equal to $f(x, x_2, \ldots, x_n)$, as shown in the Lemma 1. Note that in the ideal world the adversary who only corrupts P_1 and P_n and learns x_1, x_n, and $f(x_1, \ldots, x_n)$ cannot compute residual function $f(\cdot, x_2, \ldots, x_n)$ (except for very special functions f), and therefore the adversary in the real world has strictly more power. This finishes the proof of the Theorem 6.

Finally, we note that a similar proof can be made in case when the adversary coerces P_1 and P_n, instead of corrupting them.

References

[AOZZ15] Alwen, J., Ostrovsky, R., Zhou, H.-S., Zikas, V.: Incoercible multi-party computation and universally composable receipt-free voting. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 763–780. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_37

[BCH12] Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 266–284. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_15

[BH92] Beaver, D., Haber, S.: Cryptographic protocols provably secure against dynamic adversaries. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 307–323. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-47555-9_26

[BNNO11] Bendlin, R., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: Lower and upper bounds for deniable public-key encryption. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 125–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_7

[BT94] Benaloh, J.C., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23–25 May 1994, Montréal, Québec, Canada, pp. 544–553 (1994)

[CDMW09] Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Improved non-committing encryption with applications to adaptively secure protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 287–302. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_17

[CDNO96] Canetti, R., Dwork, C., Naor, M., Ostrovsky, R.: Deniable encryption. IACR Cryptol. ePrint Archive **1996**, 2 (1996)

[CG96] Canetti, R., Gennaro, R.: Incoercible multiparty computation (extended abstract). In: 37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14–16 October 1996, pp. 504–513 (1996)

[CGP15] Canetti, R., Goldwasser, S., Poburinnaya, O.: Adaptively secure two-party computation from indistinguishability obfuscation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 557–585. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_22

[CLOS02] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19–21 2002, Montréal, Québec, Canada, pp. 494–503 (2002)

[CPP20] Canetti, R., Park, S., Poburinnaya, O.: Fully deniable interactive encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 807–835. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_27

[CPV17] Canetti, R., Poburinnaya, O., Venkitasubramaniam, M.: Better two-round adaptive multi-party computation. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 396–427. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_14

[DKR14] Dachman-Soled, D., Katz, J., Rao, V.: Adaptively secure, universally composable, multi-party computation in constant rounds. IACR Cryptol. ePrint Archive **2014**, 858 (2014)

[DN00] Damgård, I., Nielsen, J.B.: Improved non-committing encryption schemes based on a general complexity assumption. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 432–450. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_27

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, USA, pp. 218–229 (1987)

[HOR15] Hemenway, B., Ostrovsky, R., Rosen, A.: Non-committing encryption from Φ-hiding. In: Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, 23–25 March 2015, Proceedings, Part I, pp. 591–608 (2015)

[HORR16] Hemenway, B., Ostrovsky, R., Richelson, S., Rosen, A.: Adaptive security with quasi-optimal rate. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 525–541. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_22

[MN06] Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, 20–24 August 2006, Proceedings, pp. 373–392 (2006)

[SW14] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03 2014, pp. 475–484 (2014)

[Yao86] Yao, A.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), pp. 162–167 (1986)

[YKT19] Yoshida, Y., Kitagawa, F., Tanaka, K.: Non-committing encryption with quasi-optimal ciphertext-rate based on the DDH problem. IACR Cryptol. ePrint Arch. **2019**, 1151 (2019)