Insured MPC: Efficient Secure Computation with Financial Penalties

Carsten Baum[1⋆], Bernardo David[2], and Rafael Dowsley[3⋆⋆]

[1] Aarhus University, Denmark
[2] IT University of Copenhagen, Denmark
[3] Bar-Ilan University, Israel

Abstract. Fairness in Secure Multiparty Computation (MPC) is known to be impossible to achieve in the presence of a dishonest majority. Previous works have proposed combining MPC protocols with cryptocurrencies in order to financially punish aborting adversaries, providing an incentive for parties to honestly follow the protocol. The focus of existing work is on proving that this approach is possible and unfortunately they present monolithic and mostly inefficient constructions. In this work, we put forth the first UC secure modular construction of "Insured MPC", where either the output of the private computation (which describes how to distribute funds) is fairly delivered or a proof that a set of parties has misbehaved is produced, allowing for financial punishments. Moreover, both the output and the proof of cheating are publicly verifiable, allowing third parties to independently validate an execution. We present an efficient compiler that implements Insured MPC from an MPC protocol with certain properties, a standard (non-private) Smart Contract and a publicly verifiable homomorphic commitment scheme. As an intermediate step, we propose the first construction of a publicly verifiable homomorphic commitment scheme with composability guarantees.

1 Introduction

Secure Multiparty Computation (MPC) allows a set of mutually distrusting parties to evaluate an arbitrary function on secret inputs. The participating parties learn nothing beyond the output of the computation, while malicious behavior at runtime does not alter the output. An intuitive and in practice often required feature of MPC is that if a cheating party obtains the output, then all the honest parties should do so as well. Protocols which guarantee this are also called *fair*. In his seminal work, Cleve [18] proved that fair MPC with a dishonest majority is impossible to achieve in the standard communication model. While the result can be circumvented for specific functions [3, 4, 22] in the two-party setting, the result of [18] prevents MPC from being applicable in certain interesting scenarios.

⋆ Part of this work was done while the author was with Bar-Ilan University.
⋆⋆ Part of this work was done while the author was with Aarhus University.

With the advent of cryptocurrencies, [2, 10] initiated a line of research that avoids the aforementioned drawback by imposing financial penalties on misbehaving parties. Such monetary punishments would then incentivize fair behavior of the protocol participants, assuming that they are rational and that the penalties are high enough. This is achieved by constructing a protocol which interacts with a public ledger and digital currency. The overall structure of their idea is as follows: (i) The parties run the secure computation, but delay the reconstruction of the output; (ii) Each party deposits a collateral on the public ledger; (iii) The parties reconstruct the output. Each party obtains the collateral back if it can prove that it behaved honestly during the reconstruction; and (iv) If some parties have cheated, then their share of the collateral is distributed among the honest participants. Several works [27, 29, 30] generalized this concept and improved the performance with respect to the amount of interaction with the public ledger as well as the collateral that each party needs to deposit. In particular, Kumaresan et al. [1, 2, 30] introduced the idea of MPC with cash distribution, in which the inputs and outputs of the parties consist of both data and money. In this latter case, the public ledger is used both to enforce financial penalties as well as to distribute money according to the output of the secure computation.

1.1 Related Works: Fair Computation vs. Fair Output Delivery

Before presenting our techniques and design choices, it is worthwhile to discuss first *which* adversarial behavior should be punishable: it is possible to obtain protocols that punish deviations at any point of their execution or protocols that only punish adversaries who learn the output but prevent the honest parties from learning it. In this second approach, adversaries that abort the protocol but do not learn the output are not punished. One therefore has to distinguish between two types of protocols: those that punish all cheating yield *Fair Computation with Penalties*, while the second approach only allows *Fair Output Delivery with Penalties*. One can roughly classify the state-of-the-art using this distinction, which we further explore in Section 5.

Fair Computation. [2] and [30] follow this line of work, but have high round and communication complexities overheads. As [27] correctly pointed out, care must be taken when choosing the "inner" MPC protocol (which is compiled to obtain financial penalties): to achieve this, the protocol must have a property called *Identifiable Abort* (ID-MPC, [25]). As [2, 30] use GMW [21], their specific construction achieves this property, but not every MPC protocol is suitable for their approach. On the other hand, [27] requires constant rounds but rely on expensive generic zero knowledge proofs to achieve the necessary properties.

Fair Output Delivery. This line of work has been independently initiated by [1, 10] and continued in [11, 28, 29, 31]. Most of the protocols in this line of work still require several rounds of interaction with the public ledger as well as storing all MPC protocol messages on the ledger. The currently most efficient approaches [11, 31] rely on an "inner" MPC protocol that performs the actual computation and then secret shares the result, outputting not the result itself

but commitments to each of the shares and privately giving to each party the opening for one of these commitments. The parties subsequently post all (closed) commitments to the public ledger. After the parties agree that the commitments posted on the public ledger correspond to those obtained from the MPC protocol, each party opens its commitment in public. This implicitly has identifiable abort because all parties can publicly agree if another participant has failed to post a valid opening to its commitment on the ledger. In particular, the approach of [11] relies on a smart contract that punishes parties that fail to post valid openings for their commitments to shares. However, a caveat, both from a theoretical and practical point of view, is that current protocols compute both the secret sharing of the result and the commitments to each share inside the MPC in a white-box way, which adds significant computational and communication overheads. Moreover, in order to achieve composable security, the expensive preprocessing phase of a composable commitment scheme would have to be executed as part of the circuit computed by the "inner" MPC protocol.

Other Related Work. Recently Choudhuri et al. [17] constructed fair MPC using a Bulletin Board but relying on stronger assumptions (Witness Encryption or Trusted Hardware). MPC on *permissioned* ledgers has been suggested in [9] but requires all messages to be posted on a public ledger for verification and does not support financial penalties. MPC with public verification such as [6] requires high bulletin board storage that is unsuitable for smart contracts. ID-MPC without public verifiability has been constructed in e.g. [25].

Composability and Efficiency With the exception of [27], none of the previous works have been shown to achieve composability guarantees. However, the approach of [27] incurs very high computational and communication overheads, since it compiles an "inner" MPC protocol to achieve identifiable abort and public verifiability by using expensive generic zero-knowledge proofs. Even the previous works that do not achieve composability [1, 2, 10, 28, 29, 30, 31] incur high round and communication complexities overheads, since they require non-constant extra rounds for each round of the MPC protocol in order to implement financial penalties. While the approach of [11] circumvents the need for such extra rounds by relying on a smart contract, it introduces overheads by requiring secret sharing and commitment schemes to be computed as part of the circuit evaluated by the MPC protocol after the actual function that is evaluated.

1.2 Our Contributions

In this work, we give the first universally composable modular construction of MPC achieving fair output delivery with financial penalties that can be instantiated with a concretely efficient protocol. While previous works have focused at obtaining protocols that can be instantiated using the Bitcoin or Ethereum blockchains as a public ledger, we focus instead on the MPC aspects of such constructions. We design a protocol from generic building blocks with security analysed in the Global Universal Composability framework (GUC). This modular approach directly pinpoints the properties that the "inner" MPC protocol

and other underlying protocols must have in such constructions, including precise definitions of the necessary public verifiability properties. Besides shedding light on theoretical aspects of MPC with fair output delivery with penalties, our approach also paves the way for concrete implementations, since it uses generic building blocks that have highly efficient instantiations and combines them in a way that yields highly efficient constructions. Moreover, due to its modular nature, our protocol directly benefits from any future efficiency improvements to its building blocks (*i.e.* the more efficient publicly verifiable additively homomorphic commitments recently introduced in [**?**]).

Linearly Homomorphic Commitments with Delayed Public Verifiability. This primitive acts as the central hub of our construction. Such commitment schemes are additively homomorphic, allowing one to open linear combinations of commitments without revealing the individual commitments themselves. Moreover, they allow for any third party to verify that a message is a valid opening for a given commitment. We remark that existing constructions achieving all of these properties do not have composability guarantees.

Modular Design. Based on a multiparty additively homomorphic commitment scheme with delayed public verifiability and a suitable "inner" MPC, we give a modular approach for constructing "Insured MPC": first, we combine the inner MPC with the commitment scheme to achieve *MPC with publicly verifiable output*. In this step, we leverage a property of the inner MPC output phase to avoid computing secret sharing or commitments inside the MPC itself, instead computing commitments before the actual output is revealed. Given a (non-private) Smart Contract functionality and a global clock we can then construct a cheater identifiable output reconstruction phase in a modular way where the Smart Contract mediates the reconstruction, receiving openings to the commitments obtained in the previous step. In case of disagreement during reconstruction, the Smart Contract can identify the cheaters as the parties who failed to provide commitment openings. This reconstruction phase and posterior public verification of the resulting output are mostly light-weight due to our commitment scheme, which allows for verification of openings using only calls to a Pseudorandom Generator. Our technique adds no overhead to the circuit being computed inside the MPC (differently from [11, 31]) and little overhead to the MPC protocol (differently from [27]), since each party only computes and posts to the public ledger a number of commitments linear in the output size.

Efficient Instantiation. We show how to instantiate all sub-protocols efficiently. We modify the constant-round MPC of [23] to work as the "inner" MPC while essentially keeping the same concrete efficiency. Our publicly verifiable additively homomorphic commitment scheme only performs Random Oracle calls after a small number of base Oblivious Transfers (OT) using a publicly verifiable OT scheme, achieving the same concrete efficiency as the non-publicly verifiable scheme of [19]. As we use a restricted programmable and observable global RO [12] we are then still able to prove security of all steps in GUC.

Full Version. In this extended abstract we only provide an overview of our construction, while the complete building blocks, protocols and proofs appear in the full version, which can be found in [7].

2 Preliminaries

Let $y \xleftarrow{\$} F(x)$ denote running the randomized algorithm F on input x yielding output y. Similarly, $y \leftarrow F(x)$ is used if F is deterministic. For a set $\mathcal{X}$, let $x \xleftarrow{\$} \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$. For any $k \in \mathbb{N}$ we write $[k]$ for the set $\{1, \ldots, k\}$. Let n be the number of parties in an MPC scheme, $\mathcal{A}$ be an adversary and $\mathcal{S}$ the ideal-world simulator. $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ denotes the set of parties where $I \subsetneq \mathcal{P}$ are the corrupted and $\overline{I} = \mathcal{P} \setminus I$ the uncorrupted parties. τ denotes the computational and κ the statistical security parameter. As we focus on MPC over $\mathbb{F}_2$ we use $\mathbb{F}$ for conciseness. In this work, the (G)UC framework [13, 14] is used to analyze security. We refer interested readers to the aforementioned works for more details. Several functionalities in this work allow *public verifiability*. To model this, we follow the approach of [5] and allow the set of verifiers $\mathcal{V}$ to be dynamic by adding register and de-register instructions as well as instructions that allow $\mathcal{S}$ to obtain the list of registered verifiers. Functionalities with public verifiability include the (de-)registration interfaces, which are omitted in the descriptions for simplicity. Due to space constraints, the mentioned interfaces are fully described in the full version.

We focus on *Secure Multiparty Computation* with security against a static, rushing and malicious adversary $\mathcal{A}$ corrupting up to $n-1$ of the n parties and introduce the functionality $\mathcal{F}_{\mathsf{Online}}$ which we realize in this work. This functionality, as depicted in Figure 1, realizes what we call *MPC with Punishable Abort* or *Insured MPC*: $\mathcal{F}_{\mathsf{Online}}$ computes the result $\boldsymbol{y}$ honestly, but will only output it if every party $\mathcal{P}_i$ sent coins $\mathsf{coins}(d)$ to it first. $\mathcal{F}_{\mathsf{Online}}$ hands these coins back if everyone obtains $\boldsymbol{y}$. $\mathcal{A}$ can withhold the output from honest parties, but only at the expense of losing its provided coins. In case of no cheating, $\mathcal{F}_{\mathsf{Online}}$ redistributes additional coins based on $\boldsymbol{y}$ based on a *Cash Distribution Function*.

Definition 1 (Cash Distribution Function). *Let $g : \mathbb{F}^m \times \mathbb{N}^n \to \mathbb{N}^n$ be such that $\forall \boldsymbol{y} \in \mathbb{F}^m, t^{(1)}, \ldots, t^{(n)} \in \mathbb{N}$ it holds that $\sum_i t^{(i)} = \sum_i e^{(i)}$ for $(e^{(1)}, \ldots, e^{(n)}) \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$. Then g is called a* Cash Distribution Function.

Observe that our functionality $\mathcal{F}_{\mathsf{Online}}$ allows $\mathcal{A}$ to *delay* the delivery of the correct output by some time, which is necessary for technical reasons. $\mathcal{F}_{\mathsf{Online}}$ is defined in the presence of a GUC functionality $\mathcal{F}_{\mathsf{Clock}}$ which we will not fully specify here (we use the version of [26]). $\mathcal{F}_{\mathsf{Clock}}$ provides a counter readable consistently by all parties which progresses if all honest parties send a tick signal. One would obviously like to get a result in terms of wall-clock time, but this is difficult to specify in UC. We implement $\mathcal{F}_{\mathsf{Online}}$ using a Smart Contract functionality which could emulate wall-clock time to a certain extent. We will furthermore later make use of a coin-flipping functionality $\mathcal{F}_{\mathsf{CT}}$ which outputs unbiased random bits to all parties. Both functionalities are provided in the full version.

Functionality $\mathcal{F}_{\mathsf{Online}}$ interacts with the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ as well as the global functionality $\mathcal{F}_{\mathsf{Clock}}$. This functionality is parameterized by a circuit C representing the computation with an output of length m, the compensation amount q, the security deposit $d \geq (n-1)q$ and a cash distribution function g. $\mathcal{S}$ specifies a set $I \subset [n]$ of corrupted parties.

Input: Upon first input $(\text{INPUT}, sid, i, x^{(i)})$ by $\mathcal{P}_i$ and $(\text{INPUT}, sid, i, \cdot)$ by all other parties the functionality stores the value $(i, x^{(i)})$ internally. Every further such message with the same sid and i is ignored.

Evaluate: Upon input $(\text{COMPUTE}, sid)$ by all parties and if the inputs $(i, x^{(i)})_{i \in [n]}$ for all parties have been received, compute $\boldsymbol{y} = C(x^{(1)}, \ldots, x^{(n)})$. If $\mathcal{S}$ sends (ABORT, sid) during **Input** or **Evaluate** then send (ABORT, sid) to all parties and stop.

Deposit: Wait for each party $\mathcal{P}_i$ to send $(\text{DEPOSIT}, sid, \mathsf{coins}(d + t^{(i)}))$ containing the d coins of the security deposit as well as the $t^{(i)} \geq 0$ coins that $\mathcal{P}_i$ wants to use as financial input in the computation. Send $(\text{DEPOSITED}, sid, \mathcal{P}_i, d + t^{(i)})$ to $\mathcal{S}$ upon receiving it. If all honest parties sent their deposit then send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$. Then query $\mathcal{F}_{\mathsf{Clock}}$ until $\nu = 1$. If by $\nu = 1$ some parties $j \in I$ sent $\mathsf{coins}(c^{(j)})$ with $c^{(j)} < d$ then return the collateral to all honest parties and $\mathcal{S}$. Afterwards send (ABORT, sid) to the honest parties and abort. If all went ok, then activate **Reveal**.

Reveal: Send $(\text{OUTPUT}, sid, \boldsymbol{y})$ to $\mathcal{S}$, (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$ and wait until $\nu = 2$. $\mathcal{S}$ may now either send $(\text{NO-OUTPUT}, sid)$ or $(\text{OK}, sid, \boldsymbol{y})$. Afterwards send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$ and activate **Resolve**.

Resolve: Query $\mathcal{F}_{\mathsf{Clock}}$ until $\nu = 3$. Then send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$ and query until $\nu = 4$.

1. Wait for the message $(\text{PUNISH}, sid, \texttt{punish})$ from $\mathcal{S}$ where $\texttt{punish} \subseteq I$. If $\mathcal{S}$ sent $(\text{NO-OUTPUT}, sid, \boldsymbol{y})$ in **Reveal** then $\emptyset \neq \texttt{punish}$.
2. Depending on $\texttt{punish}$ do the following:
   - If $\texttt{punish} = \emptyset$ then compute $e^{(1)}, \ldots, e^{(n)} \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$.
   - Otherwise set $e^{(i)} \leftarrow d + t^{(i)} + |\texttt{punish}| \cdot q$ for each party $\mathcal{P}_i \in \mathcal{P} \setminus \texttt{punish}$ and $e^{(i)} \leftarrow d - q \cdot (n - |\texttt{punish}|) + t^{(i)}$ for each $\mathcal{P}_i \in \texttt{punish}$.
3. For each $\mathcal{P}_i \in \mathcal{P}$ send $(\text{PAYOUT}, sid, \mathcal{P}_i, \mathsf{coins}(e^{(i)}))$ to $\mathcal{P}_i$ and $(\text{PAYOUT}, sid, \mathcal{P}_i, e^{(i)})$ to each other party.
4. If $\mathcal{S}$ sent $(\text{OK}, sid, \boldsymbol{y})$ in **Reveal** then send $(\text{OUTPUT}, sid, \boldsymbol{y})$ to each honest party, otherwise send $(\text{NO-OUTPUT}, sid)$.

Fig. 1. Functionality $\mathcal{F}_{\mathsf{Online}}$ for Secure Multiparty Computation with Punishable Abort and Cash Distribution.

3 The Building Blocks

In this section we will introduce the different building blocks for our construction.

Linearly Homomorphic Commitments with Delayed Public Verifiability. A crucial building block of our Insured MPC protocol is the multiparty commitment functionality $\mathcal{F}_{\mathsf{HCom}}$ that is additively homomorphic and allows delayed public verifiability (i.e., after the opening phase it is possible for any third party to verify the opening information). This functionality is depicted in Figure 3.

$\mathcal{F}_{\mathsf{HCom}}$ is GUC-realized with security in the restricted programmable and observable RO model of Camenisch et al. [12] using multiple building blocks as depicted in Figure 2 and sketched below. The full construction including a proof of security can be found in the full version. First, we realize a simple (non-homomorphic) commitment functionality with public verifiability $\mathcal{F}_{\mathsf{Com}}$ by observing that the canonical RO based commitment scheme shown to be UC-secure in [12] is trivially publicly verifiable.

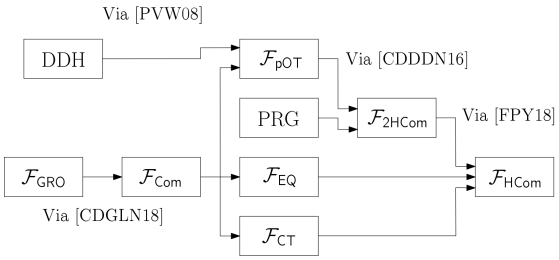

Fig. 2. The Building Blocks of the Additively Homomorphic Multiparty Commitment with Public Verifiability.

$\mathcal{F}_{\mathsf{Com}}$ is then used to realize a publicly verifiable equality testing functionality $\mathcal{F}_{\mathsf{EQ}}$ and a publicly verifiable coin tossing functionality $\mathcal{F}_{\mathsf{CT}}$. These functionalities are versions of the functionalities in Frederiksen et al. [19] that are augmented to allow public verifiability. We also use an oblivious transfer functionality with delayed public verifiability $\mathcal{F}_{\mathsf{pOT}}$ in which the receiver can activate an interface that allows any party to verify that the receiver used a given choice bit and received a given message. We show that $\mathcal{F}_{\mathsf{pOT}}$ can be realized using $\mathcal{F}_{\mathsf{Com}}$ and the DDH-based OT protocol of Peikert et al. [33]. A two-party homomorphic commitment with delayed public verifiability functionality $\mathcal{F}_{\mathsf{2HCom}}$ is then realized with a construction based on the scheme of Cascudo et al. [16], which we augment to achieve public verifiability by leveraging $\mathcal{F}_{\mathsf{pOT}}$. Finally, $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$ are used to obtain a public verifiable version of the protocol of Frederiksen et al. [19], yielding a protocol that realizes the additively homomorphic multiparty commitment functionality with public verifiability $\mathcal{F}_{\mathsf{HCom}}$.

MPC with Secret-Shared Output. In our construction we depart from a flavor of MPC that provides partial outputs which can be used to reconstruct the final output through linear operations, which is captured precisely in functionality $\mathcal{F}_{\mathsf{MPC-SO}}$ in Figure 4. This functionality provides a secret-sharing of the output value: given all shares, any party can use it to obtain the output value while even $n-1$ shares do not reveal any information about it. To reconstruct, a special function f for the reconstruction process must be used. We call this function f a *Reconstruction Function*, whose definition and use was already implicit in previous work [24, 34].

Definition 2 (Reconstruction Function). *Let $f : (\mathbb{F}^m)^{n+1} \to \mathbb{F}^m$ be a function. We call f a reconstruction function if for all $\bar{\boldsymbol{y}} \in \mathbb{F}^m$, for all $i \in [n]$ and for all $\boldsymbol{s}^{(1)}, \ldots \boldsymbol{s}^{(n-1)} \in \mathbb{F}^m$, the induced function $\hat{f}_i : \mathbb{F}^m \to \mathbb{F}^m$ such that $\hat{f}_i(\cdot) = f(\bar{\boldsymbol{y}}, \boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(i-1)}, \cdot, \boldsymbol{s}^{(i)}, \ldots, \boldsymbol{s}^{(n-1)})$ is a bijection which is poly-time computable in both directions.*

Functionality $\mathcal{F}_{\mathsf{HCom}}$ is parameterized by $k \in \mathbb{N}$ and interacts with a set of parties $\mathcal{P}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ (who may abort at any time).

Init: Upon receiving (INIT, sid) from parties $\mathcal{P}$, initialize empty lists raw and actual.

Commit: Upon receiving (COMMIT, $sid, \mathcal{I}$) from $\mathcal{P}_i \in \mathcal{P}$ where $\mathcal{I}$ is a set of unused identifiers, for every $cid \in \mathcal{I}$, sample a random $\boldsymbol{x}_{cid} \xleftarrow{\$} \mathbb{F}^k$, set $\mathsf{raw}[cid] = \boldsymbol{x}_{cid}$ and send (COMMIT-RECORDED, $sid, \mathcal{I}$) to all parties $\mathcal{P}$ and $\mathcal{S}$.

Input: Upon receiving a message (INPUT, $sid, \mathcal{P}_i, cid, \boldsymbol{y}$) from $\mathcal{P}_i \in \mathcal{P}$ and messages (INPUT, $sid, \mathcal{P}_i, cid$) from every party in $\mathcal{P}$ other than $\mathcal{P}_i$, if a message (COMMIT, $sid, \mathcal{I}$) was previously received from $\mathcal{P}_i$ and $\mathsf{raw}[cid] = \boldsymbol{x}_{cid} \neq \perp$, set $\mathsf{raw}[cid] = \perp$, set $\mathsf{actual}[cid] = \boldsymbol{y}$ and send (INPUT-RECORDED, $sid, \mathcal{P}_i, cid$) to all parties in $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast (ABORT, sid) and halt.

Random: Upon receiving (RANDOM, sid, cid) from all parties $\mathcal{P}$, if $\mathsf{raw}[cid] = \boldsymbol{x}_{cid} \neq \perp$, set $\mathsf{actual}[cid] = \boldsymbol{x}_{cid}$, set $\mathsf{raw}[cid] = \perp$ and send (RANDOM-RECORDED, sid, cid) to all parties $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast (ABORT, sid) and halt.

Linear Combination: Upon receiving (LINEAR, $sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid'$) where all $\alpha_{cid} \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^k$ from all parties $\mathcal{P}$, if $\mathsf{actual}[cid] = \boldsymbol{x}_{cid} \neq \perp$ for all $cid \in \mathcal{I}$ and $\mathsf{raw}[cid'] = \mathsf{actual}[cid'] = \perp$, set $\mathsf{actual}[cid'] = \boldsymbol{\beta} + \sum_{cid \in \mathcal{I}} \alpha_{cid} \cdot \boldsymbol{x}_{cid}$ and send (LINEAR-RECORDED, $sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid'$) to all parties $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast (ABORT, sid) and halt.

Open: Upon receiving (OPEN, sid, cid) from all parties $\mathcal{P}$, if $\mathsf{actual}[cid] = \boldsymbol{x}_{cid} \neq \perp$, send (OPEN, $sid, cid, \boldsymbol{x}_{cid}$) to $\mathcal{S}$. If $\mathcal{S}$ does not abort, send (OPEN, $sid, cid, \boldsymbol{x}_{cid}$) to all parties $\mathcal{P}$.

Check Opening: Upon receiving (CHECK-NOT-OPEN, sid, cid) from $\mathcal{P}_i \in \mathcal{P} \cup \mathcal{V}$, if parties $\{\hat{p}_1, \ldots, \hat{p}_k\} \subset \mathcal{P}$ did not send (OPEN, sid, cid), send (CHECK-NOT-OPEN, $sid, \{\hat{p}_1, \ldots, \hat{p}_k\}$) to $\mathcal{P}_i$.

Initialize Verification: Upon receiving a message (VERIFICATION-START, $sid, \mathcal{P}_i$) from a party $\mathcal{P}_i \in \mathcal{P}$, send (VERIFICATION-START, $sid, \mathcal{P}_i$) to all parties $\mathcal{P}$ and $\mathcal{V}$ and ignore all messages with this sid in all other interfaces but messages (CHECK-NOT-OPEN, sid, cid) in the Check Opening interface and messages (VERIFY, $sid, cid, \boldsymbol{x}'_{cid}$) in the Public Verification interface.

Public Verification: Upon receiving (VERIFY, $sid, cid, \boldsymbol{x}'_{cid}$) from a party $\mathcal{V}_j \in \mathcal{V}$, if a set of parties $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_m\} \subseteq \mathcal{P}$ has not sent a message (VERIFICATION-START, sid), send (VERIFY-FAIL, $sid, \{\mathcal{P}'_1, \ldots, \mathcal{P}'_m\}$) to $\mathcal{V}_j$. Otherwise, if a message (OPEN, sid, cid) has been received from all parties $\mathcal{P}$ and $\mathsf{actual}[cid] = \boldsymbol{x}_{cid} = \boldsymbol{x}'_{cid}$, set $f = 1$ (otherwise set $f = 0$) and send (VERIFIED, sid, cid, f) to $\mathcal{V}_j$.

Fig. 3. Functionality $\mathcal{F}_{\mathsf{HCom}}$ For Homomorphic Multiparty Commitment With Delayed Public Verifiability

$\mathcal{F}_{\mathsf{MPC-SO}}$ can be efficiently realized, for instance, by a slightly modified version of the constant-round preprocessed BMR protocol of Hazay et al. [23] which we provide in the full version of this work.

The Smart Contract. Central to our solution for financially fair output delivery is the smart contract functionality $\mathcal{F}_{\mathsf{SC}}$ which is described in Figure 5. This is a Global UC-functionality, meaning that other functionalities can contact it (as we will see later). It is defined with respect to the global clock $\mathcal{F}_{\mathsf{Clock}}$ although it would be possible to include this into $\mathcal{F}_{\mathsf{SC}}$ already.

Functionality $\mathcal{F}_{\mathsf{MPC-SO}}$ interacts with the parties $\mathcal{P}$ and is parametrized by a circuit C with inputs $x^{(1)}, \ldots, x^{(n)}$ and output $\boldsymbol{y} = (y_1, \ldots, y_m) \in \mathbb{F}^m$. $\mathcal{S}$ provides a set $I \subset [n]$ of corrupt parties and can at any point send (ABORT, sid) to $\mathcal{F}_{\mathsf{MPC-SO}}$, which in turn sends (ABORT, $sid, \perp$) to $\mathcal{P}$ and terminates. Let the reconstruction function f be the XOR over $\mathbb{F}$.

Input: Upon input (INPUT, $sid, i, x^{(i)}$) by $\mathcal{P}_i$ and input (INPUT, $sid, i, \cdot$) by all other parties the functionality stores the value $(sid, i, x^{(i)})$ internally. Every further such message with the same sid and i is ignored.

Evaluate: Upon input (COMPUTE, sid) by all parties in $\mathcal{P}$ and if the inputs $(sid, i, x^{(i)})_{i\in[n]}$ for all parties have been stored internally, compute $\boldsymbol{y} = (y_1, \ldots, y_m) \leftarrow C(x^{(1)}, \ldots, x^{(n)})$ and store $(sid, \boldsymbol{y})$ locally.

Share Output: Upon input (SHARE-OUTPUT, sid) and if **Evaluate** was finished:

1. For each $h \in [m]$, pick an unused cid_h and send (REQUEST-SHARES, $sid, \{cid_h\}_{h\in[m]}$) to $\mathcal{S}$. For each $i \in I$, $\mathcal{S}$ sends (OUTPUT-SHARES, $sid, \{(cid_h, s^{(i)}_{cid_h})\}_{h\in[m]}$). Then for $i \in \overline{I}$ sample $s^{(i)}_{cid_h} \xleftarrow{\$} \mathbb{F}$, store $(sid, cid_h, i, s^{(i)}_{cid_h})$ and send (OUTPUT-SHARES, $sid, \{(cid_h, s^{(i)}_{cid_h})\}_{h\in[m]}$) to $\mathcal{P}_i$.
2. For each $h \in [m]$, sample $\overline{z_{cid_h}} \in \mathbb{F}$ such that $f(\overline{z_{cid_h}}, s^{(1)}_{cid_h}, \ldots, s^{(n)}_{cid_h}) = y_h$ and store $(sid, cid_h, \overline{z_{cid_h}})$. Send (SHARE-ADVICES, $sid, \{(cid_h, \overline{z_{cid_h}})\}_{h\in[m]}$) to $\mathcal{S}$. If $\mathcal{S}$ sends (DELIVER-ADVICES, $sid, \{cid_h\}_{h\in[m]}$), then send (SHARE-ADVICES, sid, $\{(cid_h, \overline{z_{cid_h}})\}_{h\in[m]}$) to all $\mathcal{P}_i \in \overline{I}$.

Share Random Value: Upon input (SHARE-RANDOM, sid), pick $z \xleftarrow{\$} \mathbb{F}$ and an unused cid, set $\overline{z_{cid}} = 0$ and send (REQUEST-SHARES, sid, cid) to $\mathcal{S}$. For each $i \in I$, $\mathcal{S}$ sends (SHARE, $sid, cid, s^{(i)}_{cid}$). Then sample $s^{(i)}_{cid} \xleftarrow{\$} \mathbb{F}$ for $i \in \overline{I}$ such that $z = f(\overline{z_{cid}}, s^{(1)}_{cid}, \ldots, s^{(n)}_{cid})$, store $(sid, cid, i, s^{(i)}_{cid})$ and send (SHARE, $sid, cid, s^{(i)}_{cid}$) to $\mathcal{P}_i$.

Linear Combination: Upon input (LINEAR, $sid, \{(cid, \alpha_{cid})\}_{cid\in\mathcal{I}}, cid'$) from all parties $\mathcal{P}$, if all $\alpha_{cid} \in \mathbb{F}$, all $cid \in \mathcal{I}$ have stored values and cid' is unused, set $s^{(i)}_{cid'} \leftarrow \sum_{cid\in\mathcal{I}} \alpha_{cid} \cdot s^{(i)}_{cid}$ for each $i \in [n]$, $\overline{z_{cid'}} \leftarrow \sum_{cid\in\mathcal{I}} \alpha_{cid} \cdot \overline{z_{cid}}$, record $\{(sid, cid', i, s^{(i)}_{cid'})\}_{i\in[n]}$, $(sid, cid', \overline{z_{cid'}})$, and send (LINEAR-RECORDED, $sid, \{(cid, \alpha_{cid})\}_{cid\in\mathcal{I}}, cid'$) to all parties $\mathcal{P}$ and $\mathcal{S}$.

Reveal: Upon input (REVEAL, sid, cid, i) by $\mathcal{P}_i$, send (REVEAL, $sid, cid, i, s^{(i)}_{cid}$) to $\mathcal{S}$. If $\mathcal{S}$ sends (DELIVER-REVEAL, sid, cid, i), send (REVEAL, $sid, cid, i, s^{(i)}_{cid}$) to all parties.

Private Reveal: Upon input (REVEAL, sid, cid, i, j) by $\mathcal{P}_i$:

- if $\mathcal{P}_i \in I$ or $\mathcal{P}_j \in I$ then send (REVEAL, $sid, cid, i, s^{(i)}_{cid}$) to $\mathcal{S}$. If $\mathcal{S}$ sends (DELIVER-REVEAL, sid, cid, i, j), send (REVEAL, $sid, cid, i, s^{(i)}_{cid}$) to $\mathcal{P}_j$.
- else send (REVEAL, $sid, cid, i, s^{(i)}_{cid}$) to $\mathcal{P}_j$.

Fig. 4. Functionality $\mathcal{F}_{\mathsf{MPC-SO}}$ for MPC with Secret-Shared Output and Linear Secret Share Operations.

The main purpose of $\mathcal{F}_{\mathsf{SC}}$ is twofold as it abstracts two necessary properties of certain blockchains such as Ethereum. Namely, it allows parties to contribute coins towards it and upon which $\mathcal{F}_{\mathsf{SC}}$ acts in a deterministic, publicly known manner. This can be realized by a standard Smart Contract, hence the name $\mathcal{F}_{\mathsf{SC}}$. Moreover, $\mathcal{F}_{\mathsf{SC}}$ also provides a public bulletin board functionality which can also be accessed by third parties. A Bulletin Board is a publicly readable storage

Functionality $\mathcal{F}_{\mathsf{SC}}$ interacts with the parties $\mathcal{P}$ and global functionalities $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{Clock}}$. It is parameterized by the compensation q, the deposit $d \geq (n-1)q$, the reconstruction function f and the cash distribution function g. $\mathcal{F}_{\mathsf{SC}}$ has an initially empty list $\mathcal{M}$ of messages posted to the authenticated public bulletin board.

Lock-in Deposits: Upon receiving $(\text{LOCK-IN}, sid, \mathsf{coins}(d + t^{(i)}))$ from $\mathcal{P}_i$ where d coins are security deposit and $t^{(i)} \geq 0$ coins are used by $\mathcal{P}_i$ as monetary input in the computation: Query $\mathcal{F}_{\mathsf{Clock}}$ with (READ, sid). If $\nu > 0$ return the money, otherwise accept it. If this was the first message $(\text{LOCK-IN}, sid)$ send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$.

Check Deposits: If (READ, sid) to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu = 1$ for the first time: If $(\mathcal{P}_i, sid, \text{OUTPUT-SCRAMBLED}, \overline{\boldsymbol{y}}) \in \mathcal{M}$ for each $i \in [n]$ with the same $\overline{\boldsymbol{y}}$ and each $\mathcal{P}_i$ sent $(\text{LOCK-IN}, sid, \mathsf{coins}(d + t^{(i)}))$ then send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$. If not then reimburse all parties that sent coins and abort.

Check Outputs: If (READ, sid) to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu = 2$ for the first time: Let J_1 be the maximal set such that $\forall i \in J_1 : (\mathcal{P}_i, sid, \text{OUTPUT-SHARE}, \boldsymbol{z}^{(i)}) \notin \mathcal{M}$. Then send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$.

Challenge Outputs: If (READ, sid) to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu = 3$ for the first time: Let J_2 be the maximal set of parties such that $\forall i \in J_2 : (\mathcal{P}_i, sid, \text{CHALLENGE}, \top) \in \mathcal{M}$. Send (UPDATE, sid) to $\mathcal{F}_{\mathsf{Clock}}$.

Obtain Verification Data: If (READ, sid) to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu = 4$ for the first time:

1. If $J_1 \neq \emptyset$ then run $\mathsf{Punish}(J_1)$ and stop. If $J_2 = \emptyset$ then run $\mathsf{CompPay}()$ and stop.
2. If $J_2 \neq \emptyset$ then send $(\text{VERIFY}, sid, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)})$ to $\mathcal{F}_{\mathsf{Ident}}$.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\text{VERIFY-FAIL}, sid, J_3)$ then run $\mathsf{Punish}(J_3)$ and stop.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\text{REVEAL-FAIL}, sid, \mathtt{ref}^{(1)}, \ldots, \mathtt{ref}^{(n)})$ then set $J_3 \leftarrow \bigcup_{i \in [n]} \mathtt{ref}^{(i)}$. Run $\mathsf{Punish}(J_3)$ and stop.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\text{OPEN-FAIL}, sid, J_3)$ and $J_3 \neq \emptyset$ then run $\mathsf{Punish}(J_3)$ and stop. If $J_3 = \emptyset$ then run $\mathsf{CompPay}()$.

Post to Bulletin Board: Upon receiving $(\text{POST}, sid, \text{OFF}, m)$ from $\mathcal{P}_i \in \mathcal{P}$, if there is no $(\mathcal{P}_i, sid, \text{OFF}, m') \in \mathcal{M}$, append $(\mathcal{P}_i, sid, \text{OFF}, m)$ to the list $\mathcal{M}$ of authenticated messages that were posted in the public bulletin board.

Read from Bulletin Board: Upon receiving (READ, sid) from a party, return $\mathcal{M}$.

Macro $\mathsf{Punish}(\mathtt{punish})$**:** Let $\mathtt{punish} \subset [n]$ and $\mathtt{reimburse} = [n] \setminus \mathtt{punish}$. Define $e^{(i)}$ as $d - q \cdot |\mathtt{reimburse}| + t^{(i)}$ if $i \in \mathtt{punish}$ and $d + q \cdot |\mathtt{punish}| + t^{(i)}$ if $i \in \mathtt{reimburse}$ and then run $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$.

Macro $\mathsf{CompPay}$**:** Compute $\boldsymbol{y} \leftarrow f(\overline{\boldsymbol{y}}, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)})$ and $(e^{(1)}, \ldots, e^{(n)}) \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$. Then run $\mathsf{Pay}(d + e^{(1)}, \ldots, d + e^{(n)})$.

Macro $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$**:** For each $\mathcal{P}_i \in \mathcal{P}$ send $(\text{PAYOUT}, sid, \mathcal{P}_i, \mathsf{coins}(e^{(i)}))$ to $\mathcal{P}_i$ and $(\text{PAYOUT}, sid, \mathcal{P}_i, e^{(i)})$ to each other party.

Fig. 5. The stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$ that is used to enforce penalties on parties that misbehave in the multiparty computation protocol and to distribute money.

for messages which cannot be erased after being posted. We use an *authenticated* Bulletin Board, which means that messages that are posted can be related to specific parties. These can be implemented from a standard Bulletin Board and signatures.[4] As we focus on the MPC aspects rather than compatibility with a

[4] There exist impossibility results on realizing this primitive [20, 32], but we avoid these by allowing for setup, which is also necessary for UC secure MPC [15].

blockchain based public ledger, we model the public ledger as an ideal Bulletin Board that allows for parties to immediately write and read messages.

$\mathcal{F}_{\mathsf{SC}}$'s remaining interfaces are then tailored towards our application, interact with the information stored on the bulletin board and can, as mentioned before, be realized as a smart contract. Namely, the functionality ensures that first all parties deposit coins, then all parties send output shares and have the possibility to challenge outputs that they deem incorrect. In that case, $\mathcal{F}_{\mathsf{SC}}$ identifies the cheaters together with $\mathcal{F}_{\mathsf{Ident}}$ (which is defined later) and punishes cheating parties by splitting up their deposits. Conversely, if no party cheated or raises concern then $\mathcal{F}_{\mathsf{SC}}$ returns the deposits and redistributes additional coins according to a cash distribution function g that is fixed in advance.

4 Our Construction

We now describe how the aforementioned building blocks can be combined to construct $\mathcal{F}_{\mathsf{Online}}$ from $\mathcal{F}_{\mathsf{MPC-SO}}$. We will therefore proceed in two steps. First, we realize an intermediate functionality $\mathcal{F}_{\mathsf{Ident}}$ which realizes a flavor of Publicly Verifiable MPC, from which we then in a second step construct Insured MPC.

The functionality $\mathcal{F}_{\mathsf{Ident}}$ can be found in Figure 6. It describes MPC with a flavor of publicly verifiable output. Here, the parties can verify that the computation until the output reconstruction was done correctly. If so, then they run a subcomputation which reconstructs the output and which furthermore allows to determine if a party aborted or provided incorrect shares. In particular, $\mathcal{F}_{\mathsf{Ident}}$ allows for third parties to verify that either a given output was indeed obtained from the MPC or a given party has misbehaved in the output phase.

Then, using the functionality $\mathcal{F}_{\mathsf{SC}}$ we realize $\mathcal{F}_{\mathsf{Online}}$, i.e. MPC with fair output delivery with penalties. There, $\mathcal{F}_{\mathsf{SC}}$ uses the properties of $\mathcal{F}_{\mathsf{Ident}}$ to either determine the distribution of funds according to the output or punish the identified cheaters. The relations among the functionalities are summarized in Figure 7.

Building Publicly Verifiable MPC. We now sketch a protocol Π_{Ident} which realizes $\mathcal{F}_{\mathsf{Ident}}$ in the $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{CT}}$-hybrid model with the XOR function over $\mathbb{F}^m$ as the reconstruction function. The full protocol together with a proof of security is presented in the full version.

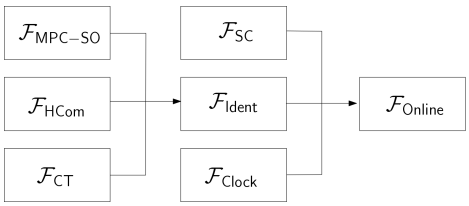

Fig. 7. Steps of the MPC Protocol Compiler.

In the protocol, the parties first use $\mathcal{F}_{\mathsf{MPC-SO}}$ to securely compute the output $\boldsymbol{y} = C(x^{(1)}, \ldots, x^{(n)})$ inside the MPC functionality. The output is then not immediately reconstructed, but instead all parties learn a vector $\boldsymbol{z} \in \mathbb{F}^m$ and each party $\mathcal{P}_i$ additionally obtains a share vector $\boldsymbol{s}^{(i)} \in \mathbb{F}^m$ such that $\boldsymbol{y} = \boldsymbol{z} \bigoplus \boldsymbol{s}^{(i)}$. Each party also generates random blinding values $\boldsymbol{r}^{(i)} \in \mathbb{F}^\kappa$ using $\mathcal{F}_{\mathsf{MPC-SO}}$.

Then each $\mathcal{P}_i$ commits to $\boldsymbol{s}^{(i)}, \boldsymbol{r}^{(i)}$ using $\mathcal{F}_{\mathsf{HCom}}$ in order to make the shares $\boldsymbol{s}^{(i)}$ publicly verifiable later. To ensure correctness, the parties use $\mathcal{F}_{\mathsf{CT}}$ to sample

Functionality $\mathcal{F}_{\mathsf{Ident}}$ interacts with the parties $\mathcal{P}$ and also provides an interface to register verifiers $\mathcal{V}$. It is parameterized by a circuit C (with inputs $x^{(1)}, \ldots, x^{(n)}$ and output $\boldsymbol{y} \in \mathbb{F}^m$) and a reconstruction function f. $\mathcal{S}$ provides a set $I \subset [n]$ of corrupt parties. Throughout **Init**, **Input**, **Evaluate** and **Share**, $\mathcal{S}$ can at any point send $(\textsc{Abort}, sid)$, upon which $\mathcal{F}_{\mathsf{Ident}}$ broadcasts $(\textsc{Abort}, sid, \perp)$ and terminates. Throughout **Reveal** and **Verify**, $\mathcal{S}$ at any point is allowed to send $(\textsc{Abort}, sid, J)$. If $J \subseteq I$ then $\mathcal{F}_{\mathsf{Ident}}$ will send $(\textsc{Abort}, sid, J)$ to all honest parties and terminate.

Init: Upon first input $(\textsc{Init}, sid)$ by all $\mathcal{P}_i \in \mathcal{P}$ set $\mathtt{rev}, \mathtt{ver}, \mathtt{ref}^{(1)}, \ldots, \mathtt{ref}^{(n)} \leftarrow \emptyset$.

Input: Upon first input $(\textsc{Input}, sid, i, x^{(i)})$ by $\mathcal{P}_i$ and input $(\textsc{Input}, sid, i, \cdot)$ by all other parties the functionality stores the value $(i, x^{(i)})$ internally. Every further such message with the same sid and i is ignored.

Evaluate: Upon first input $(\textsc{Compute}, sid)$ by all $\mathcal{P}_i \in \mathcal{P}$ and if inputs $(i, x^{(i)})_{i \in [n]}$ for all parties are stored internally, compute $\boldsymbol{y} \leftarrow C(x^{(1)}, \ldots, x^{(n)})$ and store $\boldsymbol{y}$ locally.

Share: Upon first input $(\textsc{Share}, sid)$ by $\mathcal{P}_i \in \mathcal{P}$ and if **Evaluate** was finished:

1. For each $\mathcal{P}_i \in \mathcal{P}$ sample $\boldsymbol{s}^{(i)} \xleftarrow{\$} \mathbb{F}^m$ uniformly at random and store it locally. Then send $\boldsymbol{s}^{(i)}$ for each $i \in I$ to $\mathcal{S}$.
2. Upon $(\textsc{Deliver-Share}, sid, i)$ from $\mathcal{S}$ for $i \in \overline{I}$ send $(\textsc{Output}, sid, \boldsymbol{s}^{(i)})$ to $\mathcal{P}_i$.
3. Sample $\bar{\boldsymbol{y}} \in \mathbb{F}^m$ such that $f(\bar{\boldsymbol{y}}, \boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(n)}) = \boldsymbol{y}$.
4. Send $(\textsc{Output}, sid, \bar{\boldsymbol{y}})$ to $\mathcal{S}$. If $\mathcal{S}$ sends $(\textsc{Deliver-Output}, sid, \bar{\boldsymbol{y}})$ then send $(\textsc{Output}, sid, \bar{\boldsymbol{y}})$ to all $\mathcal{P}_i \in \overline{I}$.

Reveal: Upon input $(\textsc{Reveal}, sid, i)$ by $\mathcal{P}_i$, if $i \notin \mathtt{rev}$ and $\mathtt{ref}^{(i)} = \emptyset$ send $(\textsc{Reveal}, sid, i, \boldsymbol{s}^{(i)})$ to $\mathcal{S}$.

- If $\mathcal{S}$ sends $(\textsc{Reveal-Ok}, sid, i)$ then set $\mathtt{rev} \leftarrow \mathtt{rev} \cup \{i\}$, send $(\textsc{Reveal}, sid, i, \boldsymbol{s}^{(i)})$ to all parties in $\mathcal{P}$.
- If $\mathcal{S}$ sends $(\textsc{Reveal-Not-Ok}, sid, i, J)$ with $J \subseteq I$ then send $(\textsc{Reveal-Fail}, sid, i)$ to all parties in $\mathcal{P}$ and set $\mathtt{ref}^{(i)} \leftarrow J$.

Test Reveal: Upon input $(\textsc{Test-Reveal}, sid)$ from a party in $\mathcal{P} \cup \mathcal{V}$ define $\overline{\mathtt{ref}}^{(i)} = \mathtt{ref}^{(i)}$ if $i \in \mathtt{rev}$ and $\overline{\mathtt{ref}}^{(i)} \leftarrow \mathtt{ref}^{(i)} \cup \{i\}$ otherwise. Then send $(\textsc{Reveal-Fail}, sid, \overline{\mathtt{ref}}^{(1)}, \ldots, \overline{\mathtt{ref}}^{(n)})$ to $\mathcal{P}$ and $\mathcal{V}$.

Allow Verify: Upon input $(\textsc{Start-Verify}, sid, i)$ from party $\mathcal{P}_i \in \mathcal{P}$ set $\mathtt{ver} \leftarrow \mathtt{ver} \cup \{i\}$. If $\mathtt{ver} = [n]$ then deactivate all interfaces except **Test Reveal** and **Verify**.

Verify: Upon input $(\textsc{Verify}, sid, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)})$ by $\mathcal{V}_i \in \mathcal{V}$ with $\boldsymbol{z}^{(j)} \in \mathbb{F}^m$:

- If $\mathtt{ver} \neq [n]$ then return $(\textsc{Verify-Fail}, sid, [n] \setminus \mathtt{ver})$.
- If $\mathtt{ver} = [n]$ and $\mathtt{rev} \neq [n]$ then send to $\mathcal{V}_i$ what **Test Reveal** sends.
- If $\mathtt{ver} = \mathtt{rev} = [n]$ then compute the set $\mathtt{ws} \leftarrow \{j \in [n] \mid \boldsymbol{z}^{(j)} \neq \boldsymbol{s}^{(j)}\}$ and return $(\textsc{Open-Fail}, sid, \mathtt{ws})$.

Fig. 6. Functionality $\mathcal{F}_{\mathsf{Ident}}$ for an MPC with Publicly Verifiable Output.

a random matrix $\boldsymbol{\alpha} \in \mathbb{F}_2^{\kappa \times m}$, compute and open the output $\alpha \times \boldsymbol{s}^{(i)} + \boldsymbol{r}^{(i)}$ in both $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}$ for each $\mathcal{P}_i$ and abort if this value differs for any party. Otherwise, the parties set $\overline{\boldsymbol{y}} \leftarrow \boldsymbol{z}$ as their public advice and continue the protocol with the committed values $\boldsymbol{s}^{(i)}$.

To implement the **Reveal** and **Verify**-type interfaces of $\mathcal{F}_{\mathsf{Ident}}$ we use the respective interfaces of $\mathcal{F}_{\mathsf{HCom}}$. Namely, the opening commands of $\mathcal{F}_{\mathsf{HCom}}$ can be used to generate the unreliable but identifiable opening of $\boldsymbol{s}^{(i)}$ by each party. These openings are made publicly verifiable and tested by using the **Verification** interfaces of the functionality.

Theorem 1. *The aforementioned Π_{Ident} UC-realizes $\mathcal{F}_{\mathsf{Ident}}$ (with XOR over $\mathbb{F}^m$ as the reconstruction function) against static active adversaries corrupting $< n$ parties in the $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{CT}}$-hybrid model with broadcast.*

Proof (Sketch). Define a simulator $\mathcal{S}$ where $\mathcal{F}_{\mathsf{CT}}, \mathcal{F}_{\mathsf{HCom}}$ are global and $\mathcal{F}_{\mathsf{MPC-SO}}$ a local functionality and which itself simulates the protocol Π_{Ident} with $\mathcal{A}$ using dummy honest parties. In the full proof we will first show that if a party obtains values $\boldsymbol{r}^{(i)}, \boldsymbol{s}^{(i)}$ from $\mathcal{F}_{\mathsf{MPC-SO}}$ but commits to differing values towards $\mathcal{F}_{\mathsf{HCom}}$, then the opened values $\boldsymbol{\alpha} \times \boldsymbol{s}^{(i)} + \boldsymbol{r}^{(i)}$ from $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}$ are identical with probability $O(2^{-\kappa})$ as we are essentially evaluating a universal hash function on these inputs. We then use the fact that we can extract the shares which $\mathcal{A}$ uses for the dishonest parties from our simulated $\mathcal{F}_{\mathsf{MPC-SO}}$ to provide these to $\mathcal{F}_{\mathsf{Ident}}$. The shares of the output of $\mathcal{F}_{\mathsf{MPC-SO}}$ can be altered during the opening of z so that the advice obtained by $\mathcal{A}$ is consistent with the output of $\mathcal{F}_{\mathsf{Ident}}$. That the simulation of the **Reveal**, **Verify** interfaces using $\mathcal{F}_{\mathsf{HCom}}$ is indistinguishable then follows as the values of the dishonest parties inside $\mathcal{F}_{\mathsf{HCom}}$ coincide with those provided to $\mathcal{F}_{\mathsf{Ident}}$ by $\mathcal{S}$, while the equivocability of $\mathcal{F}_{\mathsf{HCom}}$ allows to simulate the opening and verification of the $\boldsymbol{s}^{(i)}$ values.

From Publicly Verifiable MPC to Insured MPC. We now sketch a protocol Π_{Compiler} that realizes the functionality $\mathcal{F}_{\mathsf{Online}}$ with punishable abort in the $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{Clock}}$-hybrid model. In Π_{Compiler}, $\mathcal{F}_{\mathsf{Ident}}$ will obtain the inputs $x^{(i)}$ from all parties and provide both the advice $\bar{\boldsymbol{y}}$ and shares $\boldsymbol{s}^{(i)}$ that are necessary for the reconstruction of $\boldsymbol{y}$ to the parties. To reliably reconstruct $\boldsymbol{y}$, each $\mathcal{P}_i$ sends $\bar{\boldsymbol{y}}$ as well as $\mathsf{coins}(d + t^{(i)})$ to the bulletin board $\mathcal{F}_{\mathsf{SC}}$. The coins $\mathsf{coins}(d)$ are used to reimburse other parties in case $\mathcal{P}_i$ aborts, while $\mathsf{coins}(t^{(i)})$ is the input of $\mathcal{P}_i$ into the cash distribution function g. Then, $\mathcal{F}_{\mathsf{Ident}}$ is used by each party $\mathcal{P}_i$ to reveal its share $\boldsymbol{s}^{(i)}$ to all other parties and a value $\boldsymbol{z}^{(i)}$ is posted on $\mathcal{F}_{\mathsf{SC}}$ (where $\boldsymbol{z}^{(i)}$ might be different from $\boldsymbol{s}^{(i)}$ if the adversary cheats). We use $\mathcal{F}_{\mathsf{Clock}}$ to determine if all parties opened/posted their shares $\boldsymbol{z}^{(i)}$ in time and proceed if so. If a party cheats during the opening phase with $\mathcal{F}_{\mathsf{Ident}}$, the protocol instructs all parties to post a complaint on $\mathcal{F}_{\mathsf{SC}}$ within a limited time period (enforced by $\mathcal{F}_{\mathsf{Clock}}$). Once the parties have reacted to complaints by activating verification, $\mathcal{F}_{\mathsf{SC}}$ contacts $\mathcal{F}_{\mathsf{Ident}}$ to verify the correctness of the $\boldsymbol{z}^{(i)}$. An adversary may withhold his share, provide an incorrect share or abort this verification, thus preventing both $\mathcal{F}_{\mathsf{SC}}$ and the honest parties from obtaining the result. In such a case, let $\mathtt{punish} \subseteq I$ be the set of aborting or cheating parties, and $\mathtt{reimburse} = \mathcal{P} \setminus \mathtt{punish}$. Each party from $\mathtt{reimburse}$ will be reimbursed by $\mathsf{coins}(d - q \cdot |\mathtt{reimburse}| + t^{(i)})$ by $\mathcal{F}_{\mathsf{SC}}$, whereas the rest is fairly distributed among the non-cheating parties, which obtain $\mathsf{coins}(d + q \cdot |\mathtt{punish}| + t^{(i)})$. If all parties act honestly, then $\mathcal{F}_{\mathsf{SC}}$ uses g to

determine the correct payoffs that are then sent to all parties. This also happens if parties cheat by not revealing $s^{(i)}$ via $\mathcal{F}_{\mathsf{Ident}}$, but posting the correct value on $\mathcal{F}_{\mathsf{SC}}$, because we cannot distinguish a setting where a dishonest party did not reveal the correct share towards an honest party (which sends a complaint) from a dishonest party framing an honest party.

Theorem 2. *Protocol Π_{Compiler} UC-realizes $\mathcal{F}_{\mathsf{Online}}$ in the $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{SC}}$-hybrid model with global $\mathcal{F}_{\mathsf{Clock}}$ against static and active adversaries corrupting $< n$ parties.*

Proof (Sketch). Define a simulator $\mathcal{S}$ which will interact with the hybrid-world adversary $\mathcal{A}$ in the presence of $\mathcal{F}_{\mathsf{Online}}, \mathcal{F}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{SC}}$. $\mathcal{S}$ simulates an instance of Π_{Compiler} by emulating honest parties and running copies of $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{Clock}}$. Both $\mathcal{F}_{\mathsf{Online}}, \mathcal{F}_{\mathsf{Ident}}$ use the same cash distribution function g and reconstruction function f. $\mathcal{S}$ runs Π_{Compiler} with random inputs for the simulated honest parties, extracts the inputs of the dishonest parties from $\mathcal{F}_{\mathsf{Ident}}$ and forwards these to $\mathcal{F}_{\mathsf{Online}}$. $\mathcal{S}$ also inputs coins on behalf of $\mathcal{A}$ into $\mathcal{F}_{\mathsf{Online}}$ (if $\mathcal{A}$ sends these to $\mathcal{F}_{\mathsf{SC}}$) and uses the leakage from $\mathcal{F}_{\mathsf{Online}}$ to simulate coins from the emulated parties. $\mathcal{S}$ opens those shares $s^{(i)}$ of honest parties towards $\mathcal{A}$ that it obtained from $\mathcal{F}_{\mathsf{Online}}$ (same for $\overline{\boldsymbol{y}}$) and forwards any aborts of the dishonest parties to $\mathcal{F}_{\mathsf{Online}}$. Depending if $\mathcal{F}_{\mathsf{SC}}$ punishes parties or compensates them send the set used by Punish to $\mathcal{F}_{\mathsf{Online}}$ or $\emptyset$. It is easy to see that the output which $\mathcal{A}$ obtains during the simulation is consistent with $\mathcal{F}_{\mathsf{Online}}$, and so are the shares as it does not see $\boldsymbol{s}^{(i)}$ for $i \in \overline{I}$ until the output $\boldsymbol{y}$ is known to $\mathcal{S}$. The coins-values which $\mathcal{S}$ sends are consistent with those from $\mathcal{F}_{\mathsf{SC}}$ (and vice versa) and both $\mathcal{F}_{\mathsf{Online}}, \mathcal{F}_{\mathsf{SC}}$ abort in the same cases. We see that by construction if $\mathcal{F}_{\mathsf{SC}}$ calls Punish then the set given to the macro is non-empty. $\mathcal{F}_{\mathsf{SC}}$ either punishes parties that do not send $\boldsymbol{z}^{(i)}$, do not activate verification or where verification of $\boldsymbol{z}^{(i)}$ fails. All of these can only occur for dishonest parties.

Hiding the Output $\boldsymbol{y}$ while distributing cash. It is immediate that our protocol Π_{Compiler} leaks the value $\boldsymbol{y}$ to any user of the distributed ledger. By the construction of $\mathcal{F}_{\mathsf{SC}}$, we can keep it private if one only wants to obtain MPC with fair output delivery with penalties (without cash distribution). If cash distribution is indeed required, then we can augment the MPC input by $t^{(1)}, \ldots, t^{(n)}$, the output by $e^{(1)}, \ldots, e^{(n)}$ and compute the latter based on $g, \boldsymbol{y}$ inside the MPC. During the output phase we only publish the "public" part of the advice on $\mathcal{F}_{\mathsf{SC}}$, which can then perform the cash distribution reliably.

5 Efficiency and Comparison to Previous Works

Our approach preserves the efficiency of the "inner" MPC protocol and the commitment scheme used for our generic construction. No modifications are done to these components, since our constructions basically consists in executing the MPC protocol to evaluate the circuit describing the function to be computed and then executing the commitment scheme to obtain commitments (and later openings) to the MPC protocol's partial outputs. Hence, the complexity of executing

our generic protocol between n parties is essentially that of executing the "inner" MPC protocol among n parties in order to evaluate the function and then executing n commitments and openings using the commitment scheme. Moreover, our approach is "optimistic" in the sense that more expensive verification procedures are only executed in case there is a suspicion that a party has cheated. Our generic construction can be concretely instantiated in the preprocessing model based on the MPC protocol of [23] and the publicly verifiable additively homomorphic commitment scheme that we introduce. In case no party is suspected to be cheating, our online phase achieves basically the same efficiency as the MPC protocol of [23], since our commitment scheme's online phase achieves the same efficiency as the scheme of [16], which according to the benchmarks of [**?**] requires only a few microseconds for commitments/openings. An even better concrete instantiation can be obtained by employing the new publicly verifiable additively homomorphic commitment scheme recently introduced in [**?**].

Efficiency of Previous Works For an efficient implementation fair computation, one can use more efficient ID-MPC protocols (e.g. [8]), but these are significantly less efficient than MPC protocols without that property. Apart from incurring very high computational overheads in relation to the underlying MPC protocol due to the use of expensive generic non-interactive zero-knowledge proofs (NIZKs), the best scheme in this line of work [27] also requires all MPC protocol messages and associated NIZKs to be posted to the ledger at each round, which is prohibitive for practical scenarios.[5]

As mentioned already in Section 1.2 current protocols for fair output delivery such as [11, 31] compute both the secret sharing of the result and the commitments to each share inside the MPC in a white-box way, adding significant computational and communication overheads. Moreover, while these works claim that a random oracle (RO) based commitment can be used, this would preclude the resulting protocol from achieving simulation-based security notions. Notice that computing such a commitment inside the MPC means that calls to the RO itself would have to be computed by the MPC, which is not possible since the RO ideal functionality cannot be represented as a circuit. The alternative for instantiating such protocols with composabability guarantees would be using universally composable commitment schemes that can be instantiated from a common reference string, which would require the MPC to compute tens (if not hundreds) of modular exponentiations, resulting in enormous overheads.

Acknowledgments: This work has been supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and the DFF under grant agreement number 9040-00399B (TrA^2C).

[5] In private communication with the authors of [27] we have confirmed that while their generic construction achieves optimal round complexity, it does incur very high computational, communication and public ledger storage overheads that make it impractical to construct a concrete instantiation or estimate parameters.

Bibliography

[1] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *FC 2014 Workshops*, volume 8438 of *LNCS*, pages 105–121. Springer, Heidelberg, March 2014.

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

[3] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 291–316. Springer, Heidelberg, February 2014.

[4] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete characterization of fairness in secure two-party computation of Boolean functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 199–228. Springer, Heidelberg, March 2015.

[5] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.

[6] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 175–196. Springer, Heidelberg, September 2014.

[7] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured mpc: Efficient secure multiparty computation with punishable abort. Cryptology ePrint Archive, Report 2018/942, 2018. https://eprint.iacr.org/2018/942.

[8] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 461–490. Springer, Heidelberg, October / November 2016.

[9] F. Benhamouda, S. Halevi, and T. Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 357–363, April 2018.

[10] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, August 2014.

[11] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASI-*

ACRYPT 2017, Part II, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, December 2017.

[12] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[14] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.

[15] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[16] Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016.

[17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 719–728. ACM Press, October / November 2017.

[18] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[19] Tore K. Frederiksen, Benny Pinkas, and Avishay Yanai. Committed MPC - maliciously secure multiparty computation from homomorphic commitments. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 587–619. Springer, Heidelberg, March 2018.

[20] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, October 2007.

[21] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[22] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 413–422. ACM Press, May 2008.

[23] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Heidelberg, December 2017.

[24] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 483–500. Springer, Heidelberg, August 2006.

[25] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

[26] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[27] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.

[28] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 30–41. ACM Press, November 2014.

[29] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 418–429. ACM Press, October 2016.

[30] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 195–206. ACM Press, October 2015.

[31] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 406–417. ACM Press, October 2016.

[32] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. In *34th ACM STOC*, pages 514–523. ACM Press, May 2002.

[33] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[34] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.