Insured MPC: Efficient Secure Multiparty Computation with Punishable Abort

Carsten Baum[1⋆], Bernardo David[2,4], and Rafael Dowsley[3,4⋆⋆]

[1] Bar Ilan University
[2] Tokyo Institute of Technology
[3] Aarhus University
[4] IOHK

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. This approach also yields privacy-preserving Smart Contracts, where private inputs can be processed with MPC in order to determine the distribution of funds given to the contract. Unfortunately, the focus of existing work is on proving that this approach is possible and they present monolithic and mostly inefficient constructions. In this work, we put forth the first modular construction of "Insured MPC", where the result of the private computation of parties either yields an output describing how to distribute funds or a proof that a set of parties has misbehaved, 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 a highly efficient protocol which allows public verification of cheating behavior during the output stage. This scheme is constructed using a publicly verifiable homomorphic commitment scheme, for which we propose an efficient construction. Furthermore, we construct a compiler that uses any such scheme together with a Smart Contract to implement Insured MPC. This compiler requires a standard (non-private) Smart Contract. Our results are proven in the Universal Composability framework using a Global Random Oracle as the setup assumption. From a theoretical perspective, our general results provide the first characterization of sufficient properties that MPC protocols must achieve in order to be efficiently combined with Cryptocurrencies, as well as insights on publicly verifiable protocols. On the other hand, all our constructions and protocols are highly efficient and allow for a fast implementation.

⋆ 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 author acknowledges additional support from IOHK.

⋆⋆ This project has received funding from the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

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 [19] proved that fair MPC with a dishonest majority is impossible to achieve in the standard communication model. While the result can be circumvented for certain, specific functions [24, 3, 4] in the two-party setting, this barrier prevents MPC from being a useful tool for certain interesting applications.

With the advent of cryptocurrencies Andrychowicz et al. [2] (and independently Bentov & Kumaresan [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 distributed ledger and digital currency, where the overall structure of their idea is as follows: (i) Each party deposits a collateral on the distributed ledger; (ii) The parties run the secure computation, but delay the reconstruction of the output; (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 [33, 29, 32] generalized this concept and improved the performance with respect to the amount of interaction with the distributed ledger as well as the collateral that each party needs to deposit. In particular, Kumaresan et al. [1, 2, 33] 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 distributed ledger is used both to enforce financial penalties as well as to distribute money according to the output of the secure computation.

1.1 Fair Computation vs. Fair Output Delivery

Regarding a better understanding of our design choices it is worthwhile to discuss first *which* adversarial behavior should be punishable: while a fair protocol can impose fines on any deviation, an adversary aborting before the output phase will not learn any information about the result. 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.

Fair Computation. [2] and [33] follow this line of work, but use heavy zero-knowledge machinery to achieve their results. As [29] correctly pointed out, care must be taken when choosing the "inner" MPC protocol (which is compiled to have financial penalties): to achieve fair computation with penalties, it must have a property called *Identifiable Abort* (ID-MPC, [27]). As [2, 33] use GMW [23] this works in their case, but not every MPC protocol is suitable. For an efficient implementation, one can use more efficient ID-MPC protocols such as [7]. Unfortunately, the best ID-MPC protocols are still significantly slower than those protocols without that property and the amount of data that current constructions store on the ledger is highly impractical[5].

Fair Output Delivery. This line of work has been independently initiated by [1, 10] and continued in [2, 31, 32, 34, 11]. Here, the MPC protocol computes a verifiable secret sharing (VSS) of the output and shares it among the parties. All of the above protocols perform the reconstruction essentially on the ledger, and they obtain a reconstruction that only needs a constant number of rounds per party. This implicitly has identifiable abort due to the use of a VSS and because all parties can publicly agree if another participant stopped to post on the ledger. A caveat, both from a theoretical and practical point of view, is that current protocols compute the sharing of the VSS inside the MPC in a white-box way, which adds significant computational overhead.

1.2 Our Contributions

In this work, we give the first concrete MPC scheme having a fair output delivery with penalties. We depart from the current designs to have a cleaner separation between the different phases, which allows for a modular analysis in the Global Universal Composability framework (GUC). At the same time, this approach directly pinpoints the necessary properties of all building blocks involved in our construction to actually make it work. We now explain our construction and the contributions more in detail.

New Multiparty Additively Homomorphic Commitment with Delayed Public Verifiability. These primitive acts as the central hub of our construction. Such commitment schemes are additively homomorphic, allowing one to reveal linear combinations between 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. These commitments, when combined with a suitable "Base MPC" protocol, allow us to use a much more efficient and modular output secret sharing and reconstruction. While such a primitive could be realized with stand alone security by constructions such as Pedersen Commitments [38], existing constructions that achieve all of these properties do not have composability guarantees. Instead, we provide an interactive construction which only needs a small number of Oblivious Transfers (OTs) independent from the number of commitments and otherwise relies solely on symmetric primitives. This construction improves on the protocols of [16, 21], which are incompatible with public verifiability. We believe this construction is of independent interest.

[5] One can of course replace the proofs with SNARKs, but this comes with high computation costs to construct these proofs.

Modular Design. Based on such a commitment scheme and a suitable "Base MPC", we give a modular approach for constructing "Insured MPC": first, we combine the "Base MPC" with the commitment scheme to achieve *MPC with publicly verifiable output.* In this step, the protocol moves the shares of the secret sharing consistently out of the MPC. Given an extended standard Smart Contract functionality and a global clock we can then construct an identifiable reconstruction phase in a modular way where we let the Smart Contract mediate the reconstruction. In case of disagreement, this Smart Contract can then identify the cheater as all parties are required to post the openings of their committed shares to the ledger. By choice of our construction, the steps involved in the verification process are mostly light-weight due to our commitment scheme.

Efficient Instantiation. We show how to instantiate all sub-protocols with efficient primitives. More specifically, we modify the constant-round MPC protocol due to [25, 43] to work as the "Base MPC". The commitment protocol, after a small number of base OTs using a verifiable OT scheme, only performs Random Oracle (RO) calls and can be implemented using a cryptographic hash function. As we use a restricted programmable and observable global RO [13] we are then still able to prove security of all steps in GUC.

Other Related Work. Recently Choudhuri et al. [18] showed how to circumvent the impossibility result of [19] and constructed a fair MPC using a Bulletin Board. As their work either relies on Witness Encryption (which currently requires Indistinguishability Obfuscation to be constructed) or Trusted Hardware (which we also deem to be a very strong assumption) it does seem to be an incomparable alternative. The use of MPC for computing on private data in *permissioned* ledgers has been suggested in [9], where the authors suggest that an MPC protocol can have all of its messages posted on a public ledger for verification.

As mentioned before, public verifiability and identifiable abort are two crucial properties to construct fair MPC with penalties. MPC with public verification was introduced in [6, 41]. Both of protocols come with a significant overhead during the computation phase and are not suitable in our setting. Ishai et al. [27] formally studied how to construct ID-MPC using adaptively secure OT and Zero-Knowledge proofs. Subsequent work [7, 42, 20] then introduced more efficient protocols. [7, 20] also describe how to modify their approach to have a public verification procedure.

2 Preliminaries

Let $y \stackrel{\$}{\leftarrow} F(x)$ denote running the randomized algorithm F with input x and random coins, and obtaining the output y. When the coins r are specified we use $y \leftarrow F(x; r)$. Similarly, $y \leftarrow F(x)$ is used for a deterministic algorithm. For a set $\mathcal{X}$, let $x \stackrel{\$}{\leftarrow} \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$; and for a distribution $\mathcal{Y}$, let $y \xleftarrow{\$} \mathcal{Y}$ denote y sampled according to the distribution $\mathcal{Y}$. For any $k \in \mathbb{N}$ we write $[k]$ for the set $\{1, \ldots, k\}$. A function $f(x)$ is negligible in x (or $\mathrm{negl}(x)$ to denote an arbitrary such function) if $f(x)$ is positive and for every positive polynomial $p(x) \in \mathrm{poly}(x)$ there exists a $x' \in \mathbb{N}$ such that $\forall x \geq x' : \ f(x) < 1/p(x)$. Two ensembles $X = \{X_{\kappa,z}\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ and $Y = \{Y_{\kappa,z}\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ of binary random variables are said to be *statistically indistinguishable*, denoted by $X \approx_s Y$, if for all z it holds that $|\Pr[\mathcal{D}(X_{\kappa,z}) = 1] - \Pr[\mathcal{D}(Y_{\kappa,z}) = 1]|$ is negligible in κ for every probabilistic algorithm (distinguisher) $\mathcal{D}$. In case this only holds for computationally bounded (non-uniform probabilistic polynomial-time (PPT)) distinguishers we say that X and Y are *computationally indistinguishable* and denote it by $\approx_c$.

Let n be the number of parties in an MPC scheme and $\mathcal{A}$ be an adversary. Throughout this work, we will denote with $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ the set of parties and with $I \subsetneq \mathcal{P}$ the set of corrupted parties. The uncorrupted parties will be $\overline{I} = \mathcal{P} \setminus I$. We denote the ideal-world adversary as $\mathcal{S}$. We use τ to denote a computational and κ for a statistical security parameter.

Vectors of field elements are denoted by bold lower-case letters and matrices by bold upper-case letters. Concatenation of vectors is represented by $\|$. For $\boldsymbol{z} \in \mathbb{F}^k$, $\boldsymbol{z}[i]$ denotes the i'th entry of the vector, so that e.g. $\boldsymbol{z}[1]$ is the first element of $\boldsymbol{z}$. We denote by $\mathbf{0}^k$ the column vector of k components where all entries are 0. We denote the scalar product of a scalar $\alpha \in \mathbb{F}$ with a vector $\boldsymbol{x} \in \mathbb{F}^k$ by $\alpha \cdot \boldsymbol{x} = (\alpha \cdot \boldsymbol{x}[1], \ldots, \alpha \cdot \boldsymbol{x}[k])$. For a matrix $\mathbf{M} \in \mathbb{F}^{n\times k}$, we let $\mathbf{M}[\cdot, j]$ denote its j'th column and $\mathbf{M}[i, \cdot]$ denote its i'th row. This work focus on computations on $\mathbb{F}_2$, which will be denote as $\mathbb{F}$ for conciseness.

2.1 Coding Theory, Interactive Proximity Testing and Linear Time Building Blocks

We will use the interactive proximity testing technique and corresponding linear time building blocks introduced in [16]. We adopt the notation and definitions from [16], reproduced in almost verbatim form in the remainder of this section.

For a vector $\boldsymbol{x} \in \mathbb{F}^n$, we denote the Hamming-weight of $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_0 = |\{i \in [n] : \boldsymbol{x}[i] \neq 0\}|$. Let $\mathsf{C} \subset \mathbb{F}^{\mathsf{n}}$ be a linear subspace of $\mathbb{F}^n$. We say that C is an $\mathbb{F}$-linear $[n, k, s]$ code if C has dimension k and it holds for every non-zero $\boldsymbol{x} \in \mathsf{C}$ that $\|\boldsymbol{x}\|_0 \geq s$, *i.e.*, the minimum distance of C is at least s. The distance $\mathsf{dist}(\mathsf{C}, \boldsymbol{x})$ between C and a vector $\boldsymbol{x} \in \mathbb{F}^n$ is the minimum of $\|\boldsymbol{c} - \boldsymbol{x}\|_0$ when $\boldsymbol{c} \in \mathsf{C}$. The rate of an $\mathbb{F}$-linear $[n, k, s]$ code is $\frac{k}{n}$ and its relative minimum distance is $\frac{s}{n}$.

A matrix $\mathbf{G} \in \mathbb{F}^{n\times k}$ is a generator matrix of C if $\mathsf{C} = \{\mathbf{G}\boldsymbol{x} : \boldsymbol{x} \in \mathbb{F}^k\}$. The code C is systematic if it has a generator matrix $\mathbf{G}$ such that the submatrix given by the top k rows of $\mathbf{G}$ is the identity matrix $\mathbf{I} \in \mathbb{F}^{k\times k}$. A matrix $\mathbf{P} \in \mathbb{F}^{(n-k)\times n}$ of maximal rank $n-k$ is a parity check matrix of C if $\mathbf{P}\boldsymbol{c} = 0$ for all $\boldsymbol{c} \in \mathsf{C}$. When we have fixed a parity check matrix $\mathbf{P}$ of C we say that the syndrome of an element $\boldsymbol{v} \in \mathbb{F}^n$ is $\mathbf{P}\boldsymbol{v}$. For an $\mathbb{F}$-linear $[n, k, s]$ code C, we denote by $\mathsf{C}^{\odot m}$ the *m-interleaved product* of C, which is defined by $\mathsf{C}^{\odot m} = \{\mathbf{C} \in \mathbb{F}^{n\times m} : \forall i \in [m] : \mathbf{C}[\cdot, i] \in \mathsf{C}\}$. In other words, $\mathsf{C}^{\odot m}$ consists of all $\mathbb{F}^{n\times m}$ matrices for which all columns are in C. We can think of $\mathsf{C}^{\odot m}$ as a linear code with symbol alphabet $\mathbb{F}^m$, where we obtain codewords by taking m arbitrary codewords of C and bundling together the components of these codewords into symbols from $\mathbb{F}^m$. For a matrix $\mathbf{E} \in \mathbb{F}^{n\times m}$, $\|\mathbf{E}\|_0$ is the number of non-zero rows of $\mathbf{E}$, and the code $\mathsf{C}^{\odot m}$ has minimum distance at least s' if all non-zero $\mathbf{C} \in \mathsf{C}^{\odot m}$ satisfy $\|\mathbf{C}\|_0 \geq s'$. Furthermore, $\mathbf{P}$ is a parity-check matrix of C if and only if $\mathbf{PC} = \mathbf{0}$ for all $\mathbf{C} \in \mathsf{C}^{\odot m}$. If C is an $\mathbb{F}$-linear $[n, k, s]$ code, its square C^{*2} is defined as the linear subspace of $\mathbb{F}^n$ generated by all the vectors of the form $\boldsymbol{v} * \boldsymbol{w}$ with $\boldsymbol{v}, \boldsymbol{w} \in \mathsf{C}$.

Definition 1 (Almost Universal Linear Hashing [16]). *We say that a family $\mathcal{H}$ of linear functions $\mathbb{F}^n \to \mathbb{F}^s$ is ϵ-almost universal, if it holds for every non-zero $\boldsymbol{x} \in \mathbb{F}^n$ that*

$$\Pr_{\mathbf{H} \xleftarrow{\$} \mathcal{H}} [\mathbf{H}(\boldsymbol{x}) = \mathbf{0}] \leq \epsilon,$$

where $\mathbf{H}$ is chosen uniformly at random from the family $\mathcal{H}$. We say that $\mathcal{H}$ is universal, if it is $|\mathbb{F}|^{-s}$-almost universal. We will identify functions $\mathbf{H} \in \mathcal{H}$ with their transformation matrix and write $\mathbf{H}(\boldsymbol{x}) = \mathbf{H} \cdot \boldsymbol{x}$.

The interactive proximity testing technique (as introduced in [16]) consists in the following argument: suppose we sample a function $\mathbf{H}$ from a family of almost universal linear hash functions (from $\mathbb{F}^m$ to $\mathbb{F}^\ell$), and we apply $\mathbf{H}$ to each of the rows of a matrix $\mathbf{X} \in \mathbb{F}^{n\times m}$, obtaining another matrix $\mathbf{X}' \in \mathbb{F}^{n\times \ell}$; because of linearity, if $\mathbf{X}$ belonged to an interleaved code $\mathsf{C}^{\odot m}$, then $\mathbf{X}'$ belongs to the interleaved code $\mathsf{C}^{\odot \ell}$. The following Theorem (from [16]) states that we can test whether $\mathbf{X}$ is close to $\mathsf{C}^{\odot m}$ by testing instead if $\mathbf{X}'$ is close to $\mathsf{C}^{\odot \ell}$ (with high probability over the choice of the hash function) and moreover, if these elements are close to the respective codes, the set of rows that have to be modified in each of the matrices in order to correct them to codewords are the same.

Theorem 1 ([16]). *Let $\mathcal{H} : \mathbb{F}^m \to \mathbb{F}^{2s+t}$ be a family of $|\mathbb{F}|^{-2s}$-almost universal $\mathbb{F}$-linear hash functions. Further let C be an $\mathbb{F}$-linear $[n, k, s]$ code. Then for every $\mathbf{X} \in \mathbb{F}^{n\times m}$ at least one of the following statements holds, except with probability $|\mathbb{F}|^{-s}$ over the choice of $\mathbf{H} \xleftarrow{\$} \mathcal{H}$:*

1. *$\mathbf{XH}^\top$ has distance at least s from $\mathsf{C}^{\odot(2s+t)}$.*
2. *For every $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ there exists a $\mathbf{C} \in \mathsf{C}^{\odot m}$ such that $\mathbf{XH}^\top - \mathbf{C}'$ and $\mathbf{X} - \mathbf{C}$ have the same row support.*

Remark 1 ([16]). If the first item in the statement of Theorem 1 does not hold, the second one must and we can efficiently recover a codeword $\mathbf{C}$ with distance at most $s - 1$ from $\mathbf{X}$ using erasure correction, given a codeword $\mathbf{C}' \in \mathsf{C}^{\odot(2s+t)}$ with distance at most $s - 1$ from $\mathbf{XH}^\top$. More specifically, we compute the row support of $\mathbf{XH}^\top - \mathbf{C}'$, erase the corresponding rows of $\mathbf{X}$ and recover $\mathbf{C}$ from $\mathbf{X}$ using erasure correction[6]. The last step is possible as the distance between $\mathbf{X}$ and $\mathbf{C}$ is at most $s - 1$.

[6] Erasure correction for linear codes can be done efficiently via Gaussian elimination.

2.2 UC Framework and Functionalities

We use the (Global) Universal Composability or (G)UC model [14, 15] for analyzing security and refer interested readers to the original works for more details. Several functionalities in this work allow public verifiability. To model this fact, we follow the approach of Badertscher et al. [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 the adversary to get the list of registered verifiers. All functionalities with public verifiability include the following instructions (which are omitted henceforth for simplicity):

- Upon receiving $(\textsc{Register}, sid)$ from some verifier $\mathcal{V}_i$, set $\mathcal{V} = \mathcal{V} \cup \mathcal{V}_i$ and return $(\textsc{Registered}, sid, \mathcal{V}_i)$ to $\mathcal{V}_i$.
- Upon receiving $(\textsc{Deregister}, sid)$ from some verifier $\mathcal{V}_i$, set $\mathcal{V} = \mathcal{V} \setminus \mathcal{V}_i$ and return $(\textsc{Deregistered}, sid, \mathcal{V}_i)$ to $\mathcal{V}_i$.
- Upon receiving $(\textsc{Is-Registered}, sid)$ from $\mathcal{V}_i$, return $(\textsc{Is-Registered}, sid, b)$ to $\mathcal{V}_i$, where $b = 1$ if $\mathcal{V}_i \in \mathcal{V}$ and $b = 0$ otherwise.
- Upon receiving $(\textsc{Get-Registered}, sid)$ from the ideal adversary $\mathcal{S}$, the functionality returns $(\textsc{Get-Registered}, sid, \mathcal{V})$ to $\mathcal{S}$.

The above instructions can also be used by other functionalities to register as a verifier of a functionality with public verifiability. Following the approach of [5, 29, 28] we use a global clock functionality $\mathcal{F}_{\mathsf{Clock}}$. We also use the restricted programmable and observable global random oracle model $\mathcal{G}_{\mathsf{rpoRO}}$ of [13]. $\mathcal{F}_{\mathsf{Clock}}$ and $\mathcal{G}_{\mathsf{rpoRO}}$ are described in Figure 1 and Figure 2, respectively.

Functionality $\mathcal{F}_{\mathsf{Clock}}$

$\mathcal{F}_{\mathsf{Clock}}$ is parametrized by a variable ν, and sets $\mathcal{P}$ of parties and $\mathcal{F}$ of functionalities. It keeps a Boolean variable $d_{\mathcal{J}}$ for each $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$. All variables are initialized as 0.

Clock Update: Upon receiving a message $(\textsc{Clock-Update})$ from $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$, set $d_{\mathcal{J}} = 1$, run $\mathsf{RoundUpdate}$ and return $(\textsc{Clock-Update}, \mathcal{J})$ to $\mathcal{S}$.

Clock Read: Upon receiving a message $(\textsc{Clock-Read})$ from any entity, answer that entity with $(\textsc{Clock-Read}, \nu)$.

Macro $\mathsf{RoundUpdate}$: If $d_F = 1$ for all $F \in \mathcal{F}$ and $d_p = 1$ for all honest $p \in \mathcal{P}$, then set $\nu \leftarrow \nu + 1$ and reset $d_{\mathcal{J}}$ to 0 for all $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$.

Fig. 1. Functionality $\mathcal{F}_{\mathsf{Clock}}$ for a Global Clock.

2.3 Authenticated Bulletin Boards and Smart Contracts

Bulletin Boards and Smart Contracts are primitives which form the backbone of our result. A Bulletin Board is a publicly readable storage for messages which cannot be erased after been posted. We use an *authenticated* Bulletin Board.

Functionality $\mathcal{G}_{\mathsf{rpoRO}}$

$\mathcal{G}_{\mathsf{rpoRO}}$ is parameterized by an output size function ℓ and keeps initially empty lists $\mathsf{List}_{\mathcal{H}}$,$\mathsf{prog}$.

Query: On input $(\textsc{Hash-Query}, m)$ from party $(\mathcal{P}, sid)$ or $\mathcal{S}$, parse m as (s, m') and proceed as follows:

1. Look up h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, sample $h \stackrel{\$}{\leftarrow} \{0,1\}^{\ell(\tau)}$ and set $\mathsf{List}_{\mathcal{H}} = \mathsf{List}_{\mathcal{H}} \cup \{(m, h)\}$.
2. If this query is made by $\mathcal{S}$, or if $s \neq sid$, then add (s, m', h) to the (initially empty) list of illegitimate queries $\mathcal{Q}_s$.
3. Send $(\textsc{Hash-Confirm}, h)$ to the caller.

Observe: On input $(\textsc{Observe}, sid)$ from $\mathcal{S}$, if $\mathcal{Q}_{sid}$ does not exist yet, set $\mathcal{Q}_{sid} = \emptyset$. Output $(\textsc{List-Observe}, \mathcal{Q}_{sid})$ to $\mathcal{S}$.

Program: On input $(\textsc{Program-RO}, m, h)$ with $h \in \{0,1\}^{\ell(\tau)}$ from $\mathcal{S}$, ignore the input if there exists $h' \in \{0,1\}^{\ell(\tau)}$ where $(m, h') \in \mathsf{List}_{\mathcal{H}}$ and $h \neq h'$. Otherwise, set $\mathsf{List}_{\mathcal{H}} = \mathsf{List}_{\mathcal{H}} \cup \{(m, h)\}$, $\mathsf{prog} = \mathsf{prog} \cup \{m\}$ and send $(\textsc{Program-Confirm})$ to $\mathcal{S}$.

IsProgrammed: On input $(\textsc{IsProgrammed}, m)$ from a party $\mathcal{P}$ or $\mathcal{S}$, if the input was given by $(\mathcal{P}, sid)$ then parse m as (s, m') and, if $s \neq sid$, ignore this input. Set $b = 1$ if $m \in \mathsf{prog}$ and $b = 0$ otherwise. Then send $(\textsc{IsProgrammed}, b)$ to the caller.

Fig. 2. Functionality $\mathcal{G}_{\mathsf{rpoRO}}$ from [13].

This means that messages that are posted can be related to specific parties. This latter primitive can be implemented from a Bulletin Board and signatures.[7] We also assume that there exists a Smart Contract functionality. Such a smart contract is set up and parameterized by $\mathcal{P}$ and obtains further inputs from the Bulletin Board. To simplify this interaction, we will provide one functionality $\mathcal{F}_{\mathsf{SC}}$ which incorporates both the authenticated Bulletin Board and the Smart Contract. Since the Smart Contract will furthermore have to communicate with the MPC functionality for verification, we will only define it in Section 3.

2.4 Secure Multiparty Computation with Punishable Abort and Cash Distribution

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. For this setting, it is known that fairness cannot be achieved [19]. Instead, we let the functionality compute the result $\boldsymbol{y}$ without releasing it. To release it, every party $\mathcal{P}_i$ sends coins $\mathsf{coins}(d)$ to the functionality, which will hand them back if every party obtained $\boldsymbol{y}$. $\mathcal{A}$ will be able to block honest parties from obtaining $\boldsymbol{y}$, but only at the expense of losing money to the honest parties. We call this

[7] There exist impossibility results on realizing this primitive [36, 22], but we avoid these by assuming that the Bulletin Board is not realized by $\mathcal{P}$. Instead, we use a public ledger.

MPC with Punishable Abort or *Insured MPC*. In the case where no party was punished, we furthermore let the parties obtain coins based on $\boldsymbol{y}$. To formalize this step, we define a *Cash Distribution Function*.

Definition 2 (Cash Distribution Function). *Let $g : \mathbb{F}^m \times \mathbb{N}^n \to \mathbb{N}^n$ be a function such that, for all $\boldsymbol{y} \in \mathbb{F}^m, t^{(1)}, \ldots, t^{(n)} \in \mathbb{N}$ 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*.*

In Figure 3 we formally define MPC which has both properties.

Functionality $\mathcal{F}_{\mathsf{Online}}$

This functionality interacts with the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and is parametrized by a timeout limit ρ, a circuit C representing the computation that the parties want to perform, the compensation amount q and the security deposit $d \geq (n-1)q$. $\mathcal{S}$ specifies a set $I \subset [n]$ of corrupted parties. Let g be a cash distribution function.

Input: Upon first input $(\textsc{Input}, sid, i, x^{(i)})$ by $\mathcal{P}_i$ and $(\textsc{Input}, sid, i, \cdot)$ 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 input $(\textsc{Compute}, sid)$ by all parties and if the inputs $(i, x^{(i)})_{i\in[n]}$ for all parties have been stored internally, compute $\boldsymbol{y} = C(x^{(1)}, \ldots, x^{(n)})$. If $\mathcal{S}$ sends $(\textsc{Abort}, sid, i)$ during **Input** or **Evaluate** and $i \in I$ then send $(\textsc{Abort}, sid)$ to all parties and stop.

Deposit: After the evaluation is done successfully, wait for each party $\mathcal{P}_i$ to send $(\textsc{Deposit}, sid, \mathsf{coins}(d+t^{(i)}))$ containing the d coins of the security deposit, the $t^{(i)} \geq 0$ coins that the party want to use as financial input in the computation. Send $(\textsc{Deposited}, sid, \mathcal{P}_i, d+t^{(i)})$ to all parties. If some party fails to lock the coins within the timeout limit ρ (counting from the the first such deposit), then allow the parties to reclaim their coins and abort.

Open: After the deposits are done successfully, send $(\textsc{Result}, sid, \boldsymbol{y})$ to $\mathcal{S}$.

- If $\mathcal{S}$ returns $(\textsc{Ok}, sid, \boldsymbol{y})$, then output $(\textsc{Result}, sid, \boldsymbol{y})$ to the parties, compute $e^{(1)}, \ldots, e^{(n)} \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$. Then run $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$.
- If $\mathcal{S}$ returns $(\textsc{Abort}, sid, J)$ and $J \subseteq I$, $J \neq \emptyset$, set $e^{(i)} \leftarrow d + t^{(i)} + |J| \cdot q$ for each party $\mathcal{P}_i \in \mathcal{P} \setminus J$ and $e^{(i)} \leftarrow d - q \cdot (n - |J|) + t^{(i)}$ for each $\mathcal{P}_i \in J$. Then run $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$.
- If $\mathcal{S}$ returns $(\textsc{Abort-With-Result}, sid, J)$ and $J \subseteq I$, $J \neq \emptyset$, set $e^{(i)} \leftarrow d + t^{(i)} + |J| \cdot q$ for each party $\mathcal{P}_i \in \mathcal{P} \setminus J$ and $e^{(i)} \leftarrow d - q \cdot (n - |J|) + t^{(i)}$ for each $\mathcal{P}_i \in J$. Then run $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$ and send $(\textsc{Result}, sid, \boldsymbol{y})$ to each $\mathcal{P}_i \in \overline{I}$[a].

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

[a] This might be counterintuitive but $\mathcal{S}$ can always allow the honest parties to obtain the result and then abort, unless all communication runs via a public ledger.

Fig. 3. Functionality $\mathcal{F}_{\mathsf{Online}}$ for Secure Multiparty Computation with Punishable Abort and Cash Distribution.

3 Compiling Multiparty Computation to Punish Aborts

In the course of this section, we describe our approach for compiling a generic protocol with publicly verifiable output to a protocol with punishable abort ($\mathcal{F}_{\mathsf{Online}}$). The compiler will work in the following two steps: (i) We give a functionality $\mathcal{F}_{\mathsf{Ident}}$ which describes MPC with 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. We furthermore fully describe the functionality $\mathcal{F}_{\mathsf{SC}}$ which was already mentioned in Section 2. It contains both the smart contract and the authenticated bulletin board that we use. For technical reasons, it is defined using the non-interactive verification interface of $\mathcal{F}_{\mathsf{Ident}}$. (ii) Both of these functionalities are then compiled using a global clock functionality $\mathcal{F}_{\mathsf{Clock}}$ into a new protocol that allows to punish aborts and cheating during the output phase.

3.1 Identifiable Output and How to Compile

$\mathcal{F}_{\mathsf{Ident}}$ (see Figure 4 and Figure 5) 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 [26, 40].

Definition 3 (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 $\boldsymbol{s}^{(1)}, \dots \boldsymbol{s}^{(n-1)} \in \mathbb{F}^m$, the induced function $\hat{f}_i : \mathbb{F}^m \to \mathbb{F}^m$ such that $\hat{f}_i(\cdot) = f(\bar{\boldsymbol{y}}, \boldsymbol{s}^{(1)}, \dots, \boldsymbol{s}^{(i-1)}, \cdot, \boldsymbol{s}^{(i)}, \dots, \boldsymbol{s}^{(n-1)})$ is a bijection which is poly-time computable in both directions.*

The function f depends on $\mathcal{F}_{\mathsf{Ident}}$, i.e. the MPC scheme that we use inside the compiler. $\mathcal{F}_{\mathsf{Ident}}$ will provide both the advice $\bar{\boldsymbol{y}}$ and shares $\boldsymbol{s}^{(i)}$ that are necessary for the reconstruction. In the case of the instantiation presented in Section 5.2, the reconstruction function is simply the XOR between all the $\boldsymbol{s}^{(i)}$, but more sophisticated functions might be plausible.

To reliably reconstruct $\boldsymbol{y}$, each party $\mathcal{P}_i$ sends $\mathsf{coins}(d+t^{(i)})$ to $\mathcal{F}_{\mathsf{SC}}$. 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. In the next step, $\mathcal{F}_{\mathsf{Ident}}$ is used by each party $\mathcal{P}_i$ to reveal its share $\boldsymbol{s}^{(i)}$ to all other parties in a reliable way. We use $\mathcal{F}_{\mathsf{Clock}}$ to determine if all parties opened their shares in time, and $\mathcal{F}_{\mathsf{Ident}}$ will allow to externally verify that each share $\boldsymbol{s}^{(i)}$ indeed is the same that was provided with respect to $f, \overline{\boldsymbol{y}}$. If all parties open the correct shares, then they activate the cash distribution function g of $\mathcal{F}_{\mathsf{SC}}$ to send the correct payoffs to all parties. Except for deposing the money and running the cash distribution, no interaction with $\mathcal{F}_{\mathsf{SC}}$ is needed in a successful protocol run.

Functionality $\mathcal{F}_{\mathsf{Ident}}$ (part 1)

This functionality interacts with the parties $\mathcal{P}$ and also provides an interface to register external verifiers $\mathcal{V}$. It is parametrized by a circuit C with inputs $x^{(1)}, \ldots, x^{(n)}$ and output $\boldsymbol{y} \in \mathbb{F}^m$. $\mathcal{S}$ provides a set $I \subset [n]$ of parties which he corrupts. Let f be a reconstruction function.

Throughout **Init**, **Input**, **Evaluate** and **Share**, $\mathcal{S}$ can at any point send $(\textsc{Abort}, sid)$ to the functionality, upon which it sends $(\textsc{Abort}, sid, \perp)$ to all parties and terminates. Throughout **Reveal** and **Verify**, $\mathcal{S}$ at any point is allowed to send $(\textsc{Abort}, sid, J)$ to the functionality. 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 parties in $\mathcal{P}$ initialize the sets $\mathtt{rev}, \mathtt{ver}, \mathtt{ref}^{(1)}, \ldots, \mathtt{ref}^{(n)} \leftarrow \emptyset$.

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 parties in $\mathcal{P}$ and if the inputs $(i, x^{(i)})_{i \in [n]}$ for all parties have been 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 input $(\textsc{Deliver-Share}, sid, i)$ 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 $\mathcal{P}_i \in \overline{I}$.

Fig. 4. Functionality $\mathcal{F}_{\mathsf{Ident}}$ for an MPC Protocol with Publicly Verifiable Output.

If a party cheats during the opening phase, the protocol instructs all parties to post their shares using $\mathcal{F}_{\mathsf{SC}}$ within a limited time period (enforced by $\mathcal{F}_{\mathsf{Clock}}$). $\mathcal{F}_{\mathsf{SC}}$ will then contact $\mathcal{F}_{\mathsf{Ident}}$ to verify the correctness of the openings and verify if the correct shares were provided or not. An adversary may now withhold his share or provide an incorrect one, thus preventing both $\mathcal{F}_{\mathsf{SC}}$ and the honest parties from obtaining the correct result. In such a case, let $J \subseteq I$ be the set of aborting or cheating parties, and $\overline{J} = \mathcal{P} \setminus J$. Each party from J will be reimbursed by $\mathsf{coins}(d - q \cdot |\overline{J}| + t^{(i)})$, whereas the rest is fairly distributed among the non-cheating parties, which obtain $\mathsf{coins}(d + q \cdot |J| + t^{(i)})$. $\mathcal{F}_{\mathsf{SC}}$ is described in Figure 6.

3.2 The Compiler

We now give a protocol Π_{Compiler} that implements $\mathcal{F}_{\mathsf{Online}}$ in the $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{Clock}}$-hybrid model.

Functionality $\mathcal{F}_{\mathsf{Ident}}$ (part 2)

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

– If $\mathcal{S}$ sends (REVEAL-OK, sid, i) then set $\texttt{rev} \leftarrow \texttt{rev} \cup \{i\}$, send (REVEAL, $sid, i, \boldsymbol{s}^{(i)}$) to all parties in $\mathcal{P}$.
– If $\mathcal{S}$ sends (REVEAL-NOT-OK, sid, i, J) with $J \subseteq I$ then send (REVEAL-FAIL, sid, i) to all parties in $\mathcal{P}$ and set $\texttt{ref}^{(i)} \leftarrow J$.

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

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

Verify: Upon input (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 $\texttt{ver} \neq [n]$ then return (VERIFY-FAIL, $sid, [n] \setminus \texttt{ver}$).
– If $\texttt{ver} = [n]$ and $\texttt{rev} \neq [n]$ then send to $\mathcal{V}_i$ what **Test Reveal** sends.
– If $\texttt{ver} = \texttt{rev} = [n]$ then compute the set $\texttt{ws} \leftarrow \{j \in [n] \mid \boldsymbol{z}^{(j)} \neq \boldsymbol{s}^{(j)}\}$ and return (OPEN-FAIL, $sid, \texttt{ws}$).

Fig. 5. Functionality $\mathcal{F}_{\mathsf{Ident}}$ for an MPC Protocol with Publicly Verifiable Output (continued).

The compiler protocol, as depicted in Π_{Compiler} in Figure 7, runs in 4 rounds, which means that we require $\mathcal{F}_{\mathsf{Clock}}$ to tick 4 times. The first time (between $\nu = 0$ and $\nu = 1$) each party sends the scrambled output $\overline{\boldsymbol{y}}$ to the bulletin board, which later can be used to reconstruct the real output by $\mathcal{F}_{\mathsf{SC}}$ if necessary. If parties do not agree on the same $\overline{\boldsymbol{y}}$ or they do not send such a message at all, then $\mathcal{F}_{\mathsf{SC}}$ ends. Between $\nu = 1$ and $\nu = 2$ each party locks in its coins using $\mathcal{F}_{\mathsf{SC}}$. These will be returned by $\mathcal{F}_{\mathsf{SC}}$ if not all parties lock in their coins.

Between $\nu = 2$ and $\nu = 3$ the parties run the **Reveal** phase of $\mathcal{F}_{\mathsf{Ident}}$ and post the correct money distribution to $\mathcal{F}_{\mathsf{SC}}$, if all shares are revealed. If not, then $\mathcal{F}_{\mathsf{SC}}$ will inform all parties to perform conflict resolution. During this last step (between $\nu = 3$ and $\nu = 4$) each party then posts its share $\boldsymbol{s}^{(i)}$ to the bulletin board. $\mathcal{F}_{\mathsf{SC}}$ will then either obtain the right output $\mathbf{y}$ and send the correct coins according to g to every party (if they all provide the right shares) or punish cheaters.

For this compiler, the following statement can be shown:

Theorem 2. *The protocol Π_{Compiler} UC-securely implements $\mathcal{F}_{\mathsf{Online}}$ in the $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{Clock}}$-hybrid model against a static, active and rushing adversary corrupting up to $n-1$ parties.*

Proof. We construct a simulator $\mathcal{S}$ which will interact with the hybrid-world adversary $\mathcal{A}$ in the presence of $\mathcal{F}_{\mathsf{Online}}$. $\mathcal{S}$ will simulate a protocol instance of Π_{Compiler} and internally run copies of $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{Clock}}$ and $\mathcal{F}_{\mathsf{SC}}$. It therefore simulates

Functionality $\mathcal{F}_{\mathsf{SC}}$

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

Init: Upon receiving the message $(\textsc{Init}, sid)$ from all parties $\mathcal{P}$ send $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$. Afterwards, ignore all messages for **Init**.

Lock-in: Upon receiving the first message $(\textsc{Lock-In}, sid, \mathsf{coins}(d + t^{(i)}))$ from $\mathcal{P}_i$ containing the d coins of the security deposit and the $t^{(i)} \geq 0$ coins that the party wants to use as monetary input in the computation:

1. Check that each $\mathcal{P}_i$ posted $(\mathcal{P}_i, sid, \textsc{Output-Scrambled}, \overline{y})$. Otherwise reimburse $\mathcal{P}_i$ and abort.
2. If $(\textsc{Clock-Read}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu \neq 1$ then refund $\mathcal{P}_i$. Otherwise, send $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.
3. If $(\textsc{Clock-Read}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$ returns $\nu = 2$ and if $(\textsc{Lock-In}, sid, \mathsf{coins}(d+t^{(j)}))$ was not sent by every party, then reimburse each party with its coins and stop, otherwise send $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.

Agreement on the Output: If $(\textsc{Clock-Read}, sid)$ returns $\nu = 3$ and $(\mathcal{P}_i, sid, \textsc{Payout}, (e^{(1)}, \ldots, e^{(n)})) \in \mathcal{M}$ for each $\mathcal{P}_i$, then run $\mathsf{Pay}(e^{(1)}, \ldots, e^{(n)})$.

Conflict Resolution: If $(\textsc{Clock-Read}, sid)$ returns $\nu = 3$ and $(\mathcal{P}_i, sid, \textsc{Payout}, (e^{(1)}, \ldots, e^{(n)})) \in \mathcal{M}$ was not posted by each $\mathcal{P}_i$, send $(\textsc{Conflict-Resolution}, sid)$ to all parties and $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.

If $(\textsc{Clock-Read}, sid)$ returns $\nu = 4$ then do the following:

1. Check if $(\mathcal{P}_i, sid, \textsc{Output-Share}, \boldsymbol{z}^{(i)}) \in \mathcal{M}$. If not, let J be the set of parties for which $\boldsymbol{z}^{(i)}$ is not present. Then run $\mathsf{Punish}(J)$ and stop.
2. Send $(\textsc{Verify}, sid, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)})$ to $\mathcal{F}_{\mathsf{Ident}}$.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\textsc{Verify-Fail}, sid, J)$ then run $\mathsf{Punish}(J)$ and stop.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\textsc{Reveal-Fail}, sid, \mathtt{ref}^{(1)}, \ldots, \mathtt{ref}^{(n)})$ then set $J \leftarrow \bigcup_{i\in[n]} \mathtt{ref}^{(i)}$. Run $\mathsf{Punish}(J)$ and stop.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns $(\textsc{Open-Fail}, sid, J)$ and $J \neq \emptyset$ then run $\mathsf{Punish}(J)$ and stop.
3. Compute $y \leftarrow f(\overline{\boldsymbol{y}}, \boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)})$ and $(e^{(1)}, \ldots, e^{(n)}) \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$. Then run $\mathsf{Pay}(d + e^{(1)}, \ldots, d + e^{(n)})$.

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

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

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

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

Fig. 6. 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.

Protocol Π_{Compiler}

If any party sends $(\textsc{Abort}, sid)$ during **Init**, **Input** or **Evaluate** then abort. Let ρ be a timeout. Initialize $\mathcal{F}_{\mathsf{Clock}}$ with $\mathcal{P}$ and $\mathcal{F}_{\mathsf{SC}}$.

Init: All parties send $(\textsc{Init}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$.

Input: Upon input $x^{(i)} \in \mathbb{F}$ each party $\mathcal{P}_i$ sends $(\textsc{Input}, sid, i, x^{(i)})$ to $\mathcal{F}_{\mathsf{Ident}}$. It furthermore sends $(\textsc{Input}, sid, j, \cdot)$ for all $\mathcal{P}_j \in \mathcal{P} \setminus \{\mathcal{P}_i\}$.

Evaluate: All parties send $(\textsc{Compute}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$. Afterwards, each party $\mathcal{P}_i$ sends $(\textsc{Share}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$ and obtains $\boldsymbol{s}^{(i)}$ as well as $\overline{\boldsymbol{y}}$.

Deposit:

1. Each $\mathcal{P}_i$ sends $(\textsc{Post}, sid, \nu, \mathsf{output\text{-}scrambled}, \overline{\boldsymbol{y}})$ to $\mathcal{F}_{\mathsf{SC}}$.
2. After time ρ each $\mathcal{P}_i$ sends $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.
3. Send $(\textsc{Lock-In}, sid, \mathsf{coins}(d + t^{(i)}))$ to $\mathcal{F}_{\mathsf{SC}}$ and $(\textsc{Clock-Update}, sid)$
4. After time ρ each $\mathcal{P}_i$ sends $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.
5. Send $(\textsc{Read}, sid)$ to $\mathcal{F}_{\mathsf{SC}}$ to obtain $t^{(1)}, \ldots, t^{(n)}$. If $\mathcal{F}_{\mathsf{SC}}$ reimbursed parties because not all parties locked money, then abort.

Open:

1. Each party $\mathcal{P}_i$ sends $(\textsc{Reveal}, sid, i)$ to $\mathcal{F}_{\mathsf{Ident}}$.
2. If $\mathcal{P}_i$ obtains $(\textsc{Reveal}, sid, j, \boldsymbol{s}^{(j)})$ for each $j \in [n]$ until ρ time passed then locally compute $\boldsymbol{y} \leftarrow f(\overline{\boldsymbol{y}}, \boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(n)})$ as well as $(e^{(1)}, \ldots, e^{(n)}) \leftarrow g(\boldsymbol{y}, t^{(1)}, \ldots, t^{(n)})$. Then, send $(\textsc{Post}, sid, \textsc{Payout}, (e^{(1)}, \ldots, e^{(n)}))$ to $\mathcal{F}_{\mathsf{SC}}$.
3. After time ρ each $\mathcal{P}_i$ sends $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.
4. If $\mathcal{P}_i$ obtains $(\textsc{Conflict-Resolution}, sid)$ from $\mathcal{F}_{\mathsf{SC}}$ then it sends $(\textsc{Post}, sid, \textsc{Output-Share}, \boldsymbol{s}^{(i)})$ to $\mathcal{F}_{\mathsf{SC}}$ and $(\textsc{Start-Verify}, sid, i)$ to $\mathcal{F}_{\mathsf{Ident}}$. After time ρ it sends $(\textsc{Clock-Update}, sid)$ to $\mathcal{F}_{\mathsf{Clock}}$.

Fig. 7. The Compiler Protocol Π_{Compiler}

honest parties to communicate with the functionalities and the parties that are controlled by $\mathcal{A}$.

Init: Send messages in the name of the honest parties as in the protocol, send abort message to $\mathcal{F}_{\mathsf{Online}}$ if $\mathcal{A}$ aborts.

Input: Sample random inputs for the simulated honest parties and input these into $\mathcal{F}_{\mathsf{Ident}}$ during the simulation of the protocol. Furthermore, intercept inputs that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{Ident}}$ for the dishonest parties and send these to $\mathcal{F}_{\mathsf{Online}}$. Send an abort message to $\mathcal{F}_{\mathsf{Online}}$ if $\mathcal{A}$ aborts.

Evaluate: Run this step as in the protocol and obtain $\boldsymbol{s}^{(i)}$ for each simulated honest party from $\mathcal{F}_{\mathsf{Ident}}$.

Deposit: Run steps $1, 2$ honestly. Let the parties run steps $3, 4$ honestly. For each honest party $\mathcal{P}_i$, wait for $(\textsc{Deposited}, sid, \mathcal{P}_i, d + t^{(i)})$ from $\mathcal{F}_{\mathsf{Ident}}$ and then use $\mathsf{coins}(d + t^{(i)})$ for the simulated honest party in the protocol. For each dishonest party $\mathcal{P}_i$, if it sends $\mathsf{coins}(d + t^{(i)})$ to $\mathcal{F}_{\mathsf{SC}}$ then send $(\textsc{Deposit},$

$sid, \mathsf{coins}(d + t^{(i)}))$ to $\mathcal{F}_{\mathsf{Online}}$. If a dishonest party does so after step 4 then do not send the message, but instead reclaim money and abort.

Open: Obtain $(\textsc{Result}, sid, \boldsymbol{y})$ from $\mathcal{F}_{\mathsf{Online}}$. Let $\mathcal{P}_j$ be a simulated honest party, then using $\boldsymbol{y}$ compute a new share $\hat{\boldsymbol{s}}^{(j)}$ using the fact that f is a reconstruction function and fixing all inputs and the output except $\hat{\boldsymbol{s}}^{(j)}$. Then, run the protocol honestly and send $(\textsc{Reveal}, sid, i)$ in the name of each simulated honest party and for each $\mathcal{P}_i \in \mathcal{P}$ to $\mathcal{F}_{\mathsf{Ident}}$, but let $\mathcal{F}_{\mathsf{Ident}}$ change the share of $\mathcal{P}_j$ to $\hat{\boldsymbol{s}}^{(j)}$ for consistency. Then do the following for repayment:

- If $\mathcal{F}_{\mathsf{SC}}$ runs Pay in **Agreement on the Output** or step 3 of **Conflict Resolution** then send $(\textsc{Ok}, sid, \boldsymbol{y})$ to $\mathcal{F}_{\mathsf{Online}}$.
- If the honest parties post $\textsc{Payout}$-messages during step 2 of **Open** but $\mathcal{A}$ lets a party abort either during this step or later, then let J be the set of aborting/cheating parties and send $(\textsc{Abort-With-Result}, sid, J)$ to $\mathcal{F}_{\mathsf{Online}}$.
- For every other abort where $\mathcal{F}_{\mathsf{SC}}$ runs $\mathsf{Punish}(J)$ send $(\textsc{Abort}, sid, J)$ to $\mathcal{F}_{\mathsf{Online}}$.

It is easy to see that the output which $\mathcal{A}$ obtains during the protocol 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 the simulator. If the simulated honest parties obtain all shares during the protocol then no party gets punished by $\mathcal{F}_{\mathsf{SC}}$. Therefore, the simulator lets the real honest parties obtain the output in that situation. If $\mathcal{A}$ makes one of the parties abort or send an incorrect message, then this will be detected by $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{S}$ will keep consistency between the protocol and $\mathcal{F}_{\mathsf{Online}}$. If a dishonest party then wishes to deliver its result but abort, then in this case the simulator will send the correct message to $\mathcal{F}_{\mathsf{Online}}$. $\square$

Making the Opening Fully Punishable: Having to run arbitration consumes extra message space on the bulletin board. In our scheme, it might still happen that arbitration does not result in punishment of a party, even though it is only activated if cheating indeed happens. This is a problem, as parties may have to pay for using space on the bulletin board or for running computations using a smart contract.

This phenomenon occurs because the **Open** phase does, for efficiency reasons, not completely operate via the bulletin board, which saves a round of interaction with the bulletin board in case of agreement. To fix this and punish any misbehavior during the opening, one can have each party $\mathcal{P}_i$ post their share $\boldsymbol{s}^{(i)}$ after running **Reveal** instead of posting the payout message ($\mathcal{F}_{\mathsf{SC}}$ can locally compute $e^{(1)}, \ldots, e^{(n)}$). Parties are then allowed to post $\mathsf{disagreement}$ messages about this opening, and **Conflict Resolution** can punish parties for either sending incorrect messages or making false $\mathsf{disagreement}$ claims.

4 Multiparty Homomorphic Commitments with Delayed Public Verifiability

One of the main building blocks of our secure multiparty computation protocol is a (multiparty) additively homomorphic commitment scheme with delayed public verifiability, meaning that the receiver can prove that he received a (potentially) invalid opening to a given commitment after it has been opened. In order to construct such a scheme efficiently, we depart from the multiparty homomorphic commitment scheme of [21], which is in turn realized based on a two-party homomorphic commitment functionality, an equality testing functionality and a coin tossing functionality. In order to augment the construction of [21] with delayed public verifiability, we need to also augment the functionalities it is based on with similar properties. To that end, we present a two-party homomorphic commitment with delayed public verifiability functionality $\mathcal{F}_{\mathsf{2HCom}}$, a publicly verifiable coin tossing functionality $\mathcal{F}_{\mathsf{CT}}$ and a publicly verifiable equality testing functionality $\mathcal{F}_{\mathsf{EQ}}$. We realize $\mathcal{F}_{\mathsf{2HCom}}$ with a construction based on an instantiation of the scheme of [16] with an oblivious transfer with delayed public verifiability $\mathcal{F}_{\mathsf{pOT}}$. We show that $\mathcal{F}_{\mathsf{pOT}}$ can be realized in the restricted programmable and observable random oracle model of [13] by the construction of [39] plus a publicly verifiable (non-homomorphic) commitment functionality $\mathcal{F}_{\mathsf{Com}}$, which is also instrumental in realizing $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$.

Public Verification. In our modeling of public verification, we denote the parties who actively participate in executing a protocol by $\mathcal{P}$ and the parties who later verify the output of an execution of the protocol by $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_\ell\}$. In the case of functionalities with *delayed* public verification, the functionality's interface providing public verification is only activated after a subset of (or all) parties in $\mathcal{P}$ agree with its activation. This delayed activation models the fact that the protocols realizing these functionalities require that a subset (or all) of $\mathcal{P}$ reveal private information (*e.g.* private randomness or inputs) in order for the public verification procedure to be executed given publicly available transcripts and outputs. All messages broadcast by parties $\mathcal{P}$ to parties $\mathcal{V}$ in the protocols described in this section are in fact sent to the smart contract, which makes them accessible to verifiers at any later point. This eliminates the need for $\mathcal{V}$ to be involved in the protocol execution of $\mathcal{P}$, as $\mathcal{V}$ can later retrieve relevant messages from the smart contract. When a protocol in this section says a message m is broadcast, the party broadcasting m sends $(\textsc{Post}, sid, \mathsf{Off}, m)$ to $\mathcal{F}_{\mathsf{SC}}$, posting the message to a bulletin board and increases the identifier $\textsc{Off}$. All parties that expect to receive a broadcast message send $(\textsc{Read}, sid)$ to $\mathcal{F}_{\mathsf{SC}}$ and retrieve the message from the contents of the authenticated bulletin board.

4.1 Publicly Verifiable Commitments

In order to adapt the construction of [21], it is necessary to also realize functionalities $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$ with delayed public verifiability, which can be done from simple (non-homomorphic) commitments with public verifiability. We define a

Functionality $\mathcal{F}_{\mathsf{Com}}$

$\mathcal{F}_{\mathsf{Com}}$ keeps an internal (initially empty) list C and interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ through the following interfaces:

Commit: Upon receiving $(\textsc{Commit}, sid, \mathcal{P}_i, cid, \boldsymbol{x})$ from $\mathcal{P}_i \in \mathcal{P}$ (where $\boldsymbol{x} \in \mathbb{F}^\tau$) check if $(cid, \cdot, \cdot) \in C$. If yes, ignore the message, else add $(cid, \mathcal{P}_i, \boldsymbol{x})$ to C and send a public delayed output $(\textsc{Committed}, sid, \mathcal{P}_i, cid)$ to all remaining parties in $\mathcal{P}$.

Open: Upon receiving $(\textsc{Open}, sid, \mathcal{P}_i, cid)$ from $\mathcal{P}_i \in \mathcal{P}$, if $(cid, \mathcal{P}_i, \boldsymbol{x}) \in C$, send a delayed output $(\textsc{Open}, sid, \mathcal{P}_i, cid, \boldsymbol{x})$ to all parties $\mathcal{V}$ and all $\mathcal{P}_j \in \mathcal{P}$ for $j \neq i$.

Verify: Upon receiving $(\textsc{Verify}, sid, cid, \mathcal{P}_i, \boldsymbol{x})$ from $\mathcal{V}_j \in \mathcal{V}$, if $(cid, \mathcal{P}_i, \boldsymbol{x}) \in C$ set $f = 1$, otherwise, set $f = 0$. Send $(\textsc{Verified}, sid, cid, \mathcal{P}_i, \boldsymbol{x}, f)$ to $\mathcal{V}_j$.

Fig. 8. Functionality $\mathcal{F}_{\mathsf{Com}}$ for Publicly Verifiable Multiparty Commitments.

Protocol Π_{Com}

Parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and verifiers $\mathcal{V}$ interact with each other and with $\mathcal{G}_{\mathsf{rpoRO}}$ as follows:

Commit: On input $(\textsc{Commit}, sid, \mathcal{P}_i, cid, \boldsymbol{x}_i)$, a party $\mathcal{P}_i \in \mathcal{P}$ uniformly samples $\boldsymbol{r} \xleftarrow{\$} \{0,1\}^\kappa$ and queries $\mathcal{G}_{\mathsf{rpoRO}}$ on $(sid, cid, \boldsymbol{r}, \boldsymbol{x}_i)$ to obtain c. $\mathcal{P}_i$ broadcasts $(\textsc{Committed}, sid, \mathcal{P}_i, cid, c)$. All parties $\mathcal{P}_j \in \mathcal{P}$ for $j \neq i$ output $(\textsc{Committed}, sid, \mathcal{P}_i, cid)$ upon receiving this message.

Open: On input $(\textsc{Open}, sid, \mathcal{P}_i, cid)$, $\mathcal{P}_i$ broadcasts $(\textsc{Open}, sid, \mathcal{P}_i, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$. Upon receiving $(\textsc{Open}, sid, \mathcal{P}_i, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$, each party $\mathcal{P}_j$ queries $\mathcal{G}_{\mathsf{rpoRO}}$ on $(sid, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$ and checks that the answer is equal to c and that $(sid, \boldsymbol{r}', \boldsymbol{x}'_i)$ is not programmed by sending $(\textsc{IsProgrammed}, sid, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$ to $\mathcal{G}_{\mathsf{rpoRO}}$, aborting if the answer is $(\textsc{IsProgrammed}, sid, 0)$. Output $(\textsc{Open}, sid, \mathcal{P}_i, cid, \boldsymbol{x}'_i)$.

Verify: On input $(\textsc{Verify}, sid, cid, \mathcal{P}_i, \boldsymbol{x})$, $\mathcal{V}_j \in \mathcal{V}$ checks that $\boldsymbol{x} = \boldsymbol{x}'_i$ in $(\textsc{Open}, sid, \mathcal{P}_i, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$, aborting otherwise. $\mathcal{V}_j$ queries $\mathcal{G}_{\mathsf{rpoRO}}$ on $(sid, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$ and checks that the answer is equal to c and that $(sid, \boldsymbol{r}', \boldsymbol{x}'_i)$ is not programmed by sending $(\textsc{IsProgrammed}, sid, cid, \boldsymbol{r}', \boldsymbol{x}'_i)$ to $\mathcal{G}_{\mathsf{rpoRO}}$, setting $f = 0$ if the answer is $(\textsc{IsProgrammed}, sid, 0)$ and, otherwise, setting $f = 1$. Output $(\textsc{Verified}, sid, cid, \mathcal{P}_i, \boldsymbol{x}, f)$.

Fig. 9. Protocol Π_{Com} for Publicly Verifiable Multiparty Commitments.

functionality for Publicly Verifiable Commitments $\mathcal{F}_{\mathsf{Com}}$ in Figure 8 and will show that this functionality can be realized in the restricted programmable and observable random oracle model of [13]. The basic insight here is to observe that the canonical random oracle based commitment scheme proven UC-secure in [13] is trivially publicly verifiable, since any party can verify the validity of a given commitment/opening pair by querying the global random oracle. We describe protocol Π_{Com} in Figure 9. The security of Π_{Com} is stated in Theorem 3.

Theorem 3. *Protocol Π_{Com} GUC-realizes $\mathcal{F}_{\mathsf{Com}}$ in the $\mathcal{G}_{\mathsf{rpoRO}}, \mathcal{F}_{\mathsf{SC}}$ hybrid model.*

Proof (Sketch). The fact that the Commit and Open steps of protocol Π_{Com} realize the corresponding interfaces of $\mathcal{F}_{\mathsf{Com}}$ in the $\mathcal{G}_{\mathsf{rpoRO}}$ and $\mathcal{F}_{\mathsf{Auth}}$ hybrid model ($\mathcal{F}_{\mathsf{Auth}}$ is the functionality for authenticated channels) is proven in [13]. In our case $\mathcal{F}_{\mathsf{Auth}}$ is substituted by the authenticated bulleting board through which broadcasts are carried out. Public verification follows in a straightforward manner since parties $\mathcal{V}$ receive the same messages as parties $\mathcal{P}$ and perform the exact same procedures of an honest receiver to verify the validity of such messages. Notice that the strategy taken by the simulator described in [13] in exploring the restricted programmability and observability of $\mathcal{G}_{\mathsf{rpoRO}}$ allows it to equivocate commitment openings towards $\mathcal{V}$ as well. Hence, since $\mathcal{G}_{\mathsf{rpoRO}}$ is global the output obtained by $\mathcal{V}$ in the public verification procedure is 1 if and only if the output x was really obtained from a valid opening of the commitment identified by cid.

Functionality $\mathcal{F}_{\mathsf{EQ}}$

$\mathcal{F}_{\mathsf{EQ}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ through the following interfaces:

Equality: Upon receiving $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in \mathbb{F}^m$, from each party $\mathcal{P}_i \in \mathcal{P}$ (or from $\mathcal{S}$ in case $\mathcal{P}_i$ is corrupted), if $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n$, send $(\textsc{Equal}, sid)$ to $\mathcal{S}$. Otherwise, send $(\textsc{Not-Equal}, sid, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ to $\mathcal{S}$. Proceed as follows according to the answer of $\mathcal{S}$:

- If $\mathcal{S}$ answers with $(\textsc{Deliver}, sid)$, send $(\textsc{Equal}, sid)$ to all parties in $\mathcal{P}$ if $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n$ and otherwise send $(\textsc{Not-Equal}, sid, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ to them.
- If $\mathcal{S}$ answers with $(\textsc{Abort}, sid)$, then send $(\textsc{Abort}, sid)$ to all parties.

Verify: Upon receiving $(\textsc{Verify}, sid, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ from $\mathcal{V}_j \in \mathcal{V}$, if messages $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{x}_i)$ (with $\boldsymbol{x}_i \in \mathbb{F}^m$) have been received from each $\mathcal{P}_i \in \mathcal{P}$ and $\mathcal{S}$ did not send $(\textsc{Abort}, sid)$ then set $f = 1$ if $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n$ or otherwise set $f = 0$. Send $(\textsc{Verified}, sid, f)$ to $\mathcal{V}_j$.

Fig. 10. Functionality $\mathcal{F}_{\mathsf{EQ}}$ for Publicly Verifiable Equality Testing.

4.2 Publicly Verifiable Equality Testing

The functionality for Equality Testing as defined in [21] but augmented with Public Verifiability is presented in Figure 10. Notice that this functionality leaks the inputs of all parties to the adversary *after* it provides its inputs. Hence, it must not be used with inputs that must remain private after equality testing is performed. Nevertheless, this relaxed guarantee is enough for realizing the construction of [21] and the functionality $\mathcal{F}_{\mathsf{EQ}}$ itself can be realized using $\mathcal{F}_{\mathsf{Com}}$. The basic idea as proposed in [21] is to have all parties commit to the values whose equality will be tested and, after all commitments are performed, open their commitments and compare the values locally. Since $\mathcal{F}_{\mathsf{Com}}$ is publicly verifiable, the commitments and openings can be publicly verified to check the validity of the equality test. We describe protocol Π_{EQ} in Figure 11. The security of Π_{Com} is stated in Theorem 4.

Protocol Π_{EQ}

Parties $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ and verifiers $\mathcal{V}$ interact with each other and with $\mathcal{F}_{\mathsf{Com}}$ as follows:

Equality: On input $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{x}_i)$, each party $\mathcal{P}_i$ proceeds as follows:

1. Samples a fresh unused cid_i and send $(\textsc{Commit}, sid, \mathcal{P}_i, cid_i, \boldsymbol{x}_i)$ to $\mathcal{F}_{\mathsf{Com}}$.
2. After receiving $(\textsc{Committed}, sid, \mathcal{P}_j, cid_j)$ from $\mathcal{F}_{\mathsf{Com}}$ for all $j \in [n]$ with $j \neq i$, send $(\textsc{Open}, sid, \mathcal{P}_i, cid_i)$ to $\mathcal{F}_{\mathsf{Com}}$.
3. Upon receiving $(\textsc{Open}, sid, \mathcal{P}_j, cid_j, \boldsymbol{x}_j)$ from $\mathcal{F}_{\mathsf{Com}}$ for all $j \in [n]$ with $j \neq i$, output $(\textsc{Equal}, sid)$ if $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n$, otherwise, $(\textsc{Not-Equal}, sid, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$. If $(\textsc{Open}, sid, \mathcal{P}_a, cid_a, \boldsymbol{x}_a)$ is not received for some $a \in [n]$, output $(\textsc{Abort}, sid)$.

Verify: On input $(\textsc{Verify}, sid, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, $\mathcal{V}_j \in \mathcal{V}$ sends $(\textsc{Verify}, sid, cid_i, \boldsymbol{x}_i)$ to $\mathcal{F}_{\mathsf{Com}}$ for all $i \in [n]$. If $\mathcal{V}_j$ receives $(\textsc{Verified}, sid, cid, \mathcal{P}_i, \boldsymbol{x}_i, 1)$ for all $i \in [n]$, it outputs $(\textsc{Verified}, sid, 1)$ if $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n$. Otherwise, it outputs $(\textsc{Verified}, sid, 0)$.

Fig. 11. Protocol Π_{EQ} for Publicly Verifiable Equality Testing.

Theorem 4. *Protocol Π_{EQ} GUC-realizes $\mathcal{F}_{\mathsf{EQ}}$ in the $\mathcal{F}_{\mathsf{Com}}$ hybrid model.*

Proof (Sketch). We'll sketch a simulator $\mathcal{S}$ running an internal copy of the real world adversary $\mathcal{A}$ such that an execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{EQ}}$ is indistinguishable from an execution of Π_{EQ} with $\mathcal{A}$ to the environment $\mathcal{Z}$. $\mathcal{S}$ interacts with $\mathcal{A}$ emulating the honest parties of the protocol and $\mathcal{F}_{\mathsf{Com}}$. On inputs $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{x}_i)$, where $\mathcal{P}_i$ is a corrupted party, $\mathcal{S}$ sends $(\textsc{Committed}, sid, \mathcal{P}_j, cid_j)$ from each simulated honest party $\mathcal{P}_j$ emulating a commitment to a random message from $\mathcal{F}_{\mathsf{Com}}$ to $\mathcal{A}$ and waits for $\mathcal{A}$ to send a $(\textsc{Commit}, sid, \mathcal{P}_i, cid_i, \boldsymbol{x}_i)$ to $\mathcal{F}_{\mathsf{Com}}$. For each corrupted party $\mathcal{P}_i$, $\mathcal{S}$ sends $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{x}_i)$ to $\mathcal{F}_{\mathsf{EQ}}$. Upon receiving $(\textsc{Equal}, sid)$ from $\mathcal{F}_{\mathsf{EQ}}$, if all commitments from $\mathcal{A}$ have been opened with messages $(\textsc{Open}, sid, \mathcal{P}_i, cid_i)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ opens the emulated commitments from honest parties by sending $\mathcal{A}$ a message $(\textsc{Open}, sid, \mathcal{P}_j, cid_j, \boldsymbol{x})$ with $\boldsymbol{x}$ equal to value $\boldsymbol{x}_i$ contained in the messages $(\textsc{Commit}, sid, \mathcal{P}_i, cid_i, \boldsymbol{x}_i)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$ and sends $(\textsc{Deliver}, sid)$ to $\mathcal{F}_{\mathsf{EQ}}$. Upon receiving $(\textsc{Not-Equal}, sid, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ from $\mathcal{F}_{\mathsf{EQ}}$, if all commitments from $\mathcal{A}$ have been opened with messages $(\textsc{Open}, sid, \mathcal{P}_i, cid_i)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ opens the emulated commitments from honest parties by sending $\mathcal{A}$ a message $(\textsc{Open}, sid, \mathcal{P}_j, cid_j, \boldsymbol{x}_j)$ with the corresponding $\boldsymbol{x}_j$ according to $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ received from $\mathcal{F}_{\mathsf{EQ}}$. Upon receiving a message $(\textsc{Verify}, sid, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ from a party $\mathcal{V}_i$, $\mathcal{S}$ emulates Π_{EQ} exactly, given the commitment openings programmed into $\mathcal{F}_{\mathsf{Com}}$.

4.3 Publicly Verifiable Coin Tossing

The functionality for Coin Tossing as defined in [21] but augmented with Public Verifiability is presented in Figure 12. This functionality can also be implemented

Functionality $\mathcal{F}_{\mathsf{CT}}$

$\mathcal{F}_{\mathsf{CT}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ through the following interfaces:

Toss: Upon receiving $(\textsc{Toss}, sid, m, \mathbb{F})$ from all parties in $\mathcal{P}$ where $m \in \mathbb{N}$ and $\mathbb{F}$ is a description of a field, uniformly sample m random elements $x_1, \ldots, x_m \xleftarrow{\$} \mathbb{F}$ and send $(\textsc{Tossed}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ to $\mathcal{S}$. Proceed as follows according to the answer of $\mathcal{S}$:

- If $\mathcal{S}$ answers with $(\textsc{Deliver}, sid)$, send $(\textsc{Tossed}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ to all parties in $\mathcal{P}$.
- If $\mathcal{S}$ answers with $(\textsc{Abort}, sid)$, then send $(\textsc{Abort}, sid)$ to all parties.

Verify: Upon receiving $(\textsc{Verify}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ from $\mathcal{V}_j \in \mathcal{V}$, if $(\textsc{Tossed}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ has been sent to all parties in $\mathcal{P}$ set $f = 1$, otherwise, set $f = 0$. Send $(\textsc{Verified}, sid, m, \mathbb{F}, x_1, \ldots, x_m, f)$ to $\mathcal{V}_j$.

Fig. 12. Functionality $\mathcal{F}_{\mathsf{CT}}$ for Publicly Verifiable Coin Tossing.

using $\mathcal{F}_{\mathsf{Com}}$. The basic coin tossing interface is realized in the standard manner: (i) each party $\mathcal{P}_i \in \mathcal{P}$ commits to a random element $r_i \in \mathbb{F}$ (ii) wait for all other parties to perform their commitments (iii) open the commitment and obtain the opening of all other parties; and (iv) define the final random element $x = \sum_i r_i$. The public verifiability is achieved by relying on the public verifiability of $\mathcal{F}_{\mathsf{Com}}$, which allows parties to check that the openings to each commitment were presented correctly and to locally compute the final random value. We describe the protocol Π_{CT} in Figure 13. The security of Π_{Com} is stated in Theorem 5.

Theorem 5. *Protocol Π_{CT} GUC-realizes $\mathcal{F}_{\mathsf{CT}}$ in the $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{Com}}$ hybrid model.*

Proof (Sketch). We'll sketch a simulator $\mathcal{S}$ running an internal copy of the real world adversary $\mathcal{A}$ such that an execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CT}}$ is indistinguishable from an execution of Π_{CT} with $\mathcal{A}$ to the environment $\mathcal{Z}$. $\mathcal{S}$ interacts with $\mathcal{A}$ emulating the honest parties of the protocol and $\mathcal{F}_{\mathsf{Com}}$. On input $(\textsc{Toss}, sid, m, \mathbb{F})$, $\mathcal{S}$ sends $(\textsc{Toss}, sid, m, \mathbb{F})$ to $\mathcal{F}_{\mathsf{CT}}$ on behalf of the corrupted parties and emulates commitments from each honest party $\mathcal{P}_i$ by uniformly sampling $x_{i,1}, \ldots, x_{i,m} \xleftarrow{\$} \mathbb{F}$ and fresh unused identifiers $cid_{i,k}$, and sending $(\textsc{Commit}, sid, \mathcal{P}_i, cid_{i,k}, x_{i,k})$ to $\mathcal{A}$ for $k \in [m]$. Upon receiving $(\textsc{Tossed}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ from $\mathcal{F}_{\mathsf{CT}}$, if $\mathcal{A}$ opened all of its commitments by sending $(\textsc{Open}, sid, \mathcal{P}_i, cid_{i,k})$ to the emulated $\mathcal{F}_{\mathsf{Com}}$ for all corrupted parties $\mathcal{P}_i$ and $k \in [m]$, $\mathcal{S}$ emulates openings from the honest parties towards $\mathcal{A}$ with messages $(\textsc{Open}, sid, \mathcal{P}_j, cid_{j,k}, x_{j,k})$ from $\mathcal{F}_{\mathsf{Com}}$ with values $x_{j,k}$ such that $x_k = \sum_{j=1}^{n} x_{j,k}$ given values $x_{i,k}$ generated by $\mathcal{A}$ for $k \in [m]$. Upon input $(\textsc{Verify}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$, $\mathcal{S}$ exactly emulates Π_{CT} given the openings programmed into the emulated $\mathcal{F}_{\mathsf{Com}}$.

4.4 Oblivious Transfer with Delayed Public Verifiability

In order to realize $\mathcal{F}_{\mathsf{2HCom}}$, we will require an oblivious transfer functionality with delayed public verifiability with an interface that, when activated by the receiver,

Protocol $\mathit{\Pi}_{\mathsf{CT}}$

Parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and verifiers $\mathcal{V}$ interact with each other and with $\mathcal{F}_{\mathsf{Com}}$ as follows:

Toss: On input $(\textsc{Toss}, sid, m, \mathbb{F})$ where $m \in \mathbb{N}$ and $\mathbb{F}$ is a description of a field, each party $\mathcal{P}_i$ proceeds as follows:

1. Uniformly sample m random elements $x_{i,1}, \ldots, x_{i,m} \xleftarrow{\$} \mathbb{F}$, and for all $k \in [m]$, sample fresh unused identifiers $cid_{i,k}$ and send $(\textsc{Commit}, sid, \mathcal{P}_i, cid_{i,k}, x_{i,k})$ to $\mathcal{F}_{\mathsf{Com}}$.
2. After receiving $(\textsc{Committed}, sid, \mathcal{P}_j, cid_{j,k})$ from $\mathcal{F}_{\mathsf{Com}}$ for all $k \in [m]$ and all $j \in [n]$ with $i \neq j$, send $(\textsc{Open}, sid, \mathcal{P}_i, cid_{i,k})$ to $\mathcal{F}_{\mathsf{Com}}$ for all $k \in [m]$.
3. Upon receiving $(\textsc{Open}, sid, \mathcal{P}_j, cid_{j,k}, x_{j,k})$ from $\mathcal{F}_{\mathsf{Com}}$ for all $k \in [m]$ and all $j \in [n]$ with $i \neq j$, output $(\textsc{Tossed}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$ where $x_k = \sum_{j=1}^{n} x_{j,k}$. If a message $(\textsc{Open}, sid, \mathcal{P}_j, cid_{j,k}, x_{j,k})$ is not received for any value of j or k, outputs $(\textsc{Abort}, sid)$.

Verify: On input $(\textsc{Verify}, sid, m, \mathbb{F}, x_1, \ldots, x_m)$, $\mathcal{V}_j \in \mathcal{V}$ sends $(\textsc{Verify}, sid, cid_{i,k}, x_{i,k})$ to $\mathcal{F}_{\mathsf{Com}}$ for $i \in [n]$ and $k \in [m]$. If $\mathcal{V}_j$ receives $(\textsc{Verified}, sid, cid_{i,k}, \mathcal{P}_i, x_{i,k}, 1)$ for all i and k, and $x_k = \sum_{j=1}^{n} x_{j,k}$ for $k \in [m]$, $\mathcal{V}_j$ sets $f = 1$, otherwise it sets $f = 0$. Output $(\textsc{Verified}, sid, m, \mathbb{F}, x_1, \ldots, x_m, f)$.

Fig. 13. Protocol $\mathit{\Pi}_{\mathsf{CT}}$ For Publicly Verifiable Coin Tossing.

Functionality $\mathcal{F}_{\mathsf{pOT}}$

$\mathcal{F}_{\mathsf{pOT}}$ is parameterized by $\lambda \in \mathbb{N}$, which is publicly known. $\mathcal{F}_{\mathsf{pOT}}$ interacts with a sender $\mathcal{P}_i$, a receiver $\mathcal{P}_j$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$, proceeding as follows:

Transfer: Upon receiving a message $(\textsc{Send}, sid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ from $\mathcal{P}_i$, where $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{F}^{\lambda}$, store the tuple $(sid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ and send $(\textsc{Send}, sid)$ to $\mathcal{P}_i$ and $\mathcal{P}_j$. Ignore further messages from $\mathcal{P}_i$ with the same sid.

Choose: Upon receiving a message $(\textsc{Receive}, sid, c)$ from $\mathcal{P}_j$, where $c \in \{0, 1\}$, check if a tuple $(sid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ was recorded. If yes, send $(sid, \boldsymbol{x}_c)$ to $\mathcal{P}_j$ and $(\textsc{Received}, sid)$ to $\mathcal{S}$, and ignore further messages form $\mathcal{P}_j$ with the same sid. Otherwise, send nothing, but continue running.

Initialize Verification: Upon receiving a message $(\textsc{Verification-Start}, sid)$ from $\mathcal{P}_j$, ignore all other messages but start responding to messages $(\textsc{Verify}, sid, c, \boldsymbol{x})$ in the Public Verification interface.

Public Verification: Upon receiving a message $(\textsc{Verify}, sid, c, \boldsymbol{x})$ from $\mathcal{V}_k \in \mathcal{V}$ where $c \in \{0, 1\}$ and $\boldsymbol{x} \in \mathbb{F}^{\lambda}$, if verification was not activated with a message $(\textsc{Verification-Start}, sid)$ from $\mathcal{P}_j$ or if no $(\textsc{Receive}, sid, c)$ was received from $\mathcal{P}_j$, answer with $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$. If there is no tuple $(sid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ recorded, send $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$ to $\mathcal{V}_k$. Otherwise, if a message $(\textsc{Receive}, sid, c)$ was received from $\mathcal{P}_j$ and a tuple $(sid, \boldsymbol{x}_0, \boldsymbol{x}_1)$ where $\boldsymbol{x}_c = \boldsymbol{x}$ was recorded, set $f = 1$, otherwise, set $f = 0$. Send $(\textsc{Verified}, sid, c, \boldsymbol{x}, f)$ to $\mathcal{V}_k$.

Fig. 14. Functionality $\mathcal{F}_{\mathsf{pOT}}$ For Publicly Verifiable Oblivious Transfer.

allows parties to check that the receiver used a given choice bit (obtaining a given message). The basic 1-out-of-2 string OT functionality $\mathcal{F}_{\mathsf{pOT}}$ augmented with public verifiability is presented in Figure 14. This functionality can be realized by having the receiver use $\mathcal{F}_{\mathsf{Com}}$ to commit to all of its randomness (including the choice bit) before the OT protocol is executed and opening this commitment after the protocol is complete. In order for this construction to work, the OT protocol must be such that the receiver cannot generate two alternative randomness values such that each of these values result in the same (fixed) protocol messages for the receiver but in different outputs being obtained given the (fixed) sender's messages. We will show that the protocol of [39] has this property. Moreover, since we only require static security and are willing to use a protocol with more than two rounds, we will show how to use $\mathcal{F}_{\mathsf{CT}}$ to generate a CRS for the scheme of [39], which can be done in two extra rounds in the $\mathcal{G}_{\mathsf{rpoRO}}$-hybrid model using Protocol Π_{CT} to realize $\mathcal{F}_{\mathsf{CT}}$. We use the scheme of [39] along with $\mathcal{F}_{\mathsf{Com}}$ to construct Protocol Π_{pOT} presented in Figure 15. The security of Π_{pOT} is stated in Theorem 6.

Theorem 6. *Protocol Π_{pOT} GUC-realizes $\mathcal{F}_{\mathsf{pOT}}$ in the $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{CT}}$ hybrid model.*

Proof (Sketch). We'll sketch a simulator $\mathcal{S}$ running an internal copy of the real world adversary $\mathcal{A}$ such that an execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{pOT}}$ is indistinguishable from an execution of Π_{pOT} with $\mathcal{A}$ to the environment $\mathcal{Z}$. $\mathcal{S}$ operates exactly as the simulator of [39] in order to simulate the steps "2. Choose", "3. Transfer" and "4. Choose". In the "1. Generate CRS" step, if $\mathcal{P}_i$ is malicious, $\mathcal{S}$ samples $x, y \xleftarrow{\$} \mathbb{Z}_p$ and $g_0 \xleftarrow{\$} \mathbb{Z}_p$, and emulates $\mathcal{F}_{\mathsf{CT}}$ in such a way that it outputs $g_0, g_0^y, g_0^x, g_0^{xy}$, which will allow the simulator from [39] to extract $\mathcal{P}_i$'s messages. On the other hand, if $\mathcal{P}_j$ is malicious, $\mathcal{S}$ samples $x_0, x_1 \xleftarrow{\$} \mathbb{Z}_p$ and $g_0, g_1 \xleftarrow{\$} \mathbb{G}$, and emulates $\mathcal{F}_{\mathsf{CT}}$ in such a way that it outputs $g_0, g_1, g_0^{x_0}, g_1^{x_1}$, which will allow the simulator from [39] to extract $\mathcal{P}_j$'s choice bit. When simulating the "Start Verification" step, $\mathcal{S}$ allows $\mathcal{P}_j$ to open its commitment with the emulated $\mathcal{F}_{\mathsf{Com}}$. In step "Public Verification", notice that $\mathcal{V}_i$ learns $\mathsf{sk} = r, c$ from $\mathcal{F}_{\mathsf{Com}}$ and that it has also learned $(sid, \mathsf{pk} = (g, h))$ and $(sid, \mathsf{ct}_0, \mathsf{ct}_1)$ if those messages have been sent. Hence, it can trivially check that both $\mathcal{P}_i$ and $\mathcal{P}_j$ have participated in the protocol and that $\mathcal{P}_j$ has activated the public verification procedure by opening its commitment. Notice that given a fixed value for $\mathsf{pk} = (g, h)$, $\mathcal{P}_j$ cannot claim a different value of $\mathsf{sk} = (r)$ and vice versa. Given a fixed value of $\mathsf{ct}_c = (g_c^s h_c^t, g^s h^t \cdot m)$ and a fixed r (as argued before), the decryption check performed by $\mathcal{V}_i$ only passes if the c obtained from the commitment is the same that was used in the protocol, which results in the relation $\frac{g^s h^t \cdot m}{(g_c^s h_c^t)^r} = \frac{(g_c^r)^s (h_c^r)^t \cdot m}{(g_c^s h_c^t)^r}$. Hence, $\mathcal{V}_i$ only outputs $(\textsc{Verified}, sid, c, x, 1)$ if $\mathcal{P}_j$ has indeed used c and received x in the session identified by sid.

Protocol Π_{pOT}

Parties $\mathcal{P}_i$,$\mathcal{P}_j$ and verifiers $\mathcal{V}$ interact with each other, with $\mathcal{F}_{\mathsf{Com}}$ and with $\mathcal{F}_{\mathsf{CT}}$ as follows:

1. Generate CRS: When first activated, both $\mathcal{P}_i$ and $\mathcal{P}_j$ send $(\textsc{Toss}, sid, 4, \mathbb{G})$ to $\mathcal{F}_{\mathsf{CT}}$.[a] If $\mathcal{F}_{\mathsf{CT}}$ answers with $(\textsc{Tossed}, sid, m, \mathbb{G}, g_0, g_1, h_0, h_1)$, both $\mathcal{P}_i$ and $\mathcal{P}_j$ set $\mathsf{crs} = (g_0, g_1, h_0, h_1)$. If $\mathcal{F}_{\mathsf{CT}}$ answers with $(\textsc{Abort}, sid)$, both $\mathcal{P}_i$ and $\mathcal{P}_j$ output $(\textsc{Abort}, sid)$ and halt.

2. Choose: On input $(\textsc{Receive}, sid, c)$, $\mathcal{P}_j$ uniformly samples a fresh identifier cid_j and $r \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, and sends $(\textsc{Commit}, sid, \mathcal{P}_j, cid_j, c||r)$ to $\mathcal{F}_{\mathsf{Com}}$. $\mathcal{P}_j$ computes $\mathsf{pk} = (g_c^r, h_c^r), \mathsf{sk} = r$ and broadcasts (sid, pk).

3. Transfer: On input $(\textsc{Send}, sid, x_0, x_1)$, upon receiving (sid, pk) from $\mathcal{P}_j$, $\mathcal{P}_i$ outputs $(\textsc{Abort}, sid)$ and halts if it has not received $(\textsc{Committed}, sid, \mathcal{P}_j, cid_j)$ from $\mathcal{F}_{\mathsf{Com}}$. Otherwise, $\mathcal{P}_i$ parses $\mathsf{pk} = (g, h)$ and, for $c \in \{0, 1\}$, samples $s, t \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, computes $u = g_c^s h_c^t$, $v = g^s h^t$ and $\mathsf{ct}_c = (u, m \cdot v)$. $\mathcal{P}_i$ broadcasts $(sid, \mathsf{ct}_0, \mathsf{ct}_1)$.

4. Finalize Transfer: Upon receiving $(sid, \mathsf{ct}_0, \mathsf{ct}_1)$ from $\mathcal{P}_i$, $\mathcal{P}_j$ parses $\mathsf{ct}_c = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1)$ and computes $x_c = \frac{\tilde{\mathsf{ct}}_1}{\tilde{\mathsf{ct}}_0^{\mathsf{sk}}}$. $\mathcal{P}_j$ outputs $(\textsc{Received}, sid)$.

Initialize Verification: On input $(\textsc{Verification-Start}, sid)$, $\mathcal{P}_j$ sends $(\textsc{Open}, sid, \mathcal{P}_j, cid_j)$ to $\mathcal{F}_{\mathsf{Com}}$.

Public Verification: On input $(\textsc{Verify}, sid, c, x)$, $\mathcal{V}_k \in \mathcal{V}$ outputs $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$ if it has not received $(\textsc{Open}, sid, \mathcal{P}_j, cid_j, c||r)$ from $\mathcal{F}_{\mathsf{Com}}$ or (sid, pk) from $\mathcal{P}_j$. If it has not received $(sid, \mathsf{ct}_0, \mathsf{ct}_1)$ from $\mathcal{P}_i$, $\mathcal{V}_k$ outputs $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$. Otherwise, if it has received (sid, pk) from $\mathcal{P}_j$ and $(sid, \mathsf{ct}_0, \mathsf{ct}_1)$ from $\mathcal{P}_i$, and $x = \frac{\tilde{\mathsf{ct}}_1}{\tilde{\mathsf{ct}}_0^{\mathsf{sk}}}$, $\mathcal{V}_k$ sets $f = 1$ (otherwise, it sets $f = 0$) and outputs $(\textsc{Verified}, sid, c, x, f)$.

[a] We abuse notation and assume that $\mathcal{F}_{\mathsf{CT}}$ also handles representations of a group $\mathbb{G}$, which can be done by Protocols Π_{CT} and Π_{Com} using a $\mathcal{G}_{\mathsf{rpoRO}}$ where the domain is $\mathbb{G}$.

Fig. 15. Protocol Π_{pOT} for Publicly Verifiable Oblivious Transfer.

4.5 Homomorphic Two-Party Commitments with Delayed Public Verifiability

In order to construct homomorphic multiparty commitments with delayed public verifiability using the construction of [21], we will first need to define homomorphic two-party commitments with delayed public verifiability, which will serve as the main building block. This functionality performs the usual actions of a two-party homomorphic commitment but is augmented with an interface that, when activated by the receiver, allows parties to verify that the receiver obtained a given message from a given valid opening of a commitment. This is described in functionality $\mathcal{F}_{\mathsf{2HCom}}$ in Figure 18. We will show how to use the construction of [16] together with $\mathcal{F}_{\mathsf{pOT}}$ to efficiently realize $\mathcal{F}_{\mathsf{2HCom}}$. The main idea is that the receiver can reveal his view of the watchlist used by the scheme of [16] (*i.e.* the random seeds received from $\mathcal{F}_{\mathsf{ROT}}$), which can be publicly verified with $\mathcal{F}_{\mathsf{ROT}}$.

Functionality $\mathcal{F}_{\mathsf{ROT}}$

$\mathcal{F}_{\mathsf{ROT}}$ interacts with a sender $\mathcal{P}_i$, a receiver $\mathcal{P}_j$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$, proceeding as follows:

Both parties are honest: $\mathcal{F}_{\mathsf{ROT}}$ waits for messages (SENDER, sid) and (RECEIVER, sid) from $\mathcal{P}_i$ and $\mathcal{P}_j$, respectively. Then $\mathcal{F}_{\mathsf{ROT}}$ samples random bits $(b_1, \dots, b_n) \xleftarrow{\$} \{0,1\}^n$ and two random matrices $\mathbf{R_0}, \mathbf{R_1} \xleftarrow{\$} \{0,1\}^{n \times m}$ with n rows and m columns. It computes a matrix $\mathbf{S}$ such that for $i \in [n]$: $\mathbf{S}[i,\cdot] = \mathbf{R_{b_i}}[i,\cdot]$.[a] It sends $(sid, \mathbf{R_0}, \mathbf{R_1})$ to $\mathcal{P}_i$ and $(sid, b_1, \dots, b_n, \mathbf{S})$ to $\mathcal{P}_j$. That is, for each row-position, $\mathcal{P}_j$ learns a row of $\mathbf{R_0}$ or of $\mathbf{R_1}$, but $\mathcal{P}_i$ does not know the selection. Record tuples $(sid, \mathbf{R_0}, \mathbf{R_1})$ and $(sid, b_1, \dots, b_n, \mathbf{S})$.

$\mathcal{P}_i$ **is corrupted:** $\mathcal{F}_{\mathsf{ROT}}$ waits for messages (RECEIVER, sid) from $\mathcal{P}_j$ and (ADVERSARY, $sid, \mathbf{R_0}, \mathbf{R_1}$) from $\mathcal{S}$. $\mathcal{F}_{\mathsf{ROT}}$ samples $(b_1, \dots, b_n) \xleftarrow{\$} \{0,1\}^n$, sets $\mathbf{S}[i,\cdot] = \mathbf{R_{b_i}}[i,\cdot]$ for $i \in [n]$ and sends $(sid, b_1, \dots, b_n, \mathbf{S})$ to $\mathcal{P}_j$. Record tuples $(sid, \mathbf{R_0}, \mathbf{R_1})$ and $(sid, b_1, \dots, b_n, \mathbf{S})$.

$\mathcal{P}_j$ **is corrupted:** $\mathcal{F}_{\mathsf{ROT}}$ waits for messages (SENDER, sid) from $\mathcal{P}_i$ and (ADVERSARY, $sid, b_1, \dots, b_n, \mathbf{S}$) from $\mathcal{S}$. $\mathcal{F}_{\mathsf{ROT}}$ samples random matrices $\mathbf{R_0}, \mathbf{R_1} \xleftarrow{\$} \{0,1\}^{n \times m}$, subject to $\mathbf{S}[i,\cdot] = \mathbf{R_{b_i}}[i,\cdot]$, for $i \in [n]$. $\mathcal{F}_{\mathsf{ROT}}$ sends $(sid, \mathbf{R_0}, \mathbf{R_1})$ to $\mathcal{P}_i$. Record tuples $(sid, \mathbf{R_0}, \mathbf{R_1})$ and $(sid, b_1, \dots, b_n, \mathbf{S})$.

Initialize Verification: Upon receiving a message (VERIFICATION-START, sid) from $\mathcal{P}_j$, ignore all other messages but start responding to messages (VERIFY, $sid, b_1, \dots, b_n, \mathbf{S}$) in the Public Verification interface.

Public Verification: Upon receiving a message (VERIFY, $sid, b_1, \dots, b_n, \mathbf{S}$) from $\mathcal{V}_k \in \mathcal{V}$, if verification was not activated with a message (VERIFICATION-START, sid) from $\mathcal{P}_j$ or if no (RECEIVER, sid) (resp. (ADVERSARY, $sid, b_1, \dots, b_n, \mathbf{S}$)) was received from $\mathcal{P}_j$ (resp. $\mathcal{S}$), answer with (VERIFICATION-FAIL, $sid, \mathcal{P}_j$). If there is no tuple $(sid, \mathbf{R_0}, \mathbf{R_1})$ recorded, send (VERIFICATION-FAIL, $sid, \mathcal{P}_i$) to $\mathcal{V}_k$. Otherwise, if a tuple $(sid, b'_1, \dots, b'_n, \mathbf{S}')$ where $(b'_1, \dots, b'_n, \mathbf{S}') = (b_1, \dots, b_n, \mathbf{S})$ was recorded, set $f = 1$, otherwise, set $f = 0$. Send (VERIFIED, $sid, b_1, \dots, b_n, \mathbf{S}, f$) to $\mathcal{V}_k$.

[a] Notice that $\mathbf{S}$ can equivalently be specified as $\mathbf{S} = \mathbf{\Delta R_1} + (\mathbf{I} - \mathbf{\Delta})\mathbf{R_0}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{\Delta}$ is the diagonal matrix with $b_1, \dots, b_n$ on the diagonal.

Fig. 16. Functionality $\mathcal{F}_{\mathsf{ROT}}$.

Given the receiver's view of the watchlist, a commitment and corresponding opening information, any party can run the procedures of an honest receiver in the construction of [16] to verify that the commitments were indeed opened to the messages the receiver claims (or that an invalid opening was given by the sender).

Random Oblivious Transfer with Delayed Public Verifiability A random oblivious transfer functionality that works on matrices suffices for instantiating the commitment protocols described in the remainder of this section. We will add a public verification interface to the functionality presented in [16] and show how it can be instantiated in the $\mathcal{F}_{\mathsf{pOT}}$-hybrid model following the con-

Protocol Π_{ROT}

We assume that all parties have access to a pseudorandom number generator PRG. A sender $\mathcal{P}_i$, a receiver $\mathcal{P}_j$ and verifiers $\mathcal{V}$ interact with each other and with $\mathcal{F}_{\mathsf{pOT}}$ as follows:

1. **OT Phase:** For $i \in [n]$, $\mathcal{P}_i$ samples random $\boldsymbol{r_{0,i}}, \boldsymbol{r_{1,i}} \xleftarrow{\$} \{0,1\}^{\kappa}$ and sends $(\textsc{Send}, sid_i, \boldsymbol{r_{0,i}}, \boldsymbol{r_{1,i}})$ to $\mathcal{F}_{\mathsf{pOT}}$, while $\mathcal{P}_j$ samples $b_i \xleftarrow{\$} \{0,1\}$ and sends $(\textsc{Receive}, sid_i, b_i)$ to $\mathcal{F}_{\mathsf{pOT}}$.
2. **Seed Expansion Phase:** For $i \in [n]$, $\mathcal{P}_i$ sets $\mathbf{R_0}[i,\cdot] = \mathsf{PRG}(\boldsymbol{r_{0,i}})$ and $\mathbf{R_1}[i,\cdot] = \mathsf{PRG}(\boldsymbol{r_{1,i}})$, while $\mathcal{P}_j$ sets $\mathbf{S}[i,\cdot] = \mathsf{PRG}(\boldsymbol{r_{b_i,i}})$. $\mathcal{P}_i$ outputs $(\mathbf{R_0}, \mathbf{R_1})$ and $\mathcal{P}_j$ outputs $(b_1, \ldots, b_n, \mathbf{S})$.

Initialize Verification: On input $(\textsc{Verification-Start}, sid)$, $\mathcal{P}_j$ sends $(\textsc{Verification-Start}, sid)$ to $\mathcal{F}_{\mathsf{pOT}}$.

Public Verification: On input $(\textsc{verify}, sid, b_1, \ldots, b_n, \mathbf{S})$, $\mathcal{V}_k \in \mathcal{V}$ sends $(\textsc{Verify}, sid, b_i, \mathbf{S}[i,\cdot])$ to $\mathcal{F}_{\mathsf{pOT}}$ for $i \in [n]$. Upon receiving $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$ or $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{pOT}}$ for any $i \in [n]$, $\mathcal{V}_k$ outputs the same message. Upon receiving $(\textsc{Verified}, sid, b_i, \mathbf{S}[i,\cdot], 0)$ for any $i \in [n]$, $\mathcal{V}_k$ outputs $(\textsc{Verified}, sid, b_1, \ldots, b_n, \mathbf{S}, 0)$. Upon receiving $(\textsc{Verified}, sid, b_i, \mathbf{S}[i,\cdot], 1)$ for all $i \in [n]$, $\mathcal{V}_k$ outputs $(\textsc{Verified}, sid, b_1, \ldots, b_n, \mathbf{S}, 1)$.

Fig. 17. Protocol Π_{ROT}.

struction of [16]. Functionality $\mathcal{F}_{\mathsf{ROT}}$ as defined in [16] is presented in Figure 16. Protocol Π_{ROT} presented in Figure 17 (and based on the protocol presented in [16]) realizes $\mathcal{F}_{\mathsf{ROT}}$ in the $\mathcal{F}_{\mathsf{pOT}}$-hybrid model. The basic idea is to invoke several instances of $\mathcal{F}_{\mathsf{pOT}}$ where the sender inputs short seeds and the receiver inputs random choices bits. After all instances of $\mathcal{F}_{\mathsf{pOT}}$ are executed, both parties use a PRG to extend the seeds they hold. The proof follows trivially from the proof presented in [16] and the public verifiability of $\mathcal{F}_{\mathsf{pOT}}$.

Protocol $\boldsymbol{\Pi_{\mathsf{2HCom}}}$ We describe protocol Π_{2HCom} in Figure 19 and Figure 20. This protocol is basically the protocol of [16] in almost verbatim form with an interface for computing linear combinations (instead of individual additions) and added public verification steps, which are constructed using the public verification interfaces of $\mathcal{F}_{\mathsf{ROT}}$ as described above. The security of Π_{2HCom} is stated in Theorem 7.

Theorem 7. *Protocol Π_{2HCom} GUC-realizes $\mathcal{F}_{\mathsf{2HCom}}$ in the $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{ROT}}$ hybrid model.*

Proof (Sketch). In order to prove this protocol secure we observe that there exists a simulator $\mathcal{S}$ running an internal copy of the real world adversary $\mathcal{A}$ such that an execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{2HCom}}$ is indistinguishable from an execution of Π_{2HCom} with $\mathcal{A}$ to the environment $\mathcal{Z}$. $\mathcal{S}$ operates exactly as the simulator of [16] in order to simulate the Commit, Linear Combination and Opening phases. Although

Functionality $\mathcal{F}_{\mathsf{2HCom}}$

$\mathcal{F}_{\mathsf{2HCom}}$ is parameterized by $k \in \mathbb{N}$. $\mathcal{F}_{\mathsf{2HCom}}$ interacts with parties $\mathcal{P}_i, \mathcal{P}_j$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ (who may abort at any time) through the following interfaces:

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

Commit: Upon receiving $(\textsc{Commit}, sid, \mathcal{I})$ from $\mathcal{P}_i$ where $\mathcal{I}$ is a set of unused identifiers, send $(\textsc{Commit}, sid, \mathcal{I})$ to $\mathcal{S}$ and proceed as follows:

1. If $\mathcal{S}$ sends $(\textsc{Corrupt}, sid, \{(cid, \boldsymbol{x}_{cid})\}_{cid \in \mathcal{I}})$ and $\mathcal{P}_i$ is corrupted, ignore the next step and proceed to Step 3.
2. If $\mathcal{S}$ answers $(\textsc{No-Corrupt}, sid, \mathcal{I})$, for every $cid \in \mathcal{I}$, sample $\boldsymbol{x}_{cid} \xleftarrow{\$} \mathbb{F}^k$.
3. Set $\mathsf{raw}[cid] = \boldsymbol{x}_{cid}$, send $(\textsc{Commit-Recorded}, sid, \mathcal{I}, \{(cid, \boldsymbol{x}_{cid})\}_{cid \in \mathcal{I}})$ to $\mathcal{P}_i$ and send $(\textsc{Commit-Recorded}, sid, \mathcal{I})$ to $\mathcal{P}_j$ and $\mathcal{S}$.

Input: Upon receiving a message $(\textsc{Input}, sid, \mathcal{P}_i, cid, \boldsymbol{y})$ from $\mathcal{P}_i$, if $\mathsf{raw}[cid] = \boldsymbol{x}_{cid} \neq \perp$, set $\mathsf{actual}[cid] = \boldsymbol{y}$, set $\mathsf{raw}[cid] = \perp$, and send $(\textsc{Input-Recorded}, sid, \mathcal{P}_i, cid)$ to $\mathcal{P}_j$ and $\mathcal{S}$. Otherwise broadcast $(\textsc{Abort}, sid,)$ and halt.

Random: Upon receiving a message $(\textsc{Random}, sid, cid)$ from $\mathcal{P}_i$, if $\mathsf{raw}[cid] = \boldsymbol{x}_{cid} \neq \perp$, set $\mathsf{actual}[cid] = \boldsymbol{x}_{cid}$, set $\mathsf{raw}[cid] = \perp$, and send $(\textsc{Random-Recorded}, sid, cid)$ to $\mathcal{P}_j$ and $\mathcal{S}$. Otherwise broadcast $(\textsc{Abort}, sid,)$ and halt.

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

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

Initialize Verification: Upon receiving $(\textsc{Verification-Start}, sid)$ from $\mathcal{P}_i$ and $\mathcal{P}_j$, stop responding to all messages with this sid in all other interfaces but Public Verification.

Public Verification: Upon receiving $(\textsc{Verify}, sid, cid, \boldsymbol{x}'_{cid})$ from a party $\mathcal{V}_v \in \mathcal{V}$, if $\mathcal{P}_i/\mathcal{P}_j$ has not sent a message $(\textsc{Verification-Start}, sid)$, send $(\textsc{Verify-Fail}, sid, \mathcal{P}_i)/(\textsc{Verify-Fail}, sid, \mathcal{P}_j)$ to $\mathcal{V}_v$. Otherwise, if a message $(\textsc{Open}, sid, cid)$ has not been received from $\mathcal{P}_i$, send $(\textsc{Verify-Fail}, sid, \mathcal{P}_i)$ to $\mathcal{V}_v$. Otherwise, if a message $(\textsc{Open}, sid, cid)$ has been received from $\mathcal{P}_i$ and $\mathsf{actual}[cid] = \boldsymbol{x}_{cid} = \boldsymbol{x}'_{cid}$, set $f = 1$ (otherwise set $f = 0$) and send $(\textsc{Verified}, sid, cid, f)$ to $\mathcal{V}_v$.

Fig. 18. Functionality $\mathcal{F}_{\mathsf{2HCom}}$ For Homomorphic Two-party Commitment With Delayed Public Verifiability

the protocol of [16] only handles individual additions, its proof techniques can be trivially extended to handle a linear combination, which simply consists of multiple additions of commitments.

The main difference in protocol Π_{2HCom} is that it provides a public verification procedure. We will show that this procedure only succeeds if the protocol was correctly executed and only pinpoints a party as responsible for a failure if this party indeed disrupted an honest execution. First, we observe that all the messages exchanged during the protocol are broadcast to the verifier parties $\mathcal{V}$, making it impossible for either $\mathcal{P}_i$ or $\mathcal{P}_j$ to later provide an alternative protocol transcript for verification. However, the private view of $\mathcal{P}_j$ consisting of $b_1, \ldots, b_n, \mathbf{B}$ is only revealed once the verification procedure is initialized. Notice that the public verification procedure of $\mathcal{F}_{\mathsf{ROT}}$ guarantees that $\mathcal{P}_j$'s view as broadcast in the verification initialization procedure of Π_{2HCom} is correct. Given that the protocol transcript received by parties $\mathcal{V}$ through the broadcast channel are immutable and that the values $b_1, \ldots, b_n, \mathbf{B}$ are guaranteed by $\mathcal{F}_{\mathsf{ROT}}$ to be correct, a verifier $\mathcal{V}$ following the instructions of an honest receiver $\mathcal{P}_j$ will only output $(\textsc{Verified}, sid, cid, f)$ if a valid opening for the commitment identified by cid was provided by $\mathcal{P}_i$. Moreover, observing the transcript, any verifier $\mathcal{V}$ can readily check whether $\mathcal{P}_i$ has failed to provide valid messages or whether $\mathcal{P}_j$ has claimed an opening that is invalid.

4.6 Homomorphic Multiparty Commitments with Delayed Public Verifiability

In Figure 21, we present a functionality for multiparty commitments with delayed public verifiability based on the functionality of [21]. As shown in [21], versions of $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$ without delayed public verifiability can be used to realize a version of $\mathcal{F}_{\mathsf{HCom}}$ also without delayed public verifiability. We will focus on showing how delayed public verifiability can be added to the construction of [21] assuming the underlying functionalities also have this property. Using the same principle as in the construction of Π_{2HCom}, we show that the public verification mechanisms of $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$ can be used to obtain the full view of the receiving parties (including secret states). Given that the verifiers know the full transcript of the protocol and are guaranteed to have obtained the view of the receiving parties, they can run the procedure of honest verifying parties to check that a commitment was opened to an specific message. We describe Protocol Π_{HCom} in Figures 22 and 23 as presented in [21], but add the delayed public verification mechanism to it. The security of the protocol is stated in Theorem 8.

Theorem 8. *Protocol Π_{HCom} GUC-realizes $\mathcal{F}_{\mathsf{HCom}}$ in the $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{CT}}$ hybrid model.*

Proof (Sketch). In order to prove this protocol secure we observe that there exists a simulator $\mathcal{S}$ running an internal copy of the real world adversary $\mathcal{A}$ such that an execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{HCom}}$ is indistinguishable from an execution of Π_{HCom} with $\mathcal{A}$ to the environment $\mathcal{Z}$. $\mathcal{S}$ operates exactly as the simulator of [21] in order to simulate the Commit, Linear Combination and Opening phases. We will show that public verification holds given that $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{F}_{\mathsf{CT}}$ also

Protocol Π_{2HCom} (Commitment Phase)

Let C be a systematic binary linear $[n, k, s]$ code, where s is the statistical security parameter. Let $\mathcal{H}$ be a family of linear almost universal hash functions $\mathbf{H} : \{0,1\}^m \to \{0,1\}^\ell$. A sender $\mathcal{P}_i$, a receiver $\mathcal{P}_j$ and verifiers $\mathcal{V}$ interact with each other and $\mathcal{F}_{\mathsf{ROT}}$, proceeding as follows:

Init: On input $(\textsc{Init}, sid)$, $\mathcal{P}_i$ initializes empty lists $\mathsf{raw} = \mathsf{actual} = \emptyset$.
Commit: On input $(\textsc{Commit}, sid, \mathcal{I})$, where $\mathcal{I} = \{cid_1, \ldots, cid_{m-\ell}\}$, $\mathcal{P}_i$ and $\mathcal{P}_j$ proceed as follows:

1. $\mathcal{P}_i$ and $\mathcal{P}_j$ send $(\textsc{Sender}, sid)$ and $(\textsc{Receiver}, sid)$ to $\mathcal{F}_{\mathsf{pOT}}$, respectively. $\mathcal{P}_i$ receives $(sid, \mathbf{R_0}, \mathbf{R_1})$ from $\mathcal{F}_{\mathsf{ROT}}$ and sets $\mathbf{R} = \mathbf{R_0} + \mathbf{R_1}$. $\mathcal{P}_j$ receives $(sid, b_1, \ldots, b_n, \mathbf{S})$ from $\mathcal{F}_{\mathsf{ROT}}$ and sets the diagonal matrix $\mathbf{\Delta}$ such that it contains $b_1, \ldots, b_n$ on the diagonal. $\mathbf{R}$ will contain in the top k rows the data to commit to. Note that $\mathbf{R_0}, \mathbf{R_1}$ form an additive secret sharing of $\mathbf{R}$, and in each row $\mathcal{P}_j$ knows shares from either $\mathbf{R_0}$ or $\mathbf{R_1}$.
2. $\mathcal{P}_i$ now adjusts the bottom $n-k$ rows of $\mathbf{R}$ so that all columns are codewords in C, and $\mathcal{P}_j$ will adjust his shares accordingly, as follows: $\mathcal{P}_i$ constructs a matrix $\mathbf{W}$ with dimensions as $\mathbf{R}$ and 0s in the top k rows, such that $\mathbf{A} := \mathbf{R} + \mathbf{W} \in \mathsf{C}^{\odot \mathsf{m}}$ (recall that C is systematic). $\mathcal{P}_i$ broadcasts $(sid, \mathbf{W})$ (of course, only the bottom $n-k$ rows need to be sent).
3. $\mathcal{P}_i$ sets $\mathbf{A_0} = \mathbf{R_0}, \mathbf{A_1} = \mathbf{R_1} + \mathbf{W}$ and $\mathcal{P}_j$ sets $\mathbf{B} = \mathbf{\Delta W} + \mathbf{S}$. Note that now we have
$$\mathbf{A} = \mathbf{A_0} + \mathbf{A_1},\ \mathbf{B} = \mathbf{\Delta A_1} + (\mathbf{I} - \mathbf{\Delta})\mathbf{A_0},\ \mathbf{A} \in \mathsf{C}^{\odot \mathsf{m}}\ ,$$
i.e., $\mathbf{A}$ is additively shared and for each row index, $\mathcal{P}_j$ knows either a row from $\mathbf{A_0}$ or from $\mathbf{A_1}$.
4. $\mathcal{P}_j$ chooses a seed H' for a random function $\mathbf{H} \in \mathcal{H}$ and broadcasts (sid, H'), we identify the function with its matrix (recall that all functions in $\mathcal{H}$ are linear).
5. $\mathcal{P}_i$ computes $\mathbf{T_0} = \mathbf{A_0 H}, \mathbf{T_1} = \mathbf{A_1 H}$ and broadcasts $(sid, \mathbf{T_0}, \mathbf{T_1})$. Note that $\mathbf{AH} = \mathbf{A_0 H} + \mathbf{A_1 H} = \mathbf{T_0} + \mathbf{T_1}$, and $\mathbf{AH} \in \mathsf{C}^{\odot \ell}$. So we can think of $\mathbf{T_0}, \mathbf{T_1}$ as an additive sharing of $\mathbf{AH}$, where again $\mathcal{P}_j$ knows some of the shares, namely the rows of $\mathbf{BH}$.
6. $\mathcal{P}_j$ checks that $\mathbf{\Delta T_0} + (\mathbf{I} - \mathbf{\Delta})\mathbf{T_1} = \mathbf{BH}$ and that $\mathbf{T_0} + \mathbf{T_1} \in \mathsf{C}^{\odot \ell}$. If any check fails, he aborts.
7. We sacrifice some of the columns in $\mathbf{A}$ to protect $\mathcal{P}_i$'s privacy: Note that each column j in $\mathbf{AH}$ is a linear combination of some of the columns in $\mathbf{A}$, we let $\mathbf{A}(j)$ denote the index set for these columns. Now for each j the parties choose an index $a(j) \in \mathbf{A}(j)$ such that all $a(j)$'s are distinct. $\mathcal{P}_i$ and $\mathcal{P}_j$ now discard all columns in $\mathbf{A}, \mathbf{A_0}, \mathbf{A_1}$ and $\mathbf{B}$ indexed by some $a(j)$. For simplicity in the following, we renumber the remaining columns from 1.
8. $\mathcal{P}_i$ saves $\mathbf{A}, \mathbf{A_0}$ and $\mathbf{A_1}$, and $\mathcal{P}_j$ saves $\mathbf{B}$ and $\mathbf{\Delta}$ (all of which now have $m-\ell$ columns). $\mathcal{P}_i$ stores the k top rows of each column $\mathbf{A}[\cdot, \imath]$ in $\mathsf{raw}^i[cid_\imath]$ and $\mathcal{P}_j$ sets $\mathsf{raw}^j[cid_\imath] = \top$ and $\mathsf{actual}^j[cid_\imath] = \perp$, for $\imath \in [m-\ell]$.

Fig. 19. Protocol Π_{2HCom} (Commitment Phase)

Protocol Π_{2HCom} (Linear Combination, Opening and Public Verification)

After the Commit phase, the parties proceed as follows:

Input: On input $(\textsc{Input}, sid, \mathcal{P}_i, cid, \boldsymbol{x}_{cid})$, if $\mathsf{raw}[cid] \neq \perp$, $\mathcal{P}_i$ computes $\boldsymbol{w} = \boldsymbol{x}_{cid} - \mathsf{raw}^i[cid]$, sets $\mathsf{actual}^i[cid] = \mathsf{raw}^i[cid]$, sets $\mathsf{raw}^i[cid] = \perp$, and broadcasts $(\textsc{Input}, sid, cid, \boldsymbol{w})$. Upon receiving $(\textsc{Input}, sid, cid, \boldsymbol{w})$ from $\mathcal{P}_i$, $\mathcal{P}_j$ sets $\mathsf{raw}^j[cid] = \perp$ and $\mathsf{actual}^j[cid] = \boldsymbol{w}$.
Rand: On input $(\textsc{Random}, sid, cid)$, if $\mathsf{raw}[cid] \neq \perp$, $\mathcal{P}_i$ sets $\mathsf{actual}^i[cid] = \mathsf{raw}^i[cid]$ and $\mathsf{raw}^i[cid] = \perp$, and broadcasts $(\textsc{Random}, sid, cid)$. Upon receiving $(\textsc{Input}, sid, cid, \boldsymbol{w})$ from $\mathcal{P}_i$, if $\mathsf{raw}^j[cid] = \top$, $\mathcal{P}_j$ sets $\mathsf{raw}^j[cid] = \perp$, $\mathsf{actual}^j[cid] = \mathbf{0}^k$.
Linear Combination:

1. On input $(\textsc{Linear}, sid, \{(cid_\imath, \alpha_{cid_\imath})\}_{\imath \in [m']}, \boldsymbol{\beta}, cid')$ where m' is the current number of columns in $\mathbf{A}, \mathbf{A_0}, \mathbf{A_1}$ and all $\alpha_{cid} \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^k$, if $\mathsf{actual}^i[cid_\imath] = \boldsymbol{x}_{cid_\imath} \neq \perp$ for $\imath \in [m']$ and cid' is unused, $\mathcal{P}_i$ appends column $\mathsf{C}(\boldsymbol{\beta}) + \sum_{\imath \in [\mathsf{m}']} \alpha_{\mathsf{cid}_\imath} \cdot \mathbf{A}[\cdot, \imath]$ to $\mathbf{A}$ where $\mathsf{C}(\boldsymbol{\beta})$ is an encoding of $\boldsymbol{\beta}$ under C, likewise appending to $\mathbf{A_0}$ and $\mathbf{A_1}$ the corresponding linear combination of columns. $\mathcal{P}_i$ broadcasts $(\textsc{Linear}, sid, \{(cid_\imath, \alpha_{cid_\imath})\}_{\imath \in [m']}, \boldsymbol{\beta}, cid')$.
2. Upon receiving $(\textsc{Linear}, sid, \{(cid_\imath, \alpha_{cid_\imath})\}_{\imath \in [m']}, \boldsymbol{\beta}, cid')$ from $\mathcal{P}_i$, if $\mathsf{actual}^j[cid_\imath] = \boldsymbol{x}_{cid_\imath} \neq \perp$ for $\imath \in [m']$ and cid' is unused, $\mathcal{P}_j$ computes $\mathsf{actual}^j[cid'] = \boldsymbol{\beta} + \sum_{\imath \in [m']} \alpha_{cid_\imath} \cdot \mathsf{actual}^j[cid_\imath]$ appends $\mathsf{C}(\boldsymbol{\beta}) + \sum_{\imath \in [\mathsf{m}']} \alpha_{\mathsf{cid}_\imath} \cdot \mathbf{B}[\cdot, \imath]$ to $\mathbf{B}$. Note that this maintains the properties $\mathbf{A} = \mathbf{A_0} + \mathbf{A_1}$, $\mathbf{B} = \boldsymbol{\Delta}\mathbf{A_1} + (\mathbf{I} - \boldsymbol{\Delta})\mathbf{A_0}$, and $\mathbf{A} \in \mathsf{C}^{\odot \mathsf{m}'}$, where m' is the new current number of columns.

Opening Phase:

1. To open the commitment identified by $cid_\imath$, $\mathcal{P}_i$ broadcasts $(sid, \mathbf{A_0}[\cdot, \imath], \mathbf{A_1}[\cdot, \imath])$.
2. $\mathcal{P}_j$ checks that $\mathbf{A_0}[\cdot, \imath] + \mathbf{A_1}[\cdot, \imath] \in \mathsf{C}$ and that for $\jmath \in [n]$, it holds that $\mathbf{B}[\jmath, \imath] = \mathbf{A}_{b_\jmath}[\jmath, \imath]$ (recall that $b_\jmath$ is the $\jmath$'th entry on the diagonal of $\boldsymbol{\Delta}$). If this check fails, $\mathcal{P}_j$ aborts outputting $(sid, \perp)$. Otherwise, $\mathcal{P}_j$ computes $\boldsymbol{x}_{cid}$, the first k entries in $\mathbf{A_0}[\cdot, \imath] + \mathbf{A_1}[\cdot, \imath] + \mathsf{actual}^j[cid] \,\|\, \mathbf{0}^{n-k}$, and outputs $(\textsc{Open}, sid, cid, \boldsymbol{x}_{cid})$.

Initialize Verification: On input $(\textsc{Verification-Start}, sid)$, $\mathcal{P}_j$ sends $(\textsc{Verification-Start}, sid)$ to $\mathcal{F}_{\mathsf{ROT}}$ and broadcasts $(sid, b_1, \ldots, b_n, \mathbf{B})$.
Public Verification: On input $(\textsc{Verify}, sid, cid_\imath, \boldsymbol{x}'_{cid_\imath})$, a party $\mathcal{V}_v \in \mathcal{V}$ outputs $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$ if $(sid, b_1, \ldots, b_n, \mathbf{B})$ has not been broadcast by $\mathcal{P}_j$. Otherwise, $\mathcal{V}_v$ sends $(\textsc{Verify}, sid, b_1, \ldots, b_n, \mathbf{B})$ to $\mathcal{F}_{\mathsf{ROT}}$. Upon receiving $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$ or $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{ROT}}$ for any $i \in [n]$, $\mathcal{V}_v$ outputs the same message. Upon receiving $(\textsc{Verified}, sid, b_1, \ldots, b_n, \mathbf{S}, 0)$ from $\mathcal{F}_{\mathsf{pOT}}$, $\mathcal{V}_v$ outputs $(\textsc{Verification-Fail}, sid, \mathcal{P}_j)$. Otherwise, if a message $(sid, \mathbf{A_0}[\cdot, cid_\imath], \mathbf{A_1}[\cdot, cid_\imath])$ has not been broadcast by $\mathcal{P}_i$, output $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$. Otherwise, $\mathcal{V}_v$ executes the procedures of an honest $\mathcal{P}_j$ using $b_1, \ldots, b_n, \mathbf{S}$ and the messages broadcast throughout protocol execution in order to verify that the commitment identified by $cid_\imath$ was correctly opened to $\boldsymbol{x}'_{cid_\imath}$. If any of the checks performed in the steps of an honest $\mathcal{P}_j$ fail, output $(\textsc{Verification-Fail}, sid, \mathcal{P}_i)$. If all of the checks performed in the steps of an honest $\mathcal{P}_j$ succeed but the opened message is $\boldsymbol{x}_{cid_\imath}$ such that $\boldsymbol{x}'_{cid_\imath} \neq \boldsymbol{x}_{cid_\imath}$, set $f = 0$. Otherwise, if $\boldsymbol{x}'_{cid_\imath} = \boldsymbol{x}_{cid_\imath}$, set $f = 1$. Output $(\textsc{Verified}, sid, cid_\imath, f)$.

Fig. 20. Protocol Π_{2HCom} (Linear Combination, Opening and Public Verification)

Functionality $\mathcal{F}_{\mathsf{HCom}}$

$\mathcal{F}_{\mathsf{HCom}}$ is parameterized by $k \in \mathbb{N}$. $\mathcal{F}_{\mathsf{HCom}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a set of verifiers $\mathcal{V}$ and an adversary $\mathcal{S}$ (who may abort at any time) through the following interfaces:

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

Commit: Upon receiving $(\textsc{Commit}, sid, \mathcal{I})$ 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 $(\textsc{Commit-Recorded}, sid, \mathcal{I})$ to all parties $\mathcal{P}$ and $\mathcal{S}$.

Input: Upon receiving a message $(\textsc{Input}, sid, \mathcal{P}_i, cid, \boldsymbol{y})$ from $\mathcal{P}_i \in \mathcal{P}$ and messages $(\textsc{Input}, sid, \mathcal{P}_i, cid)$ from every party in $\mathcal{P}$ other than $\mathcal{P}_i$, if a message $(\textsc{Commit}, sid, \mathcal{I})$ 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 $(\textsc{Input-Recorded}, sid, \mathcal{P}_i, cid)$ to all parties in $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast $(\textsc{Abort}, sid)$ and halt.

Random: Upon receiving a message $(\textsc{Random}, sid, cid)$ 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 $(\textsc{Random-Recorded}, sid, cid)$ to all parties $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast $(\textsc{Abort}, sid)$ and halt.

Linear Combination: Upon receiving $(\textsc{Linear}, sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid')$ 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 $(\textsc{Linear-Recorded}, sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid')$ to all parties $\mathcal{P}$ and $\mathcal{S}$. Otherwise broadcast $(\textsc{Abort}, sid)$ and halt.

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

Check Opening: Upon receiving $(\textsc{Check-Not-Open}, sid, cid)$ 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 $(\textsc{Open}, sid, cid)$, send $(\textsc{Check-Not-Open}, sid, \{\hat{p}_1, \ldots, \hat{p}_k\})$ to $\mathcal{P}_i$.

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

Public Verification: Upon receiving $(\textsc{Verify}, sid, cid, \boldsymbol{x}'_{cid})$ 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 $(\textsc{Verification-Start}, sid)$, send $(\textsc{Verify-Fail}, sid, \{\mathcal{P}'_1, \ldots, \mathcal{P}'_m\})$ to $\mathcal{V}_j$. Otherwise, if a message $(\textsc{Open}, sid, cid)$ 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 $(\textsc{Verified}, sid, cid, f)$ to $\mathcal{V}_j$.

Fig. 21. Functionality $\mathcal{F}_{\mathsf{HCom}}$ For Homomorphic Multiparty Commitment With Delayed Public Verifiability

have delayed public verification interfaces. Notice that all the secret state kept by the receiving parties consists of random values sent through $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{F}_{\mathsf{CT}}$. Hence, when this state is revealed in the verification initialization phase, the verifying parties can check that all its components were correctly obtained from $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{F}_{\mathsf{CT}}$. Moreover, all the protocol transcript is received by the verifying parties $\mathcal{V}$ through the broadcast mechanism, guaranteeing that no parties can later provide alternative version. Using the secret states of the receiving parties and the protocol transcript obtained through broadcast, the verifying parties can then run the procedures of an honest receiving party in order to verify that a given commitment was opened to a specific message.

4.7 Efficiency

The commitment phase in Π_{HCom} requires n^2 calls to $\mathcal{F}_{\mathsf{2HCom}}$'s commitment phase and then n^2 commitments to $\gamma + \kappa$ arbitrary messages through $\mathcal{F}_{\mathsf{2HCom}}$, where n is the number of parties. Each call to $\mathcal{F}_{\mathsf{2HCom}}$ phase amounts to n' calls $\mathcal{F}_{\mathsf{pOT}}$, where s is the security parameter and the underlying code is $\mathsf{C}[n', k, s]$. This amounts to a concrete communication complexity of roughly $(6n'|\mathbb{B}| + (\gamma + 5s)n' + (\gamma + k) * s)n^2$ considering protocols Π_{pOT} and Π_{2HCom} for realizing functionalities $\mathcal{F}_{\mathsf{pOT}}$ and $\mathcal{F}_{\mathsf{2HCom}}$, respectively. the cost of the underlying commitments when realized by Π_{Com} is small since it employs random oracles to achieve commitments of constant communication complexity. Notice that the communication cost of the commitment phase can be amortized over many messages, but it is still prohibitive given that our protocols need to store the messages on a public ledger for verification. In order to solve this issue, we can define a compact representation of the messages in the commitment phase of Π_{HCom} by observing that all messages sent in this phase are random. Hence, instead of having all parties post their messages on the public bulletin board we instead have them commit uniformly random seeds with Π_{Com}. Since Π_{Com} generates compact commitments, the total size of this initial commitment to seeds will be simply the output size of the underlying random oracle times the number of parties. The parties then stretch these seeds using a PRG to generate the public coins of the protocols. Moreover, each party commits to the messages generate from private coins that it would post to the public ledger using Π_{Com}, posts only the compact commitment to the public ledger and sends the messages directly to the other parties. Upon receiving the messages, each party checks that they correspond to the commitments posted in the public ledger, aborting otherwise. Later on, the parties can open the commitments in order to allow for public verification.

5 MPC with Publicly Verifiable Output

In this section we provide an implementation of $\mathcal{F}_{\mathsf{Ident}}$, the MPC scheme with a publicly verifiable output as defined in Section 3. We construct it from a functionality $\mathcal{F}_{\mathsf{MPC-SO}}$ that captures MPC with secret-shared output and that supports linear operations on the secret sharing. We describe this functionality in Figure 24, which uses the XOR function over $\mathbb{F}^m$ as the reconstruction function,

Protocol Π_{HCom} (Commitments)

Parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and verifiers $\mathcal{V}$ interact with each other and $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$, proceeding as follows:

Init: On input $(\textsc{Init}, sid)$, each pair of parties $\mathcal{P}_i$ and $\mathcal{P}_j$ invoke the command $(\textsc{Init}, sid)$ of functionality $\mathcal{F}_{\mathsf{2HCom}}$ to initialize an instance denoted by $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$.

Commit: On input $(\textsc{Commit}, sid, \mathcal{I})$ where $\mathcal{I} = \{cid_1, \ldots, cid_\gamma\}$ parties $\mathcal{P}$ proceed as follows:

1. All parties $\mathcal{P}$ agree on a set of $\gamma + \kappa$ unused identifiers $\mathcal{I}'$.
2. For all $j \neq i$, $\mathcal{P}_i$ sends $(\textsc{Commit}, sid, \mathcal{I}')$ to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$, receiving $(\textsc{Commit-Recorded}, sid, \mathcal{I}', \{(cid, \boldsymbol{x}_{cid})\}_{cid \in \mathcal{I}'})$ in response and proceeding after receiving $(\textsc{Commit-Recorded}, sid, \mathcal{I}')$ from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$ for every $j \neq i$.
3. For all $cid \in \mathcal{I}'$ and every $j \in [n], j \neq i$, party $\mathcal{P}_i$ samples $\boldsymbol{x}^i \xleftarrow{\$} \mathbb{F}^k$, sends $(\textsc{Input}, sid, \mathcal{P}_i, cid, \boldsymbol{x}^i)$ to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$ and waits for $(\textsc{Input-Recorded}, sid, \mathcal{P}_j, cid)$ from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$.
4. All parties $\mathcal{P}$ agree on sets $\mathcal{I}$ and $\mathcal{K}$ such that $|\mathcal{I}| = \gamma$, $|\mathcal{K}| = \kappa$, $\mathcal{I} \cap \mathcal{K} = \emptyset$ and $\mathcal{I} \cup \mathcal{K} = \mathcal{I}'$.
5. All parties $\mathcal{P}$ send $(\textsc{Toss}, sid, \kappa \cdot \gamma, \mathbb{F})$ to $\mathcal{F}_{\mathsf{CT}}$. Parties $\mathcal{P}$ continue to the next step upon receiving $(\textsc{Tossed}, sid, \kappa \cdot \gamma, \mathbf{R})$ where $\mathbf{R} \in \mathbb{F}^{\kappa \times \gamma}$ from $\mathcal{F}_{\mathsf{CT}}$.
6. Identifying each column of $\mathbf{R}$ with a unique $cid \in \mathcal{I}$, for every $q \in \mathcal{K}$, every party $\mathcal{P}_i$ samples a fresh identifier cid'_q and, for every $j \in [n], j \neq i$, sends $(\textsc{Linear}, sid, \{\{(cid, \mathbf{R}[q, cid])\}_{cid \in \mathcal{I}}\}, \mathbf{0}^k, cid'_q)$ to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$, waits for $(\textsc{Linear-Recorded}, sid, \{\{(cid, \mathbf{R}[q, cid])\}_{cid \in \mathcal{I}}\}, \mathbf{0}^k, cid')$ from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$, sends $(\textsc{Open}, sid, cid'_q)$ to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$ and waits for $(\textsc{Open}, sid, cid'_q, \boldsymbol{s}^j_q)$ from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$.
7. For every $q \in \mathcal{K}$, each party $\mathcal{P}_i$ computes $\boldsymbol{c}^i_q = \sum_{j \in [n]} \boldsymbol{s}^j_q$ and sends $(\textsc{Equal}, sid, \mathcal{P}_i, \boldsymbol{c}^i_q)$ to $\mathcal{F}_{\mathsf{EQ}}$. Upon receiving $(\textsc{Abort}, sid)$ or $(\textsc{Not-Equal}, sid, \boldsymbol{c}^1_q, \ldots, \boldsymbol{c}^n_q)$ from $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{P}_i$ aborts. Otherwise $\mathcal{P}_i$ outputs $(\textsc{Committed}, sid, \mathcal{I})$, sets $\mathsf{raw}^i[cid] = \top$ and $\mathsf{actual}^i[cid] = \perp$ for $cid \in \mathcal{I}$.

Input: On input $(\textsc{Input}, sid, cid, \boldsymbol{y})$ for $\mathcal{P}_i$ and input $(\textsc{Input}, sid, \mathcal{P}_j, cid)$ for every $\mathcal{P}_j$ for $j \neq i$, parties $\mathcal{P}$ proceed as follows:

1. For every $j \in [n], j \neq i$, $\mathcal{P}_j$ aborts if $\mathsf{raw}^j[cid] \neq \top$. Otherwise, $\mathcal{P}_j$ sends $(\textsc{Open}, sid, cid)$ to $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$.
2. Upon receiving $(\textsc{Open}, sid, cid, \boldsymbol{x}^j)$ from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$ for every $j \in [n], j \neq i$, $\mathcal{P}_i$ computes $\boldsymbol{x}_{cid} = \sum_{j \in [n]} \boldsymbol{x}^j_{cid}$, $\boldsymbol{w}_{cid} = \boldsymbol{y} - \boldsymbol{x}_{cid}$ and broadcasts $(sid, \mathcal{P}_i, cid, \boldsymbol{w}_{cid})$.
3. Every party $\mathcal{P}_i$ sets $\mathsf{raw}^j[cid] = \perp$ and $\mathsf{actual}^j[cid] = \boldsymbol{w}_{cid}$.

Random: On input $(\textsc{Random}, sid, cid)$, if $\mathsf{raw}^i[cid] = \top$, each party $\mathcal{P}_i$ sets $\mathsf{raw}^i[cid] = \perp$ and $\mathsf{actual}^i[cid] = \mathbf{0}^k$. Otherwise output $(\textsc{Abort}, sid)$ and halt.

Fig. 22. Protocol Π_{HCom} (Commitments)

but can be generalized to other functions easily. We then present a protocol Π_{Ident} which implements $\mathcal{F}_{\mathsf{Ident}}$ in Section 5.1 and a proof that a slightly modified version of the BMR-protocol of Hazay et al. [25] realizes $\mathcal{F}_{\mathsf{MPC-SO}}$ in Section 5.2.

Protocol Π_{HCom} (Linear Combination, Opening and Public Verificationn)

Linear Combination: On input (LINEAR, $sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid'$) where all $\alpha_{cid} \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^k$, if $\mathsf{actual}^i[cid] \neq \perp$ for all $cid \in \mathcal{I}$ and cid' is unused, each party $\mathcal{P}_i \in \mathcal{P}$ computes $\mathsf{actual}^i[cid'] = \boldsymbol{\beta} + \sum_{cid \in \mathcal{I}} \alpha_{cid} \cdot \mathsf{actual}^i[cid]$ and sends (LINEAR, $sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, \boldsymbol{\beta}, cid'$) to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$. Otherwise broadcast (ABORT, sid) and halt.

Open: On input (OPEN, sid, cid), each party $\mathcal{P}_i$ sends (OPEN, sid, cid) to $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$ for $j \in [n], j \neq i$. Upon receiving (OPEN, $sid, cid, \boldsymbol{x}^j$) from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$ for every $j \in [n], j \neq i$, $\mathcal{P}_i$ computes $\boldsymbol{y} = \sum_{j \in [n]} \boldsymbol{x}^j_{cid} + \mathsf{actual}^i[cid]$ and outputs (OPEN, $sid, cid, \boldsymbol{y}$).

Check Opening: On input (CHECK-NOT-OPEN, sid, cid), $j, i \in [n], j \neq i$, each party $\mathcal{P}_i$ adds $\mathcal{P}_j$ to set $\hat{\mathcal{P}}$ if it did not receive (OPEN, $sid, cid, \boldsymbol{x}^j$) or (OPEN, sid, cid) from $\mathcal{F}^{j,i}_{\mathsf{2HCom}}$ and outputs (CHECK-NOT-OPEN, $sid, cid, \{\hat{p}\}_{\hat{p} \in \hat{\mathcal{P}}}$).

Initialize Verification: On input (VERIFICATION-START, sid), each party $\mathcal{P}_i \in \mathcal{P}$ sends (VERIFICATION-START, sid) to $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$. Moreover, each party $\mathcal{P}_i \in \mathcal{P}$ broadcasts $\mathbf{R}$ and $\boldsymbol{s}^j_q$ for $j \neq i$.

Public Verification: On input (VERIFY, $sid, cid, \boldsymbol{x}'_{cid}$), a party $\mathcal{V}_j \in \mathcal{V}$ first uses the public verification interfaces of $\mathcal{F}_{\mathsf{2HCom}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{CT}}$ to check that the Commit phase was successfully completed. If any of these functionalities return (VERIFY-FAIL, $sid, \mathcal{P}'$), for each of these cases $\mathcal{V}_j$ adds $\mathcal{P}'$ to set $\hat{\mathcal{P}}$ and if $\mathcal{F}^{i,j}_{\mathsf{2HCom}}$ returns (VERIFIED, $sid, cid, 0$) upon receiving (VERIFY, $sid, cid, \boldsymbol{x}^i_{cid}$) (where $\boldsymbol{x}^i_{cid}$ is obtained from the opening broadcasts), add $\mathcal{P}_j$ to $\hat{\mathcal{P}}$. If $\hat{\mathcal{P}} \neq \emptyset$, $\mathcal{V}_j$ outputs (VERIFY-FAIL, $sid, \hat{\mathcal{P}}$). Otherwise, return (VERIFIED, $sid, cid, 1$)

Fig. 23. Protocol Π_{HCom} (Linear Combination, Opening and Public Verification)

5.1 Protocol Π_{Ident}

We construct a protocol Π_{Ident} which implements $\mathcal{F}_{\mathsf{Ident}}$ (with the XOR function over $\mathbb{F}^m$ as the reconstruction function) in the $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{CT}}$-hybrid model. In it, the parties obtain shares of the output as well as *advice*, such that the actual output can be obtained by combining all of these values. To make this verifiable, the parties use $\mathcal{F}_{\mathsf{HCom}}$ to make them available in a publicly verifiable way. As parties may cheat during the commitment phase, we check consistency by computing random linear combinations both on the commitments and the shares inside $\mathcal{F}_{\mathsf{MPC-SO}}$ and testing for equality. The protocol is described in Figure 31 and Figure 32. Note that for this application $\mathcal{F}_{\mathsf{CT}}$ must not be publicly verifiable. For $z \in \mathbb{F}$, we use $\hat{\boldsymbol{e}}_z$ to denote the vector in $\mathbb{F}^k$ that is z in all k positions.

An important part of the proof is to show that the commitments that each party $\mathcal{P}_i$ gives are indeed well-formed. To establish this, we will later need the following lemma.

Lemma 1. *Fix values* $r_1, \ldots, r_m, s_1, \ldots, s_\kappa \in \mathbb{F}$ *and* $\overline{\boldsymbol{r}}_1, \ldots, \overline{\boldsymbol{r}}_m, \overline{\boldsymbol{s}}_1, \ldots, \overline{\boldsymbol{s}}_\kappa \in \mathbb{F}^k$. *Then pick* $\alpha_{h,j} \xleftarrow{\$} \mathbb{F}$ *for* $h \in [m], j \in [\kappa]$ *uniformly at random. If for all* $j \in [\kappa]$ *there is a* t_j *such that*

$$t_j = s_j + \sum_{h \in [m]} \alpha_{h,j} \cdot r_h \text{ and } \hat{\boldsymbol{e}}_{t_j} = \overline{\boldsymbol{s}}_j + \sum_{h \in [m]} \alpha_{h,j} \cdot \overline{\boldsymbol{r}}_h \quad ,$$

then $\overline{\boldsymbol{r}}_h = \hat{\boldsymbol{e}}_{r_h}$ *for all* $h \in [m]$ *and* $\overline{\boldsymbol{s}}_j = \hat{\boldsymbol{e}}_{s_j}$ *for all* $j \in [\kappa]$*, except with probability* $O(2^{-\kappa})$*.*

Proof. For the sake of argument, assume that the conclusion is false. Then there are three mutually distinct cases:

1. There exists $h \in [m], \ell \in [k-1]$ such that $\overline{\boldsymbol{r}}_h[\ell] \neq \overline{\boldsymbol{r}}_h[\ell+1]$.
2. There exists $h \in [m]$ such that $\overline{\boldsymbol{r}}_h = \hat{\boldsymbol{e}}_z$ for $z = 1 - r_h$.
3. For all $h \in [m]$ it holds that $\overline{\boldsymbol{r}}_h = \hat{\boldsymbol{e}}_{r_h}$ but there exists $j \in [\kappa]$ such that $\overline{\boldsymbol{s}}_j \neq \hat{\boldsymbol{e}}_{s_j}$.

It is easy to see that the third case is impossible, so we will only consider the first two.

In the first case, w.l.o.g. let $h = \ell = 1$, then $\overline{\boldsymbol{r}}_1[1] + \overline{\boldsymbol{r}}_1[2] = 1$. By assumption,

$$\overline{\boldsymbol{s}}_1[1] + \sum_{h \in [m]} \alpha_{h,1} \cdot \overline{\boldsymbol{r}}_h[1] = \overline{\boldsymbol{s}}_1[2] + \sum_{h \in [m]} \alpha_{h,1} \cdot \overline{\boldsymbol{r}}_h[2]$$

and therefore

$$\alpha_{1,1} + \sum_{h \in [m] \setminus \{1\}} \alpha_{h,1} \cdot (\overline{\boldsymbol{r}}_h[1] + \overline{\boldsymbol{r}}_h[2]) = \overline{\boldsymbol{s}}_1[1] + \overline{\boldsymbol{s}}_1[2].$$

Assume that for $h \in [m] \setminus \{1\}, j \in [\kappa]$ the values $\alpha_{h,j}$ would be fixed ahead of the above experiment. Then $\sum_{h \in [m] \setminus \{1\}} \alpha_{h,j} \cdot (\overline{\boldsymbol{r}}_h[1] + \overline{\boldsymbol{r}}_h[2]) + \overline{\boldsymbol{s}}_j[1] + \overline{\boldsymbol{s}}_j[2]$ uniquely predetermines the κ uniformly random values $\alpha_{1,j}$. This holds with probability at most $2^{-\kappa}$ and choosing $\alpha_{h,j}$ for $h \in [m] \setminus \{1\}, j \in [\kappa]$ randomly *after* $\overline{\boldsymbol{r}}_\cdot, \overline{\boldsymbol{s}}_\cdot$ are fixed does not increase the chance of winning the above game.

In the second case, this immediately implies that also $\overline{\boldsymbol{s}}_j \in \{\hat{\boldsymbol{e}}_0, \hat{\boldsymbol{e}}_1\}$. By letting $\hat{r}_h, \hat{s}_j \in \mathbb{F}$ such that $\hat{\boldsymbol{e}}_{\hat{r}_h} = \overline{\boldsymbol{r}}_h$ and $\hat{\boldsymbol{e}}_{\hat{s}_j} = \overline{\boldsymbol{s}}_j$, we then have that

$$\hat{s}_j + \sum\nolimits_{h \in [m]} \alpha_{h,j} \cdot \hat{r}_h = s_j + \sum\nolimits_{h \in [m]} \alpha_{h,j} \cdot r_h.$$

Now there must exist a $h \in [m]$ such that $r_h \neq \hat{r}_h$. By the same argument as in case one, this boils down to predicting all $\alpha_{h,j}$ which is true with probability at most $2^{-\kappa}$. □

Using the above lemma, we now prove security of Π_{Ident}.

Theorem 9. *Protocol* Π_{Ident} *UC-securely implements* $\mathcal{F}_{\mathsf{Ident}}$ *(with the XOR function over* $\mathbb{F}^m$ *as the reconstruction function) against a static malicious adversary corrupting up to* $n-1$ *parties in the* $\mathcal{F}_{\mathsf{MPC-SO}}, \mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{CT}}$*-hybrid model with broadcast.*

We first define a simulator $\mathcal{S}$ which will simulate $\mathcal{F}_{\mathsf{CT}}, \mathcal{F}_{\mathsf{HCom}}$ globally and $\mathcal{F}_{\mathsf{MPC-SO}}$ locally (meaning the former two functionalities can be global functionalities) and which itself simulates an execution of protocol Π_{Ident} with $\mathcal{A}$. We then argue why no environment $\mathcal{Z}$ using $\mathcal{A}$ can distinguish the distribution generated by Π_{Ident} and $\mathcal{A}$ from $\mathcal{S}$ which uses $\mathcal{F}_{\mathsf{Ident}}$.

Functionality $\mathcal{F}_{\mathsf{MPC-SO}}$

This functionality interacts with the parties $\mathcal{P}$. It is parametrized by a circuit C with inputs $x^{(1)}, \dots, x^{(n)}$ and output $\boldsymbol{y} = (y_1, \dots, y_m) \in \mathbb{F}^m$. $\mathcal{S}$ provides a set $I \subset [n]$ of parties which it corrupts. Let the reconstruction function f be the XOR function over $\mathbb{F}$. $\mathcal{S}$ can at any point send $(\textsc{Abort}, sid)$ to the functionality, upon which it sends $(\textsc{Abort}, sid, \perp)$ to all parties and terminates.

Input: Upon 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 $(sid, i, x^{(i)})$ internally. Every further such message with the same sid and i is ignored.

Evaluate: Upon input $(\textsc{Compute}, sid)$ 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 $\boldsymbol{y} = (y_1, \dots, y_m) \leftarrow C(x^{(1)}, \dots, x^{(n)})$ and store $(sid, \boldsymbol{y})$ locally.

Share Output: Upon input $(\textsc{Share-Output}, sid)$ and if **Evaluate** was finished:

1. For each $h \in [m]$, pick an unused cid_h and send $(\textsc{Request-Shares}, sid, \{cid_h\}_{h \in [m]})$ to $\mathcal{S}$. For each $i \in I$ $\mathcal{S}$ sends $(\textsc{Output-Shares}, sid, \{(cid_h, s^{(i)}_{cid_h})\}_{h \in [m]})$. 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 $(\textsc{Output-Shares}, sid, \{(cid_h, s^{(i)}_{cid_h})\}_{h \in [m]})$ to $\mathcal{P}_i$.
2. For each $h \in [m]$, sample $\overline{z_{cid_h}} \in \mathbb{F}$ such that $f(\overline{z_{cid_h}}, s^{(1)}_{cid_h}, \dots, s^{(n)}_{cid_h}) = y_h$ and store $(sid, cid_h, \overline{z_{cid_h}})$. Send $(\textsc{Share-Advices}, sid, \{(cid_h, \overline{z_{cid_h}})\}_{h \in [m]})$ to $\mathcal{S}$. If $\mathcal{S}$ sends $(\textsc{Deliver-Advices}, sid, \{cid_h\}_{h \in [m]})$, then send $(\textsc{Share-Advices}, sid, \{(cid_h, \overline{z_{cid_h}})\}_{h \in [m]})$ to all $\mathcal{P}_i \in \overline{I}$.

Share Random Value: Upon input $(\textsc{Share-Random}, sid)$, pick $z \xleftarrow{\$} \mathbb{F}$ and an unused cid, set $\overline{z_{cid}} = 0$ and send $(\textsc{Request-Shares}, sid, cid)$ to $\mathcal{S}$. For each $i \in I$ $\mathcal{S}$ sends $(\textsc{Share}, sid, cid, s^{(i)}_{cid})$. Afterwards sample $s^{(i)}_{cid} \xleftarrow{\$} \mathbb{F}$ for $i \in \overline{I}$ subject to the condition that $f(\overline{z_{cid}}, s^{(1)}_{cid}, \dots, s^{(n)}_{cid}) = z$, store $(sid, cid, i, s^{(i)}_{cid})$ and send $(\textsc{Share}, sid, cid, s^{(i)}_{cid})$ to $\mathcal{P}_i$.

Linear Combination: Upon input $(\textsc{Linear}, sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, cid')$ 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 $(\textsc{Linear-Recorded}, sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, cid')$ to all parties $\mathcal{P}$ and $\mathcal{S}$.

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

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

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

Fig. 24. Functionality $\mathcal{F}_{\mathsf{MPC-SO}}$ for an MPC with Secret-Shared Output and Linear Secret Share Operations.

Proof. The simulator $\mathcal{S}$ proceeds as follows in the different phases of the protocol:

Protocol Π_{Ident}

The parties evaluate the circuit C with inputs $x^{(1)}, \ldots, x^{(n)}$ and m outputs $y_1, \ldots, y_m$. For the commitment functionality $\mathcal{F}_{\mathsf{HCom}}$ we assume that $k \geq \max\{\kappa, m\}$. Let $\hat{\boldsymbol{e}}_z \in \{0,1\}^k$ be the vector that is z in all k positions. The reconstruction function f associated with $\mathcal{F}_{\mathsf{MPC-SO}}$ is the XOR function over $\mathbb{F}$ and the one obtained by the protocol is the XOR function over $\mathbb{F}^m$.

Init: The parties set up the functionality $\mathcal{F}_{\mathsf{HCom}}$ by sending $(\textsc{Init}, sid)$.

Input: Each $\mathcal{P}_i$ sends $(\textsc{Input}, sid, i, x^{(i)})$ to $\mathcal{F}_{\mathsf{MPC-SO}}$.

Evaluate: Each $\mathcal{P}_i$ sends $(\textsc{Compute}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$.

Share: The parties generate a random blinding of the output and commitments:

1. Each $\mathcal{P}_i$ sends $(\textsc{Share-Output}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ and waits to get the responses $(\textsc{Output-Shares}, sid, \{(cid_h, s^{(i)}_{cid_h})\}_{h \in [m]})$ and $(\textsc{Share-Advices}, sid, \{(cid_h, \overline{z_{cid_h}})\}_{h \in [m]})$.
2. The parties send $n(m+\kappa)$ messages $(\textsc{Share-Random}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ to get shares of random values. We order the secret-shared values such that $(m+\kappa)$ distinct values are associated with each party $\mathcal{P}_i$. Let $cid^{(i)}_{r,h}$ for $h \in [m]$ and $cid^{(i)}_{s,j}$ for $j \in [\kappa]$ denote the respective identifiers. Let $\mathcal{I}$ be the set of all cid obtained in this step. Each $\mathcal{P}_i$ sends $(\textsc{Commit}, sid, \mathcal{I})$ to $\mathcal{F}_{\mathsf{HCom}}$.
3. For $i \in [n]$, each party $\mathcal{P}_\ell$ sends messages $(\textsc{Reveal}, sid, \cdot, \ell, i)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ for all $cid^{(i)}_{r,h}$, $h \in [m]$ and all $cid^{(i)}_{s,j}$, $j \in [\kappa]$ to open the shares towards $\mathcal{P}_i$. $\mathcal{P}_i$ uses the reconstruction function f to get the secret-shared values. Let $r^{(i)}_h$ for $h \in [m]$ and $s^{(i)}_j$ for $j \in [\kappa]$ denote the respective secret-shared values.
4. For $h \in [m]$ each party $\mathcal{P}_i$ sends $(\textsc{Input}, sid, \mathcal{P}_i, cid^{(i)}_{r,h}, \hat{\boldsymbol{e}}_{r^{(i)}_h})$ to $\mathcal{F}_{\mathsf{HCom}}$. Moreover, each $\mathcal{P}_i$ for $j \in [\kappa]$ sends $(\textsc{Input}, sid, \mathcal{P}_i, cid^{(i)}_{s,j}, \hat{\boldsymbol{e}}_{s^{(i)}_j})$ to $\mathcal{F}_{\mathsf{HCom}}$.
5. Each $\mathcal{P}_i$ sends $(\textsc{Toss}, sid, m \cdot \kappa, \mathbb{F})$ to $\mathcal{F}_{\mathsf{CT}}$. They obtain bits $\{\alpha_{h,j}\}_{h \in [m], j \in [\kappa]}$.
6. For $i \in [n]$, $j \in [\kappa]$ set $\mathtt{lin}_{i,j} \leftarrow \{cid^{(i)}_{r,h}, \alpha_{h,j}\}_{h \in [m]} \cup \{cid^{(i)}_{s,j}, 1\}$. Each party sends $(\textsc{Linear}, sid, \mathtt{lin}_{i,j}, \hat{\boldsymbol{e}}_0, cid^{(i)}_{b,j})$ to $\mathcal{F}_{\mathsf{HCom}}$ and $(\textsc{Linear}, sid, \mathtt{lin}_{i,j}, cid^{(i)}_{b,j})$ to $\mathcal{F}_{\mathsf{MPC-SO}}$.
7. For $i \in [n], j \in [\kappa]$ each party $\mathcal{P}_\ell$ (a) sends $(\textsc{Open}, sid, cid^{(i)}_{b,j})$ to $\mathcal{F}_{\mathsf{HCom}}$, which outputs $\boldsymbol{o}^{(i)}_j$. If $\boldsymbol{o}^{(i)}_j = \hat{\boldsymbol{e}}_z$ for some $z \in \mathbb{F}$ then set $\overline{out}^{(i)}_j = z$, otherwise abort; and (b) sends $(\textsc{Reveal}, sid, cid^{(i)}_{b,j}, \ell)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ and after getting the shares of all parties reconstruct the value using the reconstruction function f and denote the reconstructed element as $out^{(i)}_j$.
8. If for any $i \in [n], j \in [\kappa]$ it holds that $out^{(i)}_j \neq \overline{out}^{(i)}_j$ then abort.
9. For each $h \in [m]$, set $\mathtt{lin}_h \leftarrow \{cid^{(i)}_{r,h}, -1\}_{i \in [n]} \cup \{cid_h, 1\}$. Each party sends $(\textsc{Linear}, sid, \mathtt{lin}_h, cid_{\overline{y}_h})$ to $\mathcal{F}_{\mathsf{MPC-SO}}$. Then each party $\mathcal{P}_i$ sends $(\textsc{Reveal}, sid, cid_{\overline{y}_h}, i)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ and after receiving all shares uses the reconstruction function f to obtain $\overline{y}_h$. $\mathcal{P}_i$ sets its share of the output as $\boldsymbol{r}^{(i)} \leftarrow (r^{(i)}_1, \ldots, r^{(i)}_m)$ and the advice as $\overline{\boldsymbol{y}} \leftarrow (\overline{y}_1, \ldots, \overline{y}_m)$.

Fig. 25. Protocol Π_{Ident} Implementing $\mathcal{F}_{\mathsf{Ident}}$.

Protocol Π_{Ident} (continuation)

Reveal: Combine the commitments and open them unreliably. Each party $\mathcal{P}_i$ for each $j \in [n], h \in [m]$ sends $(\textsc{Open}, sid, cid_{r,h}^{(j)})$ to $\mathcal{F}_{\mathsf{HCom}}$. Each $\mathcal{P}_i$ eventually learns $\hat{\boldsymbol{e}}_{r_h^{(i)}}$ and reconstructs $r_h^{(i)}$ using the first element of the vector.

Test Reveal: Run $\mathsf{Reveal}()$ and return its output.

Allow Verify: Each party $\mathcal{P}_i$ sends $(\textsc{Verification-Start}, sid)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Verify: Party $\mathcal{V}_i \in \mathcal{V}$ with input $(\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(n)}), \boldsymbol{z}^{(i)} \in \mathbb{F}^m$ does the following:

1. For $j \in [n], h \in [m]$ send $(\textsc{Verify}, sid, cid_{r,h}^{(j)}, \hat{\boldsymbol{e}}_{\boldsymbol{z}^{(j)}[h]} \in \mathbb{F}^k)$ to $\mathcal{F}_{\mathsf{HCom}}$.
2. If $\mathcal{F}_{\mathsf{HCom}}$ returns $(\textsc{Verify-Fail}, sid, J)$ then return $(\textsc{Verify-Fail}, sid, J)$. Otherwise, for each $j \in [n], h \in [m]$ $\mathcal{F}_{\mathsf{HCom}}$ it returns $(\textsc{Verified}, sid, cid_{r,h}^{(j)}, f_h^{(j)})$.
3. Let $(J^{(1)}, \ldots, J^{(n)}) \leftarrow \mathsf{Reveal}()$. If $\emptyset \neq \bigcup_{i \in [n]} J^{(i)}$ then return $(\textsc{Reveal-Fail}, sid, J^{(1)}, \ldots, J^{(n)})$. Else, return $(\textsc{Open-Fail}, sid, \{i \in [n] \mid \exists h \in [m] : \ f_h^{(i)} = 0\})$.

Procedure Reveal :

1. For each $i \in [n], h \in [m]$ send $(\textsc{Check-Not-Open}, sid, cid_{r,h}^{(i)})$ to $\mathcal{F}_{\mathsf{HCom}}$ and obtain $(\textsc{Check-Not-Open}, sid, J_h^{(i)})$. Set $J^{(i)} = \bigcup_{h \in [m]} J_h^{(i)}$.
2. Return $(J^{(1)}, \ldots, J^{(n)})$.

Fig. 26. Protocol Π_{Ident} Implementing $\mathcal{F}_{\mathsf{Ident}}$ (continued).

Init: Set up $\mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{CT}}$ for the simulation. Initialize $\mathcal{F}_{\mathsf{MPC-SO}}$ with the set I of corrupted parties.

Input: $\mathcal{S}$ simulates the execution of the input phase of Π_{Ident} with $\mathcal{A}$ and forwards the messages that $\mathcal{A}$ sends to the simulated $\mathcal{F}_{\mathsf{MPC-SO}}$ to $\mathcal{F}_{\mathsf{Ident}}$.

Evaluate: $\mathcal{S}$ simulates the execution of the evaluate phase of Π_{Ident} with $\mathcal{A}$ and forwards the messages that $\mathcal{A}$ sends to the simulated $\mathcal{F}_{\mathsf{MPC-SO}}$ to $\mathcal{F}_{\mathsf{Ident}}$.

Share: Obtain the $\boldsymbol{r}^{(i)}$-shares of dishonest parties from $\mathcal{F}_{\mathsf{Ident}}$ and simulate the protocol to get these and $\overline{y}_h$ right.

1. Start by simulating $\mathcal{F}_{\mathsf{MPC-SO}}$ to generate the random shares $r_h^{(i)}, s_j^{(i)}$ honestly. If all parties obtain their shares, send $(\textsc{Share}, sid)$ in the name of all dishonest parties to $\mathcal{F}_{\mathsf{Ident}}$.
2. Upon obtaining $\boldsymbol{r}^{(i)} \in \mathbb{F}^m$ for $\mathcal{P}_i \in I$ from $\mathcal{F}_{\mathsf{Ident}}$, fix an honest party and change its share of $r_h^{(i)}$ for $h \in [m]$ such that $r_h^{(i)} = \boldsymbol{r}^{(i)}[h]$.
3. Run the opening of the values $r_h^{(i)}, s_j^{(i)}$ honestly with the adjusted share of one honest party. For any $\mathcal{P}_i \in \overline{I}$, if $\mathcal{A}$ does not reveal the necessary values using $\mathcal{F}_{\mathsf{MPC-SO}}$, then send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$, otherwise send $(\textsc{Deliver-Share}, sid, i)$ to $\mathcal{F}_{\mathsf{Ident}}$.
4. For the simulated honest parties $\mathcal{P}_i$ use $\mathcal{F}_{\mathsf{HCom}}$ to commit to $\hat{\boldsymbol{e}}_{r_h^{(i)}}, \hat{\boldsymbol{e}}_{s_j^{(i)}}$ as in the protocol using $\mathcal{F}_{\mathsf{HCom}}$. If $\mathcal{A}$ commits to a value that is inconsistent with the values obtained from $\mathcal{F}_{\mathsf{MPC-SO}}$ then set $\mathtt{abort} \leftarrow \top$.
5. Run steps $5-8$ honestly, but if $\mathtt{abort} = \top$, then abort in step 8.

6. Obtain $(\textsc{Output}, sid, \overline{\boldsymbol{y}})$ with $\overline{\boldsymbol{y}} = (\overline{y}_1, \ldots, \overline{y}_m)$ from $\mathcal{F}_{\mathsf{Ident}}$. For each $\overline{y}_h$ adjust one of the shares of a simulated honest party according to the value to be revealed and simulate the opening using $\mathcal{F}_{\mathsf{MPC-SO}}$. If $\mathcal{A}$ do not reveal the necessary values using $\mathcal{F}_{\mathsf{MPC-SO}}$, then send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$, otherwise send $(\textsc{Deliver-Output}, sid, \overline{\boldsymbol{y}})$ to $\mathcal{F}_{\mathsf{Ident}}$.

Reveal: Simulate correct opening of the shares to be consistent. Obtain $(\textsc{Reveal}, sid, i, \boldsymbol{r}^{(i)})$ from $\mathcal{F}_{\mathsf{Ident}}$ and then:

1. If $i \in \overline{I}$ then $\mathcal{S}$ equivocates in $\mathcal{F}_{\mathsf{HCom}}$ for all $h \in [m]$ the values associated with $cid_{r,h}^{(i)}$ so that the open to the correct values and keep this consistent with **Verify**.
2. Let $J^{(i)}$ be the set of parties that did not send $(\textsc{Open}, sid, cid_{r,h}^{(i)})$ to $\mathcal{F}_{\mathsf{HCom}}$ for some $h \in [m]$. If $J^{(i)} = \emptyset$ then send $(\textsc{Reveal-Ok}, sid, i)$ to $\mathcal{F}_{\mathsf{Ident}}$, else send $(\textsc{Reveal-Not-Ok}, sid, i, J^{(i)})$.

Test Reveal: Send $(\textsc{Test-Reveal}, sid)$ to $\mathcal{F}_{\mathsf{Ident}}$ and output what it outputs.

Allow Verify: For each dishonest $\mathcal{P}_i \in I$ that sends $(\textsc{Verification-Start}, sid, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{HCom}}$ send $(\textsc{Start-Verify}, sid, i)$ to $\mathcal{F}_{\mathsf{Ident}}$. For each simulated honest party, send $(\textsc{Verification-Start}, sid, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Verify: Do the same as in the protocol.

We now argue why each individual part of the protocol simulation is indistinguishable.

Init, Input, Evaluate: Trivially a perfect simulation.

Share: The adversary obtains output from $\mathcal{F}_{\mathsf{Ident}}$ instead of $\mathcal{F}_{\mathsf{MPC-SO}}$, but the values are equally distributed. There are special cases in which $\mathcal{S}$ aborts where it differs from the protocol, but observe that this is a superset of those cases in which the protocol would abort. We first show that the difference in the abort probability is negligible. The protocol aborts in case that $\mathcal{A}$ commits towards $\mathcal{F}_{\mathsf{HCom}}$ to a value which differs from the value it should commit to according to Π_{Ident} (i.e. if $\texttt{abort} = \top$). By Lemma 1 we observe that in this case the protocol will only continue after passing step 8 with probability at most $O(2^{-\kappa})$. Concerning the values which $\mathcal{A}$ obtains during the protocol, $\hat{\boldsymbol{y}}$ is the same as $\overline{\boldsymbol{y}}$ that is also provided by $\mathcal{F}_{\mathsf{Ident}}$ to the real honest parties in the protocol. Furthermore, the shares $\boldsymbol{r}^{(i)}$ which $\mathcal{A}$ obtains are consistent with those from $\mathcal{F}_{\mathsf{Ident}}$. The values $out^{(i)}, \overline{out}^{(i)}$ which $\mathcal{A}$ obtains during the simulation are identical, as the simulator otherwise aborted before. Each such $out^{(i)}$ contains a linear combination of secret values $r_h^{(i)}$, XOR-ed with a uniformly-random but secret $s_j^{(i)}$ and therefore leaks no information about $r_h^{(i)}$.

Reveal: The values $\boldsymbol{r}^{(i)}$ for $i \in \overline{I}$ are consistent with those of $\mathcal{F}_{\mathsf{Ident}}$ in any further interaction. They differ from what the simulated parties committed originally but each $\boldsymbol{r}^{(i)}$ is equally likely, as any previously opened value that was derived from $\boldsymbol{r}^{(i)}$ was blinded by a uniformly random $s_j^{(i)}$.

Test Reveal: The sets that are provided by $\mathcal{F}_{\mathsf{Ident}}$ are identical with those of $\mathcal{F}_{\mathsf{HCom}}$ by construction.

Allow Verify: There is no output that $\mathcal{A}$ obtains in this step.

Verify: Due to **Allow Verify**, the parties that activated verification are identical in both $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{HCom}}$ and $(\textsc{Verify-Fail}, sid)$ is sent with the same content by both. The same holds for the parties that aborted openings as this information is provided to $\mathcal{F}_{\mathsf{Ident}}$ during the simulation of **Reveal**, so also $(\textsc{Reveal-Fail}, sid)$-messages coincide. Moreover, the shares of the honest parties from $\mathcal{F}_{\mathsf{Ident}}$ have been programmed into $\mathcal{F}_{\mathsf{HCom}}$, thus also $(\textsc{Open-Fail}, sid)$-messages are consistent. Therefore, the output of $\mathcal{F}_{\mathsf{Ident}}, \mathcal{F}_{\mathsf{HCom}}$ is identical.

□

5.2 Instantiating $\mathcal{F}_{\mathsf{MPC-SO}}$

We now show that a slightly modified version of the BMR-protocol due to Hazay et al. [25] realizes $\mathcal{F}_{\mathsf{MPC-SO}}$ in the $\mathcal{F}_{\mathsf{Offline}}$-hybrid model.

The MPC protocol evaluates a circuit C over $\mathbb{F}$ on inputs $x^{(1)}, \ldots, x^{(n)} \in \mathbb{F}$ as a preprocessing protocol which consists of three phases: (i) a constant-round circuit-independent offline phase which depends on $|C|, \tau, \kappa$, (ii) a constant-round circuit-dependent offline phase which depends on C and the previous phase; and (iii) a constant-round online phase which depends on $x^{(1)}, \ldots, x^{(n)}$ and the previous phases. The first part of our protocol is identical with that of HSS, who run a multiparty version of the TinyOT [37, 35, 12] MPC scheme (see below). This TinyOT protocol is then used to generate a garbled circuit in a distributed way, while the online phase evaluates this garbled circuit on the actual inputs. In the following, we will describe the structure of this garbling that is generated in the circuit-dependent preprocessing as well as some necessary information about computations with the TinyOT MPC scheme. Using this, we will describe the online phase of our protocol. The security of the circuit-dependent preprocessing can be found in Appendix 5.2.

Representations A value $x \in \mathbb{F}$ is called additively shared if each party $\mathcal{P}_i$ has a value $x^{(i)}$ such that $x = \sum_i x^{(i)}$. Each party $\mathcal{P}_i$ has a private secret $\boldsymbol{\Delta}^{(i)} \in \mathbb{F}^{\tau}$. We define the $[\cdot]$-representation of x as

$$[x] = \left(x^{(i)}, \{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j \in [n] \setminus \{i\}} \right)_{i \in [n]}$$

where $\boldsymbol{\chi}_j^{(i)} = \boldsymbol{\psi}_i^{(j)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$. In the $[x]$-representation the party $\mathcal{P}_i$ holds $x^{(i)}$ together with the $n-1$ MACs $\boldsymbol{\chi}_j^{(i)}$ as well as $n-1$ keys $\boldsymbol{\psi}_j^{(i)}$ protecting the share $x^{(j)}$ of each other party $\mathcal{P}_j$ using $\mathcal{P}_i$'s secret key $\boldsymbol{\Delta}^{(i)}$. It is easy to see that this representation is linear: given

$$[x] = (x^{(i)}, \{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}), \quad [y] = (y^{(i)}, \{\hat{\boldsymbol{\chi}}_j^{(i)}, \hat{\boldsymbol{\psi}}_j^{(i)}\}),$$

the sharing $[x+y]$ can be computed without interaction as

$$[x+y] = (x^{(i)} + y^{(i)}, \{\boldsymbol{\chi}_j^{(i)} + \hat{\boldsymbol{\chi}}_j^{(i)}, \boldsymbol{\psi}_j^{(i)} + \hat{\boldsymbol{\psi}}_j^{(i)}\})$$

Similarly, for $[x], c \in \mathbb{F}$ if

$$\mathcal{P}_1 \text{ sets } (x^{(1)} + c, \{\boldsymbol{\chi}_j^{(1)}, \boldsymbol{\psi}_j^{(1)}\}_{j \in [n] \setminus \{1\}})$$

and each $\mathcal{P}_i, i \neq 1$ sets

$$(x^{(i)}, \{(\boldsymbol{\chi}_1^{(i)}, \boldsymbol{\psi}_1^{(i)} + c \cdot \boldsymbol{\Delta}^{(i)})\} \cup \{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j \in [n] \setminus \{1,i\}})$$

then this is a valid sharing of $[x + c]$ and obtained with local operations only. Multiplications of two $[\cdot]$-shared values are also possible (using preprocessed data from TinyOT), but we will only introduce and use the necessary protocol Π_{Mult} in Appendix 5.2. For the online phase, we only need to be able to reliably open $[x]$-representations, i.e. open them such that sending incorrect shares can be detected.

Protocol Π_{Open}

The parties open a sharing $[x]$ publicly.

1. Each party $\mathcal{P}_i$ broadcasts $x^{(i)}$ and sends $\boldsymbol{\chi}_j^{(i)}$ to $\mathcal{P}_j$ for each $i \neq j$.
2. Each party $\mathcal{P}_i$ checks for all $j \neq i$ that $\boldsymbol{\chi}_i^{(j)} = \boldsymbol{\psi}_j^{(i)} + x^{(j)} \cdot \boldsymbol{\Delta}^{(i)}$ and broadcasts $\perp$ otherwise.
3. Each party computes $x \leftarrow \sum_i x^{(i)}$.

Fig. 27. Protocol Π_{Open} To Open A $[\cdot]$-Representation Publicly.

To achieve this, we use the protocols Π_{Open} as described in Figure 27 and Π_{POpen} from Figure 28.

Protocol Π_{POpen}

The parties open a sharing $[x]$ in private to party $\mathcal{P}_j$.

1. Each party $\mathcal{P}_i$ sends $x^{(i)}, \boldsymbol{\chi}_j^{(i)}$ to party p_j.
2. Party $\mathcal{P}_j$ checks if, for all $i \neq j$ it holds that $\boldsymbol{\chi}_j^{(i)} = \boldsymbol{\psi}_i^{(j)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$. Otherwise, it broadcasts $\perp$.
3. $\mathcal{P}_j$ locally computes $x \leftarrow \sum_i x^{(i)}$.

Fig. 28. Protocol Π_{POpen} To Open A $[\cdot]$-Representation Privately.

Multiparty Free-XOR Garbling We assume that the circuit C, which is evaluated by our MPC protocol, consists of n input wires and m output wires as well as a set of gates $\mathbb{G}$. C can be viewed as a directed acyclic graph where the edges are wires and the vertices are the gates. Each gate $g \in \mathbb{G}$ is either an AND-or a XOR-gate and has two input wires u, v as well as one output wire w, which may be input to multiple subsequent gates. Each input wire of a gate is either one of the n input wires of C or an output wire of another gate. Evaluating C in plain is done by assigning $x^{(1)}, \ldots, x^{(n)} \in \mathbb{F}$ to the n input wires and recursively applying the gate function for each gate that has inputs assigned to its input wires. Then, the values that are assigned to the m output wires $y^{(1)}, \ldots, y^{(m)}$ form the output of C when evaluated on this specific input.

To garble C classically with only one garbler, it first permutes the truth-table of the function of each gate, assigns keys $\boldsymbol{k}_{h,a} \in \{0,1\}^\tau$ to each $h \in \{u, v, w\}, a \in \{0,1\}$ according to the wire h and the truth-value a as denoted in the truth-table, and then encrypts for each row of the truth table each output key $\boldsymbol{k}_{w,\cdot}$ (based on the output bit of this row) under the two appropriate input keys $\boldsymbol{k}_{u,\cdot}, \boldsymbol{k}_{v,\cdot}$ [44, 8]. It was shown in [30] that by fixing $\boldsymbol{k}_{h,0} + \boldsymbol{k}_{h,1} = \boldsymbol{\Delta}$ to a constant value for the whole garbled circuit, one only has to garble the AND-gates and can obtain the garbled XOR-gates by linearity.

For n parties with individual global differences $\boldsymbol{\Delta}^{(i)} \in \{0,1\}^\tau$, the garbling for AND-gates in HSS then works as follows: for each AND-gate $g \in \mathbb{G}$, let u, v be the input wires and w be the output wire, $\lambda_u, \lambda_v, \lambda_w \in \{0,1\}$ be secret wire masks (that encrypt the actual value of the truth values of a gate), and $\boldsymbol{k}^{(i)}_{u,a}, \boldsymbol{k}^{(i)}_{v,b}, \boldsymbol{k}^{(i)}_{w,0} \in \{0,1\}^\tau$ be keys known to $\mathcal{P}_i$. The garbling information for a gate g can be computed as the $4n$ values

$$\begin{aligned} \boldsymbol{d}^{(i)}_{a,b}(g) = \Big(\sum\nolimits_{j=1}^{n} F_{\boldsymbol{k}^{(j)}_{u,a}, \boldsymbol{k}^{(j)}_{v,b}}(g \,\|\, i)\Big) + \boldsymbol{k}^{(i)}_{w,0} + \\ \Big(\boldsymbol{\Delta}^{(i)}((\lambda_u + a)(\lambda_v + b) + \lambda_w)\Big), \end{aligned}$$

where $(a,b) \in \{0,1\}^2, i \in [n]$ and F is a double-keyed 2-correlation robust Pseudorandom Function (PRF)[8]. Choosing keys, wire masks as well as computing the values $\boldsymbol{d}^{(i)}_{a,b}(g)$ is done during the circuit-dependent preprocessing phase $\mathcal{F}_{\mathsf{Offline}}$ as depicted in Figure 29 and Figure 30. In Appendix 5.2, we then describe how to implement $\mathcal{F}_{\mathsf{Offline}}$ in the $\mathcal{F}_{\mathsf{TinyOT}}$-hybrid model, as our $\mathcal{F}_{\mathsf{Offline}}$ differs from the version provided in HSS.

Intuition of the Online Phase We now describe how to use the encryptions $\boldsymbol{d}^{(i)}_{a,b}(g)$ from the offline phase, which are known to each party in the protocol, to perform a secure multiparty computation.

For each input $\ell \in [n]$ the input keys $\boldsymbol{k}^{(1)}_{w_\ell, \Lambda_{w_\ell}}, \ldots, \boldsymbol{k}^{(n)}_{w_\ell, \Lambda_{w_\ell}}$ are published by the respective parties, which works as follows: first, party $\mathcal{P}_i$ that holds the input computes the encrypted wire value Λ_{w_ℓ} based on its actual input $x^{(\ell)}$ and the permutation bit λ_{w_ℓ} as $\Lambda_{w_\ell} = \lambda_{w_\ell} + x^{(\ell)}$. Here, λ_{w_ℓ} is fixed for input ℓ and known to $\mathcal{P}_i$ in advance. $\mathcal{P}_i$ then broadcasts Λ_{w_ℓ} to all parties, whereupon

[8] This stronger requirement is necessary to support the garbling-free XOR gates. We do not give a definition for this primitive in this work as we will invoke the security proof of [25] for these details. See [17] for more information on these special PRFs.

Functionality $\mathcal{F}_{\mathsf{Offline}}$ (part 1)

This functionality is used by a set of parties $\mathcal{P}$ and the adversary $\mathcal{S}$ specifies a set $I \subset \mathcal{P}$ of corrupt parties. Let F be a circular 2-correlation robust PRF. The circuits that are generated consist of AND- and XOR-gates.

Init: On input $(\textsc{Init}, sid)$ from all parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and if this message has not been sent before for this sid:

1. Wait for $\mathcal{S}$ to send $\boldsymbol{\Delta}^{(i)}$ for each $\mathcal{P}_i \in I$.
2. Choose strings $\boldsymbol{\Delta}^{(i)} \xleftarrow{\$} \mathbb{F}^\tau$ uniformly at random for each honest party $\mathcal{P}_i \in \overline{I}$.

Garble: On input $(\textsc{Garble}, sid, C)$ from all parties where C is a circuit with the set of wires W and the set of AND-gates $\mathbb{G}$ and if **Init** was run before but **Garble** was not, the functionality does the following:

1. For each wire $w \in W$ in the circuit C we do the following:
   - If w is an input wire of C or the output wire of an AND-gate then sample $\lambda_w \xleftarrow{\$} \mathbb{F}$ uniformly at random. For each $\mathcal{P}_i \in I$ wait for $\boldsymbol{k}_{w,0}^{(i)} \in \mathbb{F}^\tau$ from $\mathcal{S}$, and choose $\boldsymbol{k}_{w,0}^{(i)} \xleftarrow{\$} \mathbb{F}^\tau$ uniformly at random for each honest party $\mathcal{P}_i \in \overline{I}$. Then for each $i \in [n]$ set $\boldsymbol{k}_{w,1}^{(i)} \leftarrow \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}$.
   - If w is the output wire of an XOR-gate, where the input wires u, v are already assigned, then set $\lambda_w \leftarrow \lambda_u + \lambda_v$. Moreover, for $i \in [n]$ set $\boldsymbol{k}_{w,0}^{(i)} \leftarrow \boldsymbol{k}_{u,0}^{(i)} + \boldsymbol{k}_{v,0}^{(i)}$ and $\boldsymbol{k}_{w,1}^{(i)} \leftarrow \boldsymbol{k}_{w,0} + \boldsymbol{\Delta}^{(i)}$.
2. For every AND-gate $g \in \mathbb{G}$ compute the garbled gate as
$$\boldsymbol{d}_{a,b}^{(i)}(g) = \left(\sum_{j=1}^{n} F_{\boldsymbol{k}_{u,a}^{(j)}, \boldsymbol{k}_{v,b}^{(j)}}(g \,\|\, i)\right) + \boldsymbol{k}_{w,0}^{(i)} + \left(\boldsymbol{\Delta}^{(i)}((\lambda_u + a)(\lambda_v + b) + \lambda_w)\right)$$
   for each $a, b \in \{0,1\}$ and $i \in [n]$. Then set $\boldsymbol{d}_{a,b}(g) = (\boldsymbol{d}_{a,b}^{(1)}(g) \,\|\, \cdots \,\|\, \boldsymbol{d}_{a,b}^{(n)}(g))$.
3. For each wire $w \in W$ send $\boldsymbol{k}_{w,0}^{(i)}$ to each honest party $\mathcal{P}_i \in \overline{I}$.
4. For each input wire w_i wait until $\mathcal{S}$ sends $(\textsc{Ok}, sid, w_i)$. Then send λ_{w_i} to $\mathcal{P}_i$.
5. For each output wire $\overline{w}_h$ of the circuit C with permutation bit $\lambda_{\overline{w}_h}$:
   (a) Let $\mathcal{S}$ input $\lambda_{\overline{w}_h}^{(i)}$ for each $i \in I$.
   (b) Sample uniformly random $\lambda_{\overline{w}_h}^{(i)} \xleftarrow{\$} \mathbb{F}$ for each honest $\mathcal{P}_i$ subject to the constraint $\lambda_{\overline{w}_h} = \sum_i \lambda_{\overline{w}_h}^{(i)}$.
   (c) Run $[\lambda_{\overline{w}_h}] \leftarrow \mathsf{Bracket}(\lambda_{\overline{w}_h}^{(1)}, \ldots, \lambda_{\overline{w}_h}^{(n)})$ and output $[\lambda_{\overline{w}_h}]$.

Fig. 29. Functionality $\mathcal{F}_{\mathsf{Offline}}$ For The Preprocessing Of The MPC Protocol.

each party $\mathcal{P}_j$ reacts by broadcasting its key $\boldsymbol{k}_{w_\ell, \Lambda_{w_\ell}}^{(j)}$. Once the input keys and encrypted wire values for each input of the circuit have been provided, these can be used to evaluate the garbled circuit: for each gate g with input wires u, v and respective encrypted wire values a, b as well as known keys $\{\boldsymbol{k}_{u,a}^{(i)}, \boldsymbol{k}_{v,b}^{(i)}\}_{i \in [n]}$

Functionality $\mathcal{F}_{\mathsf{Offline}}$ (part 2)

Open Garbling: On input (OPEN-GARBLING, sid) from all parties, if **Garble** was run successfully and **Open Garbling** was not run before:

1. Send $\boldsymbol{d}_{a,b}(g)$ for all $g \in \mathbb{G}$ to $\mathcal{S}$.
2. If $\mathcal{S}$ sends an additive error $\boldsymbol{e} = \{\boldsymbol{e}_{a,b}(g)\}$ for $a, b \in \{0,1\}, g \in \mathbb{G}$ then output $\tilde{\boldsymbol{d}}_{a,b}(g) = \boldsymbol{e}_{a,b}(g) + \boldsymbol{d}_{a,b}(g)$ to all honest parties, otherwise send $\boldsymbol{d}_{a,b}(g)$.

Generate Random: On input (RANDOM, sid, ℓ) by each $\mathcal{P}_i$ and if **Init** was run before send (RANDOM, sid, ℓ) to $\mathcal{S}$. Upon input $b_j^{(i)}$ for $j \in [\ell], i \in I$ by $\mathcal{S}$ sample $b_j^{(i)} \xleftarrow{\$} \mathbb{F}$ for each $i \in \overline{I}, j \in [\ell]$, compute $[b_j] \leftarrow \mathsf{Bracket}(b_j^{(1)}, \ldots, b_j^{(n)})$ for $j \in [\ell]$ and output $([b_1], \ldots, [b_\ell])$.

Macro $\mathsf{Bracket}$**:** On input $x^{(1)}, \ldots, x^{(n)}$ compute $[x]$ for each $\mathcal{P}_i$

- if $i \in I$ then $\forall j \in [n] \setminus \{i\}$ wait for $\boldsymbol{\chi}_j^{(i)}$ from $\mathcal{S}$, then compute $\boldsymbol{\psi}_i^{(j)} \leftarrow \boldsymbol{\chi}_j^{(i)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$
- if $i \in \overline{I}$ then $\forall j \in I$ wait for $\boldsymbol{\psi}_i^{(j)}$ from $\mathcal{S}$ and choose $\boldsymbol{\psi}_i^{(j)}$ honestly for all $j \in \overline{I} \setminus \{i\}$. Then compute $\boldsymbol{\chi}_j^{(i)} \leftarrow \boldsymbol{\psi}_i^{(j)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$.

Output $(x^{(i)}, \{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j \in [n] \setminus \{i\}})$ to each $\mathcal{P}_i$.

Key Queries: Upon receiving $(i, \boldsymbol{\Delta})$ for $i \in [n]$ from the adversary and if **Init** was run before, return 1 if $\boldsymbol{\Delta} = \boldsymbol{\Delta}^{(i)}$ and 0 otherwise.

Fig. 30. Functionality $\mathcal{F}_{\mathsf{Offline}}$ For The Preprocessing Of The MPC Protocol (continued).

each party locally then computes the encrypted wire value c as well as the keys $\{\boldsymbol{k}_{w,c}^{(i)}\}_{i \in [n]}$ for the output wire w as follows:

- If g is an XOR gate then set $c \leftarrow a + b$ and $\boldsymbol{k}_{w,c}^{(i)} \leftarrow \boldsymbol{k}_{u,a}^{(i)} + \boldsymbol{k}_{v,b}^{(i)}$ for all $i \in [n]$.
- If g is an AND gate then for all $i \in [n]$ compute

$$\boldsymbol{k}_{w,c}^{(i)} \leftarrow \tilde{\boldsymbol{d}}_{a,b}^{(i)}(g) + \sum_{j \in [n]} F_{\boldsymbol{k}_{u,a}^{(j)}, \boldsymbol{k}_{v,b}^{(j)}}(g \parallel i).$$

  Then set $c = 0$ if $\boldsymbol{k}_{w,c}^{(i)} = \boldsymbol{k}_{w,0}^{(i)}$ and $c = 1$ otherwise[9].

Ultimately, each party obtains the output keys $\{\boldsymbol{k}_{\overline{w}_1,\gamma_1}^{(i)}, \ldots, \boldsymbol{k}_{\overline{w}_m,\gamma_m}^{(i)}\}_{i \in [n]}$. These keys represent an encryption $\gamma_1, \ldots, \gamma_m$ of the actual outputs $y_1, \ldots, y_m$ of the circuit, and the actual outputs can be recovered using the (secret) permutation bits of the outputs.

The Protocol We now specify the protocol Π_{HSS} which implements $\mathcal{F}_{\mathsf{MPC-SO}}$ in the $\mathcal{F}_{\mathsf{Offline}}$-hybrid model. The reconstruction function f (according to Definition 3) that we use in this protocol is the XOR-function. The protocol uses auxiliary subprotocols $\Pi_{\mathsf{Open}}, \Pi_{\mathsf{POpen}}$ as given in Figure 27, Figure 28 to open either a

[9] We assume here that $\tilde{\boldsymbol{d}}_{a,b}^{(i)}(g)$ was generated correctly.

$[\cdot]$-share in public or privately, but verifiably. The specific construction of **Share Output** is an artifact of the generality of $\mathcal{F}_{\mathsf{MPC-SO}}$ - as its definition shall also capture MPC protocols that e.g. have a secret-sharing based online phase.

Protocol Π_{HSS} (part 1)

The parties evaluate the circuit C with inputs $x^{(1)}, \ldots, x^{(n)}$ and m outputs $\boldsymbol{y} = (y_1, \ldots, y_m)$.

Init: Set up functionalities and garble.

1. The parties set up the functionality $\mathcal{F}_{\mathsf{Offline}}$. They send $(\textsc{Init}, sid,)$ to $\mathcal{F}_{\mathsf{Offline}}$ and in return each $\mathcal{P}_i$ obtains $\boldsymbol{\Delta}^{(i)}$ from $\mathcal{F}_{\mathsf{Offline}}$.
2. Send $(\textsc{Garble}, sid, C)$ to $\mathcal{F}_{\mathsf{Offline}}$. Each $\mathcal{P}_i$ obtains the 0-keys $\boldsymbol{k}_{w,0}^{(i)}$ for all wires as well as λ_{w_ℓ} for its input wires. Moreover, the parties obtain sharings $[\lambda_{\overline{w}_\ell}]$ of the output permutation bits $\lambda_{\overline{w}_\ell}$.

Input: Send input keys. For each input wire $\ell \in [n]$:

1. The party $\mathcal{P}_i$ that holds that input bit $x^{(\ell)}$ computes the encrypted wire value as $\Lambda_{w_\ell} = \lambda_{w_\ell} + x^{(\ell)}$ and broadcasts it to all parties.
2. Each party $\mathcal{P}_j$ broadcasts $\boldsymbol{k}_{w_\ell, \Lambda_{w_\ell}}^{(j)}$.

Evaluate: Exchange garbling and evaluate.

1. The parties send $(\textsc{Open-Garbling}, sid)$ to $\mathcal{F}_{\mathsf{Offline}}$ to obtain $\tilde{\boldsymbol{d}}_{a,b}^{(i)}(g)$ for $i \in [n], g \in \mathbb{G}, a, b \in \{0,1\}$.
2. Traverse the circuit in topological order. For each gate g with inputs u, v having the public values a, b and keys $\boldsymbol{k}_{u,a}^{(i)}, \boldsymbol{k}_{v,b}^{(i)}$ we compute the assignment c to the output wire w as well as the keys $\boldsymbol{k}_{w,c}^{(i)}$ as follows:
   - If g is an XOR gate then set $c \leftarrow a + b$ and $\boldsymbol{k}_{w,c}^{(i)} \leftarrow \boldsymbol{k}_{u,a}^{(i)} + \boldsymbol{k}_{v,b}^{(i)}$ for all $i \in [n]$.
   - If g is an AND gate then for all $i \in [n]$ compute $\boldsymbol{k}_{w,c}^{(i)} \leftarrow \tilde{\boldsymbol{d}}_{a,b}^{(i)}(g) + \sum_{j \in [n]} F_{\boldsymbol{k}_{u,a}^{(j)}, \boldsymbol{k}_{v,b}^{(j)}}(g \,\|\, i)$. $\mathcal{P}_i$ checks if $\boldsymbol{k}_{w,c}^{(i)} \in \{\boldsymbol{k}_{w,0}^{(i)}, \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}\}$. If so then $\mathcal{P}_i$ sets $c = 0$ if $\boldsymbol{k}_{w,c}^{(i)} = \boldsymbol{k}_{w,0}^{(i)}$ and $c = 1$ otherwise. Afterwards set $(\boldsymbol{k}_{w,c}^{(1)}, \ldots, \boldsymbol{k}_{w,c}^{(n)})$ as keys of the wire w. If instead $\boldsymbol{k}_{w,c}^{(i)} \notin \{\boldsymbol{k}_{w,0}^{(i)}, \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}\}$ then $\mathcal{P}_i$ sends abort to all parties.
3. Let $\overline{w}_1, \ldots, \overline{w}_m$ be the output wires of the circuit. Each party $\mathcal{P}_i$ holds output keys $\boldsymbol{k}_{\overline{w}_1, \gamma_1}^{(i)}, \ldots, \boldsymbol{k}_{\overline{w}_m, \gamma_m}^{(i)}$ as well as public values $\gamma_1, \ldots, \gamma_m$.

Fig. 31. Protocol Π_{HSS} Implementing $\mathcal{F}_{\mathsf{MPC-SO}}$.

Theorem 10. *The protocol* Π_{HSS} *UC-securely implements* $\mathcal{F}_{\mathsf{MPC-SO}}$ *against a static malicious adversary corrupting up to* $n-1$ *parties in the* $\mathcal{F}_{\mathsf{Offline}}$*-hybrid model with broadcast.*

We first define a simulator $\mathcal{S}$ which will simulate $\mathcal{F}_{\mathsf{Offline}}$ locally. We then argue why no environment $\mathcal{Z}$ using $\mathcal{A}$ can distinguish the distribution generated by Π_{HSS} and $\mathcal{A}$ from $\mathcal{S}$ which uses $\mathcal{F}_{\mathsf{MPC-SO}}$.

Protocol Π_{HSS} (part 2)

Share Output:

1. Send $(\textsc{Random}, sid, m)$ to $\mathcal{F}_{\mathsf{Offline}}$. Let these sharings be $\{[r_h]\}_{h\in[m]}$.
2. Run Π_{Open} of $[\lambda_{\overline{w}_h} + r_h] \leftarrow [\lambda_{\overline{w}_h}] + [r_h]$ for each $h \in [m]$ to obtain $\hat{\gamma}_h$.
3. Output $[r_h]$ and $\gamma_h + \hat{\gamma}_h$ for each $h \in [m]$.

Share Random Value: Send $(\textsc{Random}, sid, 1)$ to $\mathcal{F}_{\mathsf{Offline}}$ to obtain the sharing $[z]$ for a fresh cid.

Linear Combination: The parties locally compute $[s_{cid'}] \leftarrow \sum_{cid\in\mathcal{I}} \alpha_{cid} \cdot [s_{cid}]$.

Reveal: To open the share $s^{(i)}_{cid}$ of the sharing cid to all parties:

1. Party $\mathcal{P}_i$ broadcasts $x^{(i)}$ and sends $\boldsymbol{\chi}^{(i)}_j$ to $\mathcal{P}_j$ for each $i \neq j$.
2. Each party $\mathcal{P}_j \in \mathcal{P} \setminus \{\mathcal{P}_i\}$ checks that $\boldsymbol{\chi}^{(i)}_j = \boldsymbol{\psi}^{(j)}_i + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$ and broadcasts $\perp$ otherwise.

Private Reveal: The party $\mathcal{P}_i$ opens the share $s^{(i)}_{cid}$ of the sharing cid to party $\mathcal{P}_j$.

1. Party $\mathcal{P}_i$ sends $x^{(i)}, \boldsymbol{\chi}^{(i)}_j$ to party $\mathcal{P}_j$.
2. Party $\mathcal{P}_j$ checks if it holds that $\boldsymbol{\chi}^{(i)}_j = \boldsymbol{\psi}^{(j)}_i + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$. Otherwise, it broadcasts $\perp$.

Fig. 32. Protocol Π_{HSS} Implementing $\mathcal{F}_{\mathsf{MPC-SO}}$ (continued).

Proof. Define the following simulator $\mathcal{S}$:

Init: Set up $\mathcal{F}_{\mathsf{Offline}}$ for the simulation. Initialize $\mathcal{F}_{\mathsf{Offline}}$ with the set I of corrupted parties.

1. Start simulating an honest protocol instance with $\mathcal{A}$ where the inputs of the honest parties are 0. Keep the values $\boldsymbol{\Delta}^{(i)}, i \in I$ which $\mathcal{A}$ provides for the corrupted parties.
2. Run $(\textsc{Garble}, sid, C)$ in $\mathcal{F}_{\mathsf{Offline}}$ with the adversary for the circuit C with wires W and gates $\mathbb{G}$. Therefore, for all $w \in W$ that is output of an AND-gate or an input wire record $\boldsymbol{k}^{(i)}_{w,0}$ which was provided for each $\mathcal{P}_i \in I$ by $\mathcal{A}$. Moreover, sample uniformly random $\lambda_w \xleftarrow{\$} \mathbb{F}$.
3. For each input wire w_i: if $\mathcal{A}$ sends $(\textsc{Ok}, sid, w_i)$ then forward λ_{w_i} which was chosen above.
4. For each output wire $\overline{w}_h$ run the interaction with $\mathcal{F}_{\mathsf{Offline}}$ and keep track of $[\lambda_{\overline{w}_h}]$.

Input: Extract inputs and send these to $\mathcal{F}_{\mathsf{MPC-SO}}$. Therefore, run the protocol with $\mathcal{A}$.

- For each honest party $\mathcal{P}_i$ send $(\textsc{Input}, sid, i, \cdot)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ in the name of the dishonest parties. Then send $\Lambda_{w_i} = \lambda_{w_i}$ as well as honestly sampled $\tilde{\boldsymbol{k}}^{(j)}_{w_i,\Lambda_{w_i}}$ for $j \in \overline{I}$ to $\mathcal{A}$. Store each obtained $\tilde{\boldsymbol{k}}^{(j)}_{w_i,\Lambda_{\overline{w_i}}}$ for $j \in I$ from $\mathcal{A}$.
- For each dishonest party $\mathcal{P}_i$ the adversary sends Λ_{w_i}. Set $x^{(i)} \leftarrow \Lambda_{w_i} + \lambda_{w_i}$ and send $(\textsc{Input}, sid, i, x^{(i)})$ for $\mathcal{P}_i$ and $(\textsc{Input}, sid, i, \cdot)$ for all $\mathcal{P}_j, j \in I \setminus \{i\}$.

Keep the values $\tilde{\boldsymbol{k}}_{w_i,\Lambda^{(j)}_{w_i}}$ for $j \in I$ provided by $\mathcal{A}$ and sample $\tilde{\boldsymbol{k}}^{(j)}_{w_i,\Lambda_{w_i}}$ for $j \in \overline{I}$ honestly.

After this step, all the input keys $\tilde{\boldsymbol{k}}^{(j)}_{w_i,\Lambda_{w_i}}$ that should be used during evaluation as well as the public wire values Λ_{w_i} are fixed.

Evaluate:

1. For each honest party $\mathcal{P}_i$ and for each output wire of an AND-gate $w \in W$ sample $\boldsymbol{k}^{(i)}_{w,\Lambda_w} \xleftarrow{\$} \mathbb{F}^\tau$.
2. For every output wire w of an AND-gate sample $\Lambda_w \xleftarrow{\$} \mathbb{F}$.
3. For every XOR-gate with input wires u, v and output wire w we set $\Lambda_w \leftarrow \Lambda_u + \Lambda_v$. Moreover, set $\boldsymbol{k}^{(i)}_{w,0} \leftarrow \boldsymbol{k}^{(i)}_{u,0} + \boldsymbol{k}^{(i)}_{v,0}$ as well as $\boldsymbol{k}^{(i)}_{w,1} \leftarrow \boldsymbol{k}^{(i)}_{w,0} + \boldsymbol{\Delta}^{(i)}$ for all $i \in [n]$.
4. For the outputs[10] of the circuit $\overline{w}_h$ compute $\gamma_h \leftarrow \Lambda_{\overline{w}_h}$.
5. Next, we generate the keys that are observed by $\mathcal{A}$ when evaluating the circuit. Therefore, for each AND-gate g with public values (Λ_u, Λ_v) compute
$$\boldsymbol{d}^{(j)}_{\Lambda_u,\Lambda_v}(g) \leftarrow \boldsymbol{k}^{(j)}_{w,\Lambda_w} + \sum\nolimits_{i\in[n]} F_{\boldsymbol{k}^{(i)}_{u,\Lambda_u},\boldsymbol{k}^{(i)}_{v,\Lambda_v}}(g \,\|\, j)$$
$$\boldsymbol{d}^{(j)}_{1-\Lambda_u,\Lambda_v}(g), \boldsymbol{d}^{(j)}_{\Lambda_u,1-\Lambda_v}(g), \boldsymbol{d}^{(j)}_{1-\Lambda_u,1-\Lambda_v}(g) \xleftarrow{\$} \mathbb{F}^\tau$$
for all $j \in [n]$. Then for $a, b \in \{0,1\}$ we set $\boldsymbol{d}_{a,b}(g) \leftarrow \boldsymbol{d}^{(1)}_{a,b}(g) \,\|\, \dots \,\|\, \boldsymbol{d}^{(n)}_{a,b}(g)$.
6. On input (Open-Garbling, sid) by $\mathcal{A}$ we send $\{\boldsymbol{d}_{a,b}(g)\}$ for $a, b \in \{0,1\}, g \in \mathbb{G}$. Obtain the additive error $\boldsymbol{e} = \{\boldsymbol{e}_{a,b}(g)\}$ and set $\tilde{\boldsymbol{d}}_{a,b}(g) \leftarrow \boldsymbol{d}_{a,b}(g) + \boldsymbol{e}_{a,b}(g)$.
7. Evaluate the circuit defined by $\tilde{\boldsymbol{d}}_{a,b}(g)$ using the public inputs Λ_{w_i} as well as the input keys $\tilde{\boldsymbol{k}}^{(j)}_{w_i,\Lambda_{w_i}}$ for $i, j \in [n]$. During evaluation, for every wire w obtained check if for each $i \in \overline{I}$ the key $\boldsymbol{k}^{(i)}_{w,\Lambda_w}$ is the pre-programmed key from above for this public value.

Share Output: Make a new randomized sharing of the output.

1. Send (Share-Output, sid) to $\mathcal{F}_{\mathsf{MPC-SO}}$ and obtain $\{cid_h\}_{h\in[m]}$ from it.
2. Send (Random, sid, m) for all simulated honest parties to $\mathcal{F}_{\mathsf{Offline}}$ and observe which b^i_j $\mathcal{A}$ sends. Then send (Output-Shares, sid, $\{(cid_h, b^{(i)}_{cid_h})\}$) for $i \in I$ to $\mathcal{F}_{\mathsf{MPC-SO}}$.
3. Obtain the share advices $\overline{z_{cid_h}}$ for $h \in [m]$ from $\mathcal{F}_{\mathsf{MPC-SO}}$.
4. Simulate Π_{Open} for each $[\lambda_{\overline{w}_h} + r_h]$ by adjusting the opened share of one simulated honest party, such that the honestly reconstructed result is $\gamma_h + \overline{z_{cid_h}}$. If the dishonest parties follow the protocol honestly, send (Deliver-Advices, sid, $\{cid_h\}_{h\in[m]}$) to $\mathcal{F}_{\mathsf{MPC-SO}}$. Otherwise send (Abort, sid).

[10] These values cannot simply be chosen at random as the simulation might then be inconsistent. This can happen e.g. if the outputs of two AND-gates are XOR-ed together two times, where both XORs are outputs of the circuit. If the public values of the XORs were chosen at random, then this cannot be reached during correct evaluation of the circuit.

Share Random Value:

1. Send $(\textsc{Share-Random}, sid)$ in the name of the dishonest parties to $\mathcal{F}_{\mathsf{MPC-SO}}$.
2. Upon receiving cid from $\mathcal{F}_{\mathsf{MPC-SO}}$ run **Generate Random** of $\mathcal{F}_{\mathsf{Offline}}$ honestly. Extract the shares $b^{(i)}$ for $i \in I$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{Offline}}$ and send $(\textsc{Share}, sid, cid, b^{(i)})$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ for each $i \in I$.

Linear Combination: Send $(\textsc{Linear}, sid, \{(cid, \alpha_{cid})\}_{cid \in \mathcal{I}}, cid')$ for all $i \in I$ to $\mathcal{F}_{\mathsf{MPC-SO}}$. Then apply the linear operation to the shares of the simulated honest parties locally.

Reveal: Send $(\textsc{Reveal}, sid, cid, i)$ for the dishonest parties to $\mathcal{F}_{\mathsf{MPC-SO}}$.

- If $i \in I$ simulate the protocol honestly with $\mathcal{A}$. If $\mathcal{A}$ sends incorrect shares, then send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$, otherwise send $(\textsc{Deliver-Reveal}, sid, cid, i)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$.
- If $i \notin I$ then obtain $(\textsc{Reveal}, sid, cid, i, s_{cid}^{(i)})$ from $\mathcal{F}_{\mathsf{MPC-SO}}$. Simulate $\mathcal{P}_i$ to consistently open $s_{cid}^{(i)}$ to all parties. If $\mathcal{A}$ aborts then send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$, otherwise send $(\textsc{Deliver-Reveal}, sid, cid, i)$.

Private Reveal:

- If $i \in I, j \in \overline{I}$ send $(\textsc{Reveal}, sid, cid, i, j)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$ and run the protocol with $\mathcal{A}$. If $\mathcal{A}$ sends incorrect values send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$, otherwise send $(\textsc{Deliver-Reveal}, sid, cid, i, j)$.
- If $i \in \overline{I}, j \in I$ then obtain $s_{cid}^{(i)}$ from $\mathcal{F}_{\mathsf{MPC-SO}}$. Then simulate the honest party in the protocol to open $s_{cid}^{(i)}$ consistently. If $\mathcal{A}$ aborts, send $(\textsc{Abort}, sid)$ to $\mathcal{F}_{\mathsf{MPC-SO}}$, otherwise send $(\textsc{Deliver-Reveal}, sid, cid, i, j)$.

We will argue why each individual protocol part is indistinguishable.

Init: $\mathcal{A}$ only obtains outputs so this is trivially indistinguishable.

Input: All public values Λ_{w_i} as well as keys $\boldsymbol{k}_{w_i, \Lambda_{w_i}}^{(j)}$ which $\mathcal{A}$ obtains are distributed as they are in the protocol, as these are there also chosen uniformly at random.

Evaluate: Our simulation for evaluation is built on top of the simulator of [25], and performs the exact same computation (except for hard-wiring different output values). This allows us to deduce directly that the garbled circuit which is generated is distributed correctly if no party aborts, meaning that all honest parties obtain the same output values if they do not abort (which is the output $\gamma_1, \ldots, \gamma_m$). This follows directly from [25, Lemma 5.4, 5.5 and 5.6] and F being a 2-correlation robust PRF. See the referenced works for details.

Share Output: The adversary obtains output from $\mathcal{F}_{\mathsf{MPC-SO}}$ instead of $\mathcal{F}_{\mathsf{Offline}}$, but the values are equally distributed. There are special cases in which $\mathcal{S}$ aborts where it differs from the protocol, but observe that this is a superset of those cases in which the protocol would abort. We first show that the difference in the abort probability is negligible. The abort happens whenever $\mathcal{A}$ sends incorrect shares during $\Pi_{\mathsf{Open}}, \Pi_{\mathsf{POpen}}$. It follows from the security of the TinyOT protocol that this only happens with probability $2^{-\tau}$, as $\mathcal{A}$ would have to guess $\boldsymbol{\Delta}^{(i)}$ of an honest party $\mathcal{P}_i$ correctly. As we take the shares $s^{(i)}_{cid_h}$ that $\mathcal{A}$ uses in Π_{HSS} and input them into $\mathcal{F}_{\mathsf{MPC-SO}}$ these will be consistent. We open each $[\lambda_{\overline{w}_h} + r_h]$ such that the outputs obtained by $\mathcal{A}$ are consistent with the advice obtained from $\mathcal{F}_{\mathsf{MPC-SO}}$.

Share Random Value: As we take the shares $s^{(i)}_{cid}$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{Offline}}$ and input them into $\mathcal{F}_{\mathsf{MPC-SO}}$ these shares will be consistent.

Linear Combination: This operation is entirely local.

Reveal: In the simulation, if the opened share comes from an honest party then we open to the value that $\mathcal{F}_{\mathsf{MPC-SO}}$ provides which makes the simulation consistent with the functionality. If $\mathcal{P}_i$ is controlled by $\mathcal{A}$ then we abort whenever $\mathcal{A}$ sends a value which it did not obtain from $\mathcal{F}_{\mathsf{Offline}}$ or which it did not derive correctly, which is distinguishable from Π_{HSS} only if $\mathcal{A}$ could have guessed a $\boldsymbol{\Delta}^{(j)}$.

Private Reveal: This is the same as for the case of **Reveal**.

□

In the above we were actually a bit inaccurate, as what is proven is that Π_{HSS} implements $\mathcal{F}_{\mathsf{MPC-SO}}$ with a Key Query functionality (whereas $\mathcal{F}_{\mathsf{MPC-SO}}$ as such has no such property). This gives an additional distinguishing advantage of $q/2^\tau$ to the environment, where q is the number of Key Queries which $\mathcal{A}$ can do (which is polynomial in κ). This additional advantage is thus negligible in the computational security parameter.

Implementing the Offline Functionality We present here an implementation of the functionality $\mathcal{F}_{\mathsf{Offline}}$. For this, we use a multiparty version of the TinyOT MPC protocol $\mathcal{F}_{\mathsf{TinyOT}}$ [37, 35, 12], which is depicted in Figure 33. These works implement this functionality using the same building blocks as the commitments from Section 4 (namely secure equality testing, commitments and OT) as well as hash functions. Therefore, we can reuse $\mathcal{F}_{\mathsf{pOT}}, \mathcal{F}_{\mathsf{EQ}}, \mathcal{F}_{\mathsf{Com}}$ in the construction. In practice, one would choose lighter variants as public verifiability is not necessary to implement $\mathcal{F}_{\mathsf{TinyOT}}$.

It was observed in [25, 43] that a $[x]$-representation (like the random $[\cdot]$-shares generated by **Random Bits**) can be converted into additive shares $r^{(1)}, \ldots, r^{(n)}$ of $x \cdot \boldsymbol{\Delta}^{(i)}$ for each $i \in [n]$ as

$$\mathcal{P}_i \text{ sets } r^{(i)} = x^{(i)} \cdot \boldsymbol{\Delta}^{(i)} + \sum_{k \in [n], k \neq i} \boldsymbol{\psi}_k^{(i)}$$

$$\mathcal{P}_j, i \neq j \text{ sets } r^{(j)} = \boldsymbol{\chi}_i^{(j)}$$

This is then repeated to obtain shares for each product $x \cdot \boldsymbol{\Delta}^{(i)}$ of x with all secrets $\boldsymbol{\Delta}^{(i)}$.

Proposition 1. *A representation $[x]$ can be converted locally into additive shares of $x \cdot \boldsymbol{\Delta}^{(i)}$ for each $i \in [n]$ by the above method.*

Functionality $\mathcal{F}_{\mathsf{TinyOT}}$

This functionality interacts with parties $\mathcal{P}$ and an adversary $\mathcal{S}$. Let $I \subset \mathcal{P}$ denote the set of dishonest parties chosen by $\mathcal{S}$.

Setup: On input $(\textsc{Setup}, sid)$
1. Receive $\boldsymbol{\Delta}^{(i)} \in \mathbb{F}^\tau$ for each $i \in I$ from $\mathcal{S}$.
2. For each honest party $\mathcal{P}_i \in \overline{I}$ sample $\boldsymbol{\Delta}^{(i)} \xleftarrow{\$} \mathbb{F}^\tau$ and send it to $\mathcal{P}_i$.

Random Bits: On input $(\textsc{Bits}, sid, k)$ from all parties
1. For $i \in I, j \in [k]$ wait for $b_j^{(i)} \in \mathbb{F}$ from $\mathcal{S}$.
2. For $i \in \overline{I}, j \in [k]$ sample $b_j^{(i)} \xleftarrow{\$} \mathbb{F}$.
3. For $j \in [k]$ run $[b_j] \leftarrow \mathsf{Bracket}(b_j^{(1)}, \cdots, b_j^{(n)})$.

Triples: On input $(\textsc{Triples}, sid, k)$ from all parties
1. For $i \in I, j \in [k]$ wait for $a_j^{(i)}, b_j^{(i)}, c_j^{(i)} \in \mathbb{F}$ from $\mathcal{S}$.
2. For $i \in \overline{I}, j \in [k]$ sample $a_j^{(i)}, b_j^{(i)} \xleftarrow{\$} \mathbb{F}$ at random and $c_j^{(i)} \xleftarrow{\$} \mathbb{F}$ with the constraint that $(\sum_{i\in[n]} a_j^{(i)}) \cdot (\sum_{i\in[n]} b_j^{(i)}) = \sum_{i\in[n]} c_j^{(i)}$.
3. For $j \in [k]$ run $[a_j] \leftarrow \mathsf{Bracket}(a_j^{(1)}, \ldots, a_j^{(n)})$, $[b_j] \leftarrow \mathsf{Bracket}(b_j^{(1)}, \ldots, b_j^{(n)})$ and $[c_j] \leftarrow \mathsf{Bracket}(c_j^{(1)}, \ldots, c_j^{(n)})$.

Macro $\mathsf{Bracket}$: On input $x^{(1)}, \ldots, x^{(n)}$ compute $[x]$ for each $\mathcal{P}_i$
- if $\mathcal{P}_i \in I$ then $\forall j \in [n] \setminus \{i\}$ wait for $\boldsymbol{\chi}_j^{(i)}$ from $\mathcal{S}$, then compute $\boldsymbol{\psi}_i^{(j)} = \boldsymbol{\chi}_j^{(i)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$.
- if $\mathcal{P}_i \in \overline{I}$ then $\forall j \in I$ wait for $\boldsymbol{\psi}_i^{(j)}$ from $\mathcal{S}$ and choose $\boldsymbol{\psi}_i^{(j)}$ honestly for all $j \in \overline{I} \setminus \{i\}$. Then compute $\boldsymbol{\chi}_j^{(i)} = \boldsymbol{\psi}_i^{(j)} + x^{(i)} \cdot \boldsymbol{\Delta}^{(j)}$.

Output $(x^{(i)}, \{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j\in[n]\setminus\{i\}})$ to each $\mathcal{P}_i$.

Key Queries: Upon receiving $(i, \boldsymbol{\Delta})$ for $i \in [n]$ from $\mathcal{S}$ and if **Setup** was run before return 1 if $\boldsymbol{\Delta} = \boldsymbol{\Delta}^{(i)}$ and 0 otherwise.

Fig. 33. Functionality $\mathcal{F}_{\mathsf{TinyOT}}$ For The Multiparty Computation Protocol TinyOT.

Proof. See e.g. [25, Claim 4.1] □

Our preprocessing protocol Π_{Offline} is a modified version of [25]. We nevertheless provide a full proof here.

Theorem 11. *The protocol Π_{Offline} UC-securely implements $\mathcal{F}_{\mathsf{Offline}}$ against a static, malicious adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{TinyOT}}$-hybrid model with a broadcast channel.*

Proof. To prove this statement, we construct a simulator $\mathcal{S}$ in the presence of $\mathcal{F}_{\mathsf{Offline}}$ which interacts with the PPT real-world adversary $\mathcal{A}$, and show that any PPT environment $\mathcal{Z}$ cannot distinguish the setting $\mathcal{S}, \mathcal{A}, \mathcal{F}_{\mathsf{Offline}}$ from $\mathcal{A}, \Pi_{\mathsf{HSS}}$, $\mathcal{F}_{\mathsf{TinyOT}}$. The adversary $\mathcal{A}$ corrupts a set $I \subset [n]$ at the beginning of the execution, and $\mathcal{S}$ will simulate honest parties as well as an instance of $\mathcal{F}_{\mathsf{TinyOT}}$. As $\mathcal{S}$ sees the

Protocol Π_{Mult}

Let $([a],[b],[c])$ be a triple such that $c = a \cdot b$. On input $[x],[y]$ the parties compute a sharing $[z]$ such that $z = x \cdot y$ as follows.

1. Each party locally computes $[\rho] = [a] + [x]$ as well as $[\tau] = [b] + [y]$.
2. Run Π_{Open} to open both ρ, τ reliably.
3. Each party locally computes $[z] = [c] + \rho \cdot [b] + \tau \cdot [a] + \rho \cdot \tau$.

Fig. 34. Protocol Π_{Mult} For The Multiplication Of Two $[\cdot]$-Representations Using Multiplication Triples.

random string which $\mathcal{A}$ obtains from the environment, $\mathcal{S}$ internally simulates the messages that we would expect the parties in I to send, but of course security does not rely on this as $\mathcal{A}$ may send arbitrary messages.

$\mathcal{S}$ simulates honest parties throughout the protocol, and then adjusts the output obtained during **Open Garbling** accordingly. It works as follows:

Init:
1. Set up an instance of $\mathcal{F}_{\mathsf{TinyOT}}$ and simulate it honestly, except for every Key Query of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{TinyOT}}$ which $\mathcal{S}$ forwards to $\mathcal{F}_{\mathsf{Offline}}$.
2. Forward the set of corrupted parties I to this functionality and to $\mathcal{F}_{\mathsf{Offline}}$. Send $(\textsc{Setup}, sid)$ from all honest parties to $\mathcal{F}_{\mathsf{TinyOT}}$ and forward any such messages from $\mathcal{A}$.
3. Wait for $\boldsymbol{\Delta}^{(i)}$ from $\mathcal{A}$ for each $\mathcal{P}_i \in I$ and store these internally. Keep $\boldsymbol{\Delta}^{(i)}$ for the honest $\mathcal{P}_i$ as obtained from $\mathcal{F}_{\mathsf{TinyOT}}$.
4. Send $(\textsc{Init}, sid)$ from all dishonest parties to $\mathcal{F}_{\mathsf{Offline}}$. Then send $\boldsymbol{\Delta}^{(i)}$ for each $i \in I$.

Garble:
1. Send $(\textsc{Garble}, sid, C)$ in the name of all dishonest parties to $\mathcal{F}_{\mathsf{Offline}}$. Denote with W the set of wires and $\mathbb{G}$ the set of AND-gates.
2. For each $w \in W$ that is an input wire or an output wire of an AND-gate, send $(\textsc{Bits}, sid, 1)$ to $\mathcal{F}_{\mathsf{TinyOT}}$ in the name of the simulated honest parties to obtain the shares of $[\lambda_w]$. If w instead is an output of a XOR-gate, set $[\lambda_w] \leftarrow [\lambda_u] + [\lambda_v]$ where u, v are the input wires.
3. For the PRF-keys $\boldsymbol{k}_{w,\cdot}^{(i)}$ for each $w \in W$, if w is an input wire or an output of an AND-gate then choose a uniformly random $\boldsymbol{k}_{w,0}^{(i)} \xleftarrow{\$} \mathbb{F}^{\tau}$ for each simulated honest $\mathcal{P}_i$ and compute $\boldsymbol{k}_{w,0}^{(i)}$ of the dishonest $\mathcal{P}_i$ from the input tape of the party. Then, set $\boldsymbol{k}_{w,1}^{(i)} = \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}$ for each $i \in [n]$.
4. For each AND-gate $g \in \mathbb{G}$ with input wires u, v and output wire w do the following:
   (a) Send $(\textsc{Triples}, sid, 1)$ from each simulated honest party to $\mathcal{F}_{\mathsf{TinyOT}}$, then obtain shares of the triple $([a],[b],[c])$.
   (b) Run Π_{Mult} as in Π_{Offline} to compute $[\lambda_{uv}]$. During either instance of Π_{Open} in Π_{Mult}, abort if $\mathcal{A}$ provides shares for any dishonest party which are

Protocol Π_{Offline} (part 1)

The parties $\mathcal{P}$ start by running an instance of $\mathcal{F}_{\mathsf{TinyOT}}$. For the circuit C we let $\mathbb{G}$ be the set of gates. Let $G:\ \mathbb{F}^{\tau} \to \mathbb{F}^{4n\tau|\mathbb{G}|}$ be a PRG and $F:\ \mathbb{F}^{2\tau} \times \mathbb{F}^{\tau} \to \mathbb{F}^{\tau}$ be a circular 2-correlation robust PRF.

Init: If this has not been run before, then all parties send $(\textsc{Setup}, sid)$ to $\mathcal{F}_{\mathsf{TinyOT}}$. Party $\mathcal{P}_i$ obtains $\boldsymbol{\Delta}^{(i)}$.

Garble: If this has not been run before and **Init** ran successfully, then all parties do the following:

1. All parties go through the wires of the circuit C topologically. For each wire w they do the following:
   - If w is an input wire of the circuit or an output wire of an AND-gate, then all parties send $(\textsc{Bits}, sid, 1)$ to $\mathcal{F}_{\mathsf{TinyOT}}$ and obtain a value $[\lambda_w]$. Then each party $\mathcal{P}_i$ samples $\boldsymbol{k}_{w,0}^{(i)} \xleftarrow{\$} \mathbb{F}^{\tau}$ and sets $\boldsymbol{k}_{w,1}^{(i)} \leftarrow \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}$.
   - If w is the output of a XOR-gate with input wires u, v then the parties set $[\lambda_w] \leftarrow [\lambda_u] + [\lambda_v]$. Moreover, each $\mathcal{P}_i$ sets $\boldsymbol{k}_{w,0}^{(i)} \leftarrow \boldsymbol{k}_{u,0}^{(i)} + \boldsymbol{k}_{v,0}^{(i)}$ as well as $\boldsymbol{k}_{w,1}^{(i)} \leftarrow \boldsymbol{k}_{w,0}^{(i)} + \boldsymbol{\Delta}^{(i)}$.
2. For each AND-gate $g \in \mathbb{G}$ with input wires u, v and output wire w the parties do the following
   (a) The parties send $(\textsc{Triples}, sid, 1)$ to $\mathcal{F}_{\mathsf{TinyOT}}$ to obtain a triple $([a], [b], [c])$. Then, they run Π_{Mult} with $(([a], [b], [c]), [\lambda_u], [\lambda_v])$ to compute $[\lambda_{uv}]$ and set $[\lambda_{uv} + \lambda_w] \leftarrow [\lambda_{uv}] + [\lambda_w]$ afterwards.
   (b) For each $j \in [n]$ the parties use Proposition 1 to convert $[\lambda_u], [\lambda_v], [\lambda_{uv} + \lambda_w]$ into additive shares of $\lambda_u \cdot \boldsymbol{\Delta}^{(j)}, \lambda_v \cdot \boldsymbol{\Delta}^{(j)}, (\lambda_{uv} + \lambda_w) \cdot \boldsymbol{\Delta}^{(j)}$. Write $\boldsymbol{r}_{u,j}^{(i)}$ for the share that $\mathcal{P}_i$ holds of $\lambda_u \cdot \boldsymbol{\Delta}^{(j)}$, and similarly define $\boldsymbol{r}_{v,j}^{(i)}, \boldsymbol{r}_{uv+w,j}^{(i)}$.
   (c) For each $j \in [n]$ and $a, b \in \{0, 1\}$ each $\mathcal{P}_i$ sets
$$\rho_{a,b,j}^{(i)}(g) \leftarrow \begin{cases} a \cdot \boldsymbol{r}_{v,j}^{(i)} + b \cdot \boldsymbol{r}_{u,j}^{(i)} + \boldsymbol{r}_{uv+w,j}^{(i)} & \text{if } i \neq j \\ a \cdot \boldsymbol{r}_{v,j}^{(i)} + b \cdot \boldsymbol{r}_{u,j}^{(i)} + \boldsymbol{r}_{uv+w,j}^{(i)} + a \cdot b \cdot \boldsymbol{\Delta}^{(i)} & \text{if } i = j \end{cases}$$
3. For each AND-gate $g \in \mathbb{G}$, each $a, b \in \{0, 1\}$ and each $j \in [n]$ party $\mathcal{P}_i$ computes its share of $(\boldsymbol{d}_{a,b}^{(j)}(g))^{(i)}$ as
$$(\boldsymbol{d}_{a,b}^{(j)}(g))^{(i)} \leftarrow \begin{cases} \rho_{a,b,j}^{(i)} + F_{\boldsymbol{k}_{u,a}^{(i)}, \boldsymbol{k}_{v,b}^{(i)}}(g \parallel j) + \boldsymbol{k}_{w,0}^{(i)} & \text{if } i = j \\ \rho_{a,b,j}^{(i)} + F_{\boldsymbol{k}_{u,a}^{(i)}, \boldsymbol{k}_{v,b}^{(i)}}(g \parallel j) & \text{else} \end{cases}$$
4. For each party $\mathcal{P}_i$ let w_i be the input wire corresponding to its input. Then run Π_{POpen} on $[\lambda_{w_i}]$ towards $\mathcal{P}_i$.
5. For each $\overline{w}_h \in W$ which is an output wire of the circuit, define $[\lambda_{\overline{w}_h}]$ to be the sharing of the permutation bit $\lambda_{\overline{w}_h}$.

Fig. 35. Protocol Π_{Offline} Implementing The Offline Phase $\mathcal{F}_{\mathsf{Offline}}$.

inconsistent with the shares of $[a+\lambda_u] = [a] + [\lambda_u]$ and $[b+\lambda_v] + [b] + [\lambda_v]$ which $\mathcal{A}$ obtained from $\mathcal{F}_{\mathsf{TinyOT}}$.

Protocol Π_{Offline} (part 2)

Open Garbling: This can only be run once and if **Garble** ran successfully. Each $\mathcal{P}_i$ has a share $\tilde{C}^{(i)} = \{(\boldsymbol{d}_{a,b}^{(j)}(g))^{(i)}\}_{j,a,b,g}$ of length $4n\tau|\mathbb{G}|$ from **Garble**.

1. Each $\mathcal{P}_i$ samples $n-1$ random seeds $s_j^{(i)} = \mathbb{F}^\tau$ for all $j \neq i$ and sends $s_j^{(i)}$ to $\mathcal{P}_j$.
2. Each $\mathcal{P}_i$ computes $S_i^{(i)} = \sum_{i \neq j} G(s_j^{(i)})$ and $S_i^{(j)} = G(s_i^{(j)})$ for $j \neq i$.
3. For $i \in [n] \setminus \{1\}$ the party $\mathcal{P}_i$ sends $T^{(i)} = \tilde{C}^{(i)} + \sum_{j=1}^n S_i^{(j)}$ to $\mathcal{P}_1$.
4. $\mathcal{P}_1$ computes $\tilde{C} \leftarrow \tilde{C}^{(1)} + \sum_{j=1}^n S_1^{(j)} + \sum_{i=2}^n T^{(i)}$ and broadcasts it to all parties.

Generate Random: If **Init** ran successfully, then each party sends $(\textsc{Bits}, sid, \ell)$ to $\mathcal{F}_{\mathsf{TinyOT}}$ to obtain the ℓ shares $([r_1], \dots, [r_\ell])$.

Fig. 36. Protocol Π_{Offline} Implementing The Offline Phase $\mathcal{F}_{\mathsf{Offline}}$(continued).

(c) If no abort due to the above event occurred, then set the shares $[\lambda_{uv} + \lambda_w] = [\lambda_{uv}] + [\lambda_w]$ for each simulated honest party.

5. If no abort occurred, then send $\boldsymbol{k}_{w,0}^{(i)}$ for each $i \in I$ and each $w \in W$ which is either an input-wire or the output of an AND-gate to $\mathcal{F}_{\mathsf{Offline}}$.
6. For each input wire w do the following:
   - If w belongs to an honest party, wait for the shares of $[\lambda_w]$ that $\mathcal{A}$ sends during Π_{POpen}. If any of these are inconsistent with the shares that it should have, then abort. Else, send $(\textsc{Ok}, sid, w)$ to $\mathcal{F}_{\mathsf{Offline}}$.
   - If w belongs to a dishonest party, then send $(\textsc{Ok}, sid, w)$ to $\mathcal{F}_{\mathsf{Offline}}$ to obtain $\overline{\lambda_w}$. If $\overline{\lambda_w} = \sum_i \lambda_w^{(i)}$ where $\lambda_w^{(i)}$ is the share of $\mathcal{P}_i$ of $[\lambda_w]$ then run Π_{POpen} correctly as in the protocol. Else, for one simulated honest party $\mathcal{P}_j$ adjust the share and the MACs such that the above equation holds based on $\boldsymbol{\Delta}^{(i)}$ of the dishonest parties that $\mathcal{S}$ has, and then run Π_{POpen}.
7. For each output wire $\overline{w}$ of the circuit C:
   (a) For each dishonest party $\mathcal{P}_i$ compute $\lambda_{\overline{w}}^{(i)}$ as well as $\{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j \in [n] \setminus \{i\}}\}$ based on the values that $\mathcal{P}_i$ obtained from $\mathcal{F}_{\mathsf{TinyOT}}, \Pi_{\mathsf{Mult}}$.

   (b) Send $\lambda_{\overline{w}}^{(i)}$ to $\mathcal{F}_{\mathsf{Offline}}$ as shares of the permutation bit $\lambda_{\overline{w}}$. During Bracket send $\{\boldsymbol{\chi}_j^{(i)}, \boldsymbol{\psi}_j^{(i)}\}_{j \in [n] \setminus \{i\}}\}$ for each dishonest $\mathcal{P}_i$.

Open Garbling:

1. Based on the random tapes of the dishonest parties as well as the outputs that $\mathcal{A}$ obtained from $\mathcal{F}_{\mathsf{TinyOT}}$ and during Π_{Mult} compute the share $\tilde{C}^{(i)}$ of each dishonest $\mathcal{P}_i$ as in the protocol.
2. For each $j \in \overline{I}, i \in I$ sample $S_i^{(j)} \xleftarrow{\$} \mathbb{F}^\tau$ uniformly at random and send these to $\mathcal{A}$.
3. Send $(\textsc{Open-Garbling}, sid)$ for all dishonest parties to $\mathcal{F}_{\mathsf{Offline}}$, and obtain $\tilde{C}$ in return.
4. For $i \in \overline{I}$ sample uniformly random shares $\tilde{C}^{(i)}$ subject to the constraint that $\tilde{C} = \sum_{j \in [n]} \tilde{C}^{(j)}$. Then use these shares to run the protocol honestly.

5. In the protocol, $\mathcal{P}_1$ broadcasts the circuit $\hat{C}$. Compute $\boldsymbol{e}_{a,b}(g)$ from $\tilde{C} + \hat{C}$ and send it to $\mathcal{F}_{\mathsf{Offline}}$.

Generate Random: Obtain inputs $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{TinyOT}}$, then forward these to $\mathcal{F}_{\mathsf{Offline}}$.

1. Send $(\textsc{Bits}, sid, \ell)$ to $\mathcal{F}_{\mathsf{TinyOT}}$ for all simulated honest parties. Let $(x_r^{(i)}, \{\boldsymbol{\chi}_{j,r}^{(i)}, \boldsymbol{\psi}_{j,r}^{(i)}\}_{j\in[n]\setminus i})$ be the values that $\mathcal{A}$ provides for $\mathcal{P}_i \in I$ to generate the rth random bit via $\mathsf{Bracket}$ in $\mathcal{F}_{\mathsf{TinyOT}}$ for $r \in [\ell]$.
2. Send $(\textsc{Random}, sid, \ell)$ for all dishonest parties to $\mathcal{F}_{\mathsf{Offline}}$. For each $\mathcal{P}_i \in I$ and for $r \in [\ell]$ send $(x_r^{(i)}, \{\boldsymbol{\chi}_{j,r}^{(i)}, \boldsymbol{\psi}_{j,r}^{(i)}\}_{j\in[n]\setminus i})$ to $\mathcal{F}_{\mathsf{Offline}}$.

We now argue indistinguishability of the outputs, both to the honest parties of $\mathcal{F}_{\mathsf{Offline}}$ and to $\mathcal{A}$.

Init: Only the honest parties receive outputs, and these have the same distribution in both cases.

Garble: While we run $\mathcal{S}$ with $\mathcal{A}$ we essentially run Π_{Offline} with simulated parties, thus every output that $\mathcal{A}$ obtains has the same distribution as in the protocol. The permutation bits of the inputs which the dishonest parties obtain are consistent with $\mathcal{F}_{\mathsf{Offline}}$ and the adversary can abort also in the simulation by providing incorrect openings. The shares of the permutation bits of the outputs are consistent as in the simulation the shares of $\mathcal{A}$ as well as the MACs and keys are provided to $\mathcal{F}_{\mathsf{Offline}}$, and the global difference $\boldsymbol{\Delta}^{(i)}$ of dishonest parties is the same both in $\mathcal{F}_{\mathsf{TinyOT}}$ and $\mathcal{F}_{\mathsf{Offline}}$.

Open Garbling: Both in the simulation an the real protocol, the seeds of the PRG have the same distribution. In both cases, the shares of the circuit sum up to the correct output, constrained on the shares of the circuit which the adversary has. Moreover, in case of more than one honest party the shares individually appear uniformly random in the protocol as $\mathcal{Z}$ then does not see at least one PRG seed. The error $\boldsymbol{e}_{a,b}(g)$, which is introduced by $\mathcal{A}$, is moreover identical in both cases.

Generate Random: Here, the adversary only gives inputs. Moreover, the shared values are uniformly random in both cases and the distribution of the shares, MACs and MAC keys which the honest parties obtain is the same for both $\mathcal{F}_{\mathsf{Offline}}, \mathcal{F}_{\mathsf{TinyOT}}$, also constrained of the values that $\mathcal{A}$ possesses.

$\mathcal{S}$ has two remaining differences to Π_{Offline} when considering its abort behavior, namely with respect to Π_{Open} and the Key Queries. We show that a hybrid argument allows to remove these:

1. In the first hybrid where we depart from the original $\mathcal{S}$, change $\mathcal{S}$ to always return 0 to $\mathcal{A}$ if queried for $\boldsymbol{\Delta}^{(i)}$ an honest party. This is within distance $(q+1)/2^\tau$ of $\mathcal{S}$ (where q is the number of Key Queries), which is negligible for any PPT $\mathcal{A}$.
2. In the next step, remove the aborting constraint in case $\mathcal{A}$ provides incorrect shares during $\Pi_{\mathsf{Open}}, \Pi_{\mathsf{POpen}}$ and only abort if the equations do not match. It

is easy to see that this amounts to $\mathcal{A}$ correctly guessing $\boldsymbol{\Delta}^{(i)}$ of a $\mathcal{P}_i \in \overline{I}$, which was chosen uniformly at random. As the number of openings throughout the protocol is polynomial, this hybrid is statistically close to the previous one.

3. Now forward all key queries to $\mathcal{F}_{\mathsf{TinyOT}}$. As argued before, this is statistically close to the previous hybrid. Moreover, this is now the identical setting as in Π_{Offline}.

$\square$

The offline functionality allows garbling to happen only once for a fixed instance of $\mathcal{F}_{\mathsf{TinyOT}}$, but one can change the functionality and the security proof to allow to generate multiple garblings for the same $\mathcal{F}_{\mathsf{TinyOT}}$-instance.

References

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, 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.
8. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.
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, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, December 2017.
12. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.
13. 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.
14. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
15. 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.
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. Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.
18. 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.
19. 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.
20. Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In Junji Shikata, editor, *ICITS 17*, volume 10681 of *LNCS*, pages 110–134. Springer, Heidelberg, November / December 2017.
21. 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.
22. 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.
23. 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.
24. 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.

25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.
31. 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.
32. 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.
33. 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.
34. 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.
35. Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, Heidelberg, August 2014.
36. 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.
37. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

38. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.
39. 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.
40. 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.
41. Berry Schoenmakers and Meilof Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 3–22. Springer, Heidelberg, June 2015.
42. Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In Anderson C. A. Nascimento and Paulo Barreto, editors, *ICITS 16*, volume 10015 of *LNCS*, pages 151–176. Springer, Heidelberg, August 2016.
43. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 39–56. ACM Press, October / November 2017.
44. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.