Collusion-Preserving Computation without a Mediator

Michele Ciampi*[1], Yun Lu†[1] and Vassilis Zikas‡[2]

[1]The University of Edinburgh
[2]The University of Edinburgh and IOHK

Abstract

Collusion-free (CF) and collusion-preserving (CP) protocols offer alternatives to standard multi-party computation (MPC) in settings where subliminal communication is undesirable, e.g., in decentralizing mediators in mediated games. However, all existing solutions make too strong and uninstantiable assumptions on the setups, such as physical presence of the parties, access to physical envelopes and opaque ballot boxes, or extreme isolation, where the only means of communication is a star-topology network among the parties with a special resource, the mediator, at its center—and the mediator needs to be aware of the function to be computed. The above state of affairs remained a limitation of such protocols, which was even reinforced by impossibility results. Thus, for years, it has been unclear if and how the above setup assumptions could be relaxed towards more realistic application scenarios.

In this work we provide the first solution to collusion preserving computation which uses weaker and more common assumptions than the above, i.e., an authenticated broadcast functionality and access to honestly generated trusted hardware tokens. We prove that our protocol is collusion-preserving secure (in short, *CP secure*) as long as no parties abort. In the case of an aborting adversary our protocol loses CP security, but still achieves standard security—against monolithic adversaries—and additionally identifies a corrupted party.

Leveraging the above identifiability property, we augment our protocol with a collateral and compensation mechanism which ensures that it is not profitable to abort, thereby obtaining CP security against incentive driven adversaries. To define (and prove) this latter result, we combine the Rational Protocol Design (RPD) methodology by Garay *et al.* [FOCS 2013] with the CP framework of Alwen *et al.* [CRYPTO 2012] to derive a definition of security in the presence of incentive-driven local adversaries which can be of independent interest.

Similar to existing protocols in the CP/CF literature, our protocols preserve, as a fallback, the traditional security properties—i.e., security against monolithic adversaries—even when the setup (i.e., the hardware tokens) is compromised or corrupted.

*mciampi@ed.ac.uk
†Y.Lu-59@sms.ed.ac.uk
‡vassilis.zikas@ed.ac.uk

Contents

1 Introduction

Subliminal communication channels in protocols allow parties to embed extra information into protocol messages, without being detected. The existence of subliminal channels is problematic in several applications of secure computation. In large-scale distributed systems, for instance, subliminal channels could allow two parties to coordinate their actions (i.e., collude) even if they may not have been aware of each other in advance. Such collusions have severe consequences in game theoretic applications, where stability, e.g., Nash equilibrium, is defined in terms of isolated strategies. For example, in the prototypical application of distributed cryptography, namely, playing poker in a distributed manner [GM82], one cannot use standard game-theoretic reasoning together with MPC, since the the latter introduces collusions, thereby potentially changing the rules of the game—think of playing poker against colluding opponents.

In the quest to combine mechanism design and cryptography—in particular, to cryptographically emulate mediators in mediated games—a number of works [LMPs04, LMs05, LMs05, AKL$^+$09, AsV08] put forth the notion of collusion-freeness. Intuitively, a multi-party protocol is collusion free, if any profile of non-colluding strategies for the parties (i.e., any vector of strategies) can be emulated by corresponding non-colluding ideal adversaries. However, collusion freeness is arguably impossible when parties are connected by pairwise communication channels—this is straighforward to see since such channels can directly be used for coordination. This led to the proposal of other communication models that enable collusion-freeness. The two most typical models are assuming players have access to a semi-trusted "ballot box" and can communicate publicly via (physical) envelopes [LMPs04, LMs05, LMs05], or assuming that parties are connected to a semi-trusted central node (via standard communication channels) called the *mediator* [AKL$^+$09, AsV08] in a star network topology.

To motivate the assumption that parties do not have a direct line of communication with each other, we can consider games where parties may not know each other. An example of such a scenario is online poker, where competitors are complete strangers. Another scenario is when players are seated in the same room (e.g., in a poker competition), and private communication is disallowed.

Despite providing novel constructions of collusion-free protocols in their respective models, the above security definitions are unsuitable for standard network settings such as the internet, where parties might communicate via external channels—which are not necessarily used in the protocol. In fact, as argued by Alwen et al. [AKMZ12], unlike what one might expect, a collusion-free protocol (secure according to the standard, standalone definition of [AKL$^+$09, AsV08]) does not necessarily preserve the amount of a priori and/or external correlation between corrupt parties. That is, there are collusion-free protocols which allow correlations (e.g., introduced by limited external communication) between parties that increases substantially by simply executing the protocol (cf. [AKMZ12]). Hence, one cannot hope to use any such protocol to replace, for example the poker dealer in a poker game by a decentralized protocol among the parties.

Motivated by the above, the work of [AKMZ12] made a step forward in modeling security in such a setting by introducing the notion of collusion-preserving computation (CP). Intuitively, this notion can be seen as a universally composable extension of collusion freeness, designed to explicitly address the above issue. In a nutshell, this notion requires that the protocol does not allow for *additional* subliminal communication than what parties can communicate (a bounded amount of information) via means external to the protocol (e.g., from the output of another protocol that is running concurrently or via an external channel). Importantly, CP supports composition and it is modeled with respect to arbitrary communication resources.

On the downside, Alwen et al. [AKMZ12] proved that CP is impossible when parties can communicate over a broadcast channel. Instead, following the approach of [AKL$^+$09] and [AsV08], [AKMZ12] assumed a mediator and proved that the following fallback guarantee is feasible: Assuming a global ACRS functionality [CDPW07], when parties are arranged in a star topology with the mediator in the center, there exists a protocol which (1) CP emulates any given functionality if the mediator is honest, and (2) remains GUC secure [CDPW07]—i.e., secure with abort according to the monolithic-adversary definition—even if the mediator gets corrupted. We note in passing that [AKMZ12] proves it is necessary for the mediator to be aware of the function to be computed.

An additional limitation of the results from [AKMZ12] is that they only compute the target functionality with abort, even if there is a majority of honest parties. In fact, it is straightforward to prove that this limitation is inherent to the mediated model when the mediator might get corrupted (cf. Appendix B). We note that the protocol (compiler) from [AKMZ12] takes explicit steps to ensure that an abort does not allow parties to correlate their strategies in the honest mediated setting, by making sure that abort is observable only in a fixed final round which is deterministically defined at the beginning of the protocol. However, this means that (1) their solution cannot be used to compute reactive functionalities, since a party can signal by means of aborting in an intermediate output (we refer to Appendix C for a discussion of a concrete collusion strategy which this allows), and (2) having a deterministic upper bound on the number of rounds means that if the goal is to get unanimous decision on whether or not an abort occurred even in the fallback setting where the mediator is compromised, the protocol needs a linear number of rounds; this is true because a generic compiler for such a functionality would imply deterministic broadcast which needs linearly many rounds [DS83]. In fact, in [AKMZ12], each round is emulated by a round-robin sequential interaction of each party with the mediator, yielding an additional linear multiplicative blowup in the round complexity.

In this work we circumvent several of the limitations of [AKMZ12] and extend its applicability towards a framework which allows for collusion-preserving protocols over public networks. To our knowledge, this is the first work proposing a solution that breaks the deadlock of collusion-free/preserving computation, which was believed to only apply to the mediated model or models requiring physical presence of parties in the same room.

Concretely, we show that one can replace the mediator by the arguably weaker (cf. Appendix E) assumption of an authenticated broadcast channel and honestly generated hardware tokens. As discussed below, even capturing such hardware tokens in a collusion preserving framework is a non-trivial goal, which can be of independent interest and a more realistic underlying framework for the design of CP protocols. Moreover, we combine ideas from the blockchain literature to disincentivize aborts, thereby allowing the CP emulation of reactive functionalities: We present a penalization scheme against deviating behavior, which requires no additional properties from the blockchain, other than the usual security. Our work combines and extends the CP framework with the Rational Protocol Design (RPD) methodology proposed by Garay et al. [GKM$^+$13] to capture incentive-driven attackers in a composable manner. We believe that the resulting model, which we term RPD-CP, can also be of independent interest.

As fallback in case the hardware tokens are corrupted, our proposed protocol protects the inputs of honest parties and allows identifiable (unanimous) abort while guaranteeing termination (cf. [KMTZ13]). This is the analogue—in the token-hybrid setting with a broadcast channel—of the fallback property of [AKMZ12, AsV08, AKL$^+$09] which preserves privacy against a corrupted mediator at the center of a a star-network. In fact, our fallback is stronger than what the mediated-model permits [AKMZ12]; indeed, as we show, the star-network topology makes it impossible to obtain identifiable (unanimous) abort against a corrupted mediator (cf. Appendix B).

1.1 Overview of our contributions

The goals of our work are to (1) replace the mediator-centered star-topology network as an assumption/resource for collusion-preserving computation, by weaker resources which are closer to modern communication networks, (2) get stronger fallback security properties and (3) use the penalization paradigm in combination with realistic resources—such as hardware tokens and access to a blockchain—to ensure that aborts do not occur and thereby ensure CP computation of even reactive functionalities. These goals bring the theory of CP closer to capturing an execution of a decentralized-dealer poker game. For reference, a comparison of our results to [LMs05] and [AKL$^+$09, AKMZ12] can be found in Table 1.

As a first step towards our goals, we show how to construct a collusion-preserving protocol $\Pi^{\mathtt{HT}}$ allowing n parties to CP emulate any given CP-well formed functionality—intuitively these are CP versions of well-formed functionalities [CLOS02] which, when everyone gets corrupted, give up on collusion preservation and allow arbitrary communication of the corresponding adversaries (see Definition 10). Our protocol uses as resources (1) honestly generated stateful trusted hardware tokens (HTs) and (2) an authenticated broadcast channel. Our protocol, which can CP-emulate any functionality as long as no abort occurs, improves upon the protocol of [LMs05], which is collusion-free but not preserving and does not emulate games with private actions (i.e. actions that are not publicly observable). We note that our protocol, analogous to that of [LMs05], remains (G)UC secure (but not collusion preserving) in case of abort. Intuitively, the reason is that as the protocol is over broadcast, failure of our non-abort assumption triggers a global abort. We remark that our protocol only requires two (broadcast) communication rounds. Furthermore, unlike the mediator from [AKMZ12], tokens do not need to know in advance the computation they will be used for, and only need to be initialized with correlated randomness independent of the protocol, discussed in the overview of our techniques.

The protocol $\Pi^{\mathtt{HT}}$ offers no guarantees when the tokens are compromised. This is arguably a strong assumption since in reality these tokens are locally held by the parties, and the adversary may attempt to break their security. For this reason we present a protocol compiler $\Pi^{\mathtt{HT\text{-}FBS}}$ which on input a (standard (G)UC secure) protocol with *unanimous* or *identifiable abort*, outputs a protocol with the same CP and fallback security guarantees as $\Pi^{\mathtt{HT}}$, and, additionally, preserves (G)UC security properties (i.e., security with *unanimous* or *identifiable abort*, respectively) of the compiled protocol, even when the hardware tokens are compromised/corrupted.[1] This improves upon the fallback security of [AKMZ12] where, due to model idiosyncrasies, these properties are impossible to achieve when the mediator is corrupted, even with honest majority (cf. Appendix B). We note that $\Pi^{\mathtt{HT\text{-}FBS}}$, even with the extra fallback guarantee, has round complexity that is still lower than the mediated-model solution of [AKMZ12].

As an additional contribution, we combine the RPD framework with CP and define a new CP-security notion we term *collusion preserving attack payoff* (in short, *CPAP*), which intuitively corresponds to security against any combination of incentive-driven local adversaries. Within our new model, termed *RPD-CP*, we identify a natural class of utilities under which non-aborting

[1]Recall that security with *unanimous abort* guarantees that either all or none of the honest parties receive the output while *identifiable abort* ensures any party causing an abort can be identified (and excluded from future executions).

	Channel, assumption	**Id-abort (fallback)**	**U-abort (fallback)**	**Private actions**	**Abort**
Lepinski et al., STOC 2005	Broadcast, physical envelopes	✓	✓	✗	$\mathsf{poly}(\lambda)$ bits
Alwen et al., Crypto 2009 **Alwen et al., Crypto 2012**	Star topology, honest mediator	✗	✗	✓	$\Omega(\log \lambda)$ bits for reactive functionalities
This work	Broadcast, hardware tokens	✓	✓	✓	Disincentivized

Table 1: Comparison with existent approaches. Id-abort: identifiable abort, U-abort: unanimous abort. Our protocol emulates CP functionalities, including those with private actions, when no abort occurs. We disincentivize aborts via concrete penalization schemes. In addition, we achieve fallback security maintaining identifiable abort (and unanimous abort). That is, even when the hardware token are compromised and the parties are allowed to abort we still get standard GUC security.

strategies are strictly dominant in $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$. That is, these protocols are collusion-preserving according to CPAP against adversaries bounded by this utility. We believe that RPD-CP can be used to derive formal composable versions of security statements with fair-compensation [BK14, ADMM14, KB16, KB14, KZZ16] which we think is an interesting future direction.

Finally we show a penalization scheme which uses the blockchain to induce a utility in the above class. Combining this with our previous result, we can ensure that any adversary who maximizes its revenue (or at least avoids losing coins) will not abort, making our protocol CP secure against such adversaries.

1.2 Overview of our techniques

Collusion preserving protocol via HT and non-aborting adversary. Let $\mathcal{P}$ be a set of parties who wish to compute a function f in a collusion-preserving way. We assume that each party $p_i \in \mathcal{P}$ has access to a hardware token (HT) $\mathtt{HT}_i$. All HT's contain as secret information PRF keys k_0, k_1 and master secret key msk (a secret key for a strong signature scheme). The public interface of the hardware tokens is represented by the master public key mpk for msk. We refer to the party $p_n \in \mathcal{P}$ as the *leader*. Moreover, each execution of the protocol is uniquely identified by a session id $\mathsf{sid} \in \mathbb{N}$. Roughly, our protocol works as follows: Each party $p_i \in \mathcal{P} - \{p_n\}$ provides his input x_i to $\mathtt{HT}_i$. $\mathtt{HT}_i$ uses $R_0 \leftarrow \mathsf{PRF}(k_0, \mathsf{sid})$ as randomness to generate an encryption key sk. In addition, $\mathtt{HT}_i$ uses $R_1 \leftarrow \mathsf{PRF}(k, \mathsf{sid}||i)$[2] to generate a pair of session signing-verification $(\mathsf{sigk}_i, \mathsf{vk}_i)$ keys for a strong signature scheme and certifies them by signing $\mathsf{vk}_i||\mathsf{sid}||i$ with the master secret key msk thus obtaining cert_i.[3] We refer to the encryption key sk as the *session encryption key* and to $(\mathsf{sigk}_i, \mathsf{vk}_i)$ as the *session signing-verification keys*[4]. Note that the session encryption key sk is common to all the hardware tokens (since the all the tokens share the same PRF keys).[5]

[2]As an abuse of notation we refer to the identity of a party p_i with i.

[3]We use $||$ as the concatenation operator.

[4]Unless otherwise specified, a signing-verification key always refers to a *session* signing-verification key.

[5]The intuitive reason behind the use of session-keys are related to the fact that this keys will be leaked the ideal world adversary to help him in the simulation (leaking the master secret key would completely compromise the token). We provide a more detailed discussion on this later in this section.

After the session keys have been generated, the hardware token $\mathtt{HT}_i$ encrypts his input x_i, signs this encrypted value together with f using sigk_i, and sends the encryption, the signature, and the certificate cert_i over the broadcast channel.

The leader p_n collects all the encrypted values and signatures, and gives them to his hardware token $\mathtt{HT}_n$ along with his input x_n, the function f, and sid. His hardware token $\mathtt{HT}_n$ first checks if all the certificates are valid with respect to the session id sid (i.e., the verification keys are signed together with sid using the master secret key msk). Then it checks if all the values that he has received are property signed and whether the inputs are consistent with the function f.

If all the checks are successful, then $\mathtt{HT}_n$ generates, as described earlier, the session encryption key sk and decrypts all ciphertexts using sk (we recall that the PRF key k_0 is shared among all the hardware tokens). Then $\mathtt{HT}_n$ uses these decrypted values to evaluate f together with x_n . That is, $\mathtt{HT}_n$ computes $y_1, \ldots, y_n = f(x_1, \ldots, x_n)$ and for $i = 1, \ldots, n-1$ encrypts y_i using sk and signs the (concatenation of the) encrypted values together with f using sigk_n.The encryptions and the signature uses randomness from evaluating $\mathsf{PRF}(k_1, \mathsf{sid}||n)$.

Finally, the leader p_n propagates the output of $\mathtt{HT}_n$ on the broadcast channel. Each party p_i, upon receiving a message, forwards it to $\mathtt{HT}_i$, which verifies the certificate, the signature, and the consistency between f and sid. If the checks are successful then $\mathtt{HT}_i$ uses sk to decrypt and output y_i.

Using hardware tokens allows us to achieve the following: 1) generate fresh randomness and session keys for each new session id sid, that are hidden from the parties themselves, and 2) certify that each sid is used only once. In particular, to restrict any sid to one-time use, a hardware token stops replying when an id is used more than once. To prevent an adversary from *rewinding* the hardware token, we require the hardware token to be stateful. This is necessary since an adversary using the same sid (thus the same randomness) to evaluate a function in different executions can inspect the resulting encrypted/signed messages from different inputs. Then, he can send a subliminal message by picking an input where, for example, the resulting encrypted message has its first two bits equal to the first two bits of his input—breaking collusion-preservation. This adversarial strategy was indeed observed in [LMs05], limiting the games they consider to those with publicly observable actions. We prove that $\Pi^{\mathtt{HT}}$ is collusion-preserving for non-aborting adversaries, which intuitively comes from the fact that $\Pi^{\mathtt{HT}}$ is deterministic given the tokens (and thus fixing the PRF and msk keys) and the sid we use. We note that though this may appear contradictory (i.e., to obtain a secure protocol you need entropy [GM84]), our protocol achieves security and collusion-preservation as the tokens generate fresh (pseudo)randomness for each sid (which is used only once). $\Pi^{\mathtt{HT}}$ also enjoys identifiable abort, an unachievable property in the mediated model. More interestingly, any external party observing the execution of the protocol without participating it can identify malicious behavior, by verifying signatures of messages on the channel. Following [KZZ16] we refer to this property as *publicly identifiable abort.*

Tokens in collusion-preserving computation. One may wonder why, the hardware tokens cannot directly use the master secret key to authenticate protocol messages. The reason lies in the locality restrictions that the CP model places on the ideal world adversary (the simulator). Specifically, CP requires the existence of a local simulator $\mathcal{S}_i$ for each adversarial party p_i, and mandates that simulators cannot communicate with each other except if the environment explicitly allows them to. Thus, setups (in our case, tokens) which naturally introduce correlations between parties are tricky to define and use, especially when the protocol is executed over a broadcast channel.

To understand the issue, one needs to observe that in a protocol over a broadcast channel all parties expect to see exactly the same messages from this channel. Indeed, one of the novelties of our work is to show how in the real-world, i.e., in the protocol execution, the correlations embedded in the tokens can be leveraged to ensure that every (honest-protocol) broadcast message is predictable by any token. However, this correlation in the views inherently requires the simulators' views to be correlated in some way, in order to generate the same exact protocol messages. For instance, if $\mathcal{S}_1$ (the ideal world adversary with 1 as their id) reports to the environment that he broadcasted message m in round ρ, then each $\mathcal{S}_j$ should also report to the environment that they heard this message. However, for this we need to allow the simulators to correlate their response.

A naive first approach would be to allow them to interact over some underlying communication network. However, this defeats the purpose of collusion preservation, as it explicitly introduces a venue of arbitrary correlations/collusion. A second approach would be to also offer the simulators access to correlated tokens. But this leads to a new technical issue: In order to simulate the token-hybrid protocol, our simulator needs to have extra control of the hardware tokens (e.g., be able to program it). In fact, such asymmetry between the capabilities of the adversary and the simulator is proven necessary in various related settings (e.g., programmable random oracle).

We tackle the above issue by considering the hardware token as a (global) setup functionality and embedding a trapdoor inside. To ensure that only the simulators in the ideal world can use the trapdoor, we employ a technical trick inspired by [CJS14] for the global random oracle. At a very high level, the trapdoor allows the simulators to produce the same signed messages, making their views on protocol messages consistent. Specifically, it gives access to a seed that can be used to generate correlated randomness for the simulators and it gives access to n signing-verification session keys with respective certificates that can be used only in a specific session id.

In more detail, we introduce a token global-functionality that allows functionalities registered to it to send a special command $(\mathsf{Trapdoor}, \mathsf{sid})$. The registered functionalities can then relay the trapdoor information it receives to its simulators, allowing them to complete their simulations. This information remains useless for other protocols, preserving composability with other CP protocols.

More concretely, consider an extension $\mathcal{F}^\star$ of $\mathcal{F}$, which behaves exactly as $\mathcal{F}$ but accepts an additional command $\mathsf{GetTrapdoor}$ from the ideal world adversary. In the simulation each simulator can send to $\mathcal{F}^\star$ the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. Upon receiving this command, $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to the token functionality if and only if sid is equal its session id. The token functionality, upon receiving the command $(\mathsf{Trapdoor}, \mathsf{sid})$ from $\mathcal{F}^\star$, sends to $\mathcal{F}^\star$ the trapdoor information (i.e., the seed, signing-verification session keys and their certificates). When $\mathcal{F}^\star$ receives a reply from the token functionality, it is forwarded to the simulator. Using this mechanism, all simulators obtain the same signing-verification session keys and PRG seed required for local simulation. Note that we do not leak the master secret key to the simulators as this would compromise all the sessions that are using the tokens as a global functionality and affect composability. In fact, this is the reason why, as discussed previously, for each session we create new session (e.g., signature) keys that are valid only within that specific session.

Collusion preservation with fallback security. Despite being simple and optimal in terms of round complexity, the protocol above suffers from a big limitation. That is, if the hardware tokens are corrupted (e.g., the secret keys are leaked by the token manufacturer) then not only the CP-property is lost, but we cannot even guarantee to protect the honest parties' inputs.

To rectify this issue we propose a protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ that protects the input of the honest parties (in a standard GUC-security sense) even in the case where: 1) the adversary knows all the secret keys of the hardware tokens (including those held by honest parties) and 2) the malicious parties can arbitrarily modify or replace their own hardware tokens. This protocol achieves a similar fallback security as the original collusion preserving protocol in the mediated model: When the hardware tokens are not compromised—and aborts are either excluded or deterred by means of incentives (see below)—then the protocol is collusion-preserving; and in any case (i.e., even when tokens are compromised) the protocol remains (G)UC secure—i.e., any profile of adversaries can be simulated by a monolithic simulator. We refer to a protocol that has such security guarantees as a *fallback secure* protocol. Our fallback protocol guarantees also identifiable abort and unanimous abort for functionalities that guarantee termination. Interestingly, these properties are impossible to achieve in the mediated model (see App. B for the formal proof).

To obtain such a protocol we use as a main building block an MPC protocol that is secure against a malicious adversary and which enables identifiable (unanimous) abort. Each hardware token computes protocol messages on behalf of its owner. In addition, the randomness used is jointly decided by the owner of the token and by the token itself. More precisely, the randomness used to run the MPC is the XOR of the randomness produced by the hardware token, and a random string given at run-time by the token's owner. Intuitively, this means even if an honest party's token leaks its secret keys, it will still use an honestly-generated random string in the protocol. Thus, the party's input is protected by the security of the underlying MPC protocol. If all hardware tokens are secure then the randomness of any party in the MPC protocol becomes unknown and untamperable to everyone. Thus, only messages produced by the tokens are considered valid, and no malicious party can send subliminal messages without being detected. We provide more details in the technical part of the paper.

How to deal with aborting adversaries. We note that the above CP-protocols do not prevent an adversary from sending invalid messages over the broadcast channel. Clearly, over broadcast, an adversary can always send, for example, messages encrypted using some secret key that is known only to the corrupted parties. Such attacks seem unavoidable if no assumptions are made on the topology of the network (like it is done in the mediated model [AsV08, AKL+09, AKMZ12]) and on the honesty of the nodes of this network. In this work we circumvent the above issue by considering an incentive-driven (rational) adversary. That is, we define a security notion called CPAP that captures the fact that some adversarial actions—in our case aborts—are not "for free" and instead incurs a negative payoff. As it has been done in the RPD framework proposed in [GKM+13], CPAP considers an ideal functionality $\mathcal{F}$ that a set of parties want to compute, and a relaxed functionality $\langle\mathcal{F}\rangle$ that allows the adversary to break some security properties of $\mathcal{F}$ (e.g., correctness). In our case we consider a CP-well-formed functionality $\mathcal{F}$ that can be computed in a collusion-preserving manner. That is, informally, the adversarial parties are isolated and can communicate only via $\mathcal{F}$. The relaxed version $\langle\mathcal{F}\rangle$ allows an adversarial party to collude with other adversarial parties or to abort. Then we define a function v mapping the joint view of the relaxed functionality $\langle\mathcal{F}\rangle$ and the environment $\mathcal{Z}$ to a real-valued *payoff*. For our CP protocols, we show that for an appropriate choice of payoffs—that is, sufficient penalization for aborting—the adversary will never trigger the event that allows him to subliminally communicate with other adversarial parties, as this will also trigger the event of abort.

From "ideal" to "real" payoffs. Finally, we propose two penalization mechanisms that make the payoffs concrete, by financially penalizing (e.g., using a blockchain) parties that have been caught misbehaving. Our first mechanism requires all parties to deposit a collateral before beginning a protocol, and if even one party aborts, all parties lose their collateral. This scheme has the advantage of being simple, fast and requiring minimal interaction with the blockchain; however, it is not incentive compatible, as honest parties must suffer the same penalties as adversarial parties. The second mechanism trades simplicity with the ability to penalize only cheating parties, and requires protocol messages to be posted to and stored by a functionality that can be implemented by, for example, a smart-contract. In a nutshell, this protocol works as follows. Let Π be a CP-protocol secure against non-aborting adversaries with publicly identifiable abort.

1. In the setup phase the parties create a smart contract on the blockchain corresponding to Π and deposit a fixed amount of coins.
2. The parties run Π by posting their messages on the blockchain. We assume that the blockchain can be used only to run the protocol Π. Roughly, we replace the broadcast channel with a blockchain that is dedicated to only to check the messages Π.
3. If all messages are correct (and the protocol ends successfully), everyone reclaims their deposit. Conversely, if a party p_i misbehaves, he loses his deposits, which are split among other parties.

2 Organization of the paper

In Section 3 we propose a more extensive literature review. Section 4: Preliminaries. Section 5: Our protocol $\Pi^{\mathtt{HT}}$ for CP functionalities assuming honest tokens and non-aborting parties. Section 6: Our protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ with fallback GUC security against compromised tokens. Section 7: Our new framework RPD-CP for defining CPAP—security of CP protocols against rational attackers. Section 8: Prove that our protocols $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ are CPAP secure. Section 9: Penalization schemes to make concrete the utilities defined in Section 7. In the Appendix full proofs (App. A), and further discussions. In Appendix B we prove the impossibility results in the mediated model which motivated our study.

3 Related literature

In this section we extend the comparison of our work with existing results. The work of Lepinski et al. [LMs05] achieves collusion-freeness for parties that communicate with a broadcast channel, assuming access to a physical primitive called "envelopes". They motivate the use of envelopes by proving that with only broadcast channels, collusion-freeness is not possible. The idea is that corrupted parties can share the same random tape before the start of protocol execution. Since all messages are sent through broadcast, all corrupted parties will have the same view and thus emulate a monolithic adversary. In our work, parties also communicate via a broadcast channel, but we circumvent this impossibility since the random tape of a corrupted party is not decided by the party himself, but by a hardware token (whereas Lepinski et al. generate randomness through coin-tossing and hide the result in envelopes). Using stateful hardware tokens, we can also circumvent another impossibility result of Lepinski et al., that collusion-free protocols with private actions/inputs are not possible even with envelopes. The impossibility comes from the corrupted parties being able to see (different) protocol messages that different private actions generate, and choosing his private action such that the resulting messages convey subliminal information about the private action itself. However, stateful hardware tokens can be programmed to prevent users from changing his input. That is, it is not possible for a party to see messages generated by different inputs. Thus, our protocol remains collusion-preserving even with private actions. As described earlier, in the mediated model [AsV08, AKL$^+$09] achieve collusion-freeness and [AKMZ12] achieves collusion-preservation.

The question of playing poker over the Internet has recently attracted considerable attention, fuelled by the new capabilities introduced by smart-contract-enabled (cryptocurrency) blockchains, such as Ethereum. In a nutshell, this technology made it possible to ensure that parties cannot avoid paying their bid amount when they lose without the need of a trusted escrow—by having them commit their bids on the blockchain, in a smart contract that releases them to anyone that presents evidence of winning. Furthermore, the same technology enables a mechanism that punishes cheating—or early aborting—by making parties commit collaterals that they can only claim if evidence is presented that they completed their protocol [BK14, ADMM14, KB16, KB14, KZZ16]. This gave rise to a number of proposals for decentralized poker protocols [KMB15, BKM17]. However, all these works use standard multi-party computation, thus even players who do not know each other can collude via protocol messages.

4 Preliminaries

We denote the security parameter by λ and use "$||$" as concatenation operator (i.e., if a and b are two strings then by $a||b$ we denote the concatenation of a and b). For a finite set Q, $x \overset{\$}{\leftarrow} Q$ denotes a sampling of x from Q with uniform distribution. We use the abbreviation PPT that stands for probabilistic polynomial time. We use $\mathsf{poly}(\cdot)$ to indicate a generic polynomial function. When it is necessary to refer to the randomness r used by and algorithm A we use the following notation: $A(\cdot; r)$. We assume familiarity with the notions of strong signature, secret key encryption and pseudorandom function and refer to App. H for the formal description of these notions. We say a function ν is *negligible* if for every positive integer c there is an integer N_c such that for all $x > N_c$, $|\nu(x)| < 1/x^c$. We say $a \overset{\text{negl}}{\leq} b$ for real values a, b to mean that $a \leq b + \nu(\lambda)$ for negligible function ν. We denote with $[n]$ the set $\{1, \ldots, n\}$, with $\mathbb{F}$ an arbitrary (but fixed) finite field and with $\mathbb{N}$ the set of non-negative integers.

4.1 Hardware tokens and setup assumptions.

Our protocols assume that each party has access to a stateful hardware token which might perform fresh encryptions and signatures with respect to some hidden keys. We model a hardware token as an ideal GUC functionality. In this paper we also consider the case where hardware tokens are corrupted. That is, the adversary can reprogram all malicious parties' tokens, and can learn the secret information of every (both malicious and honest) party's token. This corruption model captures the real world scenario where an adversarial party can break the security of his own token, but could only obtain the secret key of tokens that he has no physical access to. For more details on hardware tokens (HT) we refer to App. D.

4.2 Secure function evaluation.

We consider a protocol Π^{MPC} that securely (GUC-)realises any efficient functionality $\mathcal{F}$. We additionally assume that Π^{MPC} can be run over a broadcast channel.

Following [BL18] we consider MPC protocols where at each round ℓ, each party P_i broadcasts a message $\mathtt{msg}_i^{\ell}$ to all the other parties simultaneously.

Definition 1 (MPC protocol)**.** *Let n be a positive integer, m a polynomial in the security parameter, and $\mathcal{F}$ an n party-functionality. An m-round MPC protocol Π^{MPC} for $\mathcal{F}$ can be described as a tuple of deterministic polynomial-time algorithms $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$.*

Next message: $\mathtt{msg}_i^{\ell} \leftarrow \mathsf{Next}_i(1^{\lambda}, x_i, \rho_i^{\ell}, \overline{\mathtt{msg}}^{<\ell})$ is the message broadcasted by party $p_i \in \mathcal{P}$ in round $\ell \in [m]$, on input $x_i \in \{0,1\}^{\lambda}$, on random tape $\rho_i^{\ell} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, after receiving the messages $\overline{\mathtt{msg}}^{<\ell} = \{\mathtt{msg}_j^{\ell'}\}_{j\in[n],\ell'<\ell}$, where $\mathtt{msg}_j^{\ell'}$ is the message broadcasted by party p_j on round $\ell' \in [\ell-1]$. When $\overline{\mathtt{msg}}^{<m+1} = \{\mathtt{msg}_j^{\ell'}\}_{j\in[n],\ell'<m+1}$ then $y_i \leftarrow \mathsf{Next}_i(1^{\lambda}, x_i, \rho_i^{m+1}, \overline{\mathtt{msg}}^{<m+1})$ where y_i denotes the output of the party p_i.

In this paper we make use of a protocol Π^{MPC} that GUC-realizes a functionality $\mathcal{F}$ where the resource $\mathcal{R}$ is a broadcast channel. More formally, we require that $\forall \mathcal{A}\ \exists \mathsf{Sim}\ \forall \mathcal{Z}$ such that $\textsc{exec}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\mathsf{MPC}},\mathcal{A},\mathcal{Z}} \approx \textsc{exec}^{\bar{\mathcal{G}},\mathcal{F}}_{\mathsf{Sim},\mathcal{Z}}$. We refer the reader to App. I for the formal definition of GUC security.

4.3 Security with identifiable (unanimous) abort.

In this work we consider the notion of secure multi-party computation with identifiable abort, also referred to as Identifiable MPC (ID-MPC). ID-MPC allows the computation to fail (abort), but ensures that when this happens every party is informed about it, and they also agree on the index i of some corrupted party $p_i \in \mathcal{P}$ [IOZ14].

More concretely, for an arbitrary functionality $\mathcal{F}$, we define $\mathcal{F}^{\mathsf{ID}}$ to be the corresponding functionality with identifiable abort, which behaves as $\mathcal{F}$ with the following modification: upon receiving from a simulator a special command $(\mathtt{ABORT}, p_i)$ $\mathcal{F}^{\mathsf{ID}}$ sets the output of all (honest) parties to $(\perp, p_i)$.

Definition 2. *Let $\mathcal{F}$ be a functionality and $\mathcal{F}^{\mathsf{ID}}$ be the corresponding functionality with identifiable abort. We say that a protocol Π securely realizes $\mathcal{F}$ with identifiable abort if Π securely GUC-realizes the functionality $\mathcal{F}^{\mathsf{ID}}$.*

The notion of unanimous abort instead guarantees that either all or none of the honest parties abort. We denote a functionality $\mathcal{F}$ that can be GUC-realized with unanimous abort by $\mathcal{F}^{\mathsf{UNA}}$. We refer to App. K for a formal description of the functionality. We say that an adversary is *non-aborting* if he never causes an honest parties to output $(\perp, p)$ (i.e., to abort) [IKLP06].

4.4 Collusion-preserving computation.

In this section we recall the notion of collusion-preserving computation proposed in [AKMZ12]. We refer the reader to [AKMZ12] for a more thorough discussion on CP.

For n the number of parties and $\mathcal{I} \subseteq [n]$, denote by $\mathcal{A}_{\mathcal{I}}$ the set of adversaries, i.e. $\mathtt{ITMs}$ $\{\mathcal{A}_i\}_{i\in\mathcal{I}}$, where $\mathcal{A}_i$ denotes the adversary corresponding to party p_i. In collusion-preserving computation, instead of one monolithic adversary/simulator, we consider a set of (independent) PPT adversaries and simulators. In more detail, we require the following:

Split adversaries/simulators: Instead of a monolithic adversary/simulator we consider a set of n (independent) PPT adversaries $\mathcal{A}_{[n]} = \{\mathcal{A}_i, i \in [n]\}$, where $\mathcal{A}_i$ corresponds to the adversary associated with the player i (and can corrupt at most this party). We also ask that for each $\mathcal{A}_i \in \mathcal{A}_{[n]}$ there exists an (independent) simulator Sim_i.

Corrupted-Set Independence: We also require that the simulators do not depend on each other. In other words the code of simulator Sim_i is the same for any set of adversaries $\mathcal{A}_{[n]}$ and $\mathcal{B}_{[n]}$ as long as $\mathcal{A}_i = \mathcal{B}_i$.

Moreover, similar to the GUC framework (but in contrast to plain UC) we distinguish between two types of functionalities: resources which we denote with capital calligraphic font as in $\mathcal{R}$ and shared functionalities which we denote with an additional over-line as in $\bar{\mathcal{G}}$. Formally a resource $\mathcal{R}$ maintains state only with respect to a single instance of a protocol, while a shared functionality $\bar{\mathcal{G}}$ can maintain state across protocol instances. For example concurrent executions can maintain shared state via say a global CRS or via a global PKI as long as these are modeled as shared functionalities. However, although concurrent instances of a protocol π may use the same resource $\mathcal{R}$, the behavior of $\mathcal{R}$ in one execution of π must be independent of all other executions of π (and more generally of all other concurrent protocols instantiated by the environment). For clarity, in the remainder of this work we will usually refer to shared functionalities simply as setup and protocols which only share state across executions through some setup $\bar{\mathcal{G}}$ as $\bar{\mathcal{G}}$-subroutine respecting.

We denote by $\text{CP-EXEC}^{\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}}$ the output of the environment $\mathcal{Z}$ in the execution of Π with adversaries $\mathcal{A} := \mathcal{A}_{[n]}$ in the $\mathcal{R}$-hybrid model. We say that a protocol Π is $\mathcal{R}$-*exclusive* if it makes use of no resources or shared functionality other than $\mathcal{R}$. Unlike (G)UC, CP limits parties to communicate with at most one single instance of the resource. Intuitively, if $\mathcal{F}$ is a one-bit channel, then the simulator only using one instance of $\mathcal{F}$ has a completely different meaning in terms of collusion-preservation to the simulator using unlimited calls to $\mathcal{F}$.

Definition 3 (Collusion Preservation [AKMZ12])**.** *Let $\bar{\mathcal{G}}$ be a setup, $\mathcal{R}$ and $\mathcal{F}$ be n-party resources, Π be a $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol (both with n parties). Then we say that Π collusion-preservingly (CP) emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{F}\}$-hybrid world, if there exists a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \dots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds: $\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$*

4.5 Rational protocol design.

The goal of the rational protocol design framework (RPD) [GKM+13, BGM+18] is to model incentive-driven adversaries, and design protocols which optimally reduce the attacker's utility. In RPD, a protocol designer $\mathtt{D}$ engages in an *attack game* with an attacker $\mathtt{A}$. The designer first chooses a n-party protocol $\Pi \in \mathtt{ITM}^n$. Then, the attacker decides on an adversarial strategy $\mathcal{A}$ to attack the protocol. Each choice of $(\Pi, \mathcal{A})$ induces a utility for the designer and the attacker.

Consistent with [GKM+13], we consider an attack game $\mathcal{G}_{\mathcal{M}}$, where $\mathcal{M}$ is the attack model $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}})$, a vector of parameters of the game. $\mathcal{F}$ is the functionality which the designer would like to achieve, and $\langle\mathcal{F}\rangle$ is a relaxed version of $\mathcal{F}$ in the sense that it allows the simulator extra commands to break certain security properties of $\mathcal{F}$. The value function $v_{\mathtt{A}}$ allows us to define utilities of the attacker, by assigning payoffs when certain events occur in the ideal world, such as when the simulator makes queries to $\langle\mathcal{F}\rangle$. Roughly speaking, then, the goal of the attacker is to generate an adversarial strategy $\mathcal{A}$ that will "force" the simulator to cause certain security breaches in the ideal world in order to complete a successful simulation. A protocol Π is *attack-payoff secure* if, informally, an attacker can gain no more utility from attacking Π than attacking the "dummy" protocol that uses the functionality $\mathcal{F}$ as a resource. Intuitively, it means that Π is "as secure as" the dummy protocol in this model.

5 Collusion-preservation with non-aborting adversaries

In this section we present our protocol $\Pi^{\mathtt{HT}}$ for any CP-well-formed functionality (Definition 10) under the following assumptions: 1) each party has access to a hardware token (which we describe as one global ideal functionality) 2) all communication is done over broadcast, and 3) adversarial parties do not make the protocol abort. For simplicity we restrict ourselves to non-reactive functionalities, also known as secure function evaluation. (The general case can be reduced to this case using a suitable form of secret sharing to maintain the secret state of the reactive functionality.) Moreover, we describe all our protocols in a round based, synchronous manner, where messages sent in some round are delivered by the beginning of the next round. We first introduce some additional notation:

- $\mathsf{sid} \in \mathbb{N}$ is an id that uniquely identifies an execution of $\Pi^{\mathtt{HT}}$.
- $\mathcal{P} = \{p_1, \dots, p_n\}$ is the set of parties that are interested in running an execution of $\Pi^{\mathtt{HT}}$ in order to compute the function f.
- We call *leader* the party that is in charge to run a special code and we assume w.l.o.g. that the leader is $p_n \in \mathcal{P}$.
- $T^{\mathtt{HT}}$ denotes the global token functionality.

For our construction we use the following tools.

- The pseudo-random functions
  $\mathsf{PRF}_0 : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$
  $\mathsf{PRF}_1 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$,
  $\mathsf{PRF}_2 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \rightarrow \{0,1\}^{(n+2)\lambda}$,
  $\mathsf{PRF}_3 : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^{(2n+1)\lambda}$.
- A strong unforgeable signature scheme $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver})$.
- A secret-key encryption scheme $\Pi^{\mathsf{SK}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

The global setup $\bar{\mathcal{G}}$ is represented by the token functionality $T^{\mathtt{HT}}$.

We note that we use one functionality to *emulate* the behavior of the hardware tokens held by the parties that run the protocol. This token functionality replies to each party $p_i \in \mathcal{P}$ using the appropriate code and keys depending to the identity of the calling party (i.e. the functionality discriminates between leader and non-leader parties). To not overburden the notation, in the formal construction we denote the identity of a party $p_i \in \mathcal{P}$ with i.

Moreover, the token functionality exports as public information the master public key mpk, and keep as part of its private state the master secret key msk together with with the PRF keys K_0, K_1, K_2, K_3.

The parties are allowed to communicate only via a broadcast channel denoted by $\mathcal{B}$ and formally described in Fig 12, App. K. We provide a formal description of $T^{\mathtt{HT}}$ in Fig. 1. The complete formal description of the the protocol $\Pi^{\mathtt{HT}}$ for the non-leader party is proposed in Fig. 2, and the protocol run by the leader parties is provided in Fig. 3.

We assume that the ideal functionality $\mathcal{F}$ that we wish to realize is registered to the token functionality. In addition, upon receiving the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$, $\mathcal{F}$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$

to the token functionality if sid is equal its session id, and forwards the answer to the ideal adversary. We recall that this trapdoor allows us to capture the broadcast channel, on which all parties see the exact same signed messages. Equipping the ideal functionality with the trapdoor command translates this *real-world leakage* in the ideal world. We also recall that the functionality leaks to the simulators only signing keys valid within one specific session without harming the token functionality globally.

We now prove that $\Pi^{\mathtt{HT}}$ is collusion-preserving against non-aborting adversaries for well-formed functionalities. Formally, we prove the following:

Theorem 1. *Let $\bar{\mathcal{G}} = T^{\mathtt{HT}}$ be the setup as defined above, $\mathcal{R} = \mathcal{B}$ (broadcast) and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality, $\Pi^{\mathtt{HT}}$ be the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol (described by Fig. 2 and 3) and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol (both with n parties). Then $\Pi^{\mathtt{HT}}$ collusion-preservingly emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{F}\}$-hybrid world assuming a non-aborting adversary.*

Proof sketch. To prove CP, we must show n independent simulators $\mathcal{S}_1, \dots, \mathcal{S}_n$. At a very high level for all $i \in \{1, \dots, n\}$, $\mathcal{S}_i$ can query $\mathcal{F}$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}$ checks that sid is equal to its session id, and if so, it sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates the string $\widetilde{K}$, n pairs of signing-verification keys, and authenticates the verification keys using the master secret key msk. $T^{\mathtt{HT}}$ then sends this information to $\mathcal{F}$ which forwards them to the simulator $\mathcal{S}_i$. Note that if all the simulators query $\mathcal{F}$ with $(\mathsf{GetTrapdoor}, \mathsf{sid})$ they will all get the same pair of authenticated signing-verification key and the string $\widetilde{K}$. Given the above information, a simulator $\mathcal{S}_i$ can use $\widetilde{K}$ as input of a PRG to generate the randomness sufficient to: 1) to create a key $\mathsf{Kenc}^{\mathsf{sid}}$ for a secret-key encryption scheme 2) compute $n + 1$ encryptions of 0^λ and 3) authenticate the encrypted value using the appropriate signature session keys.

These authenticated messages are now used to interact with $p_i^\star$ (who is not supposed to distinguish between the encryptions of 0^λ and the encryptions that contain the actual inputs of the parties). Moreover, whenever $\mathcal{S}_i$ receives the input form $p_i^\star$ he forwards it to the $\mathcal{F}$ which returns the output y_i. The crucial observation is that the messages seen by all the corrupted parties are the same since the simulators use exactly the same strategy and randomness, and since the adversary cannot forge a signature for the strong signature scheme we are using[6]. We refer to App. A for the formal proof.

(Publicly) Identifiable aborts. Another interesting property enjoyed by $\Pi^{\mathtt{HT}}$ is *identifiable abort*. A protocol run by a set of parties $\mathcal{P}$ is said to be secure with identifiable abort if it either computes according to its specification, or it aborts with the index of some corrupted party $p_i \in \mathcal{P}$ —i.e., every honest party learns p_i (see Def 2 for the formal definition). In $\Pi^{\mathtt{HT}}$ the adversary can only deviate from the protocol specification by 1) sending a message authenticated with respect to a sid' or f' not equal to the correct sid or f the honest parties use 2) sending a message with an invalid signature or certificate, or 3) fail to send a message. Each event is verifiable by the honest parties and any third party not involved in the protocol. Indeed, with the master public key mpk, sid and function f, it is possible to claim who did abort in a run of $\Pi^{\mathtt{HT}}$ by just inspecting its transcript.

[6]We note that it is crucial to use a strong signature scheme to avoid the creation of a different valid signature for a message m from valid signatures for the same message.

```
The token functionality is parameterized by a set of parties $\mathcal{P}$ and by a list $\overline{\mathcal{F}}$ of ideal functionality
programs. The functionality manages the keys $(\mathsf{mpk}, \mathsf{msk})$ for the signature scheme $\Sigma$ and the
PRF keys $K_0, K_1, K_2, K_3$.
Input phase for non-leader parties
 - If $I = (\mathsf{Input}, \mathsf{sid}, x, f)$ is received from a non-leader party $p_j$ then do the following.
     - If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
     - Compute $R_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid})$ and $\mathsf{Kenc}^{\mathsf{sid}} \leftarrow \mathsf{Gen}(1^\lambda; R_0)$
     - Compute $R_1 \leftarrow \mathsf{PRF}_1(K_1, \mathsf{sid}||j)$ and parse $R_1$ as 4 strings of $\lambda$ bits each $\mathsf{rs}^1||\mathsf{rs}^2||r^1||r^2$.
     - $(\mathsf{sigk}_j, \mathsf{vk}_j) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}^1)$
     - $\mathsf{cert}_j \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j||\mathsf{sid}||j; \mathsf{rs}^2)$
     - Compute $\overline{x} \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, x; r^1)$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \overline{x}||f; r^2)$ and output $(\overline{x}, f, \mathsf{vk}_j, \sigma, \mathsf{cert}_j)$.
Input/output phase for the leader party
 - If $I = (\mathsf{Input}, \mathsf{sid}, x_n, f, \mathsf{sid}, (\overline{x}_1, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \ldots, (\overline{x}_{n-1}, \mathsf{vk}_{n-1}, \sigma_{n-1}.\mathsf{cert}_{n-1}))$ is received
   from the leader party $p_n$ then check if for all $j \in [n-1]$ $\mathsf{Ver}(\mathsf{vk}_j, \overline{x}_j||f, \sigma_j) = 1$ and
   $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_j||\mathsf{sid}||j, \mathsf{cert}_j) = 1$. If it is not, then output $\perp$ and stop, otherwise act as fol-
   lows.
     - If $\mathsf{ctr}_n^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_n^{\mathsf{sid}} \leftarrow 1$ else output $\perp$ and stop.
     - $R_2 \leftarrow \mathsf{PRF}_2(K_2, \mathsf{sid}||n)$ and parse $R_2$ as $n+2$ strings of $\lambda$ bits each
         $\mathsf{rs}^1||\mathsf{rs}^2||r_1||r_2||\ldots||r_{n-1}||r^\star$.
     - $(\mathsf{sigk}_n, \mathsf{vk}_n) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}^1)$
     - $\mathsf{cert}_n \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_n||\mathsf{sid}||n; \mathsf{rs}^2)$
     - Compute $R_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid})$ and $\mathsf{Kenc}^{\mathsf{sid}} \leftarrow \mathsf{Gen}(1^\lambda; R_0)$.
     - For $j = 1, \ldots, n-1$ compute $x_j \leftarrow \mathsf{Dec}(\mathsf{Kenc}^{\mathsf{sid}}, \overline{x}_j)$.
     - Compute $y_1, \ldots, y_n \leftarrow f(x_1, \ldots, x_n)$.
     - For $j = 1, \ldots, n-1$ compute $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, y_j; r_j)$.
     - $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, \overline{y}_1||\ldots||\overline{y}_{n-1}||f; r^\star)$;
     - Output $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n), y_n$
Output phase for non-leader parties
 - If $I = (\mathsf{Output}, \mathsf{sid}, z)$ is received parse $z$ as $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ and do the following.
     - if $\mathsf{Ver}(\mathsf{vk}_n, \overline{y}_1||\ldots||\overline{y}_{n-1}||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_j||\mathsf{sid}||j, \mathsf{cert}_j) = 1$ then compute and
         output $\mathsf{Dec}(\mathsf{Kenc}^{\mathsf{sid}}, \overline{y}_j)$, output $\perp$ otherwise.
Trapdoor
 - If $I = (\mathsf{Trapdoor}, \mathsf{sid})$ is received from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$ then
   do the following.
     - Pick $\widetilde{K}||\mathsf{rs}_1^1||\mathsf{rs}_1^2||\ldots||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_3(K_3, \mathsf{sid})$.
     - For all $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$.
     - Return to the calling instance $((\mathsf{sigk}_1, \mathsf{vk}_1, \mathsf{cert}_1), \ldots, (\mathsf{sigk}_n, \mathsf{vk}_n, \mathsf{cert}_n), \widetilde{K})$.
```

Figure 1: The HT functionality $T^{\mathtt{HT}}$

We assume that the party p_j is registered to the token functionality $T^{\mathtt{HT}}$. Each party is aware of the function that will be computed f, of the identifier of each execution sid, and of the parties involved in each of those executions $\mathcal{P}$.

Input.

- The party p_j on input $(\mathsf{Compute}, \mathsf{sid}, x)$ sends $(\mathsf{Input}, \mathsf{sid}, x, f)$ to $T^{\mathtt{HT}}$.
- Upon receiving the answer X from $T^{\mathtt{HT}}$, if $X = \perp$ then p_j outputs $\perp$ and stops. Otherwise, p_j sends X to p_n.

Output. The party p_j, upon receiving $z = (\overline{y}_1, \dots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ from p_n sends $(\mathsf{Output}, \mathsf{sid}, z)$ to $T^{\mathtt{HT}}$.

Upon receiving y from $T^{\mathtt{HT}}$, p_j outputs y.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, f', \mathsf{vk}, \sigma, \mathsf{cert})$ is received from a party p_i check if $f = f'$ and $\mathsf{Ver}(\mathsf{vk}_i, m||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}||\mathsf{sid}||i, \mathsf{cert}) = 1$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 2: The party p_j

We assume that the party p_n is registered to the token functionality $T^{\mathtt{HT}}$. Each party is aware of the function that will be computed f, of the identifier of each execution sid, and of the parties involved in each of those executions $\mathcal{P}$.

Input/output

- The party p_n on input $(\mathsf{Compute}, \mathsf{sid}, x_n)$ collects messages from $p_{j \in [n-1]}$ and sends $I = (\mathsf{Input}, x_n, f, \mathsf{sid}, (\overline{x}_1, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \dots, (\overline{x}_{n-1}, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1}))$ to $T^{\mathtt{HT}}$.
- Upon receiving the answer Y from $T^{\mathtt{HT}}$, if $Y = \perp$ then outputs $\perp$ and stops. Otherwise parses Y as $((m, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n), y)$, sends $(m, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to all the parties in $\mathcal{P}$ and outputs y.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, f', \mathsf{vk}, \sigma, \mathsf{cert})$ is received from a party p_i check if $f = f'$ and $\mathsf{Ver}(\mathsf{vk}_j, m||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}||\mathsf{sid}||j, \mathsf{cert}) = 1$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 3: The leader party p_n

More formally, the protocol $\Pi^{\mathtt{HT}}$ securely realizes the function $\mathcal{F}^{\mathsf{IDA}}$, where $\mathcal{F}^{\mathsf{IDA}}$ involves n parties. We now modify $\Pi^{\mathtt{HT}}$ to support an additional party p_{n+1} which takes no input, does not send any message and outputs a default value (e.g., 0). Since p_{n+1} knows the master public key mpk, she can check the validity of the signature and the certificate. Hence, she is able to identify an invalid message (in the case p_{n+1} is honest). Roughly, our protocol allows an *observer* of the protocol execution to identify a misbehaving party. Following [KZZ16] we refer to this property as *publicly identifiable abort.* In the same spirit of [KZZ16], we refer to p_{n+1} as a *judge.* The code of the judge can be used by anyone who has the public setup and wants to follow the protocol execution and decide whether it should abort or not given the parties' messages. Looking ahead, the judge's code in the protocol can be used by a blockchain ledger to decide whether or not a transaction contains a valid protocol message of Π^{HT}.

6 Collusion-preservation with fallback security

Despite being simple and optimal in terms of round complexity, the protocol described in Sec. 5 suffers from a major limitation. That is, in the case that the hardware tokens are corrupted (e.g., the secret keys are leaked by the token manufacturer) then not only the CP-property is lost, but we cannot even guarantee any protection with regard to the inputs of the honest parties. In this section we proposed a protocol that protects the inputs of the honest parties (in a standard GUC-security sense) even in the case where: 1) the adversary knows all the secret keys of the hardware tokens (also the secret keys of the hardware tokens held by the honest parties) and 2) the hardware tokens of the malicious parties can be completely reprogrammed by the adversary.

Let $\mathcal{F}$ be the function that the parties want to compute. For our construction we use the following tools.

- Pseudo-random functions $\mathsf{PRF}_0 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \rightarrow \{0,1\}^{(2m+4)\lambda}$ and $\mathsf{PRF}_1 : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^{(2n+1)\lambda}$
- A strong unforgeable signature scheme $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver})$.
- A secure MPC protocol $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$ that GUC-realizes the function $\mathcal{F}$.

The global setup $\bar{\mathcal{G}}$ is represented by the token functionality $T^{\text{HT-FBS}}$. As for the token functionality of the previous section, $T^{\text{HT-FBS}}$ has a master public key mpk and a secret state made by the corresponding master secret key msk and the PRF keys K_0, K_1. We assume without loss of generality that the setup required to run Π^{MPC} is part of $T^{\text{HT-FBS}}$.

We denote our protocol with $\Pi^{\text{HT-FBS}}$ and provide a formal description of $T^{\text{HT-FBS}}$ in Fig. 4. The complete formal description of the the protocol is proposed in Fig. 5.

6.1 Security of $\Pi^{\text{HT-FBS}}$.

We summarize the properties of the protocol $\Pi^{\text{HT-FBS}}$.

1. If the hardware tokens are honestly generated and secure, and no party aborts, then $\Pi^{\text{HT-FBS}}$ is collusion-preserving.
2. If the hardware tokens are compromised and Π^{MPC} GUC realizes $\mathcal{F}^{\mathsf{AB}}$ with $\mathsf{AB} \in \{\mathsf{IDA}, \mathsf{UNA}\}$, then $\Pi^{\text{HT-FBS}}$ GUC realizes $\mathcal{F}^{\mathsf{AB}}$.
3. If the hardware tokens are honestly generated and secure (but the malicious parties may abort), then $\Pi^{\text{HT-FBS}}$ GUC realizes the functionality $\mathcal{F}$ with publicly identifiable abort.

The properties 1 and 2 enables the fallback security of $\Pi^{\text{HT-FBS}}$. In addition, the second property states that in the case of corrupted tokens, $\Pi^{\text{HT-FBS}}$ inherits all the properties of Π^{MPC} (e.g., identifiable abort). We note passing that by assuming honest majority and a protocol $\Pi^{\mathsf{MPC}'}$ that guarantees fairness (or even output delivery) this property would be held by $\Pi^{\text{HT-FBS}}$ as well. The third property states that if an adversarial party aborts then the CP property might be lost, but the input of the honest parties are protected. We capture the case where the hardware tokens are compromised by considering the token functionality $\overline{T}^{\text{HT-FBS}}$ instead of $T^{\text{HT-FBS}}$. $\overline{T}^{\text{HT-FBS}}$ extends $T^{\text{HT-FBS}}$ with the additional command Tamper. If the adversary queries the token functionality with Tamper then $\overline{T}^{\text{HT-FBS}}$ leaks to the adversary its secret state (i.e., the master secret key msk and the PRF keys). Given the master secret key, the adversary can authenticate any message he wants and therefore acts on the behalf of the hardware token.

We now provide the formal theorems.

Theorem 2. $\Pi^{\texttt{HT-FBS}}$ *is fallback secure.*

To prove this theorem we prove the following two lemmata.

Lemma 1. *Let $\bar{\mathcal{G}} = T^{\texttt{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$ (broadcast), $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality, $\Pi^{\texttt{HT-FBS}}$ be the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol (described by Fig. 5) and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol (both with n parties). Then $\Pi^{\texttt{HT-FBS}}$ collusion-preservingly emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{F}\}$-hybrid world assuming that no parties abort.*

Lemma 2. *Let Π^{MPC} be a protocol that GUC-realizes the n-party functionality $\mathcal{F}^{\mathsf{AB}}$ with $\mathsf{AB} \in \{\mathsf{IDA}, \mathsf{UNA}\}$ that exclusively uses $\mathcal{B}$ as a resource. Let $\bar{\mathcal{G}} = \overline{T}^{\texttt{HT-FBS}}$ and $\mathcal{R} = \mathcal{B}$ then $\forall \mathcal{A}\ \exists \mathsf{Sim}\ \forall \mathcal{Z}$ $\mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\texttt{HT-FBS}},\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{F}^{\mathsf{AB}}}_{\mathsf{Sim},\mathcal{Z}}$*

To prove the security of Lemma 1 we rely on the fact that any execution of $\Pi^{\texttt{HT-FBS}}$ can be seen as an execution of Π^{MPC} among honest parties. Indeed, in the case the hardware tokens are not corrupted the adversary has no control on the messages of Π^{MPC}, and he can only act as a proxy between the hardware tokens and the broadcast channel (since we assume that the adversary is non-aborting). This allows the CP simulators $\mathcal{S}_1, \ldots, \mathcal{S}_n$ to internally run the simulator Sim of Π^{MPC} (that exists by definition) for the case where no parties are corrupted. Hence, $\mathcal{S}_1, \ldots, \mathcal{S}_n$ can just run Sim using the same correlated randomness obtained from the trapdoor information $\widetilde{K}$, following the approach proposed in Sec 5. To prove the security of Lemma 2 we can reduce the security of the entire protocol to the GUC security of Π^{MPC}. We note that in Lemma 2 the global setup is represented by $\overline{T}^{\texttt{HT-FBS}}$, which captures the scenario where the hardware tokens are compromised. We refer to App. A.2 form more detail on the proofs of the lemmata.

Theorem 3. *Let $\bar{\mathcal{G}} = T^{\texttt{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$, $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality, $\Pi^{\texttt{HT-FBS}}$ be the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol (described by Fig. 5) then $\Pi^{\texttt{HT-FBS}}$ GUC realizes $\mathcal{F}^{\mathsf{IDA}}$.*

Proof sketch. The only way that an adversary has to misbehave is by sending an invalid signature over the channel. This behavior can be detected by any party that has the verification keys of the token functionalities. Moreover, the first party that sends a unsigned message is identified as corrupted.

7 Our new model: RPD-CP

In this section, we use the RPD framework to study the feasibility of secure function evaluation (SFE) in a collusion preserving way. We consider the ideal functionality $\mathcal{F}^f$ for SFE parameterized by the function $f : \mathbb{F}^n \to \mathbb{F}$ (this form is without loss of generality—see, e.g., [LP09]). We refer the reader to Sec. K Fig. 11 for a formal definition of $\mathcal{F}^f$. Our goal is to realize SFE with collusion preservation, for incentive-driven adversaries.

The functionality is parameterized by a list $\overline{\mathcal{F}}$ of ideal functionality programs. The token functionality is parameterized by an m-round MPC protocol $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$, and a set of n parties $\mathcal{P}$. It manages a round number ℓ, initialized as 1. The functionality manages the keys $(\mathsf{mpk}, \mathsf{msk})$ for the signature scheme Σ. The functionality manages also the PRF keys K_0, K_1.

Input phase

- If $I = (\mathsf{Input}, \mathsf{sid}, x_j, R_j)$ is received from some party p_j, then do the following,
  - If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined, then define it and $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$, otherwise output $\perp$ and stop.
  - Compute $R_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid}||j) \oplus R_j$ and parse R_0 as $(2m+4)$ strings of λ bits $\mathsf{rs}_j^1||\mathsf{rs}_j^2||\rho_j^1||\ldots||\rho_j^{m+1}||r_j^1||\ldots||r_j^{m+1}$.[a]
  - $(\mathsf{sigk}_j, \mathsf{vk}_j) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}_j^1)$
  - $\mathsf{cert}_j \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j||\mathsf{sid}||j; \mathsf{rs}_j^2)$
  - Output the first message $(\mathtt{msg}_j^1, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\mathtt{msg}_j^1 = \mathsf{Next}_j(1^\lambda, x_j, \rho_j^1, \perp)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}_j^1||\Pi^{\mathsf{MPC}}||\ell; r_j^1)$

Next message function

- If $I = (\mathsf{NextMsg}, \mathsf{sid}, \{\mathtt{msg}_i^\ell, \sigma_i^\ell, \mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]})$ is received from p_j then do the following.
  - If $\ell > m$ then output $\perp$ and stop, else continue with the following steps.
  - Store $\mathtt{msg}^\ell = \{\mathtt{msg}_i^\ell\}_{i\in[n]}$.
  - For all $i \in [n]$ check if $\mathsf{Ver}(\mathsf{vk}_i, \mathtt{msg}_i^\ell||\Pi^{\mathsf{MPC}}||\ell, \sigma_i^\ell) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_i||\mathsf{sid}||i, \mathsf{cert}_i) = 1$. If it is not then output $(p_i, \perp)$ and stop. Otherwise continue with the following steps.
  - Set $\ell \leftarrow \ell + 1$, compute $\mathtt{msg}_j^\ell = \mathsf{Next}_j(1^\lambda, x_j, \rho_j^\ell, \overline{\mathtt{msg}}^{<\ell})$ with $\overline{\mathtt{msg}}^{<\ell} = \{\mathtt{msg}_i^{\ell'}\}_{i\in[n],\ell'<\ell}$ where $\mathtt{msg}_i^{\ell'}$ is the message from p_i at round ℓ'.
  - If $\ell = m+1$ then output $y_j \leftarrow \mathtt{msg}_j^\ell$
    else, output $(\mathtt{msg}_j^\ell, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_j^\ell, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\sigma_j^\ell \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}_j^\ell||\Pi^{\mathsf{MPC}}||\ell; r_j^\ell)$.

Trapdoor

- If $I = (\mathsf{Trapdoor}, \mathsf{sid})$ is received from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$ then do the following.
  - Pick $\widetilde{K}||\mathsf{rs}_1^1||\mathsf{rs}_1^2||\ldots||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_1(K_1, \mathsf{sid})$.
  - For all $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$.
  - Return to the calling instance $((\mathsf{sigk}_1, \mathsf{vk}_1, \mathsf{cert}_1), \ldots, (\mathsf{sigk}_n, \mathsf{vk}_n, \mathsf{cert}_n), \widetilde{K})$.

[a] rs_j^1 and rs_j^2 are the randomnesses used to generate the session signing-verification keys and to authenticate them using the msk; ρ_j^ℓ is the randomness used to compute the ℓ-th round message for p_j; r_j^ℓ is used to sign the message for p_j at round ℓ using the session signing key.

Figure 4: The HT functionality with fallback security $T^{\mathtt{HT\text{-}FBS}}$

We assume that the party p_j is registered to the global token functionality $T^{\texttt{HT-FBS}}$. Each party is aware of the m-round protocol that will be computed $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$ and of the parties involved in protocol execution $\mathcal{P}$.

Input and next message generation.

- The party p_j on input $(\mathsf{Compute}, \mathsf{sid}, x)$ samples uniform random $R_j \in \{0,1\}^{(2m+4)\lambda}$ and sends $I = (\mathsf{Input}, \mathsf{sid}, x_j, R_j, \Pi^{\mathsf{MPC}})$ to $T_j^{\texttt{HT-FBS}}$.
- For each $\ell \in \{1, \ldots, m\}$:
    - Upon receiving message X from $T_j^{\texttt{HT-FBS}}$ check if $X = (\perp, p_{i'})$. If it is then output $(\perp, p_{i'})$ and stop, otherwise send X over broadcast.
    - Collect message $(\texttt{msg}_i^{\ell}, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_i^{\ell}, \mathsf{vk}_i, \mathsf{cert}_i)$ for round ℓ from each party $p_i \in [n]\backslash\{j\}$ and send $(\mathsf{NextMsg}, \mathsf{sid}, \{\texttt{msg}_i^{\ell}, \sigma_i^{\ell}, \mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]})$ to $T_j^{\texttt{HT-FBS}}$.

Output phase.

- Collect the message $(\texttt{msg}_i^{\ell}, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_i^{\ell}, \mathsf{vk}_i, \mathsf{cert}_i)$ for round ℓ from each party $p_i \in [n]\backslash\{j\}$ and send $(\mathsf{NextMsg}, \mathsf{sid}, \{\texttt{msg}_i^{\ell}, \sigma_i^{\ell}, \mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]})$ to $T_j^{\texttt{HT-FBS}}$.
- Upon receiving y_j from $T_j^{\texttt{HT-FBS}}$ output it.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma, \mathsf{vk}, \mathsf{cert})$ is received from a party p_i check if $\mathsf{Ver}(\mathsf{vk}_i, m||\Pi^{\mathsf{MPC}}||\ell, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}||\mathsf{sid}||i, \mathsf{cert}) = 1$ for some $\ell \in [m]$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 5: The party p_j for fallback security

The Attack Model Following the RPD framework [GKM+13], we capture incentive-driven attackers against collusion-preservation, by considering attacks as part of an *attack game* $\mathcal{G}_{\mathcal{M}}$ between a protocol designer $\mathtt{D}$ and attacker $\mathtt{A}$. Here, $\mathtt{D}$ comes up with a protocol Π, and the attacker $\mathtt{A} \in \mathtt{ITM}$ generates a set of adversaries $\mathtt{A}(\Pi) = \mathcal{A} = \{\mathcal{A}_i\}_{i\in\mathcal{I}}$, $\mathcal{I} \subseteq [n]$ to attack it. The utility of the attacker $u_{\mathtt{A}}$ is then defined as a function of the choice of protocol Π and adversarial strategies $\mathcal{A}$. The attack model $\mathcal{M} = (\mathcal{F}^f, \langle\mathcal{F}^f\rangle, v_{\mathtt{A}})$ encompasses the parameters in this game. Our protocols aim at realizing SFE in which the parties always obtain their outputs in a collusion preserving manner. $\langle\mathcal{F}^f\rangle$ is the weaker version $\mathcal{F}^f$, which explicitly allows 1) CP to be broken by sending a colluding message to other adversarial parties and 2) the adversarial parties to abort and being identified by all the other parties (adversarial and not) that are running the protocol. We note that in contrast to monolithic adversaries and simulators, in CP the ideal adversarial parties do not automatically share their views and must use $\langle\mathcal{F}^f\rangle$ to communicate. We provide the formal description of $\langle\mathcal{F}^f\rangle$ in Fig. 6. The value function $v_{\mathtt{A}}$ defines the utility of the attacker. This function maps the joint views of the simulators (interacting with the relaxed functionality $\langle\mathcal{F}^f\rangle$) and the environment $\mathcal{Z}$, to a real-valued payoff.

7.1 Utility of the attacker $\mathtt{A}$.

The utility of the attacker $u_{\mathtt{A}}$ is a function mapping protocols and sets of adversaries, i.e. $(\Pi, \mathcal{A})$, to a real number. Intuitively, utility depends on how often must a set of simulators $\mathcal{S}$ break CP via the functionality $\langle\mathcal{F}^f\rangle$ in order to emulate $\mathcal{A}$, given Π.

As the first step of defining $u_{\mathtt{A}}$, we define the payoff of a particular set of simulators, using the value function $v_{\mathtt{A}}$ defined in the attack model. Then, we define the real payoff of a particular set of adversaries, based on the payoffs of simulators that can emulate them. Finally, we define $u_{\mathtt{A}}(\Pi, \mathcal{A})$ as the real payoff the set of adversaries $\mathcal{A}$, maximized over all possible environments $\mathcal{Z}$.

Ideal payoff of a set of simulators. We define the real-valued random variable ensemble $\{v_{\mathtt{A}}^{\langle\mathcal{F}^f\rangle,\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ ($v_{\mathtt{A}}^{\langle\mathcal{F}^f\rangle,\mathcal{S},\mathcal{Z}}$ for short) as the random variable ensemble resulting from applying value function $v_{\mathtt{A}}$ to the view of the environment $\mathcal{Z}$ and a set of simulators $\mathcal{S}=\{\mathcal{S}_i\}_{i\in\mathcal{I}}$ in the ideal execution. The ideal expected payoff of a particular set of simulators $\mathcal{S}$ with respect to $\mathcal{Z}$ is defined as the expected value: $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})=E(v_{\mathtt{A}}^{\langle\mathcal{F}^f\rangle,\mathcal{S},\mathcal{Z}})$

Real payoff of a set of adversaries. Recall that given a setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$, a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive (that is, the protocol only uses $\bar{\mathcal{G}},\mathcal{R}$) protocol Π realizes a CP-functionality $\langle\mathcal{F}^f\rangle$ if, for all $\mathcal{I}\subseteq[n]$, and independent (rather than monolithic) adversaries $\mathcal{A}=\{\mathcal{A}_i\}_{i\in\mathcal{I}}$, there exists a collection of efficiently computable transformations from ITMs to ITMs $\mathsf{Sim}=\{\mathsf{Sim}_i\}_{i\in\mathcal{I}}$ such that the simulator $\mathcal{S}_i=\mathsf{Sim}_i(\mathcal{A}_i)$ emulates $\mathcal{A}_i$. That is, the environment $\mathcal{Z}$ cannot distinguish between the real world with $\mathcal{A}$ and resource $\mathcal{R}$, and ideal world with $\mathcal{S}=\{\mathsf{Sim}_i(\mathcal{A}_i)\}_{i\in\mathcal{I}}$ and $\langle\mathcal{F}^f\rangle$. Let $\langle\mathcal{F}^f\rangle$ be a CP functionality and Π be a protocol. Denote $\mathcal{C}_{\mathcal{A}}$ as the class of simulators $\mathcal{S}=\{\mathcal{S}_i\}_{i\in\mathcal{I}}$ that can emulate the adversarial parties $\mathcal{A}=\{\mathcal{A}_i\}_{i\in\mathcal{I}}$ for $\mathcal{I}\subseteq[n]$. That is, for setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$, $\mathcal{C}_{\mathcal{A}}=\Big\{\mathsf{Sim}(\mathcal{A})=\{\mathsf{Sim}_i(\mathcal{A}_i)\}_{i\in\mathcal{I}}\ \mid\ \forall i\in\mathcal{I}\colon \mathsf{Sim}_i$ is an efficiently computable mapping from $\mathtt{ITM}$ to $\mathtt{ITM}$, $\forall\mathcal{Z}\colon \text{CP-EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}^{\bar{\mathcal{G}},\mathcal{R}}\approx\text{CP-EXEC}_{\Pi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}^{\bar{\mathcal{G}},\langle\mathcal{F}^f\rangle}\Big\}$ The expected payoff of a set of adversaries and environment $(\mathcal{A},\mathcal{Z})$ is then defined as $U_{\mathtt{A}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A},\mathcal{Z})=\inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}}\{U_{I^{\mathtt{A}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})\}$ The attacker's utility is then $u_{\mathtt{A}}(\Pi,\mathcal{A}):=\hat{U}_{\mathtt{A}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A})=\sup_{\mathcal{Z}\in\mathtt{ITM}}\{U_{\mathtt{A}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A},\mathcal{Z})\}$

The functionality interacts with a set of parties $\mathcal{P}=\{p_1,\ldots,p_n\}$.
It maintains a set of honest parties $\mathcal{H}\subseteq\mathcal{P}$, and a set of malicious parties $\mathcal{I}\subseteq\mathcal{P}$ with $\mathcal{H}\cup\mathcal{I}=\mathcal{P}$ and $\mathcal{H}\cap\mathcal{I}=\emptyset$.

1. Upon receiving $(\mathtt{COLLUDE},\mathsf{sid},m)$ from party $p_i\in\mathcal{I}$, send message $(\mathtt{SUB\_MSG},\mathsf{sid},p_i,m)$ to p_j, for all $p_j\in\mathcal{P}$-$\{p_i\}$.
2. Upon receiving $(\mathtt{ABORT},\mathsf{sid})$ from a party p_i, send $(\mathtt{ABORT},p_i)$ to p_j, for all $p_j\in\mathcal{P}$-$\{p_i\}$ and stop.
3. Upon receiving a message that is consistent with the interface of $\mathcal{F}^f$ act as $\mathcal{F}^f$ would do acting as a proxy between $\mathcal{F}^f$ and the parties in $\mathcal{P}$.

Figure 6: $\langle\mathcal{F}^f\rangle$

7.2 Utility of the protocol designer D

In [GKM+13], the attack game is assumed to be zero-sum. In order to remove this assumption, the methodology of a more recent work [BGM+18] can be used to define the utility of the protocol designer. In more detail, for each $(\Pi,\mathcal{A})$, we must assign utility for the designer using the same simulators and environments as those used for the attacker. Let $\mathcal{S}_{\mathtt{A}}$ denote the class of simulators that were used to obtain the utility of the adversary, and $\mathcal{Z}_{\mathtt{A}}$ denote the class of environments maximizing the utility for simulators in $\mathcal{S}_{\mathtt{A}}$. That is, $\mathcal{S}_{\mathtt{A}}=\Big\{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}\colon \sup_{\mathcal{Z}\in\mathtt{ITM}}\{U_{I^{\mathtt{A}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})\}=u_{\mathtt{A}}(\Pi,\mathcal{A})\Big\}$ and $\mathcal{Z}_{\mathtt{A}}=\Big\{\mathcal{Z}\in\mathtt{ITM}\colon \text{for some } \mathcal{S}\in\mathcal{S}_{\mathtt{A}}\ ,\ U_{I^{\mathtt{A}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})=u_{\mathtt{A}}(\Pi,\mathcal{A})\Big\}$

Let $v_{\mathtt{D}}^{\langle\mathcal{F}^f\rangle,\mathcal{S},\mathcal{Z}}$ and $U_{I^{\mathtt{D}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})$ be defined similar to the payoffs $v_{\mathtt{A}}^{\langle\mathcal{F}^f\rangle,\mathcal{S},\mathcal{Z}}$ and $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})$ respectively. Then, the real payoff of the designer is $U_{\mathtt{D}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A},\mathcal{Z}) = \sup_{\mathcal{S}\in\mathcal{S}_{\mathtt{A}}}\{U_{I^{\mathtt{D}}}^{\langle\mathcal{F}^f\rangle}(\mathcal{S},\mathcal{Z})\}$. The utility of the designer is then $u_{\mathtt{D}}(\Pi,\mathcal{A}) := \hat{U}_{\mathtt{D}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A}) = \inf_{\mathcal{Z}\in\mathcal{Z}_{\mathtt{A}}}\{U_{\mathtt{D}}^{\Pi,\langle\mathcal{F}^f\rangle}(\mathcal{A},\mathcal{Z})\}$. We also extend the attack model with the utility of the designer $v_{\mathtt{D}}$: $\mathcal{M} = (\mathcal{F}^f, \langle\mathcal{F}^f\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$.

7.3 Security definitions

Similar to the definition of *attack-payoff secure* in [GKM+13, BGM+18], we define collusion preserving attack payoff (CPAP). Intuitively, a protocol is CPAP with respect to an attack model $\mathcal{M} = (\mathcal{F}^f, \langle\mathcal{F}^f\rangle, v_{\mathtt{A}})$ if it enjoys security and collusion-preservation under this model. That is, no adversary can gain more utility from running our protocol, than running the dummy protocol that uses a functionality $\mathcal{F}^f$ (without the relaxations offered by $\langle\mathcal{F}^f\rangle$) as a resource.

Definition 4 (CPAP). *Let $\mathcal{M} = (\mathcal{F}^f, \langle\mathcal{F}^f\rangle, v_{\mathtt{A}})$ be an attack model and Π a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol that realizes $\langle\mathcal{F}^f\rangle$. We say that Π is* CPAP *in $\mathcal{M}$ if $\sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Pi,\mathcal{A}) \overset{negl}{\leq} \sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Phi^{\mathcal{F}^f},\mathcal{A})$ where $\Phi^{\mathcal{F}^f}$ is the dummy $\{\bar{\mathcal{G}},\mathcal{F}^f\}$-hybrid protocol which forwards all inputs to and outputs from functionality $\mathcal{F}^f$.*

To complete our framework, we also define ϵ-subgame-perfect equilibrium from [GKM+13], and define CPIC similar to the definition of *incentive compatible (IC)* in [BGM+18]. Informally, a strategy profile is an ϵ-subgame-perfect equilibrium if no deviation by the attacker nor designer can improve their utility by more than ϵ. Informally, a protocol Π is incentive compatible if both the designer and attacker are willing to stick with it. That is, there is an attacker $\mathtt{A}$ such that $(\Pi,\mathtt{A})$ is in equilibrium.

Definition 5 (ϵ-subgame-perfect equilibrium [GKM+13]). *Let $\mathcal{G}_{\mathcal{M}}$ be an attack game. A strategy profile $(\Pi,\mathtt{A})$ is an* ϵ-subgame perfect equilibrium *in $\mathcal{G}_{\mathcal{M}}$ if the following conditions hold: (1) for any $\Pi' \in \mathtt{ITM}^n$, $u_{\mathtt{D}}(\Pi',\mathtt{A}(\Pi')) \leq u_{\mathtt{D}}(\Pi,\mathtt{A}(\Pi))+\epsilon$, and (2) for any $\mathtt{A}' \in \mathtt{ITM}$, $u_{\mathtt{A}}(\Pi,\mathtt{A}'(\Pi)) \leq u_{\mathtt{A}}(\Pi,\mathtt{A}(\Pi))+\epsilon$.*

Definition 6 (CPIC). *Let Π be a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol and $\mathbf{\Pi}$ be a set of polynomial-time $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocols. We say that Π is* $\mathbf{\Pi}$-CPIC *in the attack model $\mathcal{M}$ iff for some $\mathtt{A} \in \mathtt{ITM}$, $(\Pi,\mathtt{A})$ is a $\nu(\lambda)$-subgame perfect equilibrium on the restricted attack game where the set of deviations of the designer is $\mathbf{\Pi}$.*

8 Our CPAP protocol

In this section we show that the protocol $\Pi^{\mathtt{HT}}$ presented in the previous section is *CPAP* for a natural class of value functions $v_{\mathtt{A}}$.

Concrete utility function. Let $\mathcal{F}^f$ be a CP-well-formed functionality. Consider the following events the value function $v_{\mathtt{A}}$ is concerned with. These are events defined on the views of the environment, the (relaxed) CP-functionality $\langle\mathcal{F}^f\rangle$, and the simulators $\mathcal{S} = \{\mathcal{S}_i\}_{i\in\mathcal{I}}$, given adversaries $\mathcal{A} = \{\mathcal{A}_i\}_{i\in\mathcal{I}}$:

1. Define the event $E_{\mathtt{collude}}$ as follows: For some $i \in \mathcal{I}$ and message m, the i-th simulator $\mathcal{S}_i$ sends the message $(\mathtt{COLLUDE}, \mathsf{sid}, m)$ to $\langle\mathcal{F}^f\rangle$.

2. Define the event $E_{\texttt{abort}}$ as follows: For some $i \in \mathcal{I}$, party p_i aborts and is identified by all the parties as having aborted.

Now, we define the payoffs assigned by $v_{\mathtt{A}}$ to the events above. Denote by $\gamma_{\texttt{collude}}$ the utility for the attacker obtained by sending a colluding message. Denote by $\gamma_{\texttt{abort}}$ the penalty incurred as result of a malicious party being identified by the honest parties as an aborting party. Then, $u_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z}} \left\{ \inf_{\mathcal{S} \in \mathcal{C}_{\mathcal{A}}} \left\{ \gamma_{\texttt{collude}} \Pr[E_{\texttt{collude}}] - \gamma_{\texttt{abort}} \Pr[E_{\texttt{abort}}] \right\} \right\}$

We can now prove that our protocol satisfies CPAP security under the condition that the penalty of being identified as having aborted is greater than the gain from sending a colluding message. These assumptions are natural in a game-playing setting, such as poker. If all parties are corrupted by the attacker, then the winner would always be a corrupt party, regardless whether colluding message was sent. Moreover, if a party is identified as aborting (i.e. cheating), they could be banned from playing, or even receive a severe financial penalty. In Section 9, we will discuss penalization schemes that make these penalties concrete.

Theorem 4. *Let $\mathcal{F}^f$ be an ideal CP-well-formed functionality, and $\langle \mathcal{F}^f \rangle$ be as defined in Figure 6. Let $v_{\mathtt{A}}$ be as defined above, for any $\gamma_{\texttt{collude}}$ and $\gamma_{\texttt{abort}}$ such that $\gamma_{\texttt{abort}} > \gamma_{\texttt{collude}}$. Then the protocol $\Pi^{\texttt{HT}}$ described in Sec. 5 (and the protocol $\Pi^{\texttt{HT-FBS}}$ described in Sec. 6) is* CPAP secure *in the attack model $\mathcal{M} = (\mathcal{F}^f, \langle \mathcal{F}^f \rangle, v_{\mathtt{A}})$.*

To prove this theorem we rely on the observation that $\Pi^{\texttt{HT}}$ ($\Pi^{\texttt{HT-FBS}}$) is collusion-preserving for $\mathcal{F}^f$ as long as nobody aborts. That is, the only way to do subliminal communication in $\Pi^{\texttt{HT}}$ (and in $\Pi^{\texttt{HT-FBS}}$) is by sending a message which is incompatible with the protocol description thus causing the honest parties to abort. Given the way we have set the payoffs, it is always inconvenient for the adversary to trigger the event $E_{\texttt{collude}}$ (which allows subliminal communication). We refer the reader to App. A.3 for the formal proof. We observe that in the case where we also consider the utility of the designer we can prove that our protocol is CPIC (according to Def. 6) in the case there the utilities of the designer are symmetric to the utility of the adversary.

9 How to disincentivize aborting strategy concretely.

The goal of this section is to translate the utilities defined in Section 7 to concrete, monetary values. Recall that in Section 5 and Section 6 we obtained a protocols $\Pi^{\texttt{HT}}$ and $\Pi^{\texttt{HT-FBS}}$ respectively, which achieve CP under the assumptions of honest tokens and that adversarial parties do not make the protocol abort. We proved that these protocols achieve CPAP—informally, CP against rational attackers—assuming the penalties to the attacker when adversarial parties are caught aborting, outweigh the benefits of collusion. In this section, we present penalization schemes against aborting parties, taking the (theoretical) penalties to a real, financial punishment. The first solution makes no assumption on the protocol being run, whereas the second requires publicly identifiable abort, i.e., that the correctness of protocol messages are verifiable by an external entity.

Our first simple solution $\Pi^{\text{simple}}_{\text{penalize}}$ assumes the existence of a functionality (Figure 7) which requires all players to deposit collateral, for example on the blockchain to a smart contract, prior to the start of the game. We remark that this blockchain need not be collusion-preserving, but we require that parties do not access it during protocol execution. During the offline protocol execution, the parties engage in $\Pi^{\texttt{HT}}$ or $\Pi^{\texttt{HT-FBS}}$, and are instructed to query the functionality again to reclaim the collateral only if the protocol ends without abort. Parties can only receive the deposits back if every party in the protocol sends `RECLAIM` to the functionality. Intuitively, this solution disincentivizes misbehavior assuming one honest party. Adversarial parties know that if they abort, then the honest party will never reclaim, causing everyone to lose their collateral. Thus, a rational attacker would not misbehave and honest parties would not need to lose their collateral, making this protocol CPAP.

Below, we state formally the concrete utilities implemented by the simple penalization scheme.

Theorem 5. *The n-party protocol $\Pi^{simple}_{penalize}$ with deposit amount d gives concrete negative payoff of $\gamma_{\texttt{abort}} = -td$ to the attacker and negative payoff $-(n-t)d$ to the protocol designer in the event $E_{\texttt{abort}}$.*

By Theorem 4, we obtain CPAP by setting the deposit amount d such that $td > \gamma_{\texttt{collude}}$ (where $\gamma_{\texttt{collude}}$ is the payoff to the attacker in the event he is able to collude). However, this solution is not incentive compatible (i.e., CPIC (Definition 6)) when the designer also cares about the honest parties' deposits. He has incentive to deviate from the protocol, as he loses utility in the event $E_{\texttt{abort}}$ since all parties, including honest ones, are punished for one party's misbehavior. He improves his utility by deviating from the penalization scheme and allowing everyone to reclaim their collateral, even when honest parties detect collusion.

The functionality interacts with a set of parties $\mathcal{P} = \{p_1, \ldots, p_{n'}\}$. It maintains a dictionary PSet of tuples $(\mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}})$ (initialized as $\emptyset$), indexed by pid. $\mathsf{Active}_{\mathsf{pid}}$ is the subset of $\mathcal{P}$ who have deposited money to run protocol instance pid. We denote $\mathsf{PSet.pid}$ as the tuple $(\Pi, \mathsf{Active}_{\mathsf{pid}})$ corresponding to pid, or equal to $\emptyset$ if pid is not found in PSet. The functionality is parameterized by a deposit amount d, a tuple of initial balances of each party, $\mathsf{Balance} = (b_1, \ldots, b_{n'})$ For each honest party p_i we assume that their initial balance is greater than the deposit(s) they will make.

- Upon receiving $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active})$ from party p_i: If $\mathsf{PSet.pid} = \emptyset$, Π is a n-party protocol, and for each $p_i \in \mathsf{Active}$, $b_i \geq d$:
    1. Wait: If all $p_i \in \mathsf{Active}$ have sent $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active})$, continue.
    2. Deposit collateral for each party: for each party $p_i \in \mathsf{Active}$: $b_i \leftarrow b_i - d$
    3. $\mathsf{PSet} \leftarrow \mathsf{PSet} \cup (\mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}} = \mathsf{Active})$
    4. Send the message $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}})$ to each party $p_i \in \mathsf{Active}$.
- Upon receiving $(\texttt{RECLAIM}, \mathsf{sid}, \mathsf{pid})$ from party $p_i \in \mathsf{Active}_{\mathsf{pid}}$:
    1. Wait to receive the same message $(\texttt{RECLAIM}, \mathsf{sid}, \mathsf{pid})$ from each $p_i \in \mathsf{Active}_{\mathsf{pid}}$. If received this message from all $p_i \in \mathsf{Active}_{\mathsf{pid}}$, continue to return deposits to all parties.
    2. For each $p_i \in \mathsf{Active}_{\mathsf{pid}}$, return the deposit to p_i: $b_i \leftarrow b_i + d$.
    3. $\mathsf{Active}_{\mathsf{pid}} \leftarrow \mathsf{Active}_{\mathsf{pid}} \backslash \{p_i\}$, $\mathsf{PSet} \leftarrow \mathsf{PSet} \backslash \{(\mathsf{pid}, \mathsf{PSet.pid})\}$

Figure 7: $\mathcal{F}^{\text{simple}}_{\text{penalize}}$

9.1 Penalization functionality

The solution above, while simple, is not incentive compatible (CPIC), as it requires the honest party to complain about misbehavior even at the cost of his own collateral being lost. We describe below a penalization functionality $\mathcal{F}_{\text{penalize}}$ that only punishes misbehaving parties.

Informally, our penalization functionality disincentivizes misbehavior by allowing parties to run a protocol only if all parties have deposited a collateral. Throughout the protocol, it keeps track of signed protocol messages. A party can reclaim their collateral if they have behaved correctly throughout protocol execution; otherwise, their collateral is split among the other parties. Specifically, the penalization functionality $\mathcal{F}_{\text{penalize}}$ (Fig. 8) allows parties to run multiple protocol instances (simultaneously) and penalize aborting behavior while compensating honest parties. In an ideal execution, before the protocol begins, parties register the protocol to be run and deposit collateral via `INITIALIZE`. Once all parties have deposited collateral for an initialized protocol instance, the parties can begin the protocol. The functionality keeps track of all messages sent during the execution of each protocol instance (via the parties submitting `PROTOCOL` messages to the functionality). A party can reclaim their collateral by sending message `RECLAIM`, if the protocol instance has ended and the party has behaved correctly—which can be verified for protocols whose messages that are secure with publicly identifiable abort (e.g. $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$).

We remark that $\mathcal{F}_{\text{penalize}}$ can be implemented using a blockchain, for example following the approach of [KZZ16]. There, parties make deposits on the blockchain before running an MPC protocol, and can only reclaim their deposits if they submit correct messages to the blockchain at each round. In particular, if the MPC protocol has constant number of rounds, [KZZ16] only requires a constant number of ledger rounds. We note that in such a case, parties currently engaged in the protocol (e.g. a poker game) should not be able to use the blockchain for any other purpose. That is, at each round, a party can only decide whether to send a message for a specific protocol, or not send any messages at all. Under this assumption, no party can misbehave in any way (including sending subliminal messages) without being penalized. This restriction can be achieved even when the blockchain is not collusion-preserving, assuming parties in the MPC protocol only have black box access to the blockchain (for example, the parties involved in the MPC are different from nodes in the blockchain network), and the blockchain is only used for protocol penalization.

9.2 Protocol with compensation/penalization.

We describe Π_{penalize} (Fig. 8), which uses $\mathcal{F}_{\text{penalize}}$ as penalization functionality. We denote by Π the n-party CP-protocol with publicly identifiable abort and pid the protocol instance's ID.

Initialize protocol and deposit collateral. Each party p_i takes as input a protocol Π, time the protocol starts τ, time for each round r, instance ID pid (which p_i wishes to run), and the set of parties running the protocol instance $\mathsf{Active}_{\mathsf{pid}}$.

1. Each party $p_i \in \mathsf{Active}_{\mathsf{pid}}$ submits the message $(\mathtt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}, \tau, r)$.
2. If p_i receives $(\mathtt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}}, \tau, r)$ from the functionality, continue. Otherwise, stop execution.

Protocol execution. Let pid be the ID of a protocol instance that is currently being executed. For each round where p_i should send a message m in the protocol, submit $(\mathtt{PROTOCOL}, \mathsf{sid}, \mathsf{pid}, p_i, m)$ to $\mathcal{F}_{\text{penalize}}$.

Reclaim collateral. Let pid be the ID of a terminated protocol instance. p_i submits message $(\mathtt{RECLAIM}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{F}_{\text{penalize}}$.

The above protocol, in contrast to $\Pi^{\text{simple}}_{\text{penalize}}$, results in positive payoff to the protocol designer when honest parties punish misbehavior (E_{abort}). That is, Π_{penalize} is compatible with the designer's incentives.

The functionality interacts with a clock functionality $\mathcal{G}_{\texttt{clock}}$, and with a set of parties $\mathcal{P} = \{p_1, \ldots, p_{n'}\}$. It maintains a dictionary PSet of tuples $(\mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}}, \mathsf{PState}_{\mathsf{pid}}, \tau, r)$ (initialized as $\emptyset$), indexed by pid. $\mathsf{Active}_{\mathsf{pid}}$ is the subset of $\mathcal{P}$ who have deposited money to run protocol instance pid, $\mathsf{PState}_{\mathsf{pid}}$ is the protocol state (initialized as $\emptyset$) which keeps track of all protocol messages $(\texttt{PROTOCOL}, \cdots)$ sent by parties in $\mathsf{Active}_{\mathsf{pid}}$, τ is the time which the protocol instance begins, and r is the time for each round in the protocol instance. We denote $\mathsf{PSet.pid}$ as the tuple $(\Pi, \mathsf{Active}_{\mathsf{pid}}, \mathsf{PState}_{\mathsf{pid}})$ corresponding to pid, or equal to $\emptyset$ if pid is not found in PSet.

The functionality is parameterized by a deposit amount d, and a tuple of initial balances of each party, $\mathsf{Balance} = (b_1, \ldots, b_{n'})$. For each honest party p_i we assume that their initial balance is greater than the deposit(s) they will make. We assume that each protocol Π accepted by $\texttt{INITIALIZE}$ has the property that the correctness of the protocol messages is publicly verifiable

- Upon receiving $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}, \tau, r)$ from party p_i: If $\mathsf{PSet.pid} = \emptyset$, Π is a n-party protocol, and for each $p_i \in \mathsf{Active}$, $b_i \geq d$:
    1. Wait: If all parties $p_i \in \mathsf{Active}$ send $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}, \tau, r)$ by time τ, continue.
    2. Deposit collateral for each party: for each party $p_i \in \mathsf{Active}$: $b_i \leftarrow b_i - d$
    3. Send message $(\texttt{CLOCK-READ}, \mathsf{sid})$ to $\mathcal{G}_{\texttt{clock}}$ and receive $(\texttt{CLOCK-READ}, \mathsf{sid}, \tau)$.
    4. $\mathsf{PSet} \leftarrow \mathsf{PSet} \cup (\mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}} = \mathsf{Active}, \mathsf{PState}_{\mathsf{pid}} = \emptyset, \tau, r)$
    5. Send $(\texttt{INITIALIZE}, \mathsf{sid}, \mathsf{pid}, \Pi, \mathsf{Active}_{\mathsf{pid}}, \tau, r)$ to each party $p_i \in \mathsf{Active}$.
- Upon receiving $(\texttt{PROTOCOL}, \mathsf{sid}, \mathsf{pid}, p_i, m)$ from party $p_i \in \mathsf{Active}_{\mathsf{pid}}$:
    1. Send $(\texttt{CLOCK-READ}, \mathsf{sid})$ to the clock $\mathcal{G}_{\texttt{clock}}$, and receive $(\texttt{CLOCK-READ}, \mathsf{sid}, \tau)$.
    2. $\mathsf{PState}_{\mathsf{pid}} \leftarrow \mathsf{PState}_{\mathsf{pid}} \mid\mid (\texttt{PROTOCOL}, \mathsf{sid}, \mathsf{pid}, p_i, m, \tau)$
- Upon receiving $(\texttt{RECLAIM}, \mathsf{sid}, \mathsf{pid})$ from party $p \in \mathsf{Active}_{\mathsf{pid}}$, check PState to see if the protocol has terminated. If so, for each $p_i \in \mathsf{Active}_{\mathsf{pid}}$, check if p_i behaved correctly during the protocol[a]:
    1. If p_i behaved correctly, return the deposit to p_i: $b_i \leftarrow b_i + d$.
    2. Otherwise, p_i's collateral is split as compensation for other parties. For each $p_j \in \mathsf{Active}_{\mathsf{pid}}$, $p_j \neq p_i$: $b_j \leftarrow b_j + \left\lfloor \frac{d}{n-1} \right\rfloor$
    3. $\mathsf{Active}_{\mathsf{pid}} \leftarrow \mathsf{Active}_{\mathsf{pid}} \setminus \{p_i\}$
    4. $\mathsf{PSet} \leftarrow \mathsf{PSet} \setminus \{(\mathsf{pid}, \mathsf{PSet.pid})\}$

[a]This can be checked as Π's messages are publicly verifiable and time-stamped so that we can check whether they are sent at the correct round using τ and r

Figure 8: $\mathcal{F}_{\text{penalize}}$

Theorem 6. *The n-party protocol $\Pi_{penalize}$ with deposit amount d gives concrete negative payoff of $\gamma_{\texttt{abort}} = -d$ to the attacker and positive payoff $\frac{d}{n-1}$ to the protocol designer in the event $E_{\texttt{abort}}$.*

10 Acknowledgments

Michele Ciampi is supported by H2020 project PRIVILEDGE #780477.

References

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[AKL$^+$09] Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, abhi shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 524–540, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[AKMZ12] Joël Alwen, Jonathan Katz, Ueli Maurer, and Vassilis Zikas. Collusion-preserving computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 124–143, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[AOZZ15] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multiparty computation and universally composable receipt-free voting. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 763–780, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AsV08] Joël Alwen, abhi shelat, and Ivan Visconti. Collusion-free protocols in the mediated model. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 497–514, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[BGM$^+$18] Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 34–65, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[BKM17] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 410–440, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `http://eprint.iacr.org/2003/239`.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 597–608, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[GB96] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. *Summer course Cryptography and computer security at MIT*, 1999:1999, 1996.

[GKM$^+$13] Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th Annual Symposium on Foundations of Computer Science*, pages 648–657, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 483–500, 2006.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[KB14] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 30–41, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 418–429, Vienna, Austria, October 24–28, 2016. ACM Press.

[KMB15] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 195–206, Denver, CO, USA, October 12–16, 2015. ACM Press.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 705–734, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[LMPs04] Matt Lepinski, Silvio Micali, Chris Peikert, and abhi shelat. Completely fair sfe and coalition-safe cheap talk. In Soma Chaudhuri and Shay Kutten, editors, *23rd ACM Symposium Annual on Principles of Distributed Computing*, pages 1–10, St. John's, Newfoundland, Canada, July 25–28, 2004. Association for Computing Machinery.

[LMs05] Matt Lepinski, Silvio Micali, and abhi shelat. Collusion-free protocols. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 543–552, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[PR18] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 425–458, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 260–289, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

A Security proofs

A.1 Proof of Theorem 1

Proof. We start the proof by assuming that at least one party is honest. In order to prove this part of the theorem we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\mathtt{HT}},\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$$

For $i = 1, \ldots n$, the simulator $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A}_i)$ queries$(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If it is, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates the string $\widetilde{K}$, n couples of signing-verification keys, and authenticates the verification keys using the master secret key msk (as shown in Fig 1) thus obtaining $((\mathsf{sigk}_1, \mathsf{vk}_1, \mathsf{cert}_1), \ldots, (\mathsf{sigk}_n, \mathsf{vk}_n, \mathsf{cert}_n), \widetilde{K})$. $\mathcal{F}^\star$ then sends $T^{\mathtt{HT}}$ this information to $\mathcal{F}^\star$ which forwards them to the simulator $\mathcal{S}_i$.

$\mathcal{S}_i$ then computes $\mathsf{PRG}(\widetilde{K}) = R$, parses R as $r_1||\ldots||r_{n-1}||r'_1||\ldots r'_{n-1}$ and computes $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}_n, 0^\lambda; r_j)$, $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, e_j)$ for all $j \in \{1, \ldots, n-1\}$. In addition, the simulator computes the value $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ where for $j = 1, \ldots, n-1$, $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, 0^\lambda; r'_j)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, \overline{y}_1||\ldots||\overline{y}_{n-1}||f||\mathsf{sid})$. From here onwards the behaviors of the simulators differ. Without loss of generality we describe how the simulator $\mathcal{S}_1$ works since $\mathcal{S}_i$ with $i \in \{2, \ldots, n-1\}$ will follow exactly the same strategy as $\mathcal{S}_1$. We then show how the simulator $\mathcal{S}_n$ works.

$\mathcal{S}_1$**.** The simulator $\mathcal{S}_1$ internally runs the adversary $\mathcal{A}_1$, emulates the token functionality $T^{\mathtt{HT}}$ for $\mathcal{A}_1$ for the session id sid and acts on the behalf of the parties $p_2, \ldots, p_n$. $\mathcal{S}_1$ executes the following steps.

1. Send $(e_2, f, \mathsf{vk}_2, \sigma_2, \mathsf{cert}_2), \ldots, (e_{n-1}, f, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1})$ to $\mathcal{A}_1$.
2. If $\mathcal{A}_1$ sends $I = (\mathsf{Input}, \mathsf{sid}, x, f')$ to $T^{\mathtt{HT}}$ then do the following.
    - If $\mathsf{ctr}_1^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_1^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
    - Output $(e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1)$
3. If $\mathcal{A}_1$ sends $(e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1)$ over broadcast then send (sid, x_1) to $\mathcal{F}$.
4. Upon receiving y_1 from $\mathcal{F}$ send $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to $\mathcal{A}_1$.

5. Upon receiving $(\mathsf{Output}, \mathsf{sid}, z)$ from p_1, if $z = (\overline{y}_1, \dots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ then send y_1 to $\mathcal{A}_1$, output $\perp$ otherwise.

$\mathcal{S}_n$. The simulator $\mathcal{S}_n$ internally runs the adversary $p_n^\star$, emulates the token functionality T^{HT} for $p_n^\star$ for the session id sid and acts on the behalf of the parties $p_1, \dots, p_{n-1}$. $\mathcal{S}_n$ executes the following steps.

1. Send $((e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \dots, (e_{n-1}, f, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1}))$ to $p_n^\star$.
2. If $I = (\mathsf{Input}, \mathsf{sid}, x_n, f', (e'_1, f_1, \mathsf{vk}'_1, \sigma'_1, \mathsf{cert}'_1), \dots, (e'_{n-1}, f_{n-1}, \mathsf{vk}_{n-1}, \sigma'_{n-1}, \mathsf{cert}'_{n-1}))$ is received from $p_n^\star$, then check if for $j = 1, \dots, n-1$: $e'_j = e_j$, $\sigma'_j = \sigma_j$, $\mathsf{vk}_j = \mathsf{vk}'_j$ and $\mathsf{cert}_i = \mathsf{cert}'_i$. If it is not, then output $\perp$ and stop, otherwise execute the following steps.
   - If $\mathsf{ctr}_n^{\mathsf{sid}}$ is not defined then defined it and set $\mathsf{ctr}_n^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
   - Send x_n to $\mathcal{F}$ and upon receiving y_n from $\mathcal{F}$ send $(\overline{y}_1, \dots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to $p_n^\star$.

The main differences from the real world are that the adversarial parties see dummy encryptions instead of the encryptions. We note that the simulation strongly relies on the fact that the adversaries cannot forge the signatures output of the hardware token functionalities. Indeed, by forging a signature an adversary could: 1) change the input of another party, or 2) use the inputs of the honest parties to evaluate a functions $f' \neq f$ or 3) evaluate the same function multiple times on the same honest parties inputs by changing the value sid thus completely breaking the security of the protocol.

In the case that all the parties are corrupted then we rely on the fact that $\mathcal{F}$ is CP-well-formed functionality and we allow the adversarial parties to communicate freely via $\mathcal{F}$. More precisely,

1. Whenever a message m is received on the ith adversarial interface, $\mathcal{F}$ outputs (i, m) to the first adversarial interface.

2. Whenever a message of the form (i, m) is received on the first adversarial interface, $\mathcal{F}$ outputs the message m to the ith adversarial interface.

□

A.2 Proof of Lemmata 1 and 2

A.2.1 Proof of Lemma 1.

We start the proof by assuming that at least one party is honest. In order to prove this part of the theorem we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \dots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}}, \mathcal{R}}_{\Pi^{\mathsf{HT\text{-}FBS}}, \mathcal{A}, \mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}}, \mathcal{F}}_{\phi, \mathsf{Sim}(\mathcal{A}), \mathcal{Z}}$$

By assumption on the security of the MPC protocol, we know that $\forall \mathcal{A}\ \exists \mathsf{S}\ \forall \mathcal{Z}$ such that $\text{EXEC}^{\bar{\mathcal{G}}, \mathcal{B}}_{\Pi^{\mathsf{MPC}}, \mathcal{A}, \mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}}, \mathcal{F}}_{\mathsf{S}, \mathcal{Z}}$.

We consider now S for the case where there are no corrupted parties and describe how $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A})_i$ works for $i = 1, \dots n$. Without loss of generality we formally describe how the simulator $\mathcal{S}_1$ works since the other simulators follow exactly the same strategy.

The simulator $\mathcal{S}_1$ queries $(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If it is, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\texttt{HT-FBS}}$. $T^{\texttt{HT-FBS}}$ then generates the string $\widetilde{K}$, n couples of signing-verification keys, and authenticates the verification keys using the master secret key msk (as shown in Fig 4) thus obtaining $((\mathsf{sigk}_1, \mathsf{vk}_1, \mathsf{cert}_1), \ldots, (\mathsf{sigk}_n, \mathsf{vk}_n, \mathsf{cert}_n), \widetilde{K})$. $T^{\texttt{HT-FBS}}$ sends this information to $\mathcal{F}^\star$ which forwards them to the simulator $\mathcal{S}_1$. $\mathcal{S}_1$ then computes $\mathsf{PRG}(\widetilde{K}) = R$, parses R as $\rho||r_1^1||\ldots||r_1^{m+1}||\ldots||r_n^1||\ldots||r_n^{m+1}$ and uses ρ to run the simulator of the MPC, S. Let $\{\mathtt{msg}_j^\ell\}_{j\in[n],\ell\in[m]}$ be the messages contained in the transcript obtained by the MPC simulator S. For all $j \in [n]$ and $\ell \in [m]$ computes $\sigma_j^\ell \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}_j^\ell||\Pi^{\mathsf{MPC}}||\ell; r_j^\ell)$. Then $\mathcal{S}_1$ executes the following steps.

- Send $(\mathtt{msg}_2^\ell, \mathsf{vk}_2, \sigma_2^\ell, \mathsf{cert}_2), \ldots, (\mathtt{msg}_n^\ell, \mathsf{vk}_n, \sigma_n^\ell, \mathsf{cert}_n)$ to $\mathcal{A}_1$.
- If $I = (\mathsf{Input}, \mathsf{sid}, x_1, R_1, \Pi^{\mathsf{MPC}})$ is received from $\mathcal{A}_1$ then do the following.
  - If $\mathsf{ctr}^{\mathsf{sid}} = 0$ then $\mathsf{ctr}^{\mathsf{sid}'} \leftarrow 1$ otherwise output $\perp$ and stop.
  - Set $\mathsf{x} \leftarrow x_1$, $\mathtt{l} \leftarrow 1$ and send $(\mathtt{msg}_1^1, \Pi^{\mathsf{MPC}}, \mathsf{vk}_1, \sigma_1^1, \mathsf{cert}_1)$ to $\mathcal{A}_1$.

- If $I = (\mathsf{NextMsg}, \mathsf{sid}, \{\mathtt{msg}_i^{\ell'}, \sigma_i^{\ell'}, \mathsf{vk}_i{}', \mathsf{cert}_i{}'\}_{i\in[n]})$ is received then do the following.
  - If $\ell \neq \mathtt{l}$ or $\mathtt{l} > m$ output $\perp$ and stop, otherwise continue with the following steps.
  - For all $j \in [n]$ if $\mathtt{msg}_j^\ell \neq \mathtt{msg}_j^{\ell'}$ or $\sigma_j^\ell \neq \sigma_j^{\ell'}$ or $\mathsf{cert}_i{}' \neq \mathsf{cert}_i$ or $\mathsf{vk}_i{}' \neq \mathsf{vk}_i$ then output $(\perp, p_i)$ and stop.
  - Set $\mathtt{l} \leftarrow \mathtt{l} + 1$.
  - If $\ell \leq m$ then send $(\mathtt{msg}_1^1, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_1^1)$ to $\mathcal{A}_1$.
  - If $\ell = m + 1$ then send x to $\mathcal{F}$. Upon receiving y_1 from $\mathcal{F}$ send y_1 to $\mathcal{A}_1$.

The proof relies on the observation that (unless the adversary breaks the security of the strong signature scheme), then the adversary just acts as an observer on the channel. That is, the corrupted party $\mathcal{A}_1$ can only inspect the messages generated by $T_1^{\texttt{HT-FBS}}$ and the messages received on the channel which are honestly generated using $T^{\texttt{HT-FBS}}$.[7] For this reason $\mathcal{S}_1, \ldots, \mathcal{S}_n$ can run a simulator S of Π^{MPC} that works when there are no corrupted parties. We recall that the behavior of the corrupted parties cannot influence the output of S as long as the signature scheme is secure.

A.2.2 Proof of Lemma 2.

This proof follows immediately from the GUC security of Π^{MPC}. Indeed, we note that the token functionalities simply run Π^{MPC} even in the case that a unsigned message is received (or a message is not received at all). This also enables identifiable abort (unanimous abort) if Π^{MPC} enables it.

[7]If that is not the case then either the protocol would abort, or a reduction to the security of the signature scheme can be done.

A.3 Proof of Theorem 4

Proof. In the case that all the parties are malicious then we rely on the fact that $\langle \mathcal{F}^f \rangle$ is CP-well-formed functionality and we allow the adversarial parties to communicate freely via $\langle \mathcal{F}^f \rangle$ following the same approach proposed in the proof of Theorem 1.

In the case that at least one party is honest then we need to show collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\mathtt{HT}},\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\langle \mathcal{F}^f \rangle}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$$

The simulators are equal to the simulators showed in the proof of Theorem 1 except for the following details. If $\mathcal{S}_i$ receives an invalid message m from $p_i^\star$ (i.e. a message that would yield an honest party to abort) then $\mathcal{S}_i$ sends $(\mathtt{COLLUDE}, \mathsf{sid}, m)$ to $\langle \mathcal{F}^f \rangle$ and stops. This situation captures the ability of the corrupted parties to interact with each others at the price of being detected by the honest parties. Indeed, we recall that $\Pi^{\mathtt{HT}}$ enjoys the identifiable abort property. The payoff we have defined ensures that an adversary is never incentivized to abort (i.e. break the CP property). Given that we have proved in Theorem 1 that the if no party aborts then $\Pi^{\mathtt{HT}}$ is collusion-preserving then we can claim that $\Pi^{\mathtt{HT}}$ is *CPAP*. The same arguments can be applied to $\Pi^{\mathtt{HT\text{-}FBS}}$.

□

B Impossibility results in the mediated model

We motivate our study of collusion-preservation using the broadcast channel, by proving that desirable properties—robustness, fairness, and identifiable abort—are impossible in the mediated model when the mediator may be corrupt. We recall that in the mediated model, every party communicates only to a central node called the mediator.

Intuitively, these impossibilities stem from the mediator being able to cut off communication between itself and any party, at any time. Robustness is simple to show to be impossible, since the mediator can simply stop all communication. To break fairness, the mediator can end communication in the protocol after one party receives his output and before another party receives his. This strategy can be used also to break unanimous abort and allow one honest party to receive his output while others do not. Lastly, identifiability is not possible as a corrupt mediator can "frame" an honest party as having aborted, by simply ignoring messages from this party. Since other parties only communicate through the mediator, they cannot identify whether the party or the mediator has misbehaved.

We state and prove these impossibilities formally in the theorems below.

Theorem 7. *Robustness is not possible when the mediator may be corrupted. This holds even when all other parties are honest.*

Proof. The mediator simply does nothing, and thus nothing can be computed. □

Theorem 8. *Consider a protocol in which each party interacts with the mediator a number of times that is polynomial in the security parameter. Then fairness and unanimous abort are not possible when the mediator may be corrupted, unless the output can be computed without the parties' inputs. This holds even if all other parties are honest.*

Proof. We will prove that fairness and unanimous abort are impossible. Consider three parties, honest parties Alice (A) and Bob (B), and the corrupt mediator (M) who has no inputs and some constant as output (e.g., 1). Since the mediator controls when messages are sent during the protocol, we consider the following protocol format: In odd-numbered interactions, A interacts with M; in even-numbered interactions, B interacts with M.

M's attack is the following. M guesses an interaction number in which only one of A or B has the output (this is always possible since A and B never receive messages at the same time), and aborts at the beginning of the interaction, depriving the other party of the output. Since there is a polynomial number of choices, M succeeds with non-negligible probability.

Formally: Suppose the final interaction in the protocol is between A and M. Assuming that fairness is possible, we show that if this protocol has fairness, then A and B can compute their outputs without sending any messages.

Since the final interaction does not involve B, B must have known the output prior to the final interaction. Suppose the corrupt M chooses to abort in the final interaction (Since the number of rounds is polynomial, he chooses to abort at this round with non-negligible probability). By fairness (since B knows the output without the final message), A must also be able to learn the output without the final interaction. This means both A and B learn the output without the final interaction.

Similarly, the second-to-last interaction (between B and M) does not involve A, so A must have known the output prior to this interaction. By fairness, if M chooses to abort in the second-to-last interaction, B must also be able to learn the output without this interaction. Thus both A and B learn the output without the second-to-last round.

Continue the argument for the number of interactions, and A and B both learn the output without sending any messages. □

The proof does not work when the number of interactions is exponential. This is because the mediator will only guess correctly with negligible probability, which interaction to abort at (we allow negligible failure of the fairness property).

Theorem 9. *Identifiability is not possible in the mediated model when the mediator may be corrupted, regardless of the number of honest parties.*

Proof. We show that it is impossible to distinguish between a case where the mediator is corrupt, or some other party A is corrupt. Suppose a corrupt mediator M wishes to simulate a party A aborting at round r. M follows the protocol until round $r-1$. At rounds $\geq r$, M simply ignores (does not send nor receive from) A, and does whatever it is supposed to do when A aborts. Since all parties communicate only via M, the messages they receive in this scenario are exactly the same as if the mediator were honest, but a corrupt party A has followed the protocol before round r and stops sending messages after round r. Thus, they cannot correctly identify whether M or A is corrupt in the case of an abort. □

C Motivation for using compensation paradigm for disincentivizing aborts

In this work we observe that the notions of collusion-preservation (and so collusion-freeness) do not capture all the possible attacks that an adversary can do in order to send subliminal message.

Indeed and adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ could adapt the following strategy. $\mathcal{A}_1$ aborts any time that his input has the last two bits different from 00. Clearly, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ are isolated, in the case that the protocol does not abort $\mathcal{A}_2$ has some information about $\mathcal{A}_1$'s input that the honest parties do not have. This attack becomes more interesting in the case of reactive functionalities, where the parties can get intermediate outputs. As a concrete example of reactive functionality we consider the game of poker. We can think to use a CP protocol to shuffle a deck of cards and let the parties play. An adversary $\mathcal{A}$ could have a strategy in which $\mathcal{A}_1$ aborts if he does not have an ace and a king in the second hand, and continues the game otherwise. It should be easy to see that with high probability $\mathcal{A}_1$ will be able to communicate more than one bit of information to $\mathcal{A}$. We note that this issue arises only because the CP definition allows the parties to abort, which is in general an unavoidable attack.

D Hardware token

Our protocols assume that each party has access to a stateful hardware token which might perform fresh encryptions and signatures with respect to some hidden keys. Moreover, an adversary that has physical access to the HT can only query it and get the result back, without having the possibility to inspect the intermediate steps of the computation or tamper with the token. A similar assumption is used in [AOZZ15] to generate randomness hidden from the adversary, in order to obtain receipt-freeness. The work of [PST17] provides a formal and composable abstraction of the Intel SGX hardware token, which has all the features we have just described. In this work, following [AOZZ15], we abstract the hardware token by means of an ideal functionality that has a secret state (the secret keys), a public state (the public key and the signature verification key) and code. All the outputs of the HT are signed together with the code that has been run to generate the output. This process is called *attestation* and assures parties holding the verification key of the HT that the output has been generated following a specified code. In order to simplify our token functionality description we omit the signature of the code. We note that by assuming the existence of hardware tokens we are also implicitly assuming a setup assumption that is stronger than a public key infrastructure (PKI). Indeed, we require that all parties running our protocol know each other hardware token public keys and that those public keys are generated honestly. However, we note this approach is still meaningful in the context of collusion-freeness/collusion-preservation since the hardware token (and the public keys) are generated before inputs are given to the parties and even before the function being computed is agreed upon.

E Mediated model vs hardware token

A natural question is whether trusted hardware and non-aborting adversaries are too strong as assumptions. We answer this in the negative. We show that (stateful) hardware tokens are insufficient to obtain CP when the communication resource is a broadcast channel and the parties are allowed to abort. Hence, this implies that the assumption that hardware tokens exist is weaker than assuming that a mediator exists. Conversely, we show that the an honest mediator can be used to obtain an hardware token.

To prove the first implication (that HT and broadcast are weaker than a mediator in the star topology) we just observe that an adversary can send any message over the channel without filter.

This means that nothing prevents an adversary $\mathcal{A}_1$ from sending an encrypted message to another adversarial party $\mathcal{A}_2$ using the public key of $\mathcal{A}_2$.

To prove the other implication we simply observe that an honest mediator can run any code, hence, he can also run the code of a hardware token. Since the mediator is assumed to be honest, then we implicitly have a proof that a computation made by the mediator has been done correctly. In addition, the mediator is a party that can have secret information that are not accessible to anybody else (since the adversary does not have the power to corrupt it), exactly like a hardware token.

F A note on games with short strategy description

In [LMs05] it is observed that it would be trivial to avoid subliminal communication in a model in which players commit to their strategy (when the strategy can be described in a short way) and then run a secure function evaluation on these commitments. Indeed, once the strategy of a game is committed in a collusion-preserving manner, if the parties are not allowed to abort, then any subliminal communication between the malicious parties cannot harm the outcome of the game. The situation is more complicated in the case that the malicious parties are allowed to abort since such a behaviour could cause the game to stop. The aborts can be prevented by simply using the approach proposed in [KZZ16]. In this the authors show how to promote a protocol Π that securely evaluate the function f with publicly identifiable abort into a "robust" protocol Π'. That is, either Π' terminates and the honest parties get the output, or the parties that were behaving honestly get a monetary compensation via a blockchain.

To obtain CP for games with short strategy description we simply let the parties to commit to their strategies and their inputs using a collusion-preserving protocol that allows the parties to abort (like our protocol). And if the commitment phase has been successful then we run the protocol Π' as described above using as input the committed values.

G A note on correlated equilibria

In strategic games, players can achieve correlated equilibrium by observing the same public signal. In our construction, this public signal can be the setup (e.g. public keys) or the broadcasted protocol messages. Similarly, the original CP construction by Alwen et al. introduces a public signal as a GUC-complete setup (e.g., ACRS) is assumed [Theorem 5.1, [AKMZ12]]. Indeed, this introduces additional correlated equilibria to the ideal game; however, collusion preservation ensures that no meaningful information (about the input or output) can be communicated using the additional correlation.

H Formal definitions

Definition 7 (Strong unforgeable signature scheme). *A triple of* PPT *algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is called a* signature scheme *if it satisfies the following properties.*

Completeness: *For every pair* $(s, v) \xleftarrow{\$} \mathsf{Kgen}(1^\lambda)$, *and every* $m \in \{0,1\}^\lambda$, *we have that* $\Pr[\mathsf{Ver}(v, m, \mathsf{Sign}(s, m)) = 0] < \nu(\lambda)$.

Consistency (non-repudiation): *For any m, the probability that $\mathsf{Kgen}(1^\lambda)$ generates (s, v) and $\mathsf{Ver}(v, m, \sigma)$ generates two different outputs in two independent invocations is smaller than $\nu(\lambda)$.*

(Strong) Unforgeability: *For every PPT $\mathcal{A}$, there exists a negligible function ν, such that for all auxiliary input $z \in \{0,1\}^\star$ it holds that:*

$$\Pr[(s, v) \overset{\$}{\leftarrow} \mathsf{Kgen}(1^\lambda); (m, \sigma) \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{Sign}(s,\cdot)}(z, v) \wedge \\ \mathsf{Ver}(v, m, \sigma) = 1 \wedge (m, \sigma) \notin Q] < \nu(\lambda)$$

where Q denotes the set of the couples message-signature $\{(m_i, \sigma_i)\}_{i \in \lambda}$ where m_i is requested by $\mathcal{A}$ to the oracle $\mathsf{Sign}(s, \cdot)$ which returns $\sigma_i \overset{\$}{\leftarrow} \mathsf{Sign}(s, m_i)$ for all $i \in \{1, \ldots, \lambda\}$.

In this paper we also make use of the UC-signature functionality proposed in [Can03] that we denote with $\mathcal{F}_{\mathtt{SIGN}}$. We also use the the fact that a scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ that satisfies Def. 7 can be turned into a scheme that UC-realized the functionality $\mathcal{F}_{\mathtt{SIGN}}$ [Can03, Thm 2]. We assume familiarity with $\mathcal{F}_{\mathtt{SIGN}}$, and for more discussion on this functionality we refer the reader to Sec. I.1 Fig. 9.

Definition 8 (CPA-secure Symmetric Encryption Scheme (from notes of [GB96], Definition 6.8))**.** *A triple of* PPT *algorithms* $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is called a* chosen-plaintext-attack-secure symmetric encryption scheme *if it satisfies the following properties.*

Completeness: *For every secret key $s \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$, and every $m \in \{0,1\}^\lambda$, we have that* $\Pr[\mathsf{Dec}(s, \mathsf{Enc}(s, m)) = m] = 1$.

CPA-Security: *Let the left-or-right encryption oracle be as follows, where $b \in \{0,1\}$, $m_0, m_1 \in \{0,1\}^\lambda$:*

```
Oracle $\mathsf{Enc}(\mathsf{LR}(m_0, m_1, b))$:
    if $|m_0| \neq |m_1|$ then return $\perp$
    $c \overset{\$}{\leftarrow} \mathsf{Enc}(m_b)$
    return $c$
```

Let $\mathcal{A}$ be an adversary. We consider the following experiments:

```
Experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathsf{SE}}(\mathcal{A})$ :
    $s \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$
    $d \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{Enc}(\mathsf{LR}(\cdot,\cdot,1))}$
    return $d$
```

```
Experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathsf{SE}}(\mathcal{A})$ :
    $s \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$
    $d \overset{\$}{\leftarrow} \mathcal{A}^{\mathsf{Enc}(\mathsf{LR}(\cdot,\cdot,0))}$
    return $d$
```

For every PPT *$\mathcal{A}$, there exists a negligible function ν such that it holds that:*

$$\left| \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathsf{SE}}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathsf{SE}}(\mathcal{A}) = 1] \right| < \nu(\lambda)$$

Definition 9 (Pseudo-Random Function (from book of [Gol09])). *A function* $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{c\lambda}$ *is called a* pseudo-random function *if it satisfies the following properties.*

Efficient: *For every $k \in \{0,1\}^\lambda$, and every $m \in \{0,1\}^{c\lambda}$, there exists a* PPT *algorithm to compute* $\mathsf{PRF}_k(m) = \mathsf{PRF}(k,m)$.

Indistinguishable from Random: *For every* PPT $\mathcal{A}$, *there exists a negligible function ν, such that for all auxiliary input $z \in \{0,1\}^\star$ it holds that:*

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} (\mathcal{A}^{\mathsf{PRF}_k}(z) = 1) - \Pr_{f \xleftarrow{\$} F} (\mathcal{A}^f(z) = 1) \right| < \nu(\lambda)$$

where $F = \{f\colon \{0,1\}^\lambda \to \{0,1\}^{c\lambda}\}$

Definition 10 (CP-well-formed functionality). *We say that a CP functionality $\mathcal{F}$ is a* CP-well-formed functionality *if, when all parties are corrupt $\mathcal{F}$ has the following behavior on its adversaries' interfaces:*

1. *Whenever a message m is received on the ith adversarial interface, $\mathcal{F}$ outputs (i, m) to the first adversarial interface.*

2. *Whenever a message of the form $(i, \mathtt{msg})$ is received on the first adversarial interface, $\mathcal{F}$ outputs the message m to the ith adversarial interface.*

I UC security with global setup (GUC)

The core of (GUC) security is the indistinguishability between the real and ideal worlds. In the real world, parties execute a protocol Π and communicate over a channel defined in the model. In the ideal world, parties access a functionality $\mathcal{F}$ which obtains inputs from them and returns to them the output directly. A protocol Π *securely-realizes* a functionality $\mathcal{F}$ if any adversary $\mathcal{A}$ in the real world can be emulated by a simulator Sim in the ideal world. That is, the two worlds cannot be distinguished by a distinguisher, called the environment $\mathcal{Z}$. In addition, the property of *composability* means that a protocol Π remains secure, even after replacing its calls to a subroutine $\mathcal{F}_1$ (a functionality) with calls to a protocol Π_1 that securely realizes $\mathcal{F}_1$. We call Π a $\mathcal{F}_1$-hybrid protocol.

Below, we formally define the GUC framework of Canetti et al. [CDPW07]. Let $\mathcal{R}$ and $\bar{\mathcal{G}}$ be functionalities. Let Π be a $\mathcal{R}$-hybrid protocol, $\bar{\mathcal{G}}$ a setup, $\mathcal{A}$ an adversary, and $\mathcal{Z}$ an environment. The output of the environment $\mathcal{Z}$ after an execution of Π in the GUC $\bar{\mathcal{G}}$-hybrid model in presence of $\mathcal{A}$ is denoted as $\textsc{exec}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}}$. The output of $\mathcal{Z}$ in the ideal world where the simulator Sim interacts with an ideal functionality $\mathcal{F}$ and the setup $\bar{\mathcal{G}}$ is denoted as $\textsc{exec}^{\bar{\mathcal{G}},\mathcal{F}}_{\Pi,\mathsf{Sim},\mathcal{Z}}$.

Definition 11 (UC with Global Setup). *Let $\bar{\mathcal{G}}$ be a global setup, and $\mathcal{R}$ be a resource. For an n-party efficient protocol* Π *and functionality $\mathcal{F}$, we say that* Π GUC-realizes $\mathcal{F}$ *if* $\forall\mathcal{A}\ \exists\mathsf{Sim}\ \forall\mathcal{Z}$

$$\textsc{exec}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \textsc{exec}^{\bar{\mathcal{G}},\mathcal{F}}_{\mathsf{Sim},\mathcal{Z}}$$

I.1 The basic signature functionality

In Fig. 9, we provide the basic UC signature functionality proposed in [Can03] modified to support strong unforgeability.

Key Generation.
Upon receiving a value $(\texttt{KEY\_GEN}, \mathsf{sid})$ from some party $S \in \mathcal{P}$, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid. If not, then ignore the request. Else, hand $(\texttt{KEY\_GEN}, \mathsf{sid})$ to the $\mathcal{A}$. Upon receiving $(\texttt{VERIFICATION\_KEY}, \mathsf{sid}, v)$ from the $\mathcal{A}$, output $(\texttt{VERIFICATION\_KEY}, \mathsf{sid}, v)$ to S, and record the pair (S, v).
Signature. If $I = (\mathsf{SIGN}, \mathsf{sid}, m)$ is received from party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore the request, else send $(\mathsf{SIGN}, \mathsf{sid}, m)$ to $\mathcal{A}$. Upon receiving $I = (\texttt{SIGNATURE}, \mathsf{sid}, m, \sigma)$ from $\mathcal{A}$, verify that no entry $(m, \sigma, v, 0)$ is stored. If it is, then output an error message to S and halt. Else, send $(\texttt{SIGNATURE}, \mathsf{sid}, m, \sigma)$ to S, and store the entry $(m, \sigma, v, 1)$.
Verification
Upon receiving a value $(\texttt{VERIFY}, \mathsf{sid}, m, \sigma, v')$ from some party p_i, hand $(\texttt{VERIFY}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{VERIFIED}, \mathsf{sid}, m, \phi)$ from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
2. Else, if $v' = v$, the signer is not corrupted, and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees strong unforgeability: If v' is the registered one, the signer is not corrupted, and a signature σ of m has never been generated, then the verification fails.)
3. Else, if there is an entry (m, σ, v', f') stored, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ)

Send $(\texttt{VERIFIED}, \mathsf{sid}, m, f)$ to p_i.

Figure 9: The $\mathcal{F}_{\texttt{SIGN}}$ functionality with strong unforgeability of [Can03]

J Modeling synchrony

In Figure 10 we model the global clock, which is used to keep track of the current time/round.

K Secure function evaluation and broadcast functionalities

In Fig. 11 and 12 we provide the SFE and broadcast functionalities proposed in [GKM$^+$13]. In Fig 13 we provide the formal description of the for unanimous abort functionality $\mathcal{F}^{\mathsf{UNA}}$ (the

The functionality is available to all participants. The functionality is parametrized with variable τ, a set of parties $\mathcal{P} = p_1, \ldots, p_n$, and a set F of functionalities. For each party $p_i \in \mathcal{P}$ it manages variable d_i. For each $\mathcal{F} \in F$ it manages variable $d_{\mathcal{F}}$ Initially, $\tau = 0, \mathcal{P} = \emptyset$ and $F = \emptyset$.

- Upon receiving $(\texttt{CLOCK-UPDATE}, \mathsf{sid})$ from some party $p_i \in \mathcal{P}$ set $d_i = 1$ execute *Round-Update* and forward $(\texttt{CLOCK-UPDATE}, \mathsf{sid}, p_i)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{CLOCK-UPDATE}, \mathsf{sid})$ from some functionality $\{ \in F$ set $d_{\mathcal{F}} = 1$, execute *Round-Update* and return $(\texttt{CLOCK-UPDATE}, \mathsf{sid}, F)$ to F.
- Upon receiving $(\texttt{CLOCK-READ}, \mathsf{sid})$ from any participant (including the environment, the adversary, or any ideal-shared or local-functionality) return $(\texttt{CLOCK-READ}, \mathsf{sid}, \tau)$ to the requester.

Procedure *Round-Update*: If $d_{\mathcal{F}} = 1$ for all $\mathcal{F} \in F$ and $d_i = 1$ for all honest $p_i \in \mathcal{P}$, then set $\tau = \tau + 1$ and reset $d_{\mathcal{F}} = 0$ and $d_i = 0$ for all parties in $\mathcal{P}$.

Figure 10: The functionality $\mathcal{G}_{\texttt{clock}}$

description of the functionality extends to 3-party functionality provided in [PR18]).

$\mathcal{F}^f_{\mathsf{SFE}}$ is as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^n \times R \to (\{0,1\}^*)^n$ and a set of parties $\mathcal{P}$. Initialize the variables $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ to a default value $\perp$.

- Upon receiving (Input, v) from some party $p_i \in \mathcal{P}$, set $x_i := v$ and send a message (Input, i) to the adversary.
- Upon receiving (Output) from some party $p_i \in \mathcal{P}$, do:
    1. If x_j has been set for all $j \in \mathcal{H}$, and $y_1, \ldots, y_n$ have not yet been set, then choose $r \overset{\$}{\leftarrow} R$ and set $(y_1, \ldots, y_n) := f(x_1, \ldots, x_n, r)$.
    2. Output y_i to p_i

Figure 11: The $\mathcal{F}^f_{\mathsf{SFE}}$ functionality of [GKM$^+$13]

$\mathcal{B}$ is as follows, given a set of parties $\mathcal{P}$.

- Upon receiving x_i from party $p_i \in \mathcal{P}$, send x_i to every party in $\mathcal{P}$. (If $\mathcal{B}$ is considered a UC functionality, the output is given in a delayed manner, cf. [Can00])

Figure 12: The broadcast functionality $\mathcal{B}$ of [GKM$^+$13]

- Upon receiving (Input, v) from some party $p_i \in \mathcal{P}$, set $x_i := v$ and send a message (Input, i) to the adversary. If v is outside the domain of p_i consider $x_i = \mathtt{ABORT}$.
- If there exists $i \in \{1, \ldots, n\}$ such that $x_i = \mathtt{ABORT}$ the set $(y = \perp)$ else set $y = f(x_1, \ldots, x_n)$ and send y to the adversary.
- Upon receiving `ok` from the adversary, send y to the honest parties (if they query the functionality to get the output).
- Upon receiving `ABORT` from the adversary send $\perp$ to the honest parties (if they query the functionality to get the output)

Figure 13: The $\mathcal{F}^{\mathsf{UNA}}$.