Collusion-Preserving Computation without a Mediator*

Michele Ciampi
University of Edinburgh
michele.ciampi@ed.ac.uk

Yun Lu
University of Edinburgh
Y.Lu-59@sms.ed.ac.uk

Vassilis Zikas
Purdue University
vzikas@cs.purdue.edu

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919678

***Abstract*—Collusion-free (CF) and collusion-preserving (CP) protocols enrich the standard security offered by multi-party computation (MPC), to tackle settings where subliminal communication is undesirable. However, all existing solutions make arguably unrealistic assumptions on setups, such as physical presence of the parties, access to physical envelopes, or extreme isolation, where the only means of communication is a star-topology network. 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 scenarios. Motivated also by the increasing interest in using hardware tokens for cryptographic applications, in this work we provide the first solution to collusion preserving computation which uses weaker and more common assumptions than the state of the art, i.e., an authenticated broadcast functionality and access to honestly generated trusted hardware tokens. We prove that our protocol is collusion-preserving (in short, *CP*) secure as long as no parties abort. In the case of an aborting adversary, our protocol still achieves standard (G)UC security with identifiable (and unanimous) abort.**

Leveraging the above identifiability property, we augment our protocol with a penalization scheme which ensures that it is not profitable to abort, thereby obtaining CP security against incentive-driven attackers. 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 CP/CF solutions, our protocol preserves, as a fallback, security against monolithic adversaries, even when the setup (i.e., the hardware tokens) is compromised or corrupted. In addition, our fallback solution achieves identifiable and unanimous abort, which we prove are impossible in previous CP solutions.

I. Introduction

Subliminal communication channels in protocols allow parties to embed extra information into protocol messages, often 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. An example is the prototypical application of distributed cryptography, namely, playing poker in a distributed manner [2]. An MPC protocol which allows collusions changes the rule of the game —think of playing poker against colluding opponents. In the quest to combine game theory and cryptography, a number of works [3]–[7] put forth new security notions, and in particular the notion of *collusion-freeness (CF)*. Analogous to simulation-based security, where a protocol is "secure" if the view of the adversary (controlling corrupt parties) can be emulated by a simulator, a protocol is *collusion-free*, if the view of individual corrupt parties can be emulated by individual non-colluding simulators. However, collusion freeness is impossible when parties are connected by pairwise communication channels—this is straightforward to see since such channels can directly be used for coordination.

The above led to the proposal of alternative models that enable collusion freeness. The common feature of these models is that, unlike traditional cryptographic definition that assume a worst-case monolithic adversary, these alternative models consider isolated/local adversaries.[1] The two most typical models are assuming players are physically collocated, have access to a semi-trusted "ballot box," and can communicate publicly via (physical) envelopes [3], [4], or assuming that parties are connected to a semi-trusted party (via standard communication channels) called the *mediator* [6], [7] in a star network topology. Roughly, in each round, the mediator performs a two-party computation with each party individually, in order to prevent subliminal communication between parties.

Unfortunately, collusion-freeness, as a standalone definition, is not enough to limit collusion when parties can be engaged in multiple protocols. As argued by Alwen et al. [8], unlike what one might expect, a CF protocol (secure according to the definition of [6], [7]) does not necessarily preserve its "collusion-freeness" when composed with other protocols. Alwen et al. gives the following simple counter-example. Suppose a CF protocol is augmented with the following: it allows two parties A and B to collude only if party B can provide the correct (randomly-generated at run-time) λ-bit password, but the password is given only to party A. The protocol remains CF, since B can only guess the password with negligible probability. However, if through executing another protocol party A can communicate λ-bits to B, then A can send

*A full version of this paper can be found on IACR ePrint Archive [1].
MC: Research supported in part by H2020 project PRIVILEDGE #780477. YL: Research supported in part by Sunday Group. VZ: Research supported in part by Sunday Group and by the NSF grant no. 2055599.

[1]Notably local adversaries are a generalization of the monolithic-adversary model; indeed, the latter can be captured by giving local adversaries the ability to communicate through the (assumed) network (cf. [5], [7]).

© 2022, Michele Ciampi. Under license to IEEE.
DOI 10.1109/CSF54842.2022.00013

the password and collusion-freeness is lost. This limitation motivated [8] to introduce the notion of collusion-preserving computation (CP). Intuitively, this notion can be seen as a universally composable (UC) [9] extension of CF, designed to explicitly address the above issue. (An alternative model capturing collusion-preserving universally composable computation via local adversaries was concurrently and independently proposed by Canetti and Vald [5].) As a concrete example, suppose a protocol Π is CP, and suppose while executing Π a corrupt party p_i uses an external communication channel to send another party p_j a message m (e.g. a secret value only p_i knows). CP ensures that knowing m does not allow p_j to gain *even more* information in Π, such as p_i's input or output, other than information already implied by m. Importantly, CP supports composition and it is modeled with respect to arbitrary communication resources.

Continuing the line of works in the mediated model, [8] constructed a collusion-preserving (CP) protocol, which achieves the following fallback guarantee: Assuming a global augmented common reference string (ACRS) functionality [10], 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 [10]—i.e., secure with abort according to the monolithic-adversary definition—even if the mediator gets corrupted.

On the downside, the CP protocol of [8] in the mediated model presents several limitations. First, it is straightforward to prove that desirable properties such as fairness and identifiable abort are impossible in the mediated model when the mediator might get corrupted (we refer to the full version [1] for more details). This can lead to undesirable situations—such as a poker tournament where a player can always abort the game when he realizes he is losing, without being identified as a cheater. Second, the protocol (compiler) from [8] takes explicit steps to ensure that an abort does not allow parties to correlate their strategies, by making sure that an abort is observable only in a final round that is deterministically fixed at the beginning of the protocol. However, this means that their solution cannot be used to compute reactive functionalities, since a party can signal by means of aborting in an intermediate output (in the full version we provide a description of a concrete collusion strategy which this allows). Lastly, the protocol is not round-efficient. Having a deterministic upper bound on the number of rounds means that, if the goal is to unanimously decide on whether or not an abort occurred even in the fallback setting where the mediator is compromised, the protocol always 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 [11]. In fact, in the [8] compiler, 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 [8] and extend its applicability towards a framework 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 require physical presence of parties in the same room. Concretely, our solution replaces the mediator by the strictly weaker (as we argue later) assumption of an authenticated broadcast channel and honestly generated hardware tokens. As we will show later (Sec. III): (1) Capturing such hardware tokens in a collusion preserving framework is non-trivial (2) Our protocols achieve CP when the adversary does not abort (and in fact CP is impossible with the broadcast channel when the adversary can abort). To disincentivize aborts, we show how to leverage the publicly identifiable abort (Section V-1) property of our protocol to concretely penalize (e.g., via the blockchain) aborting parties. The fact that our protocol is CP means that any external communication introduced by the blockchain will not lead to even more correlation in our protocol. To capture incentive-driven attackers in a composable manner, we combine and extend the CP framework with the Rational Protocol Design (RPD) methodology [12]. We believe that both our treatment of hardware tokens in CP, and our incentives model which we term RPD-CP, can be of independent interest.

As fallback in case the hardware tokens may be compromised, our proposed protocol protects the inputs of honest parties and ensures identifiable (unanimous) [2] abort while guaranteeing termination (cf. [13]). This is the analogue—in the token-hybrid setting with a broadcast channel—of the fallback property of [6]–[8] 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 [8]; indeed, the star-network topology makes it impossible to obtain identifiable (unanimous) abort against a corrupted mediator. The reason is that since the mediator controls the communication, it is impossible to correctly detect if a malicious party did not sent a message, or if it is the malicious mediator is pretending that an honest party has stopped replying (we refer to the full version [1] for the formal arguments).

1) Overview of our contributions: We now present our contributions in more detail. The goals of our work are to construct CP protocols that (1) replace the mediator-centered star-topology network by weaker resources which are closer to modern communication networks, (2) achieve stronger fallback security properties and (3) ensure CP computation of even reactive functionalities by proving that incentive-driven attackers will not abort, in a security model that incurs the negative cost of abort to the adversary. These goals bring the theory of CP closer to capturing real world applications such as a decentralized-dealer poker game. For reference, a comparison of our results to [4] and [6], [8] can be found in Table I. In more detail, our protocol emulates CP functionalities,

[2]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).

including those with private actions, when no abort occurs. We disincentivize aborts via a concrete penalization scheme and in addition, we achieve fallback security maintaining identifiable abort (and unanimous abort). That is, even when hardware tokens are compromised and the parties are allowed to abort we still retain standard GUC security. The abort column in the table specifies the number of bits of subliminal communication possible when an abort happens.

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 [14] which, when everyone gets corrupted, give up on collusion preservation (c.f. [1]). Our protocol uses as resources (1) honestly generated stateful trusted hardware tokens (HTs) and (2) an authenticated broadcast channel available during protocol execution[3]. Our protocol, which can CP-emulate any functionality as long as no abort occurs, improves upon the CF-protocol of [4], which is also based on broadcast but is not CP and does not emulate games with private actions (i.e. actions that are not publicly observable). We note that our protocol, analogously to that of [4], remains (G)UC secure with identifiable abort (though not collusion free) in case of abort. We remark that $\Pi^{\mathtt{HT}}$ only requires two (broadcast) communication rounds unlike that of [4] or [8]. Furthermore, unlike the mediator from [8], tokens do not need to know in advance what 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 $\Pi^{\mathtt{HT}}$ only offers security guarantees when the tokens are uncompromised. This is arguably a strong assumption since in reality the adversary may attempt to break the security of the tokens. 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 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 hardware tokens are compromised. Specifically, even if the memory/code of corrupt parties' tokens can be read/reprogrammed and the secret keys of honest parties' tokens are leaked, the protocol is still (G)UC secure. This improves upon the fallback security of [8] where, due to model idiosyncrasies, these properties are impossible to achieve when the mediator is corrupted, even with honest majority (also in this case the reason is that the mediator has full control of the network, for more detail we refer to the full version). 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 [8].

We believe that the above corruption model for the fallback solution is quite realistic, and captures most of the attacks that have been performed on hardware tokens. For this reason, our main theorems are proven in this corruption model. However, for sake of completeness, we will also briefly sketch how to construct a protocol that preserves (G)UC security even in the case where also the tokens of the honest parties are fully compromised (i.e., the memory/code of any token can be read/reprogrammed). However, this protocol will require a higher communication complexity, and we lose the property of identifiable abort.

[3]The broadcast channel used in this work guarantees that messages are always delivered to the parties [15], [16].

	Channel, assumption	Id-abort (fallback)	U-abort (fallback)	Private actions	Abort
Lepinski et al., STOC 2005	Broadcast+physical presence of parties, 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	Authenticated Broadcast, hardware tokens	✓	✓	✓	Disincentivized

TABLE I
COMPARISON WITH EXISTING APPROACHES. ID-ABORT: IDENTIFIABLE ABORT, U-ABORT: UNANIMOUS ABORT.

As an additional contribution, we combine the Rational Protocol Design (RPD) [12], [17] framework with CP and define a new model termed *RPD-CP*. Within this model we define a CP-security notion *collusion preserving attack payoff* (in short, *CPAP*), which intuitively corresponds to security against any combination of incentive-driven local adversaries. We identify a natural class of utilities under which non-aborting 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 [18]–[22] which we think is an interesting future direction. Finally we propose a concrete penalization mechanism, which may be implemented via e.g., the blockchain, that induces utilities in the above class. Combining this with the notion of CPAP, we can prove that any adversary who maximizes its revenue (or at least minimizes loss) will not abort, making our protocol CP secure against such adversaries. We note in passing that the ability of the CP security definition to make formal statements in the presence of a global blockchain demonstrates the power of the CP definition. Indeed, such a CP statement concretely treats the protocol-independent communication as an external channel; what CP ensures then is that the protocol does not increase the information communicated through this external medium. Thus, informally, if one can devise an equilibrium in an ideal poker game (with a trusted dealer), where the players can also access the blockchain, then by adapting the results from [8] we can prove that this equilibrium would be preserved when the poker dealer is replaced by our CP protocol.

2) Organization of the paper.: Sec. III: Overview of our constructions and main techniques. Sec. IV: Preliminaries. Sec. V: Protocol $\Pi^{\mathtt{HT}}$ for CP functionalities assuming tokens, broadcast and no abort. Sec. V-A: Protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ with fallback GUC security against compromised tokens. Sec. VI: New framework RPD-CP for defining CPAP—security of CP protocols against rational attackers. Sec. VI-B: Proof our protocols $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ are CPAP secure. Sec. VI-C: Penalization scheme to make the utilities defined in Sec. VI

concrete.

II. Related Works and Discussions

A. Collusion Freeness and Preservation

We extend the comparison of our work with existing results on collusion freeness (CF) and preservation (CP). The work of Lepinski et al. [4] 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 (authenticated) broadcast channels, CF is impossible, even when the adversary does not cause the computation to abort. 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 CF 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, ensure parties cannot see messages generated by different inputs. Thus, our protocol remains CP even with private actions. As described in the introduction, in the mediated model, [6], [7] achieve CF and [8] achieves CP. In [5] the authors consider the notion of UC security with local adversaries. This notion is more general than the notion of CP as it captures more security flavors and more sophisticated corruption models. In particular, the notion introduced in [5] capture the scenario where there are independent of clusters of corrupted party, whereas [8] assume that each adversary works in isolation.

1) Authenticated Broadcast Assumption: We compare authenticated broadcast with assumptions from previous works. Existing works on CF and CP either require parties to be present in person [4]—to pass around physical envelopes, or rely on restricted star-topology communication networks [6], [8]. These assumptions are used far less than the (authenticated) broadcast channel common in MPC literature, and have been criticized as overly restrictive and/or unrealistic.

In particular, the physical presence assumption of [4] makes the resulting protocols inapplicable to the standard cryptographic setting, where the protocol is played by interactive Turing machines (ITMs). Similarly, the mediated model [6], [8] requires both complete isolation of the parties and a special star-network topology where the mediator (at the center) is required to both preserve the privacy of the communication and generate (pseudo)randomness—these requirement are proven to be necessary for CP in the mediated model [8]. This is in contrast to the less demanding assumption of authenticated broadcast combined with honestly generated hardware tokens, which not only we believe is more natural—authenticated broadcast is a standard assumption in the cryptographic protocols literature and trusted hardware is a far less exotic assumption than it used to be, but is also formally weaker as discussed in Section II-C.

As in previous works, if our communication resource (i.e. broadcast) is malicious (e.g., allows undetectable communication between corrupt parties) then our protocols are no longer CP. (In fact it is easy to verify that such undetectable communication would make CP infeasible.) Nonetheless, analogous to "fallback" security in the CP setting with a corrupted mediator [8], our protocol still preserves its standard (G)UC security guarantees under a malicious broadcast resource (see Section V-A2).

B. Playing Games over the Blockchain

The question of playing games such as poker over the Internet has recently attracted considerable attention, fueled by the new capabilities introduced by smart-contract-enabled (cryptocurrency) blockchains, such as Ethereum. In a nutshell, this technology makes 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 collateral that they can only claim if evidence is presented that they completed their protocol. This method of penalization has been used by, e.g., [18]–[22], which gave rise to a number of proposals for decentralized poker protocols [23], [24]. However, all these works use standard multi-party computation, thus even players who do not know each other can collude via protocol messages.

C. Stateful Tamper-Resilient Hardware Tokens

Previous works (e.g. [25]–[32]) have based (UC-)secure protocols on stateful, tamper-resilient hardware. In addition to theory, trusted hardware such as Intel's SGX ([33]–[35]) and Bitcoin Hardware Wallets, have also been deployed in real life. The protocol of Lepinski et al. [4] is based on physical envelopes, a kind of trusted hardware as well. They constructed CF (though not CP) protocols for games with public actions, where parties pass around such envelopes.

We extend the application of hardware tokens by presenting the *first* (in our knowledge) collusion-preserving protocol based on stateful trusted hardware tokens. We emphasize that while improving upon the limitations and impossibilities of previous works, we do not remove the need for trust completely. However, instead of relying on a trusted mediator to achieve CP, we replace it with authenticated broadcast (discussed in Section II-A1) and trusted hardware tokens.

Intuitively, these assumptions are weaker than the mediator, since an honest mediator in a star topology can act as a broadcast channel and as a set of hardware tokens. In fact in the full version [1] we prove a formal separation of the assumption of a trusted mediator from the one used here (i.e., the combination of broadcast and honestly generated hardware tokens) demonstrating that our assumption is indeed strictly weaker than that of a trusted mediator ; in particular we show that a trusted mediator allows for CP even in the presence of an aborting adversary, which is impossible in our setting.

With trusted tokens, we improve upon the solution of [4] by achieving composition, games with private actions, and only requiring parties to broadcast messages instead of physically exchanging tokens. We also circumvent the impossibilities of previous works in collusion-preservation, as discussed in the introduction. To improve the practicality of our solutions, in Section V-A we propose protocols that provide fallback security in case of compromised tokens (we discuss two levels of severity of compromise), as well as address practical issues in implementation (Section V-2).

Below, we detail the properties of our token assumption. We follow the approach of [27] to model hardware tokens as ideal functionalities. Our tokens require the following properties:

- *Stateful*: The token has internal memory which may be read/updated.
- *Trusted and Tamper-resilient*: The token manufacturer is trusted, and no one except the token itself can read or write its contents. In Section V-A we also sketch solutions (with some trade-offs) which preserve GUC security given untrusted or non-tamper-resilient tokens.
- *Isolated* from its creator: Only the token owner can query and get the outputs generated by the token. In particular, we require that no other parties may communicate with the token.

The formalism of isolated, tamper-resilient and stateful hardware tokens was introduced by [26]. There, the creation process of a token is described by a "wrapper" functionality which allows parties to store and run (possibly several) ITMs representing the code and memory of tokens. The wrapper functionality models malicious token creators in the protocol. However, to achieve CP, we assume that all the parties hold honestly-generated tokens that share some common private information. Thus, following [27], we model each token as a ideal functionality without using the wrapper. Proving collusion-preservation based on hardware tokens presents several unique technical challenges. We detail these issues and their solutions in Section III.

III. Overview of our techniques

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 pseudo-random function's (PRF) keys k_0, k_1 and a 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 $\mathsf{sid} \in \mathbb{N}$.

a) Collusion preserving protocol via HT and non-aborting adversary with broadcast: Roughly, our first protocol $\Pi^{\mathtt{HT}}$ works as follows: Each party $p_i \in \mathcal{P} - \{p_n\}$ sends his input, encrypted by his token $\mathtt{HT}_i$, to a designated leader party. Upon receipt, the leader gives these messages, along with his own input, to his own token. The token then computes the output, which the leader forwards to the other parties.

In more detail, each token $\mathtt{HT}_i$ uses $R_0 \leftarrow \mathsf{PRF}(k_0, \mathsf{sid})$ as randomness to generate an encryption key sk. In addition, it uses $R_1 \leftarrow \mathsf{PRF}(k_1, \mathsf{sid}||i)$[4] 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.[5] 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*[6]. Note that the session encryption key sk is common to all the hardware tokens (since all the tokens share the same PRF keys).[7]

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 that all certificates and signatures are valid and the inputs are consistent with the function f. Then, if all the checks are successful, $\mathtt{HT}_n$ generates, as described earlier, the session encryption key sk and decrypts all the encrypted inputs of the other parties using sk (we recall that the PRF key k_0 is shared among all the hardware tokens). Using everyone's (decrypted) inputs $x_1, \ldots, x_n$, , $\mathtt{HT}_n$ evaluates $y_1, \ldots, y_n = f(x_1, \ldots, x_n)$ 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 generated 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. Intuitively, for 1), the randomness used in the computation must be hidden from the parties themselves to achieve CP over broadcast. A

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

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

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

[7] Intuitively, we use session-keys instead of the master secret key because these keys will be leaked to the simulator. Leaking the master secret key would completely compromise the token. A more detailed discussion follows later this section.

concrete example to demonstrate why hiding the randomness is important: Suppose a party Alice has access to a (limited) external channel and uses it to send this randomness to another party Bob. If, for example, the randomness is used as Alice's decryption key in the protocol, Bob now can also decrypt messages directed towards Alice, since Bob sees these encrypted messages over broadcast. This breaks CP as the limited external communication led to additional information, e.g., Alice's output, to be leaked to Bob. For 2), to restrict any sid to one-time use, our stateful hardware token stops replying when a sid is used more than once. This is necessary to prevent an adversary from using the same sid (thus the same randomness) to evaluate different inputs. Otherwise, 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 [4], limiting the games they consider to those with publicly observable actions. The proof that that $\Pi^{\mathtt{HT}}$ is CP for non-aborting adversaries intuitively comes from the fact that $\Pi^{\mathtt{HT}}$ is deterministic given the tokens (which fixes the PRF and msk keys) and the sid we use. We note that while this may appear contradictory (i.e., to obtain a secure protocol you need entropy [36]), 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, and 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 [22] we refer to this property as *publicly identifiable abort*. To reduce the amount of token memory required for checking that each sid is used only once, we propose two alternate solutions in Section V-2.

b) Tokens in collusion-preserving computation: One may wonder why hardware tokens cannot directly use their 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 [37] 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 set of pre-computed messages which contain no information about the input of the parties. These messages are indistinguishable from the messages that would be generated in the real world and are properly signed with respect to the n signing-verification session keys as they would be in a real world execution. For example, for the protocol we have just described, the trapdoor would allow the simulator to get a set of encryptions of 0, all authenticated with respect to the corresponding signing-verification session keys. To unable such a mechanism, 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.

Most importantly, this trapdoor 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 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 (e.g., the authenticated encryptions of 0). When $\mathcal{F}^{\star}$ receives a reply from the token functionality, it is forwarded to the simulator. Note that we leak only messages that can be used within a specific session id, since they are signed using signing key that are bonded to one session id. Indeed, as discussed previously, for each session we create new session (e.g., signature) keys that are valid only within that specific session.

The mechanism described above is quite natural as in the real world the parties are allowed to see signed messages passing on the channel and determine their actions based on these

messages. The same should be allowed in the ideal world. Hence, we enhance the ideal functionality $\mathcal{F}$ by constructing $\mathcal{F}^\star$ which acts exactly as $\mathcal{F}$ as described above.

c) Collusion preservation with fallback security: Despite being simple and optimal in terms of round complexity, the protocol Π^{HT} 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 $\Pi^{\text{HT-FBS}}$ 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 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, even when tokens can be broken. Interestingly, these properties are impossible to achieve in the analogous scenario of a corrupted mediator in the mediated model. To obtain such a protocol we use as a main building block a protocol Π^{MPC} that is secure against a malicious adversary and which enables identifiable (unanimous) abort. Each 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, it is the XOR of the randomness produced by the hardware token, and a random string given at runtime 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, even if a token's secret keys are leaked. On the other hand, if all hardware tokens are uncompromised, then the randomness of any party in the MPC protocol becomes unknown and untamperable to everyone. Thus, no malicious party can send subliminal messages without being detected and causing an abort.

For sake of completeness, we also sketch a protocol which, although less round-efficient and without identifiable abort, preserves standard (GUC) security even when tokens are *fully* compromised—that is, not trusted, isolated, nor tamper-resilient. This is a stronger version of a "compromised" token since the adversary may read the contents or even change the behavior of honest parties' tokens. We make a simple alteration to our solution: Any computation performed by the token, will instead be done via a secure two-party protocol between the token and its owner. In more detail, each party p_j runs a secure 2-party protocol with his token, to obtain the next message of the protocol Π^{MPC}. When tokens are uncompromised, the behavior of this solution is the same as in our original solution, achieving CP. When tokens are compromised, fallback security follows from the security of 2-party protocol which ensures the tokens learn nothing. This solution, however, cannot achieve identifiable abort against fully-compromised tokens since any token can, e.g., be forced to stop any interactions. This also means the solution can only implement functionalities with abort, even with honest majority.

d) How to deal with aborting adversaries: We note that the above CP-protocols cannot prevent, for example, a corrupt poker player from simply sending an encrypted message "I have an ace of spades; let's collude!" over the broadcast channel. While the protocol could detect such invalid messages and abort, this attack already breaks CP and seem unavoidable if no assumptions are made on the network topology (e.g., the mediated model [6]–[8]) and on the honesty of the network nodes. In this work we circumvent the above issue by considering an *incentive-driven* (rational) attacker. That is, we define a security notion called CPAP (collusion preserving attack payoff secure) that captures the fact that some adversarial actions—in our case aborts—are not "for free" and instead incur a negative payoff. For example, as our protocols have (publicly) identifiable abort, a judge in the poker game example can identify and penalize an adversarial party causing the abort. Similar to the rational protocol design (RPD) framework [12], CPAP considers a CP-well-formed functionality $\mathcal{F}$, and a relaxed functionality $\langle\mathcal{F}\rangle$ that acts the same as $\mathcal{F}$ except it explicitly includes weaknesses that allow a simulator to collude or abort. Then, we define a value function v mapping the joint view of a set of simulators[8] interacting with the relaxed functionality $\langle\mathcal{F}\rangle$ and the environment $\mathcal{Z}$, to a real-valued *payoff*. Intuitively, the real utility gained/lost by a set of adversaries for a given protocol is the payoff maximized over all environments, and minimized over all sets of simulators that successfully emulate the adversaries in the environment. For our CP protocols, we show that collusion always causes the real world execution to abort (and identify a corrupt party). Thus, when the cost of abort exceeds the gains from colluding, a rational attacker will never collude (otherwise simulator can trigger an abort in $\langle\mathcal{F}\rangle$)—intuitively it means that our protocol implements $\mathcal{F}$ for rational attackers.

e) From "ideal" to "real" payoffs.: In Sec. VI-C we construct a penalization protocol Π_{pen} which runs the computation of our CP protocols Π^{HT} or $\Pi^{\text{HT-FBS}}$, and in addition penalizes the adversary when he colludes (which can be done only by triggering an abort). For a natural class of utilities for the protocol designer and attacker, Π_{pen} enjoys both CPAP and CPIC, following similar arguments as in Sec. VI-B.

[8]Recall that in CP, adversaries are not monolithic, so we consider a set of adversaries/simulators instead of one single adversary/simulator.

IV. 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, $x \xleftarrow{\$} Q$ denotes a sampling of x from Q with uniform distribution. When it is necessary to refer to the randomness r used by and algorithm A we use the following notation: $A(\cdot; r)$. 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. We assume familiarity with the notions of secure-function evaluation, strong signatures, secret key encryption, negligible functions and pseudorandom functions and refer to the full version for more detail.

1) Security with identifiable (unanimous) abort: In this work we consider the notion of secure function evaluation (SFE) with *identifiable abort*. This notion allows the computation to fail (abort), but ensures that when this happens all the honest parties are informed about it, and they also agree on the index i of some corrupted party $p_i \in \mathcal{P}$ [38]. We denote the ideal functionality for the evaluation of the function f that captures the property of identifiable abort with $\mathcal{F}_{\mathsf{IDA}}^{f}$ (where IDA stands for identifiable abort). We also consider the notion of *unanimous abort*. This guarantees that either all or none of the honest parties abort. We denote the ideal functionality for the evaluation of the function f that captures the property of unanimous abort with $\mathcal{F}_{\mathsf{UNA}}^{f}$ (where UNA stands for unanimous abort). We refer to the full version for formal definitions of $\mathcal{F}_{\mathsf{IDA}}^{f}$ and $\mathcal{F}_{\mathsf{UNA}}^{f}$. We refer to [39] for a detailed discussion of the notions of security with non-unanimous (aka selective), unanimous, and identifiable abort, and their relation.

We will assume synchronous computation, i.e., our protocols proceed in rounds, where in each round: the uncorrupted parties generate their messages for the current round, as described in the protocol; then the messages addressed to the corrupted parties become known to the adversary; then the adversary generates the messages to be sent by the corrupted parties in this round; and finally, each uncorrupted party receives all the messages sent in this round. Although our treatment is in the (G)UC setting, to avoid overcomplicating the exposition, we will use the standard round-based language of [40], [41] to specify our protocol. Notwithstanding, such specifications can be directly translated to the synchronous UC model of Katz *et al.* [13] by assuming a clock functionality and bounded (zero) delay channels. We refer the interested reader to [13] for details.

2) Collusion-preserving computation: We now recall the notion of collusion-preserving computation proposed in [8], which we refer the reader to 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. ITMs $\{\mathcal{A}_i\}_{i \in \mathcal{I}}$, where $\mathcal{A}_i$ denotes the adversarial strategy of 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, who only has access to $\mathcal{A}_i$.
- **Corrupted-Set Independence:** We also require that the simulators do not depend on each other. In other words the code of Sim_i is the same for any set of adversaries $\mathcal{A}_{[n]}$ and $\mathcal{B}_{[n]}$ as long as $\mathcal{A}_i = \mathcal{B}_i$.

Similar to the GUC framework (but in contrast to plain UC) we distinguish between two types of functionalities: resources, denoted with capital calligraphic font as in $\mathcal{R}$, and shared functionalities, denoted 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 e.g., a global CRS or global PKI as long as they 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 that share state across executions only 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 $\mathcal{R}$. We note that 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 1 (Collusion Preservation [8])**.** *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 $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ 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:* $\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$

Following [8], we distinguish between the notion of *emulation* and its special case *realization*. For a functionality $\mathcal{F}$ we denote by $\mathcal{D}_i^{\mathcal{F}}$ the ith dummy $\mathcal{F}$-hybrid protocol which simply acts as a transparent conduit between the ith honest and adversarial interfaces of $\mathcal{F}$ and the environment $\mathcal{Z}$. Specifically, $\mathcal{D}_i^{\mathcal{F}}$ forwards all messages it receives from $\mathcal{Z}$ to the $\mathcal{F}$ (where the choice of adversarial or honest interface is

specified by $\mathcal{Z}$) and vice-versa. If for functionality $\mathcal{F}$, an $\mathcal{R}$-hybrid protocol π CP-emulates $\mathcal{D}_i^{\mathcal{F}}$ then we say that π *realizes* $\mathcal{F}$ (in the $\mathcal{R}$-hybrid world).

3) Rational protocol design: The goal of the rational protocol design framework (RPD) [12], [17] is to model security of protocols against incentive-driven attackers. 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 based on Π, the attacker decides on an adversarial strategy $\mathcal{A}$ to attack Π. Each pair $(\Pi, \mathcal{A} = \mathtt{A}(\Pi))$ (called a *strategy profile*) induces a utility for the designer and the attacker. Consistent with [12], 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 includes extra commands that break certain security properties of $\mathcal{F}$. The value function $v_{\mathtt{A}}$ allows us to define utilities of the attacker. It assigns payoffs when certain events occur in the ideal world, such as when the simulator uses the extra commands in $\langle\mathcal{F}\rangle$ to help him complete a successful simulation. Now, our goal is ensure that it is not in the attacker's best interest to force the simulator to use weaknesses of $\langle\mathcal{F}\rangle$. This is captured by the notion of *attack-payoff security*. A protocol Π is *attack-payoff secure* if no adversarial strategy attacking Π can achieve more utility than an adversary attacking the "dummy" protocol that just uses the functionality $\mathcal{F}$ (i.e. without weaknesses of $\langle\mathcal{F}\rangle$). In other words, Π is "as secure as" the dummy (trivially secure) protocol in this model. As in [17], we can augment $\mathcal{M}$ with the designer's value function $v_{\mathtt{D}}$, and consider whether the protocol is *incentive compatible* for the designer.

V. CP MPC WITH NON-ABORTING ADVERSARIES

In this section we present our protocol $\Pi^{\mathtt{HT}}$ for any CP-well-formed functionality under the following assumptions: 1) each party has access to a hardware token (which we describe as one global ideal functionality in Fig. 1) 2) all communication is done over authenticated 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 intermediate states 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:

- $\mathsf{sid} \in \mathbb{N}$ uniquely identifies an execution of $\Pi^{\mathtt{HT}}$.
- Set of parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ running $\Pi^{\mathtt{HT}}$ 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: Pseudo-random functions: $\mathsf{PRF}_0 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, $\mathsf{PRF}_1 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \to \{0,1\}^{4\lambda}$, $\mathsf{PRF}_2 : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \to \{0,1\}^{(n+2)\lambda}$, $\mathsf{PRF}_3 : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{(4n-2)\lambda}$; 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 secret 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 $\mathcal{B}$ (c.f. [1] for formal definitions). 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 messages that are 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. Then the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\mathtt{HT}}$ (described by Fig. 2 and 3) CP realizes $\mathcal{F}$ in the $\mathcal{R}$-hybrid world assuming non-aborting adversaries.*

Proof sketch. We refer to the full version for the full proof. For simplicity we assume that only the leader is honest. To prove this theorem, we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ that satisfy Definition 1. For $i = 1, \ldots n$, the simulator $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A}_i)$ queries $(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If so, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates a set of encryptions of 0^λ, a set of signing/verification keys and uses them to authenticate these encryptions (see the bottom of Fig. 1). We note that each simulator will obtain the

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 Σ and the PRF keys K_0, K_1, K_2, K_3.

If $I = (\mathsf{Get\_key}, \mathsf{sid})$ is received return mpk to the caller.

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 λ bits each $\mathsf{rs}^1||\mathsf{rs}^2||r^1||r^2$.
- $(\mathsf{sigk}_j^{\mathsf{sid}}, \mathsf{vk}_j^{\mathsf{sid}}) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}^1)$
- $\mathsf{cert}_j \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j^{\mathsf{sid}}||\mathsf{sid}||j; \mathsf{rs}^2)$
- Compute $\overline{x} \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, x; r^1)$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_j^{\mathsf{sid}}, \overline{x}||f; r^2)$ 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 follows.

- 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 λ bits each $\mathsf{rs}^1||\mathsf{rs}^2||r_1||r_2||\ldots||r_{n-1}||r^\star$.
- $(\mathsf{sigk}_n^{\mathsf{sid}}, \mathsf{vk}_n^{\mathsf{sid}}) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}^1)$
- $\mathsf{cert}_n \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_n^{\mathsf{sid}}||\mathsf{sid}||n; \mathsf{rs}^2)$
- 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 $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, y_j; r_j)$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n^{\mathsf{sid}}, \overline{y}_1||\ldots||\overline{y}_{n-1}||f; r^\star)$;
- 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.

- 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.
- Pick $r_1||\ldots||r_{n-1}||r'_1||\ldots||r'_{n-1}||\mathsf{rs}_1^1||\mathsf{rs}_1^2||\ldots||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_3(K_3, \mathsf{sid})$.
- For each $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\lambda; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$. For each $j \in [n-1]$ $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}_n, 0^\lambda; r_j)$, $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, e_j; r'_j)$
- For each $j \in [n-1]$ compute $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, 0^\lambda; r'_j)$
- $\sigma_n \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, \overline{y}_1||\ldots||\overline{y}_{n-1}||f||\mathsf{sid})$.
- Return to the calling instance $\{\mathsf{vk}_i, \mathsf{cert}_i, \sigma_i\}_{i\in[n]}, \{e_i, \overline{y}_i\}_{i\in[n-1]}$.

Fig. 1. The functionality $T^{\mathtt{HT}}$ models the behaviour of the hardware tokens.

same set of ciphertexts and verification keys. This is crucial for the proof to go through, as each individual simulator $\mathcal{S}_i$ internally runs the corrupted party p_i, and it will act on

We assume that the party p_j is registered to the token functionality $T^{\mathtt{HT}}$ and that it obtains mpk by querying it with $I = (\mathsf{Get\_key}, \mathsf{sid})$. 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 $z = (\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ 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.

Fig. 2. Protocol executed by the party p_j.

Input/output

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

Fig. 3. Protocol executed by the leader party p_n.

the behalf of the other $n-1$ parties using the authenticated cipthertexts received by $\mathcal{F}^\star$. $\mathcal{S}_i$ will also intercept all the queries the party p_i makes to $T^{\mathtt{HT}}$. Assuming that p_i is non-aborting, then at some point they will query $T^{\mathtt{HT}}$ with their input x_i. $\mathcal{S}_i$ now has the input of the corrupted party, and he will use it to query the ideal functionality. In addition, $\mathcal{S}_i$ sends to p_i (acting on the behalf of $T^{\mathtt{HT}}$) an encryption of 0 authenticated with respect to the verification key vk_i. Assuming that no party aborts, the simulation is successful since the messages generated by the individual simulator on the locally simulated broadcast channels are exactly the same for all the parties (unless the adversary breaks the signature scheme).

1) (Publicly) Identifiable abort: 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 the identity of a corrupted p_i. In $\Pi^{\mathtt{HT}}$ the adversary can only deviate from the protocol specification by either sending a message authenticated with respect to a sid' or f' not equal to the correct sid or f the honest parties use, sending a message with an invalid signature or certificate, or fail to send a message. Each event is verifiable by honest parties, and even third parties 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. Formally, the protocol $\Pi^{\mathtt{HT}}$ securely realizes the function $\mathcal{F}_{\mathsf{IDA}}^{f}$, where $\mathcal{F}_{\mathsf{IDA}}^{f}$ involves n parties. More interestingly, we can 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). That is, our protocol allows an *observer* of the protocol execution to identify a misbehaving party. Following [22] we refer to this property as *publicly identifiable abort*, and 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 who aborted the protocol given the parties' messages.

2) Note on implementing $T^{\mathtt{HT}}$ with real hardware tokens: Following the approach of [27], we describe the behavior of the hardware tokens (HTs) by means of a single ideal functionality with a single set of shared keys. In particular, the master signing/verification keys allow tokens and parties to verify whether a message was signed by a hardware token. In practice, such functionality can be replaced by tokens that each has its own secret information (e.g., the HTs do not need share the same master signing key). More precisely, only a global master verification key needs to be shared among the tokens. Each hardware token manages its own signing and verification keys, msk_i and mpk_i, along with a certificate c_i which is a signature of mpk_i created by the HT manufacturer's global master secret key. An example of this approach is Intel SGX processors, where each processor has a unique attestation key and "endorsement certificate" from the manufacturer [42]. To authenticate a message, HT HT_i signs it with its msk_i, and sends the signature together with his master verification key vk_i and the certificate c_i. Anybody that has the global master verification key can verify that the certificate c_i is valid for vk_i, and that the signature issued by HT_i is valid w.r.t. vk_i.

Another important part of $T^{\mathtt{HT}}$ is to ensure that the session ID sid must not be reused in different protocol sessions. The most simple solution stores all sids that the token has seen. Thus, if the token cannot store more sids, it will not be able to verify the freshness of new sids and must stop responding all together. This in effect makes the token no longer usable for our CP protocol. We present two alterations to improve upon the space usage of the simple solution. First, the token can transfer the burden of storing the sids to an external memory, in a hash chain data structure [43]. The token stores the head and tail of the hash chain, which ensures that no malicious party can tamper with the sids on the external memory. The solution only requires a small (i.e., constant in the number of sids) amount of storage. However, verifying a sid requires interaction with the external memory to retrieve the hash chain—to use minimal space, the token may choose to retrieve the chain one hash at a time. To eliminate interactions with external memory, one may also opt for a Bloom filter [44]. This solution trades off the need for interaction with the possibility (depending on the space allocated to the filter) of falsely marking some sids as "used". This however does not impact security, as it is equivalent to the token having seen a session with this sid.

A. Fallback solution when tokens can be compromised

While simple and optimal in terms of round complexity, the protocol $\Pi^{\mathtt{HT}}$ cannot guarantee any form of security when the hardware tokens are corrupted/compromised. In this section we propose a CP protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ that provides fallback security (in a standard GUC-security sense) in the following corruption model: 1) the secret keys of the tokens, including those of honest parties, can be leaked and 2) the memory/code of corrupt parties' tokens can be read/modified completely. Let $\mathcal{F}$ be the function that the parties wish to compute. For our construction we use the following tools.

- Pseudo-random functions $\mathsf{PRF}_0 : \{0,1\}^{\lambda} \times \{0,1\}^{2\lambda} \to \{0,1\}^{(2m+4)\lambda}$ and $\mathsf{PRF}_1 : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{((m+3)n+1)\lambda}$
- A strong unforgeable signature scheme $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver})$.
- A n-party MPC protocol $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$ that GUC-realizes functionality $\mathcal{F}$.

The global setup $\bar{\mathcal{G}}$ is represented by the token functionality $T^{\mathtt{HT\text{-}FBS}}$. Similar to the token functionality of the previous section, $T^{\mathtt{HT\text{-}FBS}}$ has a master public key mpk and a secret state consisting of 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 $T^{\mathtt{HT\text{-}FBS}}$. We denote our protocol with $\Pi^{\mathtt{HT\text{-}FBS}}$ (Fig. 5) and provide a formal description of $T^{\mathtt{HT\text{-}FBS}}$ in Fig. 4.

Security of $\Pi^{\mathtt{HT\text{-}FBS}}$:

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

1) If the hardware tokens are not compromised, and no party aborts, then $\Pi^{\mathtt{HT\text{-}FBS}}$ is collusion-preserving.
2) If the hardware tokens are compromised and Π^{MPC} GUC realizes $\mathcal{F}_{\mathsf{AB}}^{f}$ with $\mathsf{AB} \in \{\mathsf{IDA}, \mathsf{UNA}\}$, then $\Pi^{\mathtt{HT\text{-}FBS}}$ GUC realizes $\mathcal{F}_{\mathsf{AB}}^{f}$
3) If the hardware tokens are not compromised (but the malicious parties may abort), then $\Pi^{\mathtt{HT\text{-}FBS}}$ GUC realizes the functionality $\mathcal{F}$ with publicly identifiable abort.

The properties 1 and 2 above enable the fallback security of $\Pi^{\mathtt{HT\text{-}FBS}}$. In addition, the second property states that in the case of corrupted tokens, $\Pi^{\mathtt{HT\text{-}FBS}}$ inherits all the properties of Π^{MPC} (e.g., identifiable abort). We note that if the MPC protocol 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. To formally prove that $\Pi^{\text{HT-FBS}}$ is fallback secure we need to prove the following two lemmata.

Lemma 1. *Let $\bar{\mathcal{G}} = T^{\text{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$ (broadcast) and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality. Then the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\text{HT-FBS}}$ (described by Fig. 5) CP realizes $\mathcal{F}$ in the $\mathcal{R}$-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 $\bar{\mathcal{G}} = \overline{T}^{\text{HT-FBS}}$ and $\mathcal{R} = \mathcal{B}$ then $\forall \mathcal{A}\ \exists \mathsf{Sim}\ \forall \mathcal{Z}\ \text{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\text{HT-FBS}},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{F}^{\mathsf{AB}}}_{\mathsf{Sim},\mathcal{Z}}$*

Theorem 2. *Let $\bar{\mathcal{G}} = T^{\text{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$ and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality. Then, the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\text{HT-FBS}}$ (described by Fig. 5) GUC realizes $\mathcal{F}^{\mathsf{IDA}}$.*

We refer to the full version for the formal proof of the above lemmata and theorem.

1) Remark on fully-compromised tokens: As discussed in the introduction, to achieve fallback security against fully-compromised (that is, untrusted, non-tamper-resilient, non-isolated) tokens, any computation performed by the token must instead be done through a 2-party protocol between the token and the party holding it. To be specific, this 2-party protocol will implement the following functionality: On request to compute: (1) if round $\ell = 0$: Take input x_j from party p_j and $K_0, \mathsf{mpk}, \mathsf{msk}$ from p_j's token. It stores $x_j, K_0, \mathsf{mpk}, \mathsf{msk}$. (2) If $\ell \leq m$: Take input the messages from the current round from p_j, if a message fails to authenticate then the token stops and p_j aborts. (3) Finally, output the next message $(\mathtt{msg}^{\ell}_j, \cdots)$ as the token would, to p_j.

2) Remark on malicious broadcast: We also give intuition on what happens if our authenticated broadcast resource is malicious. First, we will invariably lose CP, since a malicious broadcast resource can provide a private communication channel for corrupt parties, trivially allowing collusion. However, like with a malicious mediator in [8], we still achieve (G)UC security without unanimous abort/fairness. This is an interesting feature of our model and results: our security degrades more gradually with respect to the underlying assumptions than with a corrupted mediator. When only the tokens are compromised but authenticated broadcast is honest, then we have (G)UC security with identifiable abort—impossible with a corrupt mediator. If broadcast is malicious then we lose identifiable/unanimous abort and fairness—e.g., the malicious broadcast can pass an unauthenticated invalid message to just one honest party, making only this party abort, and it can also

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.

If $I = (\mathsf{Get\_key}, \mathsf{sid})$ is received return mpk to the caller.

Input phase. If $I = (\mathsf{Input}, \mathsf{sid}, x_j, R_j)$ is received from some party p_j, then do the following,

- If $\mathsf{ctr}^{\mathsf{sid}}_j$ is not defined, then define it and $\mathsf{ctr}^{\mathsf{sid}}_j \leftarrow 1$, otherwise output $\perp$ and stop.
- Compute $R^{\mathsf{sid}}_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid}||j) \oplus R_j$ and parse R^{sid}_0 as $(2m+4)$ strings of λ bits $\mathsf{rs}^1_j||\mathsf{rs}^2_j||\rho^1_j||\ldots||\rho^{m+1}_j||r^1_j||\ldots||r^{m+1}_j$.
- $(\mathsf{sigk}^{\mathsf{sid}}_j, \mathsf{vk}^{\mathsf{sid}}_j) \leftarrow \mathsf{Kgen}(1^{\lambda}; \mathsf{rs}^1_j)$
- $\mathsf{cert}^{\mathsf{sid}}_j \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j||\mathsf{sid}||j; \mathsf{rs}^2_j)$
- Output the first message $(\mathtt{msg}^1_j, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\mathtt{msg}^1_j = \mathsf{Next}_j(1^{\lambda}, x_j, \rho^1_j, \perp)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}^1_j||\Pi^{\mathsf{MPC}}||\ell; r^1_j)$

Next message function.

If $I = (\mathsf{NextMsg}, \mathsf{sid}, \{\mathtt{msg}^{\ell}_i, \sigma^{\ell}_i, \mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]})$ is received from p_j then do the following.

- If $\ell > m$ or $\mathsf{ctrap}^{\mathsf{sid}}_j = 1$ then output $\perp$ and stop, else continue with the following steps.
- Parse R^{sid}_0 as $\mathsf{rs}^1_j||\mathsf{rs}^2_j||\rho^1_j||\ldots||\rho^{m+1}_j||r^1_j||\ldots||r^{m+1}_j$.
- Store $\mathtt{msg}^{\ell}_{\mathsf{sid}} = \{\mathtt{msg}^{\ell}_i\}_{i\in[n]}$.
- For all $i \in [n]$ check if $\mathsf{Ver}(\mathsf{vk}_i, \mathtt{msg}^{\ell}_i||\Pi^{\mathsf{MPC}}||\ell, \sigma^{\ell}_i) = 1$ 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 $\mathtt{msg}^{\ell}_j = \mathsf{Next}_j(1^{\lambda}, x_j, \rho^{\ell}_j, \overline{\mathtt{msg}}^{<\ell})$ with $\overline{\mathtt{msg}}^{<\ell} = \{\mathtt{msg}^{\ell'}_i\}_{i\in[n],\ell'<\ell}$ where $\mathtt{msg}^{\ell'}_i$ is the message from p_i at round ℓ' that was stored in $\mathtt{msg}^{\ell'}_{\mathsf{sid}_{\ell}}$
- If $\ell = m+1$ then output $y_j \leftarrow \mathtt{msg}^{\ell}_j$ else, output $(\mathtt{msg}^{\ell}_j, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma^{\ell}_j, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\sigma^{\ell}_j \leftarrow \mathsf{Sign}(\mathsf{sigk}^{\mathsf{sid}}_j, \mathtt{msg}^{\ell}_j||\Pi^{\mathsf{MPC}}||\ell; r^{\ell}_j)$.

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.

- If $\mathsf{ctr}^{\mathsf{sid}}_j$ is not defined, then define it, define $\mathsf{ctrap}^{\mathsf{sid}}_j$, set $\mathsf{ctr}^{\mathsf{sid}}_j \leftarrow 1$ and $\mathsf{ctrap}^{\mathsf{sid}}_j \leftarrow 1$. Else output $\perp$ and stop.
- Compute $\rho||r^1_1||\ldots||r^{m+1}_1||\ldots||r^1_n||\ldots||r^{m+1}_n||\mathsf{rs}^1_1||\mathsf{rs}^2_1||\ldots||\mathsf{rs}^1_n||\mathsf{rs}^2_n \leftarrow \mathsf{PRF}_1(K_1, \mathsf{sid})$.
- For all $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^{\lambda}; \mathsf{rs}^1_i)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}^2_i)$.
- Use ρ to run the simulator of the MPC, S for the case where there are no corrupted party.
- Let $\{\mathtt{msg}^{\ell}_j\}_{j\in[n],\ell\in[m]}$ be the messages contained in the transcript obtained by the MPC simulator S. For all $j \in [n]$ and $\ell \in [m]$ computes $\sigma^{\ell}_j \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}^{\ell}_j||\Pi^{\mathsf{MPC}}||\ell; r^{\ell}_j)$.
- Return $\{\mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]}, \{\mathtt{msg}^{\ell}_j, \sigma^{\ell}_j\}_{j\in[n],\ell\in[m]}$

Fig. 4. The functionality $T^{\text{HT-FBS}}$ models the behaviour of the hardware tokens, in our fallback secure protocol $\Pi^{\text{HT-FBS}}$.

We assume that the party p_j is registered to the global token functionality $T^{\text{HT-FBS}}$ and obtains mpk by querying it with $I = (\mathsf{Get\_key}, \mathsf{sid})$.

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 $T_j^{\text{HT-FBS}}$.
- For each $\ell \in \{1, \ldots, m\}$:
  - Upon receiving message X from $T_j^{\text{HT-FBS}}$ check if $X = (\bot, p_{i'})$. If it is then output $(\bot, p_{i'})$ and stop, otherwise send X over broadcast.
  - Collect message $(\mathtt{msg}_i^\ell, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_i^\ell, \mathsf{vk}_i, \mathsf{cert}_i)$ for round ℓ from each party $p_i \in [n] \backslash \{j\}$ and send $(\mathsf{NextMsg}, \mathsf{sid}, \{\mathtt{msg}_i^\ell, \sigma_i^\ell, \mathsf{vk}_i, \mathsf{cert}_i\}_{i \in [n]})$ to $T_j^{\text{HT-FBS}}$.

Output phase.

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

Fig. 5. Protocol $\Pi^{\text{HT-FBS}}$ followed by p_j to achieve fallback security.

refuse to pass the output to honest parties. Intuitively, security is preserved even with malicious broadcast since messages from the tokens are encrypted and authenticated. If tokens are in addition compromised, then our fallback protocol (when based on a MPC protocol with (G)UC security in presence of a malicious broadcast), also preserves security.

VI. CP MPC FOR RATIONAL ADVERSARIES

We start this section by using the RPD framework to study the feasibility of implementing functionalities $\mathcal{F}$ in a collusion preserving way, against incentive-driven attackers that may even choose to abort the protocol. We extend the RPD model to the case of collusion-preserving functionalities (which we call *RPD-CP*). We prove that our protocols Π^{HT} and $\Pi^{\text{HT-FBS}}$ disincentivize collusion in this model, when there is a sufficient penalty to the attacker for aborting. Recall that Π^{HT} and $\Pi^{\text{HT-FBS}}$ are CP against non-aborting adversaries, and can (publicly) identify a corrupt party in case of an abort. Finally, we show how this model can be applied in practice, e.g. using the blockchain. In particular, we can abstract the blockchain by means of an ideal functionality that allows the parties to deposit collateral, which can be reclaimed on the agreement of all parties. We provide a protocol Π_{pen} that uses this functionality to concretely penalize an attacker for aborting Π^{HT} or $\Pi^{\text{HT-FBS}}$.

A. RPD-CP: Tuning RPD to the CP setting

a) The Attack Model: Following the RPD framework [12], we capture collusion-preservation against incentive-driven attackers, 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/adversarial strategies $\mathtt{A}(\Pi) = \mathcal{A} = \{\mathcal{A}_i\}_{i \in \mathcal{I}}$, $\mathcal{I} \subseteq [n]$ to attack it. The attacker's utility $u_{\mathtt{A}}$ is then a function of the choice of protocol Π and adversarial strategies $\mathcal{A}$. The attack model $\mathcal{M} = (\mathcal{F}, \langle \mathcal{F} \rangle, v_{\mathtt{A}})$ (and $v_{\mathtt{D}}$, if we also consider the designer's utility) encompasses the parameters in this game—$\langle \mathcal{F} \rangle$ is the weaker version of the functionality $\mathcal{F}$ we wish to implement. $\langle \mathcal{F} \rangle$ 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 that are running the protocol. Note that in contrast to monolithic adversaries and simulators, in CP the ideal adversarial parties do not automatically share their views and must use $\langle \mathcal{F} \rangle$ to collude (see. Fig. 6). Lastly, the attacker's utility $u_{\mathtt{A}}$ is defined based on a value function $v_{\mathtt{A}}$, which assigns payoffs to events occurring in the ideal world—more details below.

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 $\mathcal{I} \subseteq \mathcal{P}$

- 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 $p_j \in \mathcal{P} - \{p_i\}$.
- 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.
- Upon receiving a message that is consistent with the interface of $\mathcal{F}$ act as $\mathcal{F}$ would do acting as a proxy between $\mathcal{F}$ and the parties in $\mathcal{P}$.

Fig. 6. $\langle \mathcal{F} \rangle$ weakens $\mathcal{F}$ with commands $\mathtt{COLLUDE}$ and $\mathtt{ABORT}$.

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. the strategy profile $(\Pi, \mathcal{A})$, to a real number. In our case, utility depends on whether a set of simulators must collude via a weakness in $\langle \mathcal{F} \rangle$ in order to emulate $\mathcal{A}$ in Π, and whether the simulators trigger an abort. More formally: First, we have a value function $v_{\mathtt{A}}$, defined in the attack model, which maps the views of the simulators and environment in the ideal world to a real value. Then, we define the real payoff of a particular $\mathcal{A}$ attacking the protocol, as the minimum payoff over all simulators that can emulate $\mathcal{A}$. Finally, $u_{\mathtt{A}}(\Pi, \mathcal{A})$ is the real payoff of $\mathcal{A}$, maximized over all possible environments $\mathcal{Z}$.

a) Ideal payoff of a set of simulators: In more detail, we define the real-valued random variable ensemble $\{v_{\mathtt{A}}^{\langle \mathcal{F} \rangle, \mathcal{S}, \mathcal{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$ (or $v_{\mathtt{A}}^{\langle \mathcal{F} \rangle, \mathcal{S}, \mathcal{Z}}$ 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: $U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}) = E(v_{\mathtt{A}}^{\langle \mathcal{F} \rangle, \mathcal{S}, \mathcal{Z}})$.

b) 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 $\langle \mathcal{F} \rangle$ 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 $\langle\mathcal{F}\rangle$. Let $\langle\mathcal{F}\rangle$ 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 $\mathcal{R}$, $\mathcal{C}_\mathcal{A} = \Big\{\mathsf{Sim}(\mathcal{A}) = \{\mathsf{Sim}_i(\mathcal{A}_i)\}_{i\in\mathcal{I}} \mid \forall i \in \mathcal{I}\colon \mathsf{Sim}_i$ an efficiently computable mapping from ITM to ITM, $\forall\mathcal{Z}\colon \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\langle\mathcal{F}\rangle}_{\Pi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}\Big\}$. The expected payoff of a set of adversaries and environment $(\mathcal{A},\mathcal{Z})$ is then defined as $U^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{A}}(\mathcal{A},\mathcal{Z}) = \inf_{\mathcal{S}\in\mathcal{C}_\mathcal{A}}\{U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{A}}}(\mathcal{S},\mathcal{Z})\}$. The attacker's utility is then maximized over all environments $\mathcal{Z}$, i.e., $u_{\mathtt{A}}(\Pi,\mathcal{A}) := \hat{U}^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{A}}(\mathcal{A}) = \sup_{\mathcal{Z}\in\mathtt{ITM}}\{U^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{A}}(\mathcal{A},\mathcal{Z})\}$.

2) Utility of the protocol designer D*:* In [12], the attack game is assumed to be zero-sum, i.e. the designer's utility $u_{\mathtt{D}} = -u_{\mathtt{A}}$. To remove this assumption, we follow the methodology of a more recent work [17] to define $u_{\mathtt{D}}$. 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 attacker, and $\mathcal{Z}_{\mathtt{A}}$ denote the class of environments maximizing the utility for simulators in $\mathcal{S}_{\mathtt{A}}$. That is, $\mathcal{S}_{\mathtt{A}} = \Big\{\mathcal{S}\in\mathcal{C}_\mathcal{A}\colon \sup_{\mathcal{Z}\in\mathtt{ITM}}\{U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{A}}}(\mathcal{S},\mathcal{Z})\} = u_{\mathtt{A}}(\Pi,\mathcal{A})\Big\}$ and $\mathcal{Z}_{\mathtt{A}} = \Big\{\mathcal{Z}\in\mathtt{ITM}\colon \text{for some } \mathcal{S}\in\mathcal{S}_{\mathtt{A}}\ ,\ U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{A}}}(\mathcal{S},\mathcal{Z}) = u_{\mathtt{A}}(\Pi,\mathcal{A})\Big\}$.

Then, let $v^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}_{\mathtt{D}}$ and $U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{D}}}(\mathcal{S},\mathcal{Z})$ be defined similar to the payoffs $v^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}_{\mathtt{A}}$ and $U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{A}}}(\mathcal{S},\mathcal{Z})$ respectively. Again following the definitions of [17], the real payoff of the designer is $U^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{D}}(\mathcal{A},\mathcal{Z}) = \sup_{\mathcal{S}\in\mathcal{S}_{\mathtt{A}}}\{U^{\langle\mathcal{F}\rangle}_{I^{\mathtt{D}}}(\mathcal{S},\mathcal{Z})\}$. The utility of the designer is then $u_{\mathtt{D}}(\Pi,\mathcal{A}) := \hat{U}^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{D}}(\mathcal{A}) = \inf_{\mathcal{Z}\in\mathcal{Z}_{\mathtt{A}}}\{U^{\Pi,\langle\mathcal{F}\rangle}_{\mathtt{D}}(\mathcal{A},\mathcal{Z})\}$ We can extend the attack model with the value function of the designer $v_{\mathtt{D}}$: $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle,v_{\mathtt{A}},v_{\mathtt{D}})$.

3) Attack-payoff security with collusion-preservation: Similar to the definition of *attack-payoff secure* in [12], [17], we define collusion preserving attack payoff (CPAP). Intuitively, a protocol is CPAP with respect to an attack model $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle,v_{\mathtt{A}})$ if it enjoys security and collusion-preservation under this model. That is, no attacker can gain more utility from running our protocol, than running the dummy protocol that uses a functionality $\mathcal{F}$ as a resource.

Definition 2 (CPAP)**.** *Let $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle,v_{\mathtt{A}})$ be an attack model and Π a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol that realizes $\langle\mathcal{F}\rangle$. We say that Π is* CPAP *in $\mathcal{M}$ if $\sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Pi,\mathcal{A}) \overset{negl}{\leq} \sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Phi^\mathcal{F},\mathcal{A})$ where $\Phi^\mathcal{F}$ is the dummy $\{\bar{\mathcal{G}},\mathcal{F}\}$-hybrid protocol which forwards all inputs to and outputs from functionality $\mathcal{F}$.*

To complete our framework, we also define ϵ-subgame-perfect equilibrium from [12], and define CPIC similarly to the definition of *incentive compatible (IC)* in [17]. Informally, a strategy profile is an ϵ-subgame-perfect equilibrium if no deviation could improve utilities by more than ϵ. Intuitively, a protocol Π is incentive compatible when even the designer is incentivized to stick with it.

Definition 3 (ϵ-subgame-perfect equilibrium [12])**.** *Let $\mathcal{G}_\mathcal{M}$ be an attack game. A strategy profile $(\Pi,\mathtt{A}(\Pi))$ 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 4 (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 $\mathtt{A} \in \mathtt{ITM}$, $(\Pi,\mathtt{A}(\Pi))$ is a $\nu(\lambda)$-subgame perfect equilibrium on the restricted attack game where the set of deviations of the designer is $\mathbf{\Pi}$.*

B. $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ with incentives

In this section we show that the protocols $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ presented in the previous sections are *CPAP* for a natural class of utilities $u_{\mathtt{A}}$. For simplicity 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., [45]), which we assume is a CP-well-formed functionality. We refer the reader to the full version for a formal definition of $\mathcal{F}^f$. We 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 $\mathcal{A} = \{\mathcal{A}_i\}_{i\in\mathcal{I}}$:

- Define the event $E_{\mathtt{collude}}$ as follows: For some $i \in \mathcal{I}$ and message m, the ith simulator $\mathcal{S}_i$ sends the message $(\mathtt{COLLUDE},\mathsf{sid},m)$ to $\langle\mathcal{F}^f\rangle$.
- Define the event $E_{\mathtt{abort}}$ 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 $\gamma_{\mathtt{collude}}$ the utility for the attacker obtained by triggering $E_{\mathtt{collude}}$. Denote by $\gamma_{\mathtt{abort}}$ the penalty incurred as result of a malicious party being identified by the honest parties as an aborting party. Then, the utility of the attacker is: $u_{\mathtt{A}}(\Pi,\mathcal{A}) = \sup_\mathcal{Z}\left\{\inf_{\mathcal{S}\in\mathcal{C}_\mathcal{A}}\left\{\gamma_{\mathtt{collude}}\Pr[E_{\mathtt{collude}}] - \gamma_{\mathtt{abort}}\Pr[E_{\mathtt{abort}}]\right\}\right\}$

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. In Sec. VI-C, we will discuss a penalization scheme that makes these penalties concrete.

Theorem 3. *Let $\mathcal{F}^f$ be an ideal CP-well-formed functionality, and $\langle\mathcal{F}^f\rangle$ be as defined in Fig. 6. Let $v_{\mathtt{A}}$ be as defined above, for any $\gamma_{\mathtt{collude}}$ and $\gamma_{\mathtt{abort}}$ such that $\gamma_{\mathtt{abort}} > \gamma_{\mathtt{collude}}$. Then the protocol $\Pi^{\mathtt{HT}}$ described in Sec. V (and the protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ described in Sec. V-A) is* CPAP secure *in the attack model $\mathcal{M} = (\mathcal{F}^f,\langle\mathcal{F}^f\rangle,v_{\mathtt{A}})$.*

Proof sketch (see full version for full proof). To prove this theorem we rely on the observation that $\Pi^{\mathtt{HT}}$ ($\Pi^{\mathtt{HT\text{-}FBS}}$) is collusion-preserving for $\mathcal{F}^f$ as long as nobody aborts. Moreover, $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ achieve (publicly) identifiable abort, since the only way subliminally communicate is by

sending a message which is incompatible with the protocol description. Given the way we have set the payoffs, it is always inconvenient for the adversary to trigger the collusion event $E_{\texttt{collude}}$, as it causes the abort event $E_{\texttt{abort}}$.

C. Realizing the incentives

The goal of this section is to create a penalization scheme that translates the utilities defined above to concrete (monetary) values. Below, we describe a simple protocol Π_{pen} that realizes a CP functionality $\mathcal{F}$ assuming no aborts, and disincentivizes aborts with penalization. As we show in Theorem 4 below, Π_{pen} achieves CPAP, and is incentive compatible for the designer (i.e. is CPIC), assuming the existence of one honest party. The protocol assumes a functionality $\mathcal{F}_{\text{pen}}$, which allows each party to deposit a collateral of amount d (a parameter of the functionality). Parties can reclaim their collateral *if and only if* all parties send the functionality a RECLAIM message (for more detail on $\mathcal{F}_{\text{pen}}$, see the full version [1]). In Π_{pen}, honest parties only send RECLAIM if they do not detect an abort during execution of Π^{HT} or $\Pi^{\text{HT-FBS}}$. The protocol Π_{pen} works as follows.

1) Each party sends (DEPOSIT) to $\mathcal{F}_{\text{pen}}$, to deposit a collateral of amount d. If the functionality returns (DEPOSIT, 1), proceed. Otherwise, stop.
2) Run Π^{HT} (resp. $\Pi^{\text{HT-FBS}}$) (possibly multiple times for reactive functionality, using secret sharing to maintain secret intermediate state) on the broadcast channel $\mathcal{B}$.
3) If Π^{HT} (resp. $\Pi^{\text{HT-FBS}}$) did not abort, then all parties send (RECLAIM) to $\mathcal{F}_{\text{pen}}$. $\mathcal{F}_{\text{pen}}$ ensures that parties receive their deposits back if and only if everyone sends (RECLAIM).

We define below a class of utilities for the above protocol, making the arguably natural assumption that the attacker (who chooses the adversarial strategies for all corrupted parties) cares about the sum of deposits lost/gained by all corrupt parties, and similarly, the protocol designer cares about the deposits of all honest parties. Let $E_{\texttt{collude}}$ be the event the simulator sends a COLLUDE message in $\langle\mathcal{F}\rangle$; $E^{\texttt{A}}_{\texttt{deposit}}(t)$ (resp. $E^{\texttt{D}}_{\texttt{deposit}}(m)$) is the event t corrupt (resp. m honest) parties send (DEPOSIT) to $\mathcal{F}_{\text{pen}}$ and $\mathcal{F}_{\text{pen}}$ returns (DEPOSIT, 1); $E^{\texttt{A}}_{\texttt{reclaim}}(t)$ (resp. $E^{\texttt{D}}_{\texttt{reclaim}}(m)$) is the event where t corrupt (resp. m honest) parties receive the message (RECLAIM, 1). Sets $\mathcal{C}_{\mathcal{A}}, \mathcal{S}_{\texttt{A}}, \mathcal{Z}_{\texttt{A}}$ are defined in Section VI-A.

$$u_{\texttt{A}}(\Pi,\mathcal{A}) = \sup_{\mathcal{Z}}\Big\{\inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}}\Big\{\gamma_{\texttt{collude}}\Pr[E_{\texttt{collude}}] - \sum_{t\in[n]} td\cdot\Pr[E^{\texttt{A}}_{\texttt{deposit}}(t)] + \sum_{t\in[n]} td\cdot\Pr[E^{\texttt{A}}_{\texttt{reclaim}}(t)]\Big\}\Big\}$$

$$u_{\texttt{D}}(\Pi,\mathcal{A}) = \inf_{\mathcal{Z}\in\mathcal{Z}_{\texttt{A}}}\Big\{\sup_{\mathcal{S}\in\mathcal{S}_{\texttt{A}}}\Big\{-\gamma_{\texttt{collude}}\Pr[E_{\texttt{collude}}] - \sum_{m\in[n]} md\cdot\Pr[E^{\texttt{D}}_{\texttt{deposit}}(m)] + \sum_{m\in[n]} md\cdot\Pr[E^{\texttt{D}}_{\texttt{reclaim}}(m)]\Big\}\Big\}$$

Informally, we show CPIC and CPAP by proving that the strategy profile $(\Pi_{\text{pen}}, \mathcal{A})$, where the $\mathcal{A}$ is the passive adversarial strategy that just follows Π_{pen} (and does not collude/abort), is an equilibrium solution. We observe that corrupt parties know that if they try to gain more utility by colluding (and thus abort), there is at least one honest party to ensure they lose their collateral, which means they have no incentive to deviate from Π_{pen}. The protocol designer also has no incentive to deviate from Π_{pen} if the adversary is passive, which means this strategy profile is an equilibrium.

Theorem 4 (Proof in [1]). *Let $\mathcal{F}$ be an ideal CP-well-formed functionality, and $\langle\mathcal{F}\rangle$ be as defined in Fig. 6. Let $v_{\texttt{A}}$ and $v_{\texttt{D}}$ be defined in the utility functions above, let $td > \gamma_{\texttt{collude}}$ where t is the number of corrupt parties, and assume there is at least one honest party. Then, the $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol Π_{pen}, where $\mathcal{R}$ is broadcast and $\bar{\mathcal{G}} = T^{\text{HT}}, \mathcal{F}_{pen}$ (resp. $\bar{\mathcal{G}} = T^{\text{HT-FBS}}, \mathcal{F}_{pen}$) :*

- *collusion-preservingly emulates the $\{\bar{\mathcal{G}},\mathcal{F}\}$-exclusive protocol ϕ assuming the adversary does not abort,*
- *is CPAP secure in the attack model $\mathcal{M}$, and*
- *is $\mathbf{\Pi}$-CPIC in $\mathcal{M}$.*

Where $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\texttt{A}}, v_{\texttt{D}})$ and $\mathbf{\Pi}$ is the set of all protocols.

One could implement deposit/reclaim via, e.g., a smart contract-enabled blockchain. We refer to [1] for further discussion on penalization and on how to avoid that also honest parties are penalized in the case a misbehavior is detected.

REFERENCES

[1] M. Ciampi, Y. Lu, and V. Zikas, "Collusion-preserving computation without a mediator," Cryptology ePrint Archive, Report 2020/497, 2020, https://eprint.iacr.org/2020/497.

[2] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," in *14th Annual ACM Symposium on Theory of Computing*. ACM Press, May 1982, pp. 365–377.

[3] M. Lepinski, S. Micali, C. Peikert, and a. shelat, "Completely fair sfe and coalition-safe cheap talk," in *23rd ACM Symposium Annual on Principles of Distributed Computing*, S. Chaudhuri and S. Kutten, Eds. Association for Computing Machinery, Jul. 2004, pp. 1–10.

[4] M. Lepinski, S. Micali, and a. shelat, "Collusion-free protocols," in *37th Annual ACM Symposium on Theory of Computing*, H. N. Gabow and R. Fagin, Eds. ACM Press, May 2005, pp. 543–552.

[5] R. Canetti and M. Vald, "Universally composable security with local adversaries," in *SCN 12: 8th International Conference on Security in Communication Networks*, ser. Lecture Notes in Computer Science, I. Visconti and R. D. Prisco, Eds., vol. 7485. Springer, Heidelberg, Sep. 2012, pp. 281–301.

[6] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. shelat, and I. Visconti, "Collusion-free multiparty computation in the mediated model," in *Advances in Cryptology – CRYPTO 2009*, ser. Lecture Notes in Computer Science, S. Halevi, Ed., vol. 5677. Springer, Heidelberg, Aug. 2009, pp. 524–540.

[7] J. Alwen, a. shelat, and I. Visconti, "Collusion-free protocols in the mediated model," in *Advances in Cryptology – CRYPTO 2008*, ser. Lecture Notes in Computer Science, D. Wagner, Ed., vol. 5157. Springer, Heidelberg, Aug. 2008, pp. 497–514.

[8] J. Alwen, J. Katz, U. Maurer, and V. Zikas, "Collusion-preserving computation," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Heidelberg, Aug. 2012, pp. 124–143.

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

[10] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC 2007: 4th Theory of Cryptography Conference*, ser. Lecture Notes in Computer Science, S. P. Vadhan, Ed., vol. 4392. Springer, Heidelberg, Feb. 2007, pp. 61–85.

[11] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[12] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Rational protocol design: Cryptography against incentive-driven adversaries," in *54th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 2013, pp. 648–657.

[13] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *TCC 2013: 10th Theory of Cryptography Conference*, ser. Lecture Notes in Computer Science, A. Sahai, Ed., vol. 7785. Springer, Heidelberg, Mar. 2013, pp. 477–498.

[14] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *34th Annual ACM Symposium on Theory of Computing*. ACM Press, May 2002, pp. 494–503.

[15] M. Hirt and V. Zikas, "Adaptively secure broadcast," in *Advances in Cryptology – EUROCRYPT 2010*, ser. Lecture Notes in Computer Science, H. Gilbert, Ed., vol. 6110. Springer, Heidelberg, May / Jun. 2010, pp. 466–485.

[16] J. A. Garay, J. Katz, R. Kumaresan, and H.-S. Zhou, "Adaptively secure broadcast, revisited," in *30th ACM Symposium Annual on Principles of Distributed Computing*, C. Gavoille and P. Fraigniaud, Eds. Association for Computing Machinery, Jun. 2011, pp. 179–186.

[17] C. Badertscher, J. A. Garay, U. Maurer, D. Tschudi, and V. Zikas, "But why does it work? A rational protocol design treatment of bitcoin," in *Advances in Cryptology – EUROCRYPT 2018, Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, Heidelberg, Apr. / May 2018, pp. 34–65.

[18] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology – CRYPTO 2014, Part II*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617. Springer, Heidelberg, Aug. 2014, pp. 421–439.

[19] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 443–458.

[20] R. Kumaresan and I. Bentov, "Amortizing secure computation with penalties," in *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM Press, Oct. 2016, pp. 418–429.

[21] ——, "How to use bitcoin to incentivize correct computations," in *ACM CCS 2014: 21st Conference on Computer and Communications Security*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM Press, Nov. 2014, pp. 30–41.

[22] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Advances in Cryptology – EUROCRYPT 2016, Part II*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Heidelberg, May 2016, pp. 705–734.

[23] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, I. Ray, N. Li, and C. Kruegel, Eds. ACM Press, Oct. 2015, pp. 195–206.

[24] I. Bentov, R. Kumaresan, and A. Miller, "Instantaneous decentralized poker," in *Advances in Cryptology – ASIACRYPT 2017, Part II*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10625. Springer, Heidelberg, Dec. 2017, pp. 410–440.

[25] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "One-time programs," in *Advances in Cryptology – CRYPTO 2008*, ser. Lecture Notes in Computer Science, D. Wagner, Ed., vol. 5157. Springer, Heidelberg, Aug. 2008, pp. 39–56.

[26] J. Katz, "Universally composable multi-party computation using tamper-proof hardware," in *Advances in Cryptology – EUROCRYPT 2007*, ser. Lecture Notes in Computer Science, M. Naor, Ed., vol. 4515. Springer, Heidelberg, May 2007, pp. 115–128.

[27] J. Alwen, R. Ostrovsky, H.-S. Zhou, and V. Zikas, "Incoercible multi-party computation and universally composable receipt-free voting," in *Advances in Cryptology – CRYPTO 2015, Part II*, ser. Lecture Notes in Computer Science, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9216. Springer, Heidelberg, Aug. 2015, pp. 763–780.

[28] T. Moran and G. Segev, "David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware," in *Advances in Cryptology – EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., vol. 4965. Springer, Heidelberg, Apr. 2008, pp. 527–544.

[29] N. Döttling, D. Kraschewski, and J. Müller-Quade, "Unconditional and composable security using a single stateful tamper-proof hardware token," in *TCC 2011: 8th Theory of Cryptography Conference*, ser. Lecture Notes in Computer Science, Y. Ishai, Ed., vol. 6597. Springer, Heidelberg, Mar. 2011, pp. 164–181.

[30] I. Damgård, J. B. Nielsen, and D. Wichs, "Universally composable multiparty computation with partially isolated parties," in *TCC 2009: 6th Theory of Cryptography Conference*, ser. Lecture Notes in Computer Science, O. Reingold, Ed., vol. 5444. Springer, Heidelberg, Mar. 2009, pp. 315–331.

[31] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia, "Founding cryptography on tamper-proof hardware tokens," in *TCC 2010: 7th Theory of Cryptography Conference*, ser. Lecture Notes in Computer Science, D. Micciancio, Ed., vol. 5978. Springer, Heidelberg, Feb. 2010, pp. 308–326.

[32] T. Nilges, "The cryptographic strength of tamper-proof hardware," Ph.D. dissertation, Karlsruhe Institute of Technology, 2015.

[33] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "IRON: Functional encryption using intel SGX," in *ACM CCS 2017: 24th Conference on Computer and Communications Security*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 765–782.

[34] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from SGX," in *FC 2017: 21st International Conference on Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, A. Kiayias, Ed., vol. 10322. Springer, Heidelberg, Apr. 2017, pp. 477–497.

[35] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi, "Foundations of hardware-based attested computation and application to sgx," in *2016 IEEE European Symposium on Security and Privacy (EuroSP)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2016, pp. 245–260. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EuroSP.2016.28

[36] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[37] R. Canetti, A. Jain, and A. Scafuro, "Practical UC security with a global random oracle," in *ACM CCS 2014: 21st Conference on Computer and Communications Security*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM Press, Nov. 2014, pp. 597–608.

[38] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," in *Advances in Cryptology – CRYPTO 2014, Part II*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617. Springer, Heidelberg, Aug. 2014, pp. 369–386.

[39] R. Cohen, J. A. Garay, and V. Zikas, "Broadcast-optimal two-round MPC," in *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds., vol. 12106. Springer, 2020, pp. 828–858. [Online]. Available: https://doi.org/10.1007/978-3-030-45724-2_28

[40] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, Jan. 2000.

[41] J. B. Nielsen, *On protocol security in the cryptographic model*. Citeseer, 2003.

[42] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, 2016, http://eprint.iacr.org/2016/086.

[43] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, p. 770772, Nov. 1981. [Online]. Available: https://doi.org/10.1145/358790.358797

[44] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422426, Jul. 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[45] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, Apr. 2009.