Synchronizable Fair Exchange

Ranjit Kumaresan* Srinivasan Raghuraman† Adam Sealfon‡

October 7, 2020

Abstract

Fitzi, Garay, Maurer, and Ostrovsky (Journal of Cryptology 2005) showed that in the presence of a dishonest majority, no primitive of cardinality $n-1$ is complete for realizing an arbitrary n-party functionality with *guaranteed output delivery*. In this work, we introduce a new 2-party primitive $\mathcal{F}_{\mathsf{SyX}}$ ("synchronizable fair exchange") and show that it is complete for realizing any n-party functionality with *fairness* in a setting where all n parties are pairwise connected by independent instances of $\mathcal{F}_{\mathsf{SyX}}$.

In the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model, the two parties *load* $\mathcal{F}_{\mathsf{SyX}}$ with some input, and following this, either party can *trigger* $\mathcal{F}_{\mathsf{SyX}}$ with a suitable "witness" at a later time to receive the output from $\mathcal{F}_{\mathsf{SyX}}$. Crucially the other party also receives output from $\mathcal{F}_{\mathsf{SyX}}$ when $\mathcal{F}_{\mathsf{SyX}}$ is triggered. The trigger witnesses allow us to *synchronize* the trigger phases of multiple instances of $\mathcal{F}_{\mathsf{SyX}}$, thereby aiding in the design of fair multiparty protocols. Additionally, a pair of parties may *reuse* a single *a priori* loaded instance of $\mathcal{F}_{\mathsf{SyX}}$ in any number of multiparty protocols (possibly involving different sets of parties).

Keywords: secure multiparty computation, fair exchange, completeness, preprocessing.

*Visa Research, rakumare@visa.com.

†Visa Research, srraghur@visa.com. This work was done in part while the author was at MIT.

‡UC Berkeley, asealfon@berkeley.edu. This work was done in part while the author was at MIT.

Contents

1 Introduction

Secure multiparty computation (MPC) allows a set of mutually mistrusting parties to perform a joint computation on their inputs that reveals only the outcome of the computation and nothing else. Showing feasibility [Yao86, GMW87, BGW88, CCD88, RB89] of this seemingly impossible to achieve notion has been one of the most striking contributions of modern cryptography. However, definitions of secure computation do vary across models, in part owing to the general impossibility results for fair coin-tossing [Cle86]. In settings where the majority of the participating parties are dishonest (including the two party setting), a protocol for secure computation only provides *security-with-abort*, and in particular is not required to guarantee important properties such as guaranteed output delivery or even fairness[1]. On the other hand, when up to $t < n/3$ parties are corrupt, then there exist protocols for n-party secure computation that guarantee output delivery [BGW88, CCD88]. This result can be extended to a setting where up to $t < n/2$ parties are corrupt assuming the existence of a broadcast channel [GMW87, RB89].

Given the state of affairs, there has been extensive research to better understand the problem of fairness and guaranteed output delivery in secure computation in setting where $t \geq n/2$. For instance, while Cleve [Cle86] showed that two-party fair coin tossing is impossible, the works of Gordon et al. [GHKL11, GK09, Ash14, ABMO15] showed the existence of non-trivial functions for which fair secure computation is possible in the dishonest majority setting. On the other hand, *partially fair* secure computation [GK10, BLOO11] provides a solution for a relaxed notion of fairness in secure computation where fairness may be breached but only with some parameterizable (inverse polynomial) probability.

Most relevant to our work is the work of Fitzi, Garay, Maurer, and Ostrovsky [FGMO05] who studied complete primitives for secure computation *with guaranteed output delivery*. They showed that no primitive of cardinality $n-1$ is complete for n-party secure computation. More generally, for $n \geq 3$ and $k < n$, they show that no primitive of cardinality k is complete when $t \geq \lceil \frac{k-1}{k+1} \cdot n \rceil$. It follows that when $t \geq \lceil n/3 \rceil$, no primitive of cardinality 2 is complete for secure computation. Also, when $t \geq n-2$, no primitive of cardinality $k < n$ is complete for secure computation. They also show a primitive of cardinality n that is complete for n-party secure computation when $t \geq n-2$.

It is interesting to note that the above impossibility results are derived in [FGMO05] by showing the *impossibility of broadcast* (or Byzantine agreement) given a primitive of cardinality k. In this context, note that Cohen and Lindell [CL17] showed that the presence of a broadcast channel is inconsequential to achieving the goal of fairness, i.e., they showed that any protocol for fair computation that uses a broadcast channel can be compiled into one that does not use a broadcast channel. They also showed that assuming the existence of a broadcast channel, any protocol for fair secure computation can be compiled into one that provides guaranteed output delivery. Importantly, all these transformations only require primitives of cardinality 2.

Our contributions. Given the above, one wonders whether the impossibility result of [FGMO05] can be bypassed if we restrict our attention to *fair secure computation* alone. In this work, we introduce a new 2-party primitive $\mathcal{F}_{\mathsf{SyX}}$ ("synchronizable fair exchange," or simply "synchronizable exchange") and show that it is complete for realizing any n-party functionality with *fairness* in a setting where all n parties are pairwise connected by independent instances of $\mathcal{F}_{\mathsf{SyX}}$. Our work,

[1] Fairness means that either all parties get the output or none do. Guaranteed output delivery means that all parties get the output.

combined with [CL17] and [FGMO05], clarifies the power of broadcast in obtaining fairness. Additionally, a pair of parties may *reuse* a single instance of $\mathcal{F}_{\mathsf{SyX}}$ in any number of multiparty protocols, possibly involving different sets of parties.

Synchronizable exchange $\mathcal{F}_{\mathsf{SyX}}$ is a two-party symmetric primitive that is reactive (like the commitment functionality $\mathcal{F}_{\mathsf{com}}$ [CF01]) and works in two phases. In the first phase, which we call the *load phase*, parties submit their private inputs x_1, x_2 along with public inputs $(f_1, f_2, \phi_1, \phi_2)$. Here f_1, f_2 are 2-input 2-output functions, and ϕ_1, ϕ_2 are boolean predicates. The public input must be submitted by both parties, and the submitted values must match. Upon receiving these inputs, $\mathcal{F}_{\mathsf{SyX}}$ computes $f_1(x_1, x_2)$ and delivers the respective outputs to both parties. Next, in the *trigger phase*, which can be initiated at any later time after the load phase, party P_i can send a "witness" w_i to $\mathcal{F}_{\mathsf{SyX}}$ following which $\mathcal{F}_{\mathsf{SyX}}$ checks if $\phi_i(w_i) = 1$. If that is indeed the case, then $\mathcal{F}_{\mathsf{SyX}}$ computes $f_2(x_1, x_2)$ and delivers the respective outputs along with w_i to both parties. We stress that $\mathcal{F}_{\mathsf{SyX}}$ guarantees that both parties get the output of f_2.

To use multiple pairwise instances of synchronizable exchange to achieve n-wise fair secure computation, the main idea is to keep different instances of $\mathcal{F}_{\mathsf{SyX}}$ "in sync" with each other throughout the protocol execution. That is, we need to ensure that all pairwise $\mathcal{F}_{\mathsf{SyX}}$ instances are, loosely speaking, *simultaneously loaded*, and if so, *simultaneously triggered*. Ensuring this in the presence of byzantine adversaries is somewhat tricky, and we outline our techniques below.

Reduction to fair reconstruction. First, we let parties run an (unfair) MPC protocol for a function f that accepts parties' inputs and computes the function output, then computes secret shares of the function output, and then computes commitments on these secret shares. Finally, the MPC outputs to all parties the set of all commitments computed above, and to each individual party the corresponding share of the function output. Since the MPC protocol itself does not guarantee fairness, it may be that some honest party does not receive the output. In that case, all parties terminate and abort the protocol, and no party learns the function output. If the protocol has not terminated, then all that is left to perform a fair reconstruction of the function output from the secret shares. The above technique of reducing fair computation of a function to fair reconstruction of a (non-malleable) additive secret sharing scheme is a well-known technique [GIM+10].

Synchronization via trigger conditions. The commitments generated in the above step are used to define the trigger conditions, specifically the trigger witness must include (among other things) openings to the commitments (i.e., the secret shares). That is, each pair of parties initiate the load phase with their $\mathcal{F}_{\mathsf{SyX}}$ instance. We will need to ensure that the protocol proceeds only if all $\mathcal{F}_{\mathsf{SyX}}$ instances were loaded. To do this, we let the load phase of each $\mathcal{F}_{\mathsf{SyX}}$ instance to output a *receipt* (think of these as signatures on some special instance-specific message) that indicates that the $\mathcal{F}_{\mathsf{SyX}}$ instance has been loaded. Following this parties broadcast to all other parties the receipts they have obtained in the load phase. (Note that by [CL17], we can assume a broadcast channel while developing our protocol, and then use their compiler to remove the broadcast channel from our protocol.) In an honest execution, at the end of this broadcast step, each party would possess receipts from every pairwise $\mathcal{F}_{\mathsf{SyX}}$. On the other hand, corrupt parties may not broadcast some receipts, resulting in a setting where corrupt parties possess all receipts, but honest parties do not.

To maintain that $\mathcal{F}_{\mathsf{SyX}}$ instances remain in sync, we let the trigger conditions ask for all receipts (each individual $\mathcal{F}_{\mathsf{SyX}}$ instance can verify these load receipts using, e.g., digital signature

verification). This way, we ensure that any $\mathcal{F}_{\mathsf{SyX}}$ instance can be triggered only if all $\mathcal{F}_{\mathsf{SyX}}$ instances were loaded. Recall that by definition, $\mathcal{F}_{\mathsf{SyX}}$ outputs the trigger witness along with the output of f_2. This in turn ensures that if, say an $\mathcal{F}_{\mathsf{SyX}}$ instance between P_i and P_j was triggered by P_i, then P_j would obtain the load receipts which it can then use as part of trigger witnesses for other $\mathcal{F}_{\mathsf{SyX}}$ instances associated with P_j. Finally, because parties only receive additive secret shares of the output, to get the final output the adversary will need to trigger at least one $\mathcal{F}_{\mathsf{SyX}}$ instance associated with an honest party. The ideas outlined above ensures that that honest party (and consequently every honest party) will be able to continue triggering other $\mathcal{F}_{\mathsf{SyX}}$ instances associated with it, and obtain the final output. An additional detail to note is that in our constructions, we let the boolean predicates ϕ_1, ϕ_2 depend on *time*. This is required to ensure termination of our protocols (i.e., force a time limit on when the adversary must begin triggering the $\mathcal{F}_{\mathsf{SyX}}$ instances to obtain output). Therefore, in the terminology of [PST17], our functionality $\mathcal{F}_{\mathsf{SyX}}$ is clock-aware. The techniques we use to ensure termination may be reminiscent of techniques used in the design of broadcast protocols from point-to-point channels in the dishonest majority setting [DS83].

Complexity, preprocessing, assumptions, and implementation. The complexity of $\mathcal{F}_{\mathsf{SyX}}$ is the sum of the complexities of the functions f_1, f_2, and the predicates ϕ_1, ϕ_2. In our constructions, the complexity of each $\mathcal{F}_{\mathsf{SyX}}$ instance is $\mathcal{O}(n^2\lambda\ell_{\mathsf{out}})$ and is independent of the size of the function that is being computed.[2] With additional assumptions, specifically with a *non-interactive non-committing encryption* [Nie02] (alternatively, a *programmable random oracle*), the use of $\mathcal{F}_{\mathsf{SyX}}$ can be preprocessed in a network-independent manner to support any number of executions.[3] That is, a pair of parties can preprocess an instance of $\mathcal{F}_{\mathsf{SyX}}$ by loading it once, and then re-using it across multiple independent (possibly concurrent) executions of secure computation involving different sets of parties. This type of preprocessing is reminiscent of OT preprocessing [IPS08, KRS16]. Of course, to enable this type of preprocessing, we rely on a variant of $\mathcal{F}_{\mathsf{SyX}}$ which can be *triggered multiple times* (but loaded only once). In this case, the complexity of f_1 is $\mathcal{O}(\lambda)$, while the complexity of f_2 is $\mathcal{O}(\lambda)$ per trigger invocation, and the complexities of ϕ_1, ϕ_2 would be $\mathcal{O}(n^2\lambda)$ per trigger invocation for a protocol involving n parties. We emphasize that in the preprocessing setting, $\mathcal{F}_{\mathsf{SyX}}$ need not be triggered when the protocol participants behave honestly. Using ideas similar to [CGJ+17, SGK19], a follow-up work shows, among other things, how to implement a single $\mathcal{F}_{\mathsf{SyX}}$ instance using trusted execution environments (e.g., Intel SGX) and a bulletin board abstraction. Note that different $\mathcal{F}_{\mathsf{SyX}}$ instances can use different bulletin boards, and still be usable by our protocols.

Relationship to other primitives. [FGMO05] investigate a number of interesting primitives that are complete for secure computation with guaranteed output delivery for various parameter regimes. (See [FGMO05] for a discussion of complete primitives for secure computation with abort.) For $t < n/3$, they identify *secure channels* (with cardinality 2) as a complete primitive. For $t < n/2$, they identify two complete primitives *converge cast* and *oblivious cast*. Both these have cardinality 3. For $t < n$, they identify *universal black box* (UBB) as a complete primitive of cardinality n. Improving on this, [GIM+10] show *fair reconstruction of a non-malleable secret sharing scheme* as a complete primitive of cardinality n, whose *complexity* is independent of the

[2] The dependence on ℓ_{out} can be removed in the programmable random oracle model.

[3] Preprocessing for a bounded number of executions may be achieved by assuming only *receiver non-committing encryption* [CHK05].

function being computed (this was not the case for UBB). In addition, [GIM+10] investigate the power of primitives that guarantee fairness but are restricted in other ways (i.e., inapplicable to fairly computing arbitrary functions). For instance, they study *fair coin flipping* and *simultaneous broadcast*, and show that neither of them are complete for fair computation. Note that simultaneous broadcast was shown in [Kat07] to be complete for partial fairness [GK10]. Continuing, [GIM+10] show that (1) no primitive of size $\mathcal{O}(\log \lambda)$ is complete for fair computation (where λ is the security parameter), and (2) for every "short" m (when the adversary can run in time poly(m)), no m-bit primitive can be used to construct even a $m+1$-bit simultaneous broadcast. Note that none of the primitives discussed in [FGMO05, GIM+10, Kat07] are *reactive*.

Timed commitments [BN00] (and numerous related works such as [GJ02, GP03]) can be used to enable a fair exchange of digital signatures, fair auctions, and more under a somewhat non-standard security notion. Other works with similar security notions that consider fairness in secure computation include [Pin03, GMPY11, PST17] (see also numerous references therein). Another line of research investigates the use of physical/hardware assumptions to enforce fairness. For example, [LMPS04] relies on physical envelopes which provide some form of synchronizability. Recent work [CGJ+17, SGK19] (following [PST17]) has shown that fair secure computation is possible assuming the existence of trusted execution environments (alternatively, witness encryption [CGJ+17]) and a blockchain to which all parties have read/write access. In these works, the blockchain can be interpreted as a component that helps in synchronizing the TEEs. We note that the above works use blockchain and envelopes as a cardinality n primitive in their constructions. There are numerous works in the optimistic model (cf. [ASW97, ASW00] and several follow-up works) that minimize the use of a trusted third party to restore fairness when breached. Another line of research [ADMM14, ADMM16, BK14] investigates a non-standard notion of fair secure computation, called *secure computation with penalties* [BK14] where participants who do not obtain output are instead compensated monetarily (via cryptocurrency). [BK14] identifies a cardinality 2 primitive called *claim-or-refund* which is complete for this notion. (The presentation in [BK14] is strictly speaking not cardinality 2, but follow-up works clarify this.) That said, claim-or-refund shares many features with synchronizable exchange in that (1) both primitives operate in two phases (in the context of claim-or-refund, these phases are deposit and claim/refund), (2) the second phase is triggered via witnesses, (3) are clock-aware, and (4) both primitives can be preprocessed. Note that the claim-or-refund primitive locks up monetary funds which can be claimed by a designated receiver within a certain time by providing a witness to the trigger conditions, else the funds are refunded to the sender. Implicit in [ADMM14] is a primitive that releases monetary funds after a certain time has elapsed but during which time, the funds can be reversed by providing a trigger witness. This primitive is complete for fair coin tossing and fair lottery under the relaxed notion of secure computation with penalties [ADMM14].

Organization. In Section 2, we provide preliminaries including definitions of security, notations, etc. Following that, in Section 3 we describe synchronizable exchange and provide the ideal functionality definition for $\mathcal{F}_{\mathsf{SyX}}$. Then, in Section 4, we present our protocol for fair secure computation in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model. Finally, in Section 5, we show how $\mathcal{F}_{\mathsf{SyX}}$ can be preprocessed.

2 Preliminaries

2.1 Notation and definitions

For $n \in \mathbb{N}$, let $[n] = \{1, 2, \ldots, n\}$. Let $\lambda \in \mathbb{N}$ denote the security parameter. Symbols in with an arrow over them such as $\overrightarrow{a}$ denote vectors. By a_i we denote the i-th element of the vector $\overrightarrow{a}$. For a vector $\overrightarrow{a}$ of length $n \in \mathbb{N}$ and an index set $I \subseteq [n]$, we denote by $\overrightarrow{a}|_I$ the vector consisting of (ordered) elements from the set $\{a_i\}_{i \in I}$. By $\text{poly}(\cdot)$, we denote any function which is bounded by a polynomial in its argument. An algorithm $\mathcal{T}$ is said to be PPT if it is modeled as a probabilistic Turing machine that runs in time polynomial in λ. Informally, we say that a function is negligible, denoted by negl, if it vanishes faster than the inverse of any polynomial. If S is a set, then $x \xleftarrow{\$} S$ indicates the process of selecting x uniformly at random over S (which in particular assumes that S can be sampled efficiently). Similarly, $x \xleftarrow{\$} \mathcal{A}(\cdot)$ denotes the random variable that is the output of a randomized algorithm $\mathcal{A}$. Let $\mathcal{X}, \mathcal{Y}$ be two probability distributions over some set S. Their *statistical distance* is

$$\mathbf{SD}(\mathcal{X}, \mathcal{Y}) \stackrel{\text{def}}{=} \max_{T \subseteq S} \{|\Pr[\mathcal{X} \in T] - \Pr[\mathcal{Y} \in T]|\}$$

We say that $\mathcal{X}$ and $\mathcal{Y}$ are ϵ-close if $\mathbf{SD}(\mathcal{X}, \mathcal{Y}) \leq \epsilon$ and this is denoted by $\mathcal{X} \approx_\epsilon \mathcal{Y}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are identical if $\mathbf{SD}(\mathcal{X}, \mathcal{Y}) = 0$ and this is denoted by $\mathcal{X} \equiv \mathcal{Y}$.

2.2 Secure Computation

We recall most of the definitions regarding secure computation from [GHKL11] and [CL17]. We present them here for the sake of completeness and self-containedness. Consider the scenario of n parties $P_1, \ldots, P_n$ with private inputs $x_1, \ldots, x_n \in \mathcal{X}$[4]. We denote $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$.

2.2.1 Functionalities

A functionality f is a randomized process that maps n-tuples of inputs to n-tuples of outputs, that is, $f : \mathcal{X}^n \to \mathcal{Y}^n$[5]. We write $f = (f^1, \ldots, f^n)$ if we wish to emphasize the n outputs of f, but stress that if $f^1, \ldots, f^n$ are randomized, then the outputs of $f^1, \ldots, f^n$ are correlated random variables.

2.2.2 Adversaries

We consider security against *static t-threshold adversaries*, that is, adversaries that corrupt a set of at most t parties, where $0 \leq t < n$[6]. We assume the adversary to be malicious. That is, the corrupted parties may deviate arbitrarily from an assigned protocol.

2.2.3 Model

We assume the parties are connected via a fully connected point-to-point network; we refer to this model as the point-to-point model. We sometimes assume that the parties are given access to a

[4] Here we have assumed that the domains of the inputs of all parties is $\mathcal{X}$ for simplicity of notation. This can be easily adapted to consider setting where the domains are different.

[5] Here we have assumed that the domains of the outputs of all parties is $\mathcal{Y}$ for simplicity of notation. This can be easily adapted to consider setting where the domains are different.

[6] Note that when $t = n$, there is nothing to prove.

physical broadcast channel (defined in Section 2.7)[7] in addition to the point-to-point network; we refer to this model as the broadcast model. The communication lines between parties are assumed to be ideally authenticated and private (and thus an adversary cannot read or modify messages sent between two honest parties). Furthermore, the delivery of messages between honest parties is guaranteed. We sometimes assume the parties are connected via a fully pairwise connected network of oblivious transfer channels (defined in Section 2.6)[8] in addition to a fully connected point-to-point network; we refer to this model as the OT-network model. We sometimes assume that the parties are given access to a physical broadcast channel in addition to the complete pairwise oblivious transfer network and a fully connected point-to-point network; we refer to this model as the OT-broadcast model[9].

2.2.4 Protocol

An n-party protocol for computing a functionality f is a protocol running in polynomial time and satisfying the following functional requirement: if for every $i \in [n]$, party P_i begins with private input $x_i \in \mathcal{X}$, then the joint distribution of the outputs of the parties is statistically close to $(f^1(\overrightarrow{x}), \ldots, f^n(\overrightarrow{x}))$. We assume that the protocol is executed in a synchronous network, that is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their message for this round) followed by a *receive* phase (where they receive messages from other parties). The adversary, being malicious, is also *rushing* which means that it can see the messages the honest parties send in a round, before determining the messages that the corrupted parties send in that round.

2.2.5 Security with Guaranteed Output Delivery

The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted party exists) can do no more harm than if it were involved in the above-described ideal computation.

Execution in the ideal model. The parties are $P_1, \ldots, P_n$, and there is an adversary $\mathcal{A}$ who has corrupted at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. An ideal execution for the computation of f proceeds as follows:

- **Inputs:** $P_1, \ldots, P_n$ hold their private inputs $x_1, \ldots, x_n \in \mathcal{X}$; the adversary $\mathcal{A}$ receives an auxiliary input z.
- **Send inputs to trusted party:** The honest parties send their inputs to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may send any values of their choice. Denote the inputs sent to the trusted party by $x'_1, \ldots, x'_n$.

[7] This can also be viewed as working in the $\mathcal{F}_{\mathsf{bc}}$-hybrid model. See Section 2.3.
[8] This can also be viewed as working in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. See Section 2.3.
[9] This can also be viewed as working in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model. See Section 2.3.

- **Trusted party sends outputs:** If $x'_i \notin \mathcal{X}$ for any $i \in [n]$, the trusted party sets x'_i to some default input in $\mathcal{X}$. Then, the trusted party chooses r uniformly at random and sends $f^i(x'_1, \ldots, x'_n; r)$ to party P_i for every $i \in [n]$.
- **Outputs:** The honest parties output whatever was sent by the trusted party. The corrupted parties output nothing and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\text{IDEAL}^{\text{g.d.}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model described above.

Execution in the real model. We next consider the real model in which an n-party protocol π is executed by $P_1, \ldots, P_n$ (and there is no trusted party). In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and sends all messages on behalf of these parties, using an arbitrary polynomial-time strategy. The honest parties follow the instructions of π.

Let f be as above and let π be an n-party protocol computing f. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(x_1, \ldots, x_n, \lambda)$ be the random variable consisting of the view of the adversary and the output of the honest parties following an execution of π where P_i begins by holding x_i for every $i \in [n]$.

Security as emulation of an ideal execution in the real model. Having defined the ideal and real models, we can now define security of a protocol. Loosely speaking, the definition asserts that a secure protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated as follows.

Definition 1. *Protocol π is said to securely compute f with guaranteed output delivery if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\text{IDEAL}^{\text{g.d.}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*} \equiv \left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*}$$

We will sometimes relax security to statistical or computational definitions. A protocol is statistically secure if the random variables $\text{IDEAL}^{\text{g.d.}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ and $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)$ are statistically close, and computationally secure if they are computationally indistinguishable.

2.2.6 Security with Fairness

In this definition, the execution of the protocol can terminate in two possible ways: the first is when all parties receive their prescribed output (as in the case of guaranteed output delivery) and the second is when all parties (including the corrupted parties) abort without receiving output. The only change from the definition in Section 2.2.5 is with regard to the ideal model for computing f, which is now defined as follows:

Execution in the ideal model. The parties are $P_1, \ldots, P_n$, and there is an adversary $\mathcal{A}$ who has corrupted at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. An ideal execution for the computation of f proceeds as follows:

- **Inputs:** $P_1, \ldots, P_n$ hold their private inputs $x_1, \ldots, x_n \in \mathcal{X}$; the adversary $\mathcal{A}$ receives an auxiliary input z.
- **Send inputs to trusted party:** The honest parties send their inputs to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may send any values of their choice. In addition, there exists a special abort input. Denote the inputs sent to the trusted party by $x'_1, \ldots, x'_n$.
- **Trusted party sends outputs:** If $x'_i \notin \mathcal{X}$ for any $i \in [n]$, the trusted party sets x'_i to some default input in $\mathcal{X}$. If there exists an $i \in [n]$ such that $x'_i = \mathsf{abort}$, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party chooses r uniformly at random, computes $z_i = f^i(x'_1, \ldots, x'_n; r)$ for every $i \in [n]$ and sends z_i to P_i for every $i \in [n]$.
- **Outputs:** The honest parties output whatever was sent by the trusted party. The corrupted parties output nothing and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\textsc{Ideal}^{\mathsf{fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model described above.

Definition 2. *Protocol π is said to securely compute f with fairness if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\textsc{Ideal}^{\mathsf{fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*} \equiv \left\{\textsc{Real}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*}$$

We will sometimes relax security to statistical or computational definitions. A protocol is statistically secure if the random variables $\textsc{Ideal}^{\mathsf{fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ and $\textsc{Real}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)$ are statistically close, and computationally secure if they are computationally indistinguishable.

2.2.7 Security with Fairness and Identifiable Abort

This definition is identical to the one for fairness, except that if the adversary aborts the computation, all honest parties learn the identity of one of the corrupted parties. The only change from the definition in Section 2.2.5 is with regard to the ideal model for computing f, which is now defined as follows:

Execution in the ideal model. The parties are $P_1, \ldots, P_n$, and there is an adversary $\mathcal{A}$ who has corrupted at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. An ideal execution for the computation of f proceeds as follows:

- **Inputs:** $P_1, \ldots, P_n$ hold their private inputs $x_1, \ldots, x_n \in \mathcal{X}$; the adversary $\mathcal{A}$ receives an auxiliary input z.
- **Send inputs to trusted party:** The honest parties send their inputs to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may send any values of their choice. In addition, there exists a special abort input. In case the adversary instructs P_i to send abort, it chooses an index of a corrupted party $i^* \in \mathcal{I}$ and sets $x'_i = (\mathsf{abort}, i^*)$. Denote the inputs sent to the trusted party by $x'_1, \ldots, x'_n$.

- **Trusted party sends outputs:** If $x'_i \notin \mathcal{X}$ for any $i \in [n]$, the trusted party sets x'_i to some default input in $\mathcal{X}$. If there exists an $i \in [n]$ such that $x'_i = (\mathsf{abort}, i^*)$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party chooses r uniformly at random, computes $z_i = f^i(x'_1, \ldots, x'_n; r)$ for every $i \in [n]$ and sends z_i to P_i for every $i \in [n]$.

- **Outputs:** The honest parties output whatever was sent by the trusted party. The corrupted parties output nothing and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\textsc{Ideal}^{\mathsf{id\text{-}fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model described above.

Definition 3. *Protocol π is said to securely compute f with fairness and identifiable abort if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\textsc{Ideal}^{\mathsf{id\text{-}fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*} \equiv \left\{\textsc{Real}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*}$$

We will sometimes relax security to statistical or computational definitions. A protocol is statistically secure if the random variables $\textsc{Ideal}^{\mathsf{id\text{-}fair}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ and $\textsc{Real}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)$ are statistically close, and computationally secure if they are computationally indistinguishable.

2.2.8 Security with Abort

This definition is the standard one for secure computation [Gol04] in that it allows *early abort*; that is, the adversary may receive its own output even though the honest party does not. However, if one honest party receives output, then so do all honest parties. Thus, this is the notion of *unanimous abort*. The only change from the definition in Section 2.2.5 is with regard to the ideal model for computing f, which is now defined as follows:

Execution in the ideal model. The parties are $P_1, \ldots, P_n$, and there is an adversary $\mathcal{A}$ who has corrupted at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. An ideal execution for the computation of f proceeds as follows:

- **Inputs:** $P_1, \ldots, P_n$ hold their private inputs $x_1, \ldots, x_n \in \mathcal{X}$; the adversary $\mathcal{A}$ receives an auxiliary input z.

- **Send inputs to trusted party:** The honest parties send their inputs to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may send any values of their choice. In addition, there exists a special abort input. Denote the inputs sent to the trusted party by $x'_1, \ldots, x'_n$.

- **Trusted party sends outputs to the adversary:** If $x'_i \notin \mathcal{X}$ for any $i \in [n]$, the trusted party sets x'_i to some default input in $\mathcal{X}$. If there exists an $i \in [n]$ such that $x'_i = \mathsf{abort}$, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party chooses r uniformly at random, computes $z_i = f^i(x'_1, \ldots, x'_n; r)$ for every $i \in [n]$ and sends z_i to P_i for every $i \in \mathcal{I}$ (that is, to the adversary $\mathcal{A}$).

- **Trusted party sends outputs to the honest parties:** After receiving its output (as described above), the adversary either sends abort or $\mathsf{continue}$ to the trusted party. In the former case the trusted party sends $\perp$ to the honest parties, and in the latter case the trusted party send z_j to P_j for every $j \in [n] \setminus \mathcal{I}$.

- **Outputs:** The honest parties output whatever was sent by the trusted party. The corrupted parties output nothing and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x},\lambda)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model described above.

Definition 4. *Protocol π is said to securely compute f with abort if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x},\lambda)\right\}_{\overrightarrow{x}\in\mathcal{X}^n,z\in\{0,1\}^*} \equiv \left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x},\lambda)\right\}_{\overrightarrow{x}\in\mathcal{X}^n,z\in\{0,1\}^*}$$

We will sometimes relax security to statistical or computational definitions. A protocol is statistically secure if the random variables $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x},\lambda)$ and $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x},\lambda)$ are statistically close, and computationally secure if they are computationally indistinguishable.

2.2.9 Security with Identifiable Abort

This definition is identical to the one for abort, except that if the adversary aborts the computation, all honest parties learn the identity of one of the corrupted parties. The only change from the definition in Section 2.2.5 is with regard to the ideal model for computing f, which is now defined as follows:

Execution in the ideal model. The parties are $P_1,\ldots,P_n$, and there is an adversary $\mathcal{A}$ who has corrupted at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. An ideal execution for the computation of f proceeds as follows:

- **Inputs:** $P_1,\ldots,P_n$ hold their private inputs $x_1,\ldots,x_n \in \mathcal{X}$; the adversary $\mathcal{A}$ receives an auxiliary input z.

- **Send inputs to trusted party:** The honest parties send their inputs to the trusted party. The corrupted parties controlled by $\mathcal{A}$ may send any values of their choice. In addition, there exists a special abort input. In case the adversary instructs P_i to send abort, it chooses an index of a corrupted party $i^* \in \mathcal{I}$ and sets $x'_i = (\mathsf{abort}, i^*)$. Denote the inputs sent to the trusted party by $x'_1,\ldots,x'_n$.

- **Trusted party sends outputs to the adversary:** If $x'_i \notin \mathcal{X}$ for any $i \in [n]$, the trusted party sets x'_i to some default input in $\mathcal{X}$. If there exists an $i \in [n]$ such that $x'_i = (\mathsf{abort}, i^*)$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party chooses r uniformly at random, computes $z_i = f^i(x'_1,\ldots,x'_n;r)$ for every $i \in [n]$ and sends z_i to P_i for every $i \in \mathcal{I}$ (that is, to the adversary $\mathcal{A}$).

- **Trusted party sends outputs to the honest parties:** After receiving its output (as described above), the adversary either sends (abort, i^*) where $i^* \in \mathcal{I}$, or $\mathsf{continue}$ to the trusted party. In the former case the trusted party sends $(\perp, i^*)$ to the honest parties, and in the latter case the trusted party send z_j to P_j for every $j \in [n] \setminus \mathcal{I}$.

- **Outputs:** The honest parties output whatever was sent by the trusted party. The corrupted parties output nothing and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ be the random variable consisting of the output of the adversary and the output of the honest parties following an execution in the ideal model described above.

Definition 5. *Protocol π is said to securely compute f with identifiable abort if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*} \equiv \left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)\right\}_{\overrightarrow{x} \in \mathcal{X}^n, z \in \{0,1\}^*}$$

We will sometimes relax security to statistical or computational definitions. A protocol is statistically secure if the random variables $\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathcal{S}(z)}(\overrightarrow{x}, \lambda)$ and $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)$ are statistically close, and computationally secure if they are computationally indistinguishable.

2.3 The Hybrid Model

We recall the definition of the hybrid model from [GHKL11] and [CL17]. The hybrid model combines both the real and ideal worlds. Specifically, an execution of a protocol π in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, involves parties sending normal messages to each other (as in the real model) and, in addition, having access to a trusted party computing $\mathcal{G}$. The parties communicate with this trusted party in exactly the same way as in the ideal models described above; the question of which ideal model is taken (that with or without abort) must be specified. In this paper, we always consider a hybrid model where the functionality $\mathcal{G}$ is computed according to the ideal model *with abort*. In all our protocols in the $\mathcal{G}$-hybrid model there will only be *sequential* calls to $\mathcal{G}$, that is, there is at most a single call to $\mathcal{G}$ per round, and no other messages are sent during any round in which $\mathcal{G}$ is called. This is especially important for reactive functionalities, where the calls to f are carried out in phases, and a new invocation of f cannot take place before all the phases of the previous invocation complete.

Let $\mathsf{type} \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let $\mathcal{G}$ be a functionality and let π be an n-party protocol for computing some functionality f, where π includes real messages between the parties as well as calls to $\mathcal{G}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. $\mathcal{A}$ corrupts at most t parties, where $0 \leq t < n$. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. Let $\text{HYBRID}^{\mathcal{G},\mathsf{type}}_{\pi,\mathcal{I},\mathcal{A}(z)}(\overrightarrow{x}, \lambda)$ be the random variable consisting of the view of the adversary and the output of the honest parties, following an execution of π with ideal calls to a trusted party computing $\mathcal{G}$ according to the ideal model "type" where P_i begins by holding x_i for every $i \in [n]$. Security in the model "type" can be defined via natural modifications of Definitions 1, 2, 3, 4 and 5. We call this the $(\mathcal{G}, \mathsf{type})$-hybrid model.

The hybrid model gives a powerful tool for proving the security of protocols. Specifically, we may design a real-world protocol for securely computing some functionality f by first constructing

a protocol for computing f in the $\mathcal{G}$-hybrid model. Letting π denote the protocol thus constructed (in the $\mathcal{G}$-hybrid model), we denote by π^ρ the real-world protocol in which calls to $\mathcal{G}$ are replaced by sequential execution of a real-world protocol ρ that computes $\mathcal{G}$ in the ideal model "type". "Sequential" here implies that only one execution of ρ is carried out at any time, and no other π-protocol messages are sent during the execution of ρ. The results of [Can00] then imply that if π securely computes f in the $(\mathcal{G}, \mathsf{type})$-hybrid model, and ρ securely computes $\mathcal{G}$, then the composed protocol π^ρ securely computes f (in the real world). For completeness, we state this result formally as we will use it in this work.

Lemma 1. *Let* $\mathsf{type}_1, \mathsf{type}_2 \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. *Let* $\mathcal{G}$ *be an* n*-party functionality. Let* ρ *be a protocol that securely computes* $\mathcal{G}$ *with* type_1*, and let* π *be a protocol that securely computes* f *with* type_2 *in the* $(\mathcal{G}, \mathsf{type}_1)$*-hybrid model. Then protocol* π^ρ *securely computes* f *with* type_2 *in the real model.*

Sometimes, while working in a hybrid model, say the $(\mathcal{G}, \mathsf{type})$-hybrid model, we will suppress type and simply state that we are working in the $\mathcal{G}$-hybrid model. This is because type is implied by the context, $\mathcal{G}$. For instance, unless specified otherwise:

- When $\mathcal{G} = \mathcal{F}_{\mathsf{bc}}$[10], $\mathsf{type} = \mathsf{g.d.}$.
- When $\mathcal{G} = \mathcal{F}_{\mathsf{OT}}$[11], $\mathsf{type} = \mathsf{abort}$.
- When $\mathcal{G} = \mathcal{F}_{\mathsf{2PC}}$[12], $\mathsf{type} = \mathsf{abort}$.
- When $\mathcal{G} = \mathcal{F}_{\mathsf{MPC}}$[13], $\mathsf{type} = \mathsf{abort}$.
- When $\mathcal{G} = \mathcal{F}_{\mathsf{SyX}}$[14], $\mathsf{type} = \mathsf{g.d.}$.

When working in a hybrid model that uses multiple ideal functionalities, $\mathcal{G}_1, \ldots, \mathcal{G}_k$ with associated types $\mathsf{type}_1, \ldots, \mathsf{type}_k$ for some $k \in \mathbb{N}$, we call it the $(\mathcal{G}_1, \mathsf{type}_1, \ldots, \mathcal{G}_k, \mathsf{type}_k)$-hybrid model. Furthermore, we will suppress type_j when type_j is implied by the context, $\mathcal{G}_j$ for $j \in [k]$.

2.4 Fairness versus Guaranteed Output Delivery

We recall here some of the results from [CL17].

Lemma 2. *[CL17] Consider* n *parties* $P_1, \ldots, P_n$ *in a model without a broadcast channel. Then, there exists a functionality* $f : \mathcal{X}^n \to \mathcal{Y}^n$ *such that* f *cannot be securely computed with guaranteed output delivery in the presence of* t*-threshold adversaries for* $n/3 \leq t < n$.

Lemma 3. *[CL17] Consider* n *parties* $P_1, \ldots, P_n$ *in a model with a broadcast channel. Then, assuming the existence of one-way functions, for any functionality* $f : \mathcal{X}^n \to \mathcal{Y}^n$*, if there exists a protocol* π *which securely computes* f *with fairness, then there exists a protocol* π' *which securely computes* f *with guaranteed output delivery.*

[10] See Section 2.7.
[11] See Section 2.6.
[12] See Section 3.
[13] See Section 2.6.
[14] See Section 3.

Preliminaries: $x_0, x_1 \in \{0,1\}^m$; $b \in \{0,1\}$. The functionality proceeds as follows:

- Upon receiving inputs (x_0, x_1) from the sender P_1 and b from the receiver P_2, send $\perp$ to P_1 and x_b to P_2.

Figure 1: The ideal functionality $\mathcal{F}_{\mathsf{OT}}$.

Preliminaries: $x_1, \ldots, x_n \in \{0,1\}^*$; $f_1, \ldots, f_n$ is an n-input, n-output functionalities. The functionality proceeds as follows:

- Upon receiving inputs (x_i, f_i) from P_i for all $i \in [n]$, check if $f = f_i$ for all $i \in [n]$. If not, abort. Else, send $f^i(x_1, \ldots, x_n)$ to P_i for all $i \in [n]$.

Figure 2: The ideal functionality $\mathcal{F}_{\mathsf{MPC}}$.

Lemma 4. *[CL17] Consider n parties $P_1, \ldots, P_n$ in a model with a broadcast channel. Then, assuming the existence of one-way functions, for any functionality $f : \mathcal{X}^n \to \mathcal{Y}^n$, if there exists a protocol π which securely computes f with fairness, then there exists a protocol π' which securely computes f with fairness and does not make use of the broadcast channel.*

2.5 Computing with an Honest Majority

We recall here some of the known results regarding feasibility of information-theoretic multiparty computation in the presence of an honest majority.

Lemma 5. *[GMW87] Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with guaranteed output delivery in the presence of t-threshold adversaries for any $0 \leq t < n/3$.*

Lemma 6. *[FGMvR02] Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the presence of t-threshold adversaries for any $0 \leq t < n/2$.*

Lemma 7. *[GMW87, RB89] Consider n parties $P_1, \ldots, P_n$ in the* broadcast *model. Then, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with guaranteed output delivery in the presence of t-threshold adversaries for any $0 \leq t < n/2$.*

2.6 Oblivious Transfer

In this work, oblivious transfer, or OT, refers to 1-out-of-2 oblivious transfer defined as in Figure 1. We note that in the definition of $\mathcal{F}_{\mathsf{OT}}$, one party, namely P_1, is seen as the sender, while the other, namely P_2, is seen as the receiver. However, from [WW06], OT is symmetric, which implies that the roles of the sender and the receiver can be reversed. Thus, if two parties P_1 and P_2 have access to the ideal functionality $\mathcal{F}_{\mathsf{OT}}$, they can perform 1-out-of-2 oblivious transfer with either party as a sender and the other as the receiver. It is known that OT is complete for secure multiparty computation with abort. We state this result formally below.

Preliminaries: $x \in \{0,1\}^*$. The functionality proceeds as follows:

- Upon receiving the input x from the sender P_1, send x to all parties $P_1, \ldots, P_n$.

Figure 3: The ideal functionality $\mathcal{F}_{\mathsf{bc}}$.

Lemma 8. *[Kil88, GV87, IPS08] Consider n parties $P_1, \ldots, P_n$ in the OT-network model. Then, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with abort in the presence of t-threshold adversaries for any $0 \leq t < n$.*

2.7 Broadcast

Broadcast is defined as in Figure 3. We recall that the ideal functionality for broadcast, namely $\mathcal{F}_{\mathsf{bc}}$, can be securely computed with guaranteed output delivery in the presence of t-threshold adversaries if and only if $0 \leq t < n/3$ [PSL80, LSP82]. Furthermore, $\mathcal{F}_{\mathsf{bc}}$ can be securely computed with fairness in the presence of t-threshold adversaries for any $0 \leq t < n$ [FGH+02]. Furthermore, these results hold irrespective of the model we are working in so long as we do not have explicit access to $\mathcal{F}_{\mathsf{bc}}$.

2.8 Honest-Binding Commitment Schemes

We recall the notion of honest-binding commitments from [GKKZ11]. Commitment schemes are a standard cryptographic tool. Roughly, a commitment scheme allows a sender S to generate a commitment c to a message m in such a way that (1) the sender can later open the commitment to the original value m (correctness); (2) the sender cannot generate a commitment that can be opened to two different values (binding); and (3) the commitment reveals nothing about the sender's value m until it is opened (hiding). For our application, we need a variant of standard commitments that guarantees binding when the sender is honest but ensures that binding can be violated if the sender is dishonest. (In the latter case, we need some additional properties as well; these will become clear in what follows.) Looking ahead, we will use such commitment schemes to enable a simulator in security proofs to generate a commitment dishonestly. This will give the simulator the flexibility to break binding and open the commitment to any desired message (if needed), while also being able to ensure binding (when desired) by claiming that it generated the commitment honestly.

We consider only non-interactive commitment schemes. For simplicity, we define our schemes in such a way that the decommitment information consists of the sender's random coins ω that it used when generating the commitment.

Definition 6. *A (non-interactive) commitment scheme for message space $\mathcal{M}_\lambda$ is a pair of PPT algorithms $(\mathsf{Com}, \mathsf{Open})$ such that for all $\lambda \in \mathbb{N}$, all messages $m \in \mathcal{M}_\lambda$, and all random coins ω it holds that*

$$\mathsf{Open}(\mathsf{Com}(1^\lambda, m; \omega), \omega, m) = 1$$

A commitment scheme for message space $\mathcal{M}_\lambda$ is honest-binding if it satisfies the following:

Binding (for an honest sender). *For all PPT algorithms $\mathcal{A}$ (that maintain state throughout their execution), the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{l} m \xleftarrow{\$} \mathcal{A}(1^k); \omega \xleftarrow{\$} \{0,1\}^* \\ \quad c = \mathsf{Com}(1^\lambda, m; \omega) \\ \quad (m', \omega') \xleftarrow{\$} \mathcal{A}(c, \omega) \end{array} \;:\; \mathsf{Open}(c, m', \omega') = 1 \;\wedge\; m \neq m' \right]$$

Equivocation. *There is a pair of algorithms $\left(\widetilde{\mathsf{Com}}, \widetilde{\mathsf{Open}}\right)$ such that for all PPT algorithms $\mathcal{A}$ (that maintain state throughout their execution), the following quantity is negligible in λ:*

$$\left| \begin{array}{r} \Pr\left[m \xleftarrow{\$} \mathcal{A}(1^\lambda); \omega \xleftarrow{\$} \{0,1\}^*; c = \mathsf{Com}(1^\lambda, m; \omega) : \mathcal{A}(1^\lambda, c, \omega) = 1\right] \\ - \Pr\left[(c, \mathsf{state}) \xleftarrow{\$} \widetilde{\mathsf{Com}}(1^\lambda), m \xleftarrow{\$} \mathcal{A}(1^\lambda); \omega \xleftarrow{\$} \widetilde{\mathsf{Open}}(\mathsf{state}, m) : \mathcal{A}(1^\lambda, c, \omega) = 1\right] \end{array} \right|$$

Equivocation implies the standard hiding property, namely, that for all PPT algorithms $\mathcal{A}$ (that maintain state throughout their execution) the quantity is negligible in λ:

$$\left| \Pr\left[(m_0, m_1) \xleftarrow{\$} \mathcal{A}(1^\lambda); b \xleftarrow{\$} \{0,1\}; c \xleftarrow{\$} \mathsf{Com}(1^\lambda, m_b) : \mathcal{A}(c) = b\right] \right|$$

We also observe that if (c, ω) are generated by $\left(\widetilde{\mathsf{Com}}, \widetilde{\mathsf{Open}}\right)$ for some message m as in the definition above, then binding still holds: namely, no PPT adversary given (m, c, ω) can find (m', ω') with $m' \neq m$ such that $\mathsf{Open}(c, m', \omega') = 1$.

We will sometimes use the notation $(c, \omega) \xleftarrow{\$} \mathsf{Com}(m)$ to mean $c = \mathsf{Com}(1^\lambda, m; \omega)$, suppressing λ when it is clear from the context and having the committing algorithm Com return the commitment and the decommitment information or opening. [GKKZ11] provides constructions of honest-binding commitments for bits assuming the existence of one-way functions.

2.9 Digital Signatures

Definition 7. *A (digital) signature scheme for message space $\mathcal{M}_\lambda$ is triple of PPT algorithms $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ such that for all $\lambda \in \mathbb{N}$ and all messages $m \in \mathcal{M}_\lambda$,*

$$\Pr\left[\begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathcal{V}.\mathsf{Gen}(1^\lambda) \\ \sigma \xleftarrow{\$} \mathcal{V}.\mathsf{Sign}(m; \mathsf{sk}) \end{array} : \mathcal{V}.\mathsf{Verify}(\sigma, m; \mathsf{vk}) = 1\right] = 1$$

A signature scheme for message space $\mathcal{M}_\lambda$ is existentially unforgeable if for any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr\left[\begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathcal{V}.\mathsf{Gen}(1^\lambda) \\ \mathcal{Q} = \emptyset \\ \left\{\begin{array}{l} m_i \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathcal{Q}) \\ \sigma_i \xleftarrow{\$} \mathcal{V}.\mathsf{Sign}(m_i; \mathsf{sk}) \\ \mathcal{Q} = \mathcal{Q} \cup \{(m_i, \sigma_i)\} \end{array}\right\}_i \end{array} : \mathcal{V}.\mathsf{Verify}(\sigma, m; \mathsf{vk}) = 1 \;\wedge\; (m, \sigma) \notin \mathcal{Q}\right]$$

[Rom90] provides constructions of existentially unforgeable signatures assuming the existence of one-way functions.

2.10 Receiver Non-Committing Encryption

We recall the notion of receiver non-committing encryption from [CHK05]. On a high level, a receiver non-committing encryption scheme is one in which a simulator can generate a single "fake ciphertext" and later "open" this ciphertext (by showing an appropriate secret key) as any given message. These "fake ciphertexts" should be indistinguishable from real ciphertexts, even when an adversary is given access to a decryption oracle before the fake ciphertext is known.

Formally, a receiver non-committing encryption scheme $\mathcal{E}$ consists of the following five PPT algorithms:

- $\mathcal{E}.\mathsf{Gen}(1^\lambda)$: Given the security parameter, λ, the key generation algorithm outputs a key-pair and some auxiliary information. This is denoted by: $(\mathsf{pk}, \mathsf{sk}, z) \xleftarrow{\$} \mathcal{E}.\mathsf{Gen}(1^\lambda)$. The public key pk defines a message space $\mathcal{M}_\lambda$.
- $\mathcal{E}.\mathsf{Enc}(m; \mathsf{pk})$: Given the public key pk and a message $m \in \mathcal{M}_\lambda$, the encryption algorithm returns a ciphertext $\mathsf{ct} \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(m; \mathsf{pk})$.
- $\mathcal{E}.\mathsf{Dec}(\mathsf{ct}; \mathsf{sk})$: Given the secret key sk and a ciphertext ct, the decryption algorithm returns a message $m \xleftarrow{\$} \mathcal{E}.\mathsf{Dec}(\mathsf{ct}; \mathsf{sk})$, where $m \in \mathcal{M}_\lambda \cup \{\perp\}$.
- $\mathcal{E}.\widetilde{\mathsf{Enc}}(\mathsf{pk}, \mathsf{sk}, z)$: Given the triple $(\mathsf{pk}, \mathsf{sk}, z)$ output by $\mathcal{E}.\mathsf{Gen}$, the fake encryption algorithm outputs a "fake ciphertext" $\widetilde{\mathsf{ct}} \xleftarrow{\$} \mathcal{E}.\widetilde{\mathsf{Enc}}(\mathsf{pk}, \mathsf{sk}, z)$.
- $\mathcal{E}.\widetilde{\mathsf{Dec}}(\mathsf{pk}, \mathsf{sk}, z, \widetilde{\mathsf{ct}}, m)$: Given the triple $(\mathsf{pk}, \mathsf{sk}, z)$ output by $\mathcal{E}.\mathsf{Gen}$, a "fake ciphertext" $\widetilde{\mathsf{ct}}$ output by $\mathcal{E}.\widetilde{\mathsf{Enc}}$ and a message $m \in \mathcal{M}_\lambda$, the "fake decryption" algorithm outputs a "fake secret key" $\widetilde{\mathsf{sk}} \xleftarrow{\$} \mathcal{E}.\widetilde{\mathsf{Dec}}(\mathsf{pk}, \mathsf{sk}, z, \widetilde{\mathsf{ct}}, m)$. (Intuitively, $\widetilde{\mathsf{sk}}$ is a valid-looking secret key for which $\widetilde{\mathsf{ct}}$ decrypts to m.)

We make the standard correctness requirement; namely, for any $(\mathsf{pk}, \mathsf{sk}, z)$ output by $\mathcal{E}.\mathsf{Gen}$ and any $m \in \mathcal{M}_\lambda$, we have $\mathcal{E}.\mathsf{Dec}(\mathcal{E}.\mathsf{Enc}(m; \mathsf{pk}); \mathsf{sk}) = m$. Our definition of security requires, informally, that for any message m an adversary cannot distinguish whether it has been given a "real" encryption of m along with a "real" secret key, or a "fake" ciphertext along with a "fake" secret key under which the ciphertext decrypts to m. This should hold even when the adversary has non-adaptive access to a decryption oracle. We now give the formal definition.

Definition 8. *Let $\mathcal{E}$ be a receiver non-committing encryption scheme. We say that $\mathcal{E}$ is secure if the advantage of any PPT algorithm $\mathcal{A}$ in the game below is negligible in λ:*

1. *The key generation algorithm $\mathcal{E}.\mathsf{Gen}(1^\lambda)$ is run to get $(\mathsf{pk}, \mathsf{sk}, z)$.*
2. *The algorithm $\mathcal{A}$ is given 1^λ and pk as input, and is also given access to a decryption oracle $\mathcal{E}.\mathsf{Dec}(\cdot; \mathsf{sk})$. It then outputs a challenge message $m \in \mathcal{M}_\lambda$.*
3. *A bit b is chosen at random. If $b = 1$ then a ciphertext $\mathsf{ct} \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(m; \mathsf{pk})$ is computed, and $\mathcal{A}$ receives $(\mathsf{ct}, \mathsf{sk})$. Otherwise, a "fake" ciphertext $\widetilde{\mathsf{ct}} \xleftarrow{\$} \mathcal{E}.\widetilde{\mathsf{Enc}}(\mathsf{pk}, \mathsf{sk}, z)$ and a "fake" secret key $\widetilde{\mathsf{sk}} \xleftarrow{\$} \mathcal{E}.\widetilde{\mathsf{Dec}}(\mathsf{pk}, \mathsf{sk}, z, \widetilde{\mathsf{ct}}, m)$ are computed, and $\mathcal{A}$ receives $(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{sk}})$. (After this point, $\mathcal{A}$ can no longer query its decryption oracle.) $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.*

The advantage of $\mathcal{A}$ is defined as $2 \cdot \left|\Pr[b = b'] - \frac{1}{2}\right|$.

2.11 Non-interactive Non-Committing Encryption

We recall the notion of non-interactive non-committing encryption from [Nie02]. We do so in two ways. The first way of looking at non-interactive non-committing encryption is that it is the same as receiver non-committing encryption, except that it can equivocate multiple ciphertexts as opposed to one. On a high level, a non-interactive non-committing encryption scheme is one in which a simulator can generate multiple "fake ciphertexts" and later "open" them (by showing an appropriate secret key) as any given message vector. We first note that the receiver non-committing encryption scheme of [CHK05] can be extended, as noted by them, to support equivocation of any bounded number of ciphertexts. However, the size of the key of the scheme would grow linearly with the number of outstanding ciphertexts. Such schemes can be constructed based on standard assumptions such as the quadratic residuosity assumption. If no bound on the number of outstanding texts is known *apriori*, then as noted in [Nie02], constructing such schemes is impossible in the standard model. The other way of looking at non-interactive non-committing encryption is that it is a realization of the ideal functionality for public key encryption, namely, $\mathcal{F}_{\mathsf{PKE}}$. We refer the reader to [CKN03, CHK05] for further details.

For the sake of completeness and ease of later presentation, we recall the non-interactive non-committing encryption scheme of [Nie02] in the random-oracle model. Let $\mathcal{F} = (\mathcal{K}, F)$ be a collection of trapdoor permutations, where $\mathcal{K}$ denotes an index set and $F = \{f_k\}_{k \in \mathcal{K}}$ is a set of permutations with efficiently samplable domains. For every $k \in \mathcal{K}$, we denote by t_k the trapdoor associated with k which enables inversion of f_k. We assume the existence of a generation algorithm $\mathcal{G}$ which on input the security parameter λ outputs a key-trapdoor pair (k, t_k) uniformly at random. Let $H : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ be a random oracle (instantiated by an appropriate hash function). The non-interactive non-committing encryption scheme $\mathcal{E}$ consists of the following algorithms:

- $\mathcal{E}.\mathsf{Gen}(1^\lambda)$: Given the security parameter, λ, the key generation algorithm obtains (k, t_k) by executing $\mathcal{G}$ with the security parameter λ as input. It then outputs the public and private keys $\mathsf{pk} = (k, f_k, H)$ and $\mathsf{sk} = t_k$. The message space is defined to be $\mathcal{M}_\lambda = \{0,1\}^{\ell(\lambda)}$.
- $\mathcal{E}.\mathsf{Enc}(m; \mathsf{pk})$: Given the public key pk and a message $m \in \mathcal{M}_\lambda$, the encryption algorithm samples x from the domain of f_k and returns a ciphertext $\mathsf{ct} = (f_k(x), H(x) \oplus m)$.
- $\mathcal{E}.\mathsf{Dec}(\mathsf{ct}; \mathsf{sk})$: Given the secret key sk and a ciphertext $\mathsf{ct} = (\mathsf{ct}^1, \mathsf{ct}^2)$, the decryption algorithm computes x by inverting ct^1 using t_k and returns the message $m = H(x) \oplus \mathsf{ct}^2$.

We refer the reader to [Nie02] for a complete proof that the scheme defined above is a non-interactive non-committing encryption scheme. The sketch the proof here. The scheme is clearly non-interactive. We now need to design a simulator $\mathcal{S}$ which can generate multiple "fake ciphertexts" and later "open" them to an arbitrary sequence of messages. Note that this is easy to do. To generate n "fake ciphertexts", $\mathcal{S}$ samples $x_1, \ldots, x_n$ independently at random from the domain of f_k. It then samples $y_1, \ldots, y_n \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$. The m ciphertexts are defined to be $\{\mathsf{ct}_i\}_{i \in [n]}$ where $\mathsf{ct}_i = (f_k(x_i), y_i)$. Then, in order to open the n ciphertexts to a message vector $\overrightarrow{m} = (m_1, \ldots, m_n) \in \mathcal{M}_\lambda^n$, $\mathcal{S}$ would program the random oracle H such that $H(x_i) = m_i \oplus y_i$. Note that this ensures that the "fake ciphertexts" do in fact "open" to the message vector $\overrightarrow{m}$. We also stress, as this will be required for us later, that the simulator need not know n in advance, that is, it can produce any (polynomially bounded) number of "fake ciphertexts" and later "open" them as required. This is also precisely the difference from receiver non-committing encryption as described earlier which necessitates the use of random oracles as noted in [Nie02].

Preliminaries: $x_1, x_2 \in \{0,1\}^*$; f_1, f_2 are 2-input, 2-output functionalities. The functionality proceeds as follows:

- Upon receiving inputs (x_1, f_1) from P_1 and (x_2, f_2) from P_2, check if $f = f_1 = f_2$. If not, abort. Else, send $f^1(x_1, x_2)$ to P_1 and $f^2(x_1, x_2)$ to P_2.

Figure 4: The ideal functionality $\mathcal{F}_{\mathsf{2PC}}$.

3 Synchronizable Exchange

We are interested in solving the problem of securely computing functionalities with fairness, most commonly referred to as fair secure computation. We begin with the case of two parties. It is known that fair two-party secure computation is impossible in the standard model as well as in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model [Cle86]. This result generalizes to the setting of n parties that are trying to compute in the presence of a t-threshold adversary for any $n/2 \leq t < n$.

One could define the ideal functionality, $\mathcal{F}_{\mathsf{2PC}}$ as in Figure 4. Clearly, any 2-party functionality can be securely computed with fairness in the $(\mathcal{F}_{\mathsf{2PC}}, \mathsf{fair})$-hybrid model. One can then ask the following question in the context of $n > 2$ parties:

Consider n parties $P_1, \ldots, P_n$ in the OT-$\mathsf{broadcast}$ *model. Does there exist a protocol that securely computes $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the $(\mathcal{F}_{\mathsf{2PC}}, \mathsf{fair})$-hybrid model?*

We are interested in security in the presence of a t-threshold adversary for any $n/2 \leq t < n$. While we do not know the answer to this question, it seems that the answer to this question would be negative. The intuition for this is that the various invocations of the ideal functionality $\mathcal{F}_{\mathsf{2PC}}$ cannot "synchronize" with each other and thus we would run into issues similar to the those highlighted by the impossibility result in [Cle86], namely, some party/parties obtain information about the output of the protocol before the others and if these parties were corrupt, they may choose to abort the protocol without the honest parties receiving their outputs.

Equipped with this intuition, we propose the primitive, $\mathcal{F}_{\mathsf{SyX}}$, which we call "synchronizable exchange". We define the ideal functionality for $\mathcal{F}_{\mathsf{SyX}}$ in Figure 5. We associate the type $\mathsf{g.d.}$ to the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ when working in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model. When interacting with this functionality, parties first submit their inputs to $\mathcal{F}_{\mathsf{SyX}}$ which then gives them a "receipt" acknowledging the end of the input submission phase. Following this, the functionality simply waits for a trigger from one of the parties. Once the trigger is received (we specify conditions for the validity of a trigger), the functionality will deliver the outputs according to the specification. In the formal description, we allow parties P_1, P_2 to submit two functions f_1, f_2 and two Boolean predicates (that check validity of a trigger value) ϕ_1, ϕ_2 along with their inputs x_1, x_2. $\mathcal{F}_{\mathsf{SyX}}$ then computes $f_1(x_1, x_2)$ and sends this value as a "receipt" that the input submission phase has ended. The actual output of the computation is $f_2(x_1, x_2)$ and this will be provided to the parties at the end of the trigger phase. Note that the trigger phase can be activated by either party P_i. However, P_i would need to provide a "witness" w that satisfies ϕ_i. Also, more generally, the output of the computation $f_2(x_1, x_2, w)$ can additionally depend on the witness w that was provided as well.

Note that $\mathcal{F}_{\mathsf{SyX}}$ is at least as strong as $\mathcal{F}_{\mathsf{2PC}}$. In order to realize $\mathcal{F}_{\mathsf{2PC}}$ in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model, we set $f_1 = \varepsilon$ (the empty string), $f_2 = f$, $\phi_1 = \phi_2 = 1$. The hope in defining this reactive functionality, however, is to achieve synchronization of multiple invocations of the ideal functionality

Preliminaries: $x_1, x_2 \in \{0,1\}^*$; f_1, f_2 are 2-output functions; ϕ_1, ϕ_2 are boolean predicates. The functionality proceeds as follows:

- **Load phase.** Upon receiving inputs $(x_1, f = (f_1, f_2, \phi_1, \phi_2))$ from P_1 and (x_2, f') from P_2, check if $f = f'$. If not, abort. Else, compute $f_1(x_1, x_2)$. If $f_1(x_1, x_2) = \perp$[a], abort. Else, send $f_1^i(x_1, x_2)$ to P_i for $i \in \{1, 2\}$, and go to next phase.
- **Trigger phase.** Upon receiving input w from party P_i, check if $\phi_i(w) = 1$. If yes, then send $(w, f_2^j(x_1, x_2, w))$ to both parties P_j for $j \in \{1, 2\}$.

[a]We crucially require that $\perp$ is a special symbol different from the empty string. We use $\perp$ as a means of signalling that the load phase of $\mathcal{F}_{\mathsf{SyX}}$ did not complete successfully. We will however allow parties to attempt to invoke the load phase of the functionality at a later time. As we proceed, we will also have our functionality be clock-aware and thus only accept invocations to the load phase until a certain point in time. After the load phase times out, the functionality is rendered completely unusable. Similarly, if the load phase has been completed successfully, a clock-oblivious version of the functionality can be triggered at any point in time as long as a valid witness is provided, no matter the number of failed attempts. The clock-aware version of the functionality, however, will only accept invocations of the trigger phase until a certain point in time. After the trigger phase times out, the functionality is rendered completely unusable.

Figure 5: The ideal functionality $\mathcal{F}_{\mathsf{SyX}}$.

$\mathcal{F}_{\mathsf{SyX}}$. In a nutshell, the synchronization of multiple invocations of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ is enabled by the "trigger" phase of functionality. We will be using f_1 to provide a proof to parties other than P_1, P_2 that the input submission phase has ended for parties P_1, P_2. In other words, if we wish to synchronize multiple invocations of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$, we set the witness for the trigger phase of each of the invocations to be the set of all receipts obtained from the inputs phases of the invocations. The set of all receipts acts as a proof that every invocation of the ideal functionality completed its load phase successfully. We use this feature of $\mathcal{F}_{\mathsf{SyX}}$ in order to design a protocol for fair secure computation.

Multiple Triggers and Witnesses. Note that as described, the load phase of the functionality $\mathcal{F}_{\mathsf{SyX}}$ can only be executed successfully once. And, once it has been successfully executed, the functionality is in the trigger phase. However, whilst in the trigger phase, the primitive may be triggered any number of times successfully or unsuccessfully. Furthermore, triggering the primitive with different witnesses may actually produce different outputs, as modeled by having the output f_2 depend on the witness w in addition to x_1, x_2. This will be important for us in Section 5.

Clock-awareness. A technicality that arises in the protocol is that of guaranteed termination. Specifically, we will need our ideal functionality to be "clock-aware". The issue of modeling a trusted clock has been studied in the literature. In this work, we stick to the formalism outlined in [PST17]. We recall the main ideas here. We assume a synchronous execution model, where protocol execution proceeds in atomic time steps called *rounds*. We assume that the trusted clocks of attested execution processors and the network rounds advance at the same rate. It is easy to adapt our model and results if the trusted clocks of the processors and the network rounds do not advance at the same rate. In each round, the environment must activate each party one by

Preliminaries: $x_1, x_2 \in \{0,1\}^*$; f_1, f_2 are 2-output functions; ϕ_1, ϕ_2 are boolean predicates; r denotes the current round number; $\mathsf{INPUT\_TIMEOUT} < \mathsf{TRIGGER\_TIMEOUT}$ are round numbers representing time outs. The functionality proceeds as follows:

- **Load phase.** If $r > \mathsf{INPUT\_TIMEOUT}$, abort. Otherwise, upon receiving inputs of the form $(x_1, f = (f_1, f_2, \phi_1, \phi_2))$ from P_1 and (x_2, f') from P_2, check if $f = f'$. If not, abort. Else, compute $f_1(x_1, x_2, r)$. If $f_1(x_1, x_2, r) = \perp$, abort. Else, send $f_1^i(x_1, x_2, r)$ to P_i for $i \in \{1, 2\}$, and go to next phase.
- **Trigger phase.** If $r > \mathsf{TRIGGER\_TIMEOUT}$, abort. Otherwise, upon receiving input w from party P_i, check if $\phi_i(w, r) = 1$. If yes, then send $(w, f_2^j(x_1, x_2, w, r))$ to both parties P_j for $j \in \{1, 2\}$.

Figure 6: The clock-aware ideal functionality $\mathcal{F}_{\mathsf{SyX}}$.

one, and therefore, all parties can naturally keep track of the current round number. We will use the symbol r to denote the current round number. A party can perform any fixed polynomial (in λ) amount of computation when activated, and send messages. We consider a synchronous communication model where messages sent by an honest party will be delivered at the beginning of the next round. Whenever a party is activated in a round, it can read a buffer of incoming messages to receive messages sent to itself in the previous round. To model trusted clocks in attested execution processors, we will provide a special instruction such that ideal functionalities, in particular $\mathcal{F}_{\mathsf{SyX}}$ can query the current round number. We say that a functionality $\mathcal{F}$ is *clock-aware* if the functionality queries the local time; otherwise we say that the functionality $\mathcal{F}$ is *clock-oblivious*. For the rest of the work, we will always assume that $\mathcal{F}_{\mathsf{SyX}}$ is clock-aware. We would also like to stress that we require only relative clocks – in other words, trusted clocks of all functionalities need not be synchronized, since our protocol will only make use of the number of rounds that have elapsed since initialization. Therefore, we will assume that when a functionality reads the clock, a relative round number since the first invocation of the functionality is returned. Thus, when working in this model, we assume that every party and every invocation of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ has access to a variable r that reflects the current round number. More generally, every function and predicate that is part of the specification of $\mathcal{F}_{\mathsf{SyX}}$ may also take r as an input. Finally, the functionality may also time out after a pre-programmed amount of time. We describe this clock-aware functionality in Figure 6.

Infinite Timeouts. We note here that it is possible to set either one or both of $\mathsf{INPUT\_TIMEOUT}$ and $\mathsf{TRIGGER\_TIMEOUT}$ to be ∞. What this means is that the functionality retains its state even if it goes offline. Its state would comprise $f_1, f_2, \phi_1, \phi_2, x_1, x_2$ and which phase (input or trigger) it is currently in. We also require that if the functionality does go offline and come back online, it can still access the current value of the clock, r. The only time we use this feature of the primitive is in Section 5 where we are able to preprocess the functionality for an unbounded number of fair multiparty computations that would be run in the future. In this case, we would need to trigger this functionality whenever an adversary attempts to break fairness. Since we have no bound on how many computations we will run, we will set the $\mathsf{TRIGGER\_TIMEOUT}$ to be ∞. In practice,

one could also just set TRIGGER_TIMEOUT to be a very large number. We stress however that the functionality is stateful and able to read time irrespective of whether it goes offline intermittently.

4 Fair Secure Computation in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model

In this section, we will describe how a set of n parties in the OT-network model that have pairwise access to the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ can implement n-party fair secure function evaluation. To begin with, we will assume that the n-parties are in the point-to-point model and develop a protocol in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model. We first provide some intuition for our construction.

4.1 Intuition

We first start with the 3-party case as a warm-up. Let P_1, P_2, and P_3 be the three parties with inputs x_1, x_2 and x_3 respectively. For $i, j \in \{1, 2, 3\}$ with $i < j$, we have that parties P_i and P_j have access to the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. In particular, let $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ represent the instantiation of the $\mathcal{F}_{\mathsf{SyX}}$ functionality used by parties P_i, P_j. We wish to perform fair secure function evaluation of some 3-input 3-output functionality F.

Reduction to single output functionalities. Let $(y_1, y_2, y_3) \xleftarrow{\$} F(x_1, x_2, x_3)$ be the output of the function evaluation. We define a new four input single output functionality F' such that

$$F'(x_1, x_2, x_3, z) = F^1(x_1, x_2, x_3) \| F^2(x_1, x_2, x_3) \| F^3(x_1, x_2, x_3) \oplus z = y_1 \| y_2 \| y_3 \oplus z$$

where $z = z_1 \| z_2 \| z_3$ and $|y_i| = |z_i|$ for all $i \in [3]$. The idea is that the party P_i will obtain $z' = F'(x_1, x_2, x_3, z)$ and z_i. Viewing $z' = z'_1 \| z'_2 \| z'_3$ where $|z'_i| = |z_i|$[15] for all $i \in [3]$, party P_i reconstructs its output as

$$y_i = z_i \oplus z'_i$$

Now, we may assume that the input of party P_i is (x_i, z_i) (or we can generate random z_is as part of the computation) which determines z. It thus suffices to consider fair secure function evaluation of single output functionalities.

Reduction to fair reconstruction. We will use ideas similar to [GIM+10, KVV16] where instead of focusing on fair secure evaluation of an arbitrary function, we only focus on fair reconstruction of an additive secret sharing scheme. The main idea is to let the three parties run a secure computation protocol that computes the output of the secure function evaluation on the parties' inputs, and then additively secret shares the output. Given this step, fair secure computation then reduces to fair reconstruction of the underlying additive secret sharing scheme.

The underlying additive secret sharing scheme. We use an additive secret sharing of the output y. Let the shares be y_i for $i \in [3]$. That is, it holds that

$$y = \bigoplus_{i \in [3]} y_i$$

We would like party P_i to reconstruct y by obtaining all shares y_i for each $i \in [3]$. Initially, each party P_i is given y_i. Therefore, each party P_i only needs to obtain y_j and y_k for $j, k \neq i$.

[15] We may assume without loss of generality that the lengths of the outputs of each party are known beforehand.

Fair reconstruction via $\mathcal{F}_{\mathsf{SyX}}$. We assume that the secure function evaluation also provides commitments to all the shares of the output. That is, P_i receives $(y_i, \overrightarrow{c})$ for each $i \in [3]$, where Com is a commitment scheme and

$$\overrightarrow{c} = \{\mathsf{Com}(y_1), \mathsf{Com}(y_2), \mathsf{Com}(y_3)\}$$

Furthermore, we assume that each party P_i picks its own verification key vk_i and signing key sk_i with respect to a signature scheme with a signing algorithm Sign and a verification algorithm Verify, for each $i \in [3]$. All parties then broadcast their verification keys to all parties. Let

$$\overrightarrow{\mathsf{vk}} = \{\mathsf{vk}_1, \mathsf{vk}_2, \mathsf{vk}_3\}$$

Each pair of parties P_i and P_j then initializes $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ with inputs

$$x_i = \left(\overrightarrow{\mathsf{vk}}, \mathsf{sk}_i, y_i, \overrightarrow{c}\right)$$

and

$$x_j = \left(\overrightarrow{\mathsf{vk}}, \mathsf{sk}_j, y_j, \overrightarrow{c}\right)$$

The function f_1 checks if both parties provided the same value for $\overrightarrow{\mathsf{vk}}, \overrightarrow{c}$ and checks the y_i and y_j are valid openings to the corresponding commitments. It also checks that the signing keys provided by the parties are consistent with the corresponding verification keys (more precisely, we will ask for randomness provided to the key generation algorithm of the signature scheme). If all checks pass, then $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ computes

$$\sigma_{i,j} = \mathsf{Sign}((i,j); \mathsf{sk}_i) \| \mathsf{Sign}((i,j); \mathsf{sk}_j)$$

This completes the description of f_1.

Synchronization step. The output of f_1 for each of the $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ will provide a way to synchronize all $\mathcal{F}_{\mathsf{SyX}}$ instances. By synchronization, we mean that an $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ instance cannot be triggered unless every other instance has already completed its load phase successfully. We achieve synchronization by setting the predicate $\phi_k(w)$ (for $k \in \{i,j\}$) to output 1 if and only if w consists of all signatures

$$\overrightarrow{\sigma} = \{\sigma_{i,j}\}_{i<j}$$

That is, each instance $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ will accept the same trigger $w = \overrightarrow{\sigma}$. We define f_2 to simply output both y_i and y_j to both parties if $\phi_k(w) = 1$.

Protocol intuition. We briefly discuss certain malicious behaviors and how we handle them. From the description above, it is clear that parties have no information about the output until one of the $\mathcal{F}_{\mathsf{SyX}}$ instances is triggered. Furthermore, note that this implies that the corrupt parties must successfully complete the load phases of the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it shares with all of the honest parties in order to obtain the witness that can be used to trigger the $\mathcal{F}_{\mathsf{SyX}}$ instances. Following the load phases of all of the $\mathcal{F}_{\mathsf{SyX}}$ instances, we ask each party to broadcast the receipt $\sigma_{i,j}$ obtained from $\mathcal{F}_{\mathsf{SyX}}^{i,j}$. Now suppose parties P_i and P_j are both dishonest, and suppose they do not broadcast $\sigma_{i,j}$. Note also that since P_i and P_j collude, they do not need the help of $\mathcal{F}_{\mathsf{SyX}}$ to compute $\sigma_{i,j}$. Since honest P_k does not know the synchronizing witness $\overrightarrow{\sigma}$, it will not be able to trigger any of

the $\mathcal{F}_{\mathsf{SyX}}$ instances. However, note that for the adversary to learn the output of the computation, the corrupt party P_i (without loss of generality) will need to trigger $\mathcal{F}_{\mathsf{SyX}}^{i,k}$ to obtain P_k's share of the key. However, once P_i triggers $\mathcal{F}_{\mathsf{SyX}}^{i,k}$, it follows that P_k would obtain the synchronizing witness $\overrightarrow{\sigma}$ using which it can trigger both $\mathcal{F}_{\mathsf{SyX}}^{i,k}$ and $\mathcal{F}_{\mathsf{SyX}}^{j,k}$ and learn its output.

Termination. The protocol as described up until this point does not have guaranteed termination. In particular, the honest parties will need to wait for the corrupted parties to broadcast their receipts or trigger a channel with an honest party in order to be able to trigger the instances of $\mathcal{F}_{\mathsf{SyX}}$ and obtain the output. Time outs do not help in this case as the adversary may simply wait until the last moment to trigger instances of $\mathcal{F}_{\mathsf{SyX}}$ and obtain their outputs leaving insufficient time for the honest parties to trigger their instances of $\mathcal{F}_{\mathsf{SyX}}$ and obtain their outputs. In order to ensure termination, we make use of the clock. The main invariant that we want to guarantee is that if an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an (honest) party is triggered, then every other instance of $\mathcal{F}_{\mathsf{SyX}}$ that the (honest) party is involved in, also needs to be triggered. One way to implement this idea is to assume that all instances of $\mathcal{F}_{\mathsf{SyX}}$ time out after

$$T = \binom{3}{2} = 3$$

rounds. Furthermore, an instance of $\mathcal{F}_{\mathsf{SyX}}$ accepts triggers in some round $\tau \in [T]$ (that is, until it times out) if and only if you provide a proof that $t-1$ other instances of $\mathcal{F}_{\mathsf{SyX}}$ were triggered until now. As before, we will have $\mathcal{F}_{\mathsf{SyX}}$ leak the triggering witness to the parties. Thus, if $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ is triggered in some round t, then P_i (and/or P_j) can trigger all the other $\mathcal{F}_{\mathsf{SyX}}^{i,k}$ (and/or $\mathcal{F}_{\mathsf{SyX}}^{j,k}$) channels that it is involved in, in round $\tau + 1$.

Suppose some honest party, say P_i, does not obtain the output of the computation while the adversary has learned the output. Since the adversary learned the output, this means that the adversary triggered $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ for some j (otherwise the adversary would not have learnt y_i and would not have received the output). That means P_i would have been able to trigger all the other channels that it is involved in and generate the final output in the next round. The only issue with this argument would be when $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ was triggered last, that is, in round $\tau = T$. However this is not possible since until this time, at most $T - n + 1 < T - 1$, assuming $n \geq 3$, instances of $\mathcal{F}_{\mathsf{SyX}}$ could have been be triggered. This is because $n-1$ instances of $\mathcal{F}_{\mathsf{SyX}}$ must be left untriggered in round $\tau = T - 1$ since the honest party didn't get its output.

Reducing the duration of time outs. A more clever solution will allow us to terminate within $T = n$ rounds. In order to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ in some round $\tau \in [T]$, you must provide a proof that other instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least τ different parties have been triggered. Consider the first round τ in which P_i is an honest party and $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ is triggered for some j. If $\tau = 1$, then the single invocation already gives a proof that channels involving two parties, namely, i, j, have been triggered. Otherwise, by assumption, proofs of invocations of instances of $\mathcal{F}_{\mathsf{SyX}}$ involving τ different parties were needed to trigger $\mathcal{F}_{\mathsf{SyX}}^{i,j}$. But P_i is not one of these parties as τ is the first round in which $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ was triggered for any j. Consequently, P_i, on this invocation, obtains a proof that instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least $\tau + 1$ parties have been triggered, and can thus trigger all channels in round $\tau + 1$. As before, the only gap in the argument is the case $\tau = T$. One can

trivially see that since $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ has not been triggered for any j, it is impossible to obtain a proof that instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least T different parties have been triggered.

Simulation. We look ahead for the issues that come up while trying to prove security, that is, during the simulation. The simulator will release to the adversary, the adversary's shares of the output, which can be simulated. But, it also releases commitments to all the shares of the output. Since the simulator does not know the output *apriori*, and does not know whether the adversary is going to abort the computation, in which case, no one knows the output, it has to produce commitments that it can later *equivocate*. In this context, we use, not regular commitments, but honest-binding commitments. In this case, the simulator can produce commitments to garbage but can later open them to be *valid* shares of the output. The rest of the computations can be trivially simulated. The only other detail to be looked into is that of the clock. We need to determine if the adversary has decided to abort the computation, that is, if the adversary is going to receive the output of the computation of not. This is done by noticing if and when the adversary decides to trigger the instances of $\mathcal{F}_{\mathsf{SyX}}$ that involve honest parties. We know that if the adversary ever triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, then all parties will be in a position to receive the output. Thus, the simulator can simply run the adversary to determine whether it has decided to enable parties to obtain the output, in which case the simulator would ask the trusted party to continue, or not, in which case the simulator would ask the trusted party to abort.

4.2 Protocol

We now present the protocol for fair secure computation in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model.

Preliminaries. F is the n-input n-output functionality to be computed; x_i is the input of party P_i for $i \in [n]$; $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ represents the instantiation of the $\mathcal{F}_{\mathsf{SyX}}$ functionality used by parties P_a, P_b with time out round numbers $\mathsf{INPUT\_TIMEOUT} = 0$ and $\mathsf{TRIGGER\_TIMEOUT} = n$ for $a < b$, where $a, b \in [n]$; $\left(\mathsf{Com}, \mathsf{Open}, \widetilde{\mathsf{Com}}, \widetilde{\mathsf{Open}}\right)$ is an honest-binding commitment scheme; $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a signature scheme; r denotes the current round number.

Protocol. The protocol Π_{FMPC} proceeds as follows:

- Define F' to the be the following n-input n-output functionality: On input $\overrightarrow{x} = (x_1, \ldots, x_n)$:
  - Let $(y_1, \ldots, y_n) = F(x_1, \ldots, x_n)$ and let
    $$y = y_1 \| \ldots \| y_n$$
    Sample random strings $\alpha_i \xleftarrow{\$} \{0,1\}^*$ such that $|\alpha_i| = |y_i|$ for each $i \in [n]$. Let
    $$\alpha = \alpha_1 \| \ldots \| \alpha_n$$
    Let $z = y \oplus \alpha$.
  - Sample a random additive n-out-of-n secret sharing $z_1, \ldots, z_n$ of z such that
    $$z = \bigoplus_{i \in [n]} z_i$$

– Compute commitments along with their openings $(c_i^z, \omega_i^z) \xleftarrow{\$} \mathsf{Com}(z_i)$ to each of the shares z_i for each $i \in [n]$. Let

$$\overrightarrow{c^z} = (c_1^z, \ldots, c_n^z)$$

– Sample random proof values $\pi_1, \ldots, \pi_n \xleftarrow{\$} \{0,1\}^\lambda$. Compute commitments along with their openings $(c_i^\pi, \omega_i^\pi) \xleftarrow{\$} \mathsf{Com}(\pi_i)$ to each of the proof values π_i for each $i \in [n]$. Let

$$\overrightarrow{c^\pi} = (c_1^\pi, \ldots, c_n^\pi)$$

– Party P_i receives output $\left(\alpha_i, \overrightarrow{c^z}, \omega_i^z, z_i, \overrightarrow{c^\pi}, \omega_i^\pi, \pi_i\right)$ for each $i \in [n]$.

- The parties invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ with inputs $((x_1, F'), \ldots, (x_n, F'))$. If the ideal functionality returns $\perp$ to party P_i, then P_i aborts for any $i \in [n]$[16]. Otherwise, party P_i receives output $\left(\alpha_i, \overrightarrow{c^z}, \omega_i^z, z_i, \overrightarrow{c^\pi}, \omega_i^\pi, \pi_i\right)$ for each $i \in [n]$.

- Each party P_i, for each $i \in [n]$, picks a random $\beta_i \in \{0,1\}^*$ and uses this randomness to pick a signing and verification key pair $(\mathsf{sk}_i, \mathsf{vk}_i) = \mathcal{V}.\mathsf{Gen}(1^\lambda; \beta_i)$. It then invokes the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ and broadcasts vk_i to all other parties. If it does not receive vk_j for all $j \neq i$, it aborts. Otherwise, it obtains

$$\overrightarrow{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$$

- For each $a, b \in [n]$ with $a < b$, define the following functions.

  – Let $f_1^{a,b}$ be the function that takes as input (γ, γ') and parses

  $$\gamma = \left(\overrightarrow{\mathsf{vk}}, \mathsf{sk}, \beta, \overrightarrow{c^z}, \omega^z, z, \overrightarrow{c^\pi}, \omega^\pi, \pi\right)$$

  and

  $$\gamma' = \left(\overrightarrow{\mathsf{vk}'}, \mathsf{sk}', \beta', \overrightarrow{c^z}', \omega^{z\prime}, z', \overrightarrow{c^\pi}', \omega^{\pi\prime}, \pi'\right)$$

  It checks that:
  * $\overrightarrow{\mathsf{vk}} = \overrightarrow{\mathsf{vk}'}$, $\overrightarrow{c^z} = \overrightarrow{c^z}'$, $\overrightarrow{c^\pi} = \overrightarrow{c^\pi}'$
  * $(\mathsf{sk}, \mathsf{vk}_a) = \mathcal{V}.\mathsf{Gen}(1^\lambda; \beta)$, $(\mathsf{sk}', \mathsf{vk}_b) = \mathcal{V}.\mathsf{Gen}(1^\lambda; \beta')$
  * $\mathsf{Open}(c_a^z, \omega^z, z) = \mathsf{Open}(c_b^z, \omega^{z\prime}, z') = 1$
  * $\mathsf{Open}(c_a^\pi, \omega^\pi, \pi) = \mathsf{Open}(c_b^\pi, \omega^{\pi\prime}, \pi') = 1$

[16] In the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, let $\pi_{F'}$ denote the protocol for the functionality F' defined in Lemma 8. The parties execute $\pi_{F'}$. If the execution of $\pi_{F'}$ aborts, we are assuming that all (honest) parties are aware of the round when the execution of $\pi_{F'}$ aborts, that is, when the adversary has decided to abort the execution of $\pi_{F'}$. Since we are working in the $\mathcal{F}_{\mathsf{MPC}}$-hybrid model, we know that in the ideal model, this is the case when the honest parties receive $\perp$ as their output. If we assume that in the case when the the adversary decides to let the honest parties obtain their outputs, no honest party ever receives $\perp$, this could be used to identify the scenario when the adversary has decided to abort the execution of $\pi_{F'}$. Thus, we could, in principle, replace this instruction with: If party P_i receives $\perp$ as its output, it aborts. Furthermore, since we are considering the case of *unanimous abort*, if the adversary has decided to abort the execution of $\pi_{F'}$, all honest parties abort the protocol.

If all of these checks pass, then $f_1^{a,b}$ outputs

$$\sigma_{a,b} = (\mathcal{V}.\mathsf{Sign}((a,b);\mathsf{sk}_a), \mathcal{V}.\mathsf{Sign}((a,b);\mathsf{sk}_b))$$

and otherwise it outputs $\perp$.

– Let $\phi_1^{a,b}$ be the function that takes as input a witness w, which is either of the form $(0, \overrightarrow{\sigma})$ or of the form $\left(1, \overrightarrow{\sigma}, \overrightarrow{z}, \overrightarrow{\omega^z}, \overrightarrow{\pi}, \overrightarrow{\omega^\pi}, \overrightarrow{\mathsf{ind}}\right)$.

∗ If w is of the first form, then it tests if $r = 1$ and

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,1}, (a,b); \mathsf{vk}_a\right) = 1$$

and

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,2}, (a,b); \mathsf{vk}_b\right) = 1$$

for all $a, b \in [n]$ with $a < b$, outputting 1 if so and 0 if not.

∗ If w is of the second form, then it checks that:

·

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,1}, (a,b); \mathsf{vk}_a\right) = 1$$

and

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,2}, (a,b); \mathsf{vk}_b\right) = 1$$

for all $a, b \in [n]$ with $a < b$

· $|\overrightarrow{\pi}| = \left|\overrightarrow{\omega^\pi}\right| = \left|\overrightarrow{\mathsf{ind}}\right| = r$

· $\overrightarrow{\mathsf{ind}}$ consists of distinct indices in $[n]$.

· $\mathsf{Open}\left(c^z_{\mathsf{ind}_j}, \omega^z_j, z_j\right) = 1$ for every $j \in [r]$.

· $\mathsf{Open}\left(c^\pi_{\mathsf{ind}_j}, \omega^\pi_j, \pi_j\right) = 1$ for every $j \in [r]$.

If all of these checks pass, then $\phi_1^{a,b}$ outputs 1 and otherwise it outputs 0.

– Let $\phi_2^{a,b}$ be identical to $\phi_1^{a,b}$.

– Let $f_2^{a,b}$ be the function that takes as input (γ, γ') where γ, γ' are as above, and outputs $(\omega^z, z, \omega^\pi, \pi, \omega^{z\prime}, z', \omega^{\pi\prime}, \pi')$.

- Set $r = 0$[17]. Each party P_a for each $a \in [n]$ will now run the load phase to set up each instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in. For each pair of parties P_a, P_b with $a \neq b$ for $a, b \in [n]$, let $a' = \mathsf{min}(a,b)$ and $b' = \mathsf{max}(a,b)$. For each such pair of parties P_a, P_b, party P_a runs the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a',b'}$, providing inputs (x_a, f), where

$$x_a = \left(\overrightarrow{\mathsf{vk}}, \mathsf{sk}_a, \beta_a, \overrightarrow{c^z}, \omega^z_a, z_a, \overrightarrow{c^\pi}, \omega^\pi_a, \pi_a\right)$$

and

$$f = \left(f_1^{a',b'}, f_2^{a',b'}, \phi_1^{a',b'}, \phi_2^{a',b'}\right)$$

[17]This does not entail actually setting $r = 0$, but rather viewing the current round as round zero and henceforth referencing rounds with respect to it, that is, viewing r as the round number relative to the round number when this statement was executed.

- If $r > n$, abort. Otherwise, while $r \leq n$,
  - If a party P_a for $a \in [n]$ receives $\sigma_{a',b'}$ from each $\mathcal{F}_{\mathsf{SyX}}^{a',b'}$ it is involved in, indicating that the load phase of all such $\mathcal{F}_{\mathsf{SyX}}$ functionalities were completed successfully, and $r = 0$, it invokes the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ and broadcasts
  $$\overrightarrow{\sigma_a} = \left\{\sigma_{a',b'}\right\}_{a'=a \ \vee \ b'=a}$$
  to all the parties. Otherwise, it invokes the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ when $r = 1$ and broadcasts abort to all the parties and aborts.
  - If a party P_a for $a \in [n]$ receives $\overrightarrow{\sigma}$ such that
  $$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,1}, (a,b); \mathsf{vk}_a\right) = 1$$
  and
  $$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,2}, (a,b); \mathsf{vk}_b\right) = 1$$
  for all $a, b \in [n]$ with $a < b$, and $r = 1$, then it uses the witness $w = (0, \overrightarrow{\sigma})$ to invoke the trigger phase of each instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in. Once all such instances of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a have been triggered, use the shares $z_1, \ldots, z_n$ to reconstruct z, parses z as $z_1 \| \ldots \| z_n$ where $|z_i| = |y_i|$ for all $i \in [n]$[18] and computes $y_i = z_i \oplus \alpha_i$ to obtain the output of the computation.
  - If party P_a for $a \in [n]$ has not received the output of the computation and an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a is first triggered in round $1 \leq r < n$, it triggers each instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in during round $r + 1$ using the output out it receives from the instance of $\mathcal{F}_{\mathsf{SyX}}$ as follows:
    - If $\mathsf{out}_1 = (0, \overrightarrow{\sigma})$, then $r = 1$. Let $\mathcal{F}_{\mathsf{SyX}}^{a',b'}$ be the instance of $\mathcal{F}_{\mathsf{SyX}}$ that was triggered, where $a' = a \ \vee \ b' = a$. Parse $\mathsf{out}_2 = (\omega^z, z, \omega^\pi, \pi, \omega^{z\prime}, z', \omega^{\pi\prime}, \pi')$. It prepares the witness
    $$w = \left(1, (z, z'), \left(\omega^z, \omega^{z\prime}\right), (\pi, \pi'), \left(\omega^\pi, \omega^{\pi\prime}\right), (a', b')\right)$$
    - If $\mathsf{out}_1 = \left(1, \overrightarrow{\sigma}, \overrightarrow{z}, \overrightarrow{\omega^z}, \overrightarrow{\pi}, \overrightarrow{\omega^\pi}, \overrightarrow{\mathsf{ind}}\right)$, it prepares the witness
    $$w = \left(1, \overrightarrow{\sigma}, \overrightarrow{z}', \overrightarrow{\omega^z}', \overrightarrow{\pi}', \overrightarrow{\omega^\pi}', \overrightarrow{\mathsf{ind}}'\right)$$
    where
      - $|\overrightarrow{z}'| = r + 1, \ \overrightarrow{z}'|_{[r]} = \overrightarrow{z}|_{[r]}, \ z'_{r+1} = z_a$
      - $\left|\overrightarrow{\omega^z}'\right| = r + 1, \ \overrightarrow{\omega^z}'\Big|_{[r]} = \overrightarrow{\omega^z}\Big|_{[r]}, \ {\omega^z_{r+1}}' = \omega^z_a$
      - $|\overrightarrow{\pi}'| = r + 1, \ \overrightarrow{\pi}'|_{[r]} = \overrightarrow{\pi}|_{[r]}, \ \pi'_{r+1} = \pi_a$
      - $\left|\overrightarrow{\omega^\pi}'\right| = r + 1, \ \overrightarrow{\omega^\pi}'\Big|_{[r]} = \overrightarrow{\omega^\pi}\Big|_{[r]}, \ {\omega^\pi_{r+1}}' = \omega^\pi_a$
      - $\left|\overrightarrow{\mathsf{ind}}'\right| = r + 1, \ \overrightarrow{\mathsf{ind}}'\Big|_{[r]} = \overrightarrow{\mathsf{ind}}\Big|_{[r]}, \ \mathsf{ind}'_{r+1} = a$

[18] We may assume without loss of generality that the lengths of the outputs of each party are known beforehand.

Once all instances of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a have been triggered, it uses the shares $z_1, \ldots, z_n$ to reconstruct z, parses z as $z_1 \| \ldots \| z_n$ where $|z_i| = |y_i|$ for all $i \in [n]$ and computes $y_i = z_i \oplus \alpha_i$ to obtain the output of the computation.

– If party P_a for $a \in [n]$ has not received the output of the computation and an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a is triggered and $r = n$, it receives all shares of z. It uses the shares $z_1, \ldots, z_n$ to reconstruct z, parses z as $z_1 \| \ldots \| z_n$ where $|z_i| = |y_i|$ for all $i \in [n]$ and computes $y_i = z_i \oplus \alpha_i$ to obtain the output of the computation.

Remark. It is possible to replace the $\mathcal{O}(n^2)$ signatures with n other commitments to n other independent random proof values (akin to π) that can be used to prove that all the instances of $\mathcal{F}_{\mathsf{SyX}}$ completed their load phases successfully.

4.3 Proof sketch of Security

We sketch the proof of security of the above protocol. The correctness of the computation of the functionality F' follows by definition from the correctness of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Furthermore, we have that at the end of the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, either all honest parties *unanimously abort* or all honest parties *unanimously continue*. Thus, assuming that $\mathcal{F}_{\mathsf{MPC}}$ did not abort, every party receives the output of F'. For every $i \in [n]$, let $\overrightarrow{\mathsf{vk}_i}$ denote the set of verification keys that were obtained by party P_i. Note that, by the correctness of the ideal functionality $\mathcal{F}_{\mathsf{bc}}$,

$$\overrightarrow{\mathsf{vk}} = \overrightarrow{\mathsf{vk}_i}$$

for all $i \in [n]$. If $\overrightarrow{\mathsf{vk}}$ does not contain vk_j for every $j \in [n]$, which would happen in the case that some corrupt parties do not broadcast their verification keys, all honest parties *unanimously abort*. Otherwise, all honest parties *unanimously continue*. Assuming the honest parties have not aborted, we note that if the corrupt parties do not provide valid inputs to the load phase of even one of the instances of $\mathcal{F}_{\mathsf{SyX}}$ that they are involved in along with an honest party, say P_i for some $i \in [n]$, by the correctness of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ and the binding property for the honestly generated commitments, that particular instance of $\mathcal{F}_{\mathsf{SyX}}$ will not complete its load phase successfully. In this case P_i will force all honest parties to *unanimously abort*, since no party (not even the corrupt ones) can obtain their output. We thus consider the case where all instances of $\mathcal{F}_{\mathsf{SyX}}$ have completed their load phases successfully. At this point, if all parties broadcast all the signatures they obtained from the instances of $\mathcal{F}_{\mathsf{SyX}}$, all parties can trigger the instances of $\mathcal{F}_{\mathsf{SyX}}$ that they are involved in to receive all the shares of z, reconstruct z and finally obtain their output correctly. The issue arises when some corrupt parties do not broadcast the signatures they obtained from the instances of $\mathcal{F}_{\mathsf{SyX}}$. If a corrupt party triggers any instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_i for some $i \in [n]$, with a witness of the form $(0, \overrightarrow{\sigma})$ in round 1, then the honest party obtains a tuple of values $(\overrightarrow{\sigma}, z, \omega^z, \pi, \omega^\pi)$ from the corrupt party. In addition to its own such tuple of values, it obtains a valid witness to trigger all the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round 2. Since $n \geq 2$, P_i succeeds in doing this and obtaining the shares of z, z and hence finally its output correctly. Consider any honest party P_j for $j \neq i$. Since $n > 2$, P_j, as did P_i, proceeds to trigger all the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round 3. If no corrupt party triggers any instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party with a witness of the form $(0, \overrightarrow{\sigma})$ in round 1, if the adversary is to obtain the output, it must instruct a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in along with an honest party, but now using a witness of the form $\left(1, \overrightarrow{\sigma}, \overrightarrow{z}, \overrightarrow{\omega^z}, \overrightarrow{\pi}, \overrightarrow{\omega^\pi}, \overrightarrow{\mathsf{ind}}\right)$. Let

r be the first round when a corrupt party triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in along with an honest party, say P_i for some $i \in [n]$, using a witness of the form $\left(1, \overrightarrow{\sigma}, \overrightarrow{z}, \overrightarrow{\omega^z}, \overrightarrow{\pi}, \overrightarrow{\omega^\pi}, \overrightarrow{\mathsf{ind}}\right)$. Then, it must be the case that $i \notin \overrightarrow{\mathsf{ind}}$ and that P_i now obtains the tuple of values $(\overrightarrow{\sigma}, z, \omega^z, \pi, \omega^\pi)$ corresponding to r parties other than itself. Combining this information with its own tuple of values $(z, \omega^z, \pi, \omega^\pi)$, it obtains a valid witness to trigger all the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round $r+1$. If $r < n$, P_i succeeds in doing this and obtaining the shares of z, z and hence finally its output correctly. Consider any honest party P_j for $j \neq i$. If $r+1 = n$, then P_j receives all the shares of z and consequently its output correctly. If $r+1 < n$, then P_j, as did P_i, proceeds to trigger all the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round $r+2$. Finally, we note that $r < n$ since r is the first round when a corrupt party triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in along with an honest party, which means that the witness it used to trigger the instance of $\mathcal{F}_{\mathsf{SyX}}$ can have the tuple of values $(z, \omega^z, \pi, \omega^\pi)$ corresponding to at most $n-1$ parties as at least one of the parties is honest. If this does not happen, then no party (not even the corrupt ones) obtains their output. This completes the proof of security.

4.4 Security

We now prove the following lemma.

Lemma 9. *If* $\left(\mathsf{Com}, \mathsf{Open}, \widetilde{\mathsf{Com}}, \widetilde{\mathsf{Open}}\right)$ *is an honest-binding commitment scheme and* $\mathcal{V}$ *is a signature scheme, then the protocol* Π_{FMPC} *securely computes* $\mathcal{F}_{\mathsf{MPC}}$ *with fairness in the* $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$*-hybrid model.*

Proof. Let $\mathcal{A}$ be an adversary attacking the execution of the protocol described in Section 4.2 in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model. We construct an ideal-model adversary $\mathcal{S}$ in the ideal model of type fair. Let F be the n-input n-output functionality to be computed. Let $\mathcal{I}$ be the set of corrupted parties. If $\mathcal{I}$ is empty, then there is nothing to simulate. $\mathcal{S}$ begins by simulating the first step of the protocol, namely, the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Here, $\mathcal{S}$ behaves as the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Recall that the type of $\mathcal{F}_{\mathsf{MPC}}$ is abort. $\mathcal{S}$ obtains the inputs $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ of the corrupted parties from $\mathcal{A}$. If $(x_i, f_i) = \mathsf{abort}$ for any $i \in \mathcal{I}$, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, receives $\perp$ as the output of all parties, which it forwards $\mathcal{A}$. Suppose $(x_i, f_i) \neq \mathsf{abort}$ for all $i \in \mathcal{I}$. If there exists a $j \in \mathcal{I}$ such that $f_j \neq F'$ as defined in protocol Π_{FMPC}, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, which aborts, and then aborts itself. If there exists a $j \in \mathcal{I}$ such that (x_j, f_j) is not of the specified format, $\mathcal{S}$ replaces (x_j, f_j) with a default value. Going forward, we assume that for all $i \in \mathcal{I}$, (x_i, f_i) is well-formed and that $f_i = F'$ as defined in Π_{FMPC}.

$\mathcal{S}$ now needs to simulate the outputs received by the corrupted parties from the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. For each $i \in [n]$, $\mathcal{S}$ samples a random string $\alpha_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the length of the ith output of F. Let

$$\alpha = \alpha_1 \| \ldots \| \alpha_n$$

For each $i \in \mathcal{I}$, $\mathcal{S}$ samples a random string $z_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the sum of the lengths of all the outputs of F. It then computes commitments along with their openings $(c_i^z, \omega_i^z) \xleftarrow{\$} \mathsf{Com}(z_i)$ to each of the shares z_i for each $i \in \mathcal{I}$. For each $i \in [n] \setminus \mathcal{I}$, it samples a equivocable commitment $(c_i^z, \mathsf{state}_i) \xleftarrow{\$} \widetilde{\mathsf{Com}}(1^\lambda)$. Let

$$\overrightarrow{c^z} = (c_1^z, \ldots, c_n^z)$$

For each $i \in [n]$, $\mathcal{S}$ samples random proof values $\pi_1, \ldots, \pi_n \xleftarrow{\$} \{0,1\}^\lambda$ and compute commitments along with their openings $(c_i^\pi, \omega_i^\pi) \xleftarrow{\$} \mathsf{Com}(\pi_i)$ to each of the proof values π_i. Let

$$\overrightarrow{c^\pi} = (c_1^\pi, \ldots, c_n^\pi)$$

and

$$\overrightarrow{\omega^\pi} = (\omega_1^\pi, \ldots, \omega_n^\pi)$$

Thus, the simulator constructs the output $\left(\alpha_i, \overrightarrow{c^z}, \omega_i^z, z_i, \overrightarrow{c^\pi}, \omega_i^\pi, \pi_i\right)$ for each $i \in \mathcal{I}$ and forwards it to $\mathcal{A}$. If $\mathcal{A}$ then sends abort, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, which it forwards $\mathcal{A}$. Otherwise, $\mathcal{A}$ responds with $\mathsf{continue}$. At this point, $\mathcal{S}$ has completed simulating the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$.

For each $i \in [n] \setminus \mathcal{I}$, $\mathcal{S}$ picks a random $\beta_i \in \{0,1\}^*$ and uses this randomness to pick a signing and verification key pair $(\mathsf{sk}_i, \mathsf{vk}_i) = \mathcal{V}.\mathsf{Gen}(1^\lambda; \beta_i)$. Now, $\mathcal{S}$ must simulate the invocations of the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ by the corrupt parties. Here, $\mathcal{S}$ behaves as the ideal functionality $\mathcal{F}_{\mathsf{bc}}$. Recall that the type of $\mathcal{F}_{\mathsf{bc}}$ is $\mathsf{g.d.}$. For all $i \in [n] \setminus \mathcal{I}$, $\mathcal{S}$ "broadcasts" vk_i to all the corrupt parties. For any $i \in \mathcal{I}$, if $\mathcal{A}$ instructs P_i to invoke $\mathcal{F}_{\mathsf{bc}}$ with input vk_i, $\mathcal{S}$ "broadcasts" vk_i to all the corrupt parties and stores vk_i. At the end of this round, if $\mathcal{A}$ did not instruct some corrupt party to invoke $\mathcal{F}_{\mathsf{bc}}$, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, and aborts itself. Otherwise, $\mathcal{S}$ successfully constructs

$$\overrightarrow{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$$

At this point, $\mathcal{S}$ has completed simulating the invocations of the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ used to broadcast the verification keys of all the parties.

$\mathcal{S}$ maintains a virtual round counter and initializes it to zero. Now, $\mathcal{S}$ has to simulate the invocations of the load phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that involve corrupt parties. Here, $\mathcal{S}$ behaves as the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. Recall that the type of $\mathcal{F}_{\mathsf{SyX}}$ is $\mathsf{g.d.}$. For any $a, b \in [n]$ with $a < b$ and $a \in \mathcal{I}$ and $b \in [n] \setminus \mathcal{I}$, if $\mathcal{A}$ instructs P_a to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ with inputs

$$\gamma = \left(\overrightarrow{\mathsf{vk}}', \mathsf{sk}_a, \beta_a, \overrightarrow{c^z}', \omega^z, z, \overrightarrow{c^\pi}', \omega^\pi, \pi\right)$$

$\mathcal{S}$ computes $f_1^{a,b}(\gamma, \gamma')$ as defined in Π_{FMPC}, where

$$\gamma' = \left(\overrightarrow{\mathsf{vk}}, \mathsf{sk}_b, \beta_b, \overrightarrow{c^z}, \omega_b^z, z_b, \overrightarrow{c^\pi}, \omega_b^\pi, \pi_b\right)$$

Note that since $b \in [n] \setminus \mathcal{I}$, $\mathcal{S}$ does in fact have $\mathsf{sk}_b, \beta_b, \omega_b^\pi, \pi_b$. The only values it does not have are ω_b^z, z_b. In the execution of $f_1^{a,b}$, ω_b^z, z_b are needed to check that

$$\mathsf{Open}(c_b^z, \omega_b^z, z_b) = 1$$

Note that since P_b is an honest party, it would always supply inputs such that this check passes. Furthermore, the outcome of this check does not depend on any input that the adversary sends. Thus, in simulating the computation of $f_1^{a,b}$, $\mathcal{S}$ performs all the checks that $f_1^{a,b}$, except this one. If all the checks pass, $\mathcal{S}$ computes

$$\sigma_{a,b} = (\mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_a), \mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_b))$$

and forwards $\sigma_{a,b}$ to the adversary. $\mathcal{S}$ also stores sk_a, β_a. If any of the checks do not pass, $\mathcal{S}$ simply aborts simulating the load phase of this particular instance $\mathcal{F}_{\mathsf{SyX}}^{a,b}$. $\mathcal{S}$ behaves symmetrically if for any $a, b \in [n]$ with $a < b$ and $b \in \mathcal{I}$ and $a \in [n] \setminus \mathcal{I}$, if $\mathcal{A}$ instructs P_b to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$. The final case to consider is if for any $a, b \in [n]$ with $a < b$ and $a, b \in \mathcal{I}$, if $\mathcal{A}$ instructs P_a, P_b to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ with inputs

$$\gamma = \left(\overrightarrow{\mathsf{vk}}', \mathsf{sk}_a, \beta_a, \overrightarrow{c^z}', \omega^z, z, \overrightarrow{c^\pi}', \omega^\pi, \pi\right)$$

and

$$\gamma' = \left(\overrightarrow{\mathsf{vk}}'', \mathsf{sk}_b, \beta_b, \overrightarrow{c^z}'', \omega^{z\prime}, z', \overrightarrow{c^\pi}'', \omega^{\pi\prime}, \pi'\right)$$

$\mathcal{S}$ computes $f_1^{a,b}(\gamma, \gamma')$ as defined in Π_{FMPC}. If all the checks pass, $\mathcal{S}$ computes

$$\sigma_{a,b} = (\mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_a), \mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_b))$$

and forwards $\sigma_{a,b}$ to the adversary. $\mathcal{S}$ also stores $\mathsf{sk}_a, \beta_a, \mathsf{sk}_b, \beta_b$. If any of the checks do not pass, $\mathcal{S}$ simply aborts simulating the load phase of this particular instance $\mathcal{F}_{\mathsf{SyX}}^{a,b}$. At the end of this round, let $\mathsf{LoadFailed}$ denote the set of all i such that P_i is an honest party and $\mathcal{A}$ did not instruct some corrupt party to invoke the load phase of an instance of $\mathcal{F}_{\mathsf{SyX}}$ that it was involved in with P_i. If $\mathsf{LoadFailed}$ is not empty, for each $i \in \mathsf{LoadFailed}$, $\mathcal{S}$ must simulate the invocations of the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ by party P_i to broadcast abort. For each $i \in \mathsf{LoadFailed}$, $\mathcal{S}$ "broadcasts" abort to all the corrupt parties. $\mathcal{S}$ then forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, and aborts itself. Otherwise, $\mathcal{S}$ successfully constructs

$$\overrightarrow{\mathsf{sk}} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$$

and

$$\overrightarrow{\beta} = (\beta_1, \ldots, \beta_n)$$

$\mathcal{S}$ computes

$$\sigma_{a,b} = (\mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_a), \mathcal{V}.\mathsf{Sign}((a,b); \mathsf{sk}_b))$$

for every $a < b \in [n]$ and defines

$$\overrightarrow{\sigma} = \{\sigma_{a,b}\}_{a<b, a,b \in [n]}$$

Now, $\mathcal{S}$ must simulate the invocations of the ideal functionality $\mathcal{F}_{\mathsf{bc}}$ by the corrupt parties. For all $a \in [n] \setminus \mathcal{I}$, $\mathcal{S}$ "broadcasts"

$$\overrightarrow{\sigma_i} = \left\{\sigma_{a',b'}\right\}_{a'=i \ \vee \ b'=i}$$

to all the corrupt parties. For any $i \in \mathcal{I}$, if $\mathcal{A}$ instructs P_i to invoke $\mathcal{F}_{\mathsf{bc}}$ with input $\overrightarrow{\sigma_i}$, $\mathcal{S}$ "broadcasts" $\overrightarrow{\sigma_i}$ to all the corrupt parties.

Once round 0 is completed, $\mathcal{S}$ has completed simulating the invocations of the load phase of all the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ and the ideal functionality $\mathcal{F}_{\mathsf{bc}}$. What remains is to determine whether the adversary wishes to obtain its output and to simulate the invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that the adversary instructs corrupt parties to trigger. We consider two cases. First, we make the following definition: a witness w is *valid* if

$$w = (0, \overrightarrow{\sigma})$$

in round 1 with

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,1},(a,b);\mathsf{vk}_a\right)=1$$

and

$$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,2},(a,b);\mathsf{vk}_b\right)=1$$

for all $a,b\in[n]$ with $a<b$, or

$$w=\left(1,\overrightarrow{\sigma},\overrightarrow{z},\overrightarrow{\omega^z},\overrightarrow{\pi},\overrightarrow{\omega^\pi},\overrightarrow{\mathsf{ind}}\right)$$

in round $1\leq r\leq n$ with

- $$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,1},(a,b);\mathsf{vk}_a\right)=1$$

  and

  $$\mathcal{V}.\mathsf{Verify}\left(\sigma_{a,b,2},(a,b);\mathsf{vk}_b\right)=1$$

  for all $a,b\in[n]$ with $a<b$
- $|\overrightarrow{\pi}|=\left|\overrightarrow{\omega^\pi}\right|=\left|\overrightarrow{\mathsf{ind}}\right|=r$
- $\overrightarrow{\mathsf{ind}}$ consists of distinct indices in $[n]$.
- $\mathsf{Open}\left(c^z_{\mathsf{ind}_j},\omega^z_j,z_j\right)=1$ for every $j\in[r]$.
- $\mathsf{Open}\left(c^\pi_{\mathsf{ind}_j},\omega^\pi_j,\pi_j\right)=1$ for every $j\in[r]$.

Case A. The adversary instructed all corrupt parties to broadcast all of their signatures. $\mathcal{S}$ forwards $\{(x_i,f_i)\}_{i\in\mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness. It receives the corrupt parties outputs, namely, $\{y_i\}_{i\in\mathcal{I}}$. $\mathcal{S}$ chooses the outputs of the honest party completely at random, that is, it samples random strings $y_i\xleftarrow{\$}\{0,1\}^*$ of length equal to the length of the ith output of F, for $i\in[n]\setminus\mathcal{I}$. $\mathcal{S}$ then constructs

$$y=y_1\|\ldots\|y_n$$

It then defines

$$z=y\oplus\alpha$$

Let j be an arbitrary index in $[n]\setminus\mathcal{I}$. $\mathcal{S}$ samples random strings $z_i\xleftarrow{\$}\{0,1\}^*$ of length equal to the sum of the lengths of all the outputs of F, for $i\in[n]\setminus(\mathcal{I}\cup\{j\})$. $\mathcal{S}$ then computes

$$z_j=z\oplus\bigoplus_{i\in[n]\setminus\{j\}}z_i$$

and constructs

$$\overrightarrow{z}=(z_1,\ldots,z_n)$$

$\mathcal{S}$ computes $\omega_i^z \xleftarrow{\$} \widetilde{\mathsf{Open}}(\mathsf{state}_i, z_i)$ for each $i \in [n] \setminus \mathcal{I}$ and constructs

$$\overrightarrow{\omega^z} = (\omega_1^z, \ldots, \omega_n^z)$$

Note that, at this point, $\mathcal{S}$ has every value ever used in the protocol. For every $i \in [n] \setminus \mathcal{I}$ and every $j \in \mathcal{I}$, letting $a = \mathsf{min}(i,j)$ and $b = \mathsf{max}(i,j)$, $\mathcal{S}$ sends $\left((0, \overrightarrow{\sigma}), (\omega_a^z, z_a, \omega_a^\pi, \pi_a, \omega_b^z, z_b, \omega_b^\pi, \pi_b)\right)$ to P_j. Going forward, $\mathcal{S}$ simulates invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that the adversary instructs corrupt parties to trigger as follows.

- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving another corrupt party, say P_j for $j \in \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$[19] to parties P_i and P_j.
- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_j for $j \in [n] \setminus \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to P_i.
- Suppose the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an *invalid* witness. $\mathcal{S}$ simply sends no response.
- Suppose the an honest party, say P_i for $i \in [n] \setminus \mathcal{I}$, triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving a corrupt party, say P_j for $j \in \mathcal{I}$. $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to P_j.

Case B. The adversary did not instruct all corrupt parties to broadcast all of their signatures. We first discuss how $\mathcal{S}$ simulates certain invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that the adversary instructs the corrupt parties to trigger.

- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving another corrupt party, say P_j for $j \in \mathcal{I}$, with a *valid* witness w. $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to parties P_i and P_j.
- Suppose the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an *invalid* witness. $\mathcal{S}$ simply sends no response.

Suppose the adversary does not instruct a corrupt party, say P_i for some $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_j for some $j \in [n] \setminus \mathcal{I}$, with a *valid* witness and the round counter exceeds n, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, and aborts itself. Otherwise, at the first instant the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party with a *valid* witness w, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness. It receives the corrupt parties outputs, namely, $\{y_i\}_{i \in \mathcal{I}}$. $\mathcal{S}$ chooses the outputs of the honest party completely at random, that is, it samples random strings $y_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the length of the ith output of F, for $i \in [n] \setminus \mathcal{I}$. $\mathcal{S}$ then constructs

$$y = y_1 \| \ldots \| y_n$$

[19] Technically, this would have to be reordered as before. We ignore this technicality for ease of presentation.

It then defines

$$z = y \oplus \alpha$$

Let j be an arbitrary index in $[n] \setminus \mathcal{I}$. $\mathcal{S}$ samples random strings $z_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the sum of the lengths of all the outputs of F, for $i \in [n] \setminus (\mathcal{I} \cup \{j\})$. $\mathcal{S}$ then computes

$$z_j = z \oplus \bigoplus_{i \in [n] \setminus \{j\}} z_i$$

and constructs

$$\overrightarrow{z} = (z_1, \ldots, z_n)$$

$\mathcal{S}$ computes $\omega_i^z \xleftarrow{\$} \widetilde{\mathsf{Open}}(\mathsf{state}_i, z_i)$ for each $i \in [n] \setminus \mathcal{I}$ and constructs

$$\overrightarrow{\omega^z} = (\omega_1^z, \ldots, \omega_n^z)$$

Note that, at this point, $\mathcal{S}$ has every value ever used in the protocol. $\mathcal{S}$ sends the tuple of values $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to P_i. Going forward, $\mathcal{S}$ simulates invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that involve corrupt parties as follows.

- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving another corrupt party, say P_j for $j \in \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to parties P_i and P_j.
- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_j for $j \in [n] \setminus \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to P_i.
- Suppose the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an *invalid* witness. $\mathcal{S}$ simply sends no response.
- Suppose the an honest party, say P_i for $i \in [n] \setminus \mathcal{I}$, triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving a corrupt party, say P_j for $j \in \mathcal{I}$. $\mathcal{S}$ sends $(w, (\omega_i^z, z_i, \omega_i^\pi, \pi_i, \omega_j^z, z_j, \omega_j^\pi, \pi_j))$ to P_j.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. It is easy to see that the view of $\mathcal{A}$ is indistinguishable in the execution of the protocol Π_{FMPC} and the simulation with $\mathcal{S}$, if $\left(\mathsf{Com}, \mathsf{Open}, \widetilde{\mathsf{Com}}, \widetilde{\mathsf{Open}}\right)$ is an honest-binding commitment scheme and $\mathcal{V}$ is a signature scheme. We therefore conclude that the protocol Π_{FMPC} securely computes $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model, as required. □

Remark. In the proof of Lemma 9, we ignore some annoying technicalities. For instance, the adversary may cause the honest parties to abort, will be unable to obtain its output but still pointlessly interact with some of the ideal functionalities. In the proof, however, the simulator would have aborted. We note that these details are not particularly enlightening and are of no consequence. One can deal with these sorts of attacks by asking the simulator to wait in these scenarios until the adversary says that it is done and then finally abort if it has to. Thus, we assume, for the purpose of the proof, that if the adversary forces the honest parties to abort in a situation where it will be unable to obtain its output, without loss of generality, it halts. Other examples of such technicalities are when the adversary sends "unexpected" messages, "incomplete" messages, etc. Note that such messages can be easily detected and ignored, and do not affect the protocol in any way.

4.5 Getting to the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model

Combining Lemmas 1, 4, 8 and 9, we obtain the following theorem.

Theorem 1. *Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, assuming the existence of one-way functions, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the presence of t-threshold adversaries for any $0 \leq t < n$ in the $(\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model.*

As discussed in Section 3, $\mathcal{F}_{\mathsf{2PC}}$, and hence $\mathcal{F}_{\mathsf{OT}}$, can be realized in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model. We thus have the following theorem.

Theorem 2. *Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, assuming the existence of one-way functions, there exists a protocol π which securely computes $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the presence of t-threshold adversaries for any $0 \leq t < n$ in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model.*

It is important to note that via this transformation, we have not introduced a need for the parties to have access to multiple instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ as opposed to one. This is because, in the protocol Π_{FMPC}, the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ will only be used to emulate the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. During this stage, we do not make any use of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. Once we are done with the single invocation of $\mathcal{F}_{\mathsf{MPC}}$, we only invoke the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. As a consequence, parties can reuse the same instance of $\mathcal{F}_{\mathsf{SyX}}$ to first emulate $\mathcal{F}_{\mathsf{OT}}$ and then as a complete $\mathcal{F}_{\mathsf{SyX}}$ functionality. We note that this however does increase the number of times the functionality is invoked.

5 Preprocessing $\mathcal{F}_{\mathsf{SyX}}$

In this section, we will describe how a pair of parties can "preprocess" an instance of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$.We first describe what we mean by "preprocess". What we would like to enable is the following. We already know that the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ allows the pair of parties to perform fair two-party computations. We would like to set up the $\mathcal{F}_{\mathsf{SyX}}$ functionality such that after a *single* invocation of the load phase, the two parties can perform an arbitrary (*apriori* unknown) polynomial number of fair two-party computations. Furthermore, if the parties are honest, they would not need to invoke the ideal functionality, that is, the "preprocessing" of the functionality is optimistic. Combining this with the protocol for fair multiparty computation in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model from Section 4, we are able to show how an arbitrary set of n parties in the point-to-point model that have pairwise access to the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that has been preprocessed, can perform an arbitrary (*apriori* unknown) polynomial number of fair multiparty computations. To begin with, we will assume that the n-parties are in the point-to-point model and develop a protocol in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model. We first provide some intuition for our construction.

5.1 Intuition

We first start with the 3-party case as a warm-up. Let P_1, P_2, and P_3 be the three parties, subsets (or all) of which would like to perform an unbounded (*apriori* unknown polynomial) number of secure function evaluations. For $i, j \in \{1, 2, 3\}$ with $i < j$, we have that parties P_i and P_j have access to the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. In particular, let $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ represent the instantiation of the $\mathcal{F}_{\mathsf{SyX}}$ functionality used by parties P_i, P_j. We wish to perform fair secure function evaluation of some 3-input 3-output functionality F.

Reduction to single output functionalities. Let $(y_1, y_2, y_3) \xleftarrow{\$} F(x_1, x_2, x_3)$ be the output of a function evaluation[20]. We define a new four input single output functionality F' such that

$$F'(x_1, x_2, x_3, z) = F^1(x_1, x_2, x_3) \| F^2(x_1, x_2, x_3) \| F^3(x_1, x_2, x_3) \oplus z = y_1 \| y_2 \| y_3 \oplus z$$

where $z = z_1 \| z_2 \| z_3$ and $|y_i| = |z_i|$ for all $i \in [3]$. The idea is that the party P_i will obtain $z' = F'(x_1, x_2, x_3, z)$ and z_i. Viewing $z' = z'_1 \| z'_2 \| z'_3$ where $|z'_i| = |z_i|$[21] for all $i \in [3]$, party P_i reconstructs its output as

$$y_i = z_i \oplus z'_i$$

Now, we may assume that the input of party P_i is (x_i, z_i) (or we can generate random z_is as part of the computation) which determines z. It thus suffices to consider fair secure function evaluation of single output functionalities.

Reduction to fair reconstruction. We will use ideas similar to [GIM+10, KVV16] where instead of focusing on fair secure evaluation of an arbitrary function, we only focus on fair reconstruction of an additive secret sharing scheme. The main idea is to let the parties run a secure computation protocol that computes the output of the secure function evaluation on the parties' inputs, and then additively secret shares the output. Given this step, fair secure computation then reduces to fair reconstruction of the underlying additive secret sharing scheme.

Instance and party independence. Looking ahead, as in Section 4, we will use the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ to perform fair reconstruction. In order to be able to preprocess the instances of the functionality for arbitrary reconstructions, what is being reconstructed must be independent of the secure function evaluation and, in particular, the inputs of the parties. Furthermore, it must also be independent of the specific parties that are performing the reconstruction. However, until now, we have been assuming that the output of the secure function evaluation on the parties' inputs is what is being reconstructed, which does not satisfy our requirements and hence would not allow preprocessing. In order to fix this, we assume that the output of the secure function evaluation on the parties' inputs is encrypted under a key and that key is what will be reconstructed fairly. Note that the key can be chosen independent of the secure function evaluation and the parties' inputs. We would also like it to be the case that even after reconstructing once, our preprocessing is valid. This would require that the preprocessing allows for the generation and fair reconstruction of multiple independent (to a computational adversary) keys, one for each secure function evaluation. Thus, what is actually done during the preprocessing phase is the following. Each pair of parties P_i and P_j initializes $\mathcal{F}^{i,j}_{\mathsf{SyX}}$. The function f_1 samples two random values $v_{i,j}, v_{j,i} \xleftarrow{\$} \{0,1\}^\lambda$ and computes the output of f_1 as

$$V_{i,j} = V_{j,i} = v_{i,j} \oplus v_{j,i}$$

along with commitments to these values to ensure that only these values are used by parties in protocols. This completes the description of f_1.

[20] This discussion can be trivially extended to function evaluations with two inputs as opposed to three.

[21] We may assume without loss of generality that the lengths of the outputs of each party are known beforehand.

The underlying additive secret sharing scheme. For the instance of secure function evaluation with identifier id, we sample a unique key, K_{id}, to encrypt the output y_{id} of the secure function evaluation. Let Enc denote the encryption algorithm of an encryption scheme. The parties would receive $\mathsf{ct}_{\mathsf{id}} = \mathsf{Enc}(y_{\mathsf{id}}; K_{\mathsf{id}})$ and then fairly reconstruct K_{id}. We use an independent additive secret sharing of the key K_{id} for each party. Let the shares be $k_{\mathsf{id},i,j}$ for $i, j \in [3]$. That is, it holds that

$$K_{\mathsf{id}} = \bigoplus_{j \in [3]} k_{\mathsf{id},i,j}$$

for each $i \in [3]$. We would like party P_i to reconstruct K_{id} by obtaining all shares $k_{\mathsf{id},i,j}$ for each $j \in [3]$. Initially, each party P_i is given $k_{\mathsf{id},i,i}$. Therefore, each party P_i only needs to obtain $k_{\mathsf{id},i,j}$ and $k_{\mathsf{id},i,j'}$ for $j, j' \neq i$. Looking ahead, we would use the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ to allow parties to fairly learn all their shares of K_{id}. However, since we are preprocessing the instances, the information needed to compute these shares must be independent of the instance of secure function evaluation. The value that the instance $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ would release fairly to parties P_i and P_j is $V_{i,j}$. Thus, party P_i additionally receives

$$\mathsf{ct}_{\mathsf{id},i,j} = \mathsf{Enc}(k_{\mathsf{id},i,j}; h_{\mathsf{id},i,j})$$

where

$$h_{\mathsf{id},i,j} = H(V_{i,j} \| \mathsf{id})$$

where H is a hash function (random oracle). The intuition is that the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ to allow parties to fairly learn the $V_{i,j}$s, and hence the $h_{\mathsf{id},i,j}$s and finally $k_{\mathsf{id},i,j}$s, thus fairly reconstructing K_{id}. It is important to note that using $V_{i,j}$s that are independent of the instance of secure function evaluation, we can fairly reconstruct, using per-instance (computationally independent) hash values $h_{\mathsf{id},i,j}$ generated using $V_{i,j}$s, per-instance (independent) encryption keys K_{id}.

An attempt at fair reconstruction via $\mathcal{F}_{\mathsf{SyX}}$. We assume that the secure function evaluation with identifier id provides the encryption $\mathsf{ct}_{\mathsf{id}}$ of the output y_{id} of the secure function evaluation. Additionally, party P_i receives $\mathsf{ct}_{\mathsf{id},i,j}$ for each $j \in [n]$ and $k_{\mathsf{id},i,i}$. From our earlier discussion, the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ allow parties to fairly learn the $V_{i,j}$s. In order to allow reuse of the preprocessing, however, the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ must only allow parties to fairly learn the $h_{\mathsf{id},i,j}$s. As a first attempt to ensure this, we require the secure function evaluation to also give party P_i a signature σ_i on id. That is, P_i receives

$$\left(\mathsf{ct}_{\mathsf{id}}, \{\mathsf{ct}_{\mathsf{id},i,j}\}_{j \in [3]}, k_{\mathsf{id},i,i}, \sigma_i\right)$$

We will have the parties fairly learn the $h_{\mathsf{id},i,j}$s using the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. We achieve this by setting the predicate $\phi_k(w)$ (for $k \in \{i, j\}$) to output 1 if and only if w consists of both signatures (σ_i, σ_j). That is, each instance $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ will accept the trigger $w_{i,j} = (\mathsf{id}, \sigma_i, \sigma_j)$. We define f_2 to simply output $h_{\mathsf{id},i,j}$ to both parties if $\phi_k(w) = 1$. Parties learn signatures of other parties by broadcasting their signatures and waiting for other parties to do so. If party P_i receives signatures from every other party, it can trigger every instance of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ it is involved in, thus learning $h_{\mathsf{id},i,j}$ for each $j \in [3]$ and finally learning K_{id}. Malicious parties may however not broadcast their signatures. Concretely, we have the following attack: Suppose P_1 is

honest while P_2 and P_3 are corrupt. P_2 and P_3 already know σ_2 and σ_3 and only need σ_1 to learn the output. P_1 broadcasts σ_1 while P_2 and P_3 do not broadcast σ_2 and σ_3. Finally, P_2 triggers the ideal functionality $\mathcal{F}_{\mathsf{SyX}}^{1,2}$ using (σ_1, σ_2) and learns the output. P_1, on the other hand, only learns σ_2 and hence does not learn the output.

Fair reconstruction via $\mathcal{F}_{\mathsf{SyX}}$. We fix the protocol sketch described above using a technique we developed for termination of the protocol described in Section 4. The protocol for reconstruction proceeds in $T = n$ rounds. In order to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ in some round $\tau \in [T]$, you must provide a proof that other instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least τ different parties have been triggered. Consider the first round τ in which P_i is an honest party and $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ is triggered for some j. If $\tau = 1$, then the single invocation already gives a proof that channels involving two parties, namely, i, j, have been triggered. Otherwise, by assumption, proofs of invocations of instances of $\mathcal{F}_{\mathsf{SyX}}$ involving τ different parties were needed to trigger $\mathcal{F}_{\mathsf{SyX}}^{i,j}$. But P_i is not one of these parties as τ is the first round in which $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ was triggered for any j. Consequently, P_i, on this invocation, obtains a proof that instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least $\tau + 1$ parties have been triggered, and can thus trigger all channels in round $\tau + 1$. The only gap in the argument is the case $\tau = T$. One can trivially see that since $\mathcal{F}_{\mathsf{SyX}}^{i,j}$ has not been triggered for any j, it is impossible to obtain a proof that instances of $\mathcal{F}_{\mathsf{SyX}}$ involving at least T different parties have been triggered.

Optimistic preprocessing. In the case where parties are honest, we can simply have the secure function evaluation provide the output instead of parties having to trigger their instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. We are guaranteed, by virtue of the fair reconstruction techniques dicussed thus far, that in the case where parties behave adversarially, the honest parties do have a way to obtain the output of the computation. In this way, in the optimistic setting, parties never have to trigger the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. Combined with the fact that the actual preprocessing phase is extremely simple, we see that this paradigm makes fair secure function evaluation just as efficient as secure function evaluation with abort in the optimisitic case.

Simulation. We look ahead for the issues that come up while trying to prove security, that is, during the simulation. The simulator will release to the adversary, the encryption of the output and encryptions of the adversary's shares of the key used to encrypt the output. Since the simulator does not know the output *apriori*, and does not know whether the adversary is going to abort the computation, in which case, no one knows the output, it has to produce encryptions that it can later *equivocate*. In this context, we use, not a regular encryption scheme, but non-interactive non-committing encryption commitments. In this case, the simulator can produce encryptions to garbage but can later decrypt them to be *valid* shares of the key and the actual output. The rest of the computations can be trivially simulated. The only other detail to be looked into is that of the clock. We need to determine if the adversary has decided to abort the computation, that is, if the adversary is going to receive the output of the computation of not. This is done by noticing if and when the adversary decides to trigger the instances of $\mathcal{F}_{\mathsf{SyX}}$ that involve honest parties. We know that if the adversary ever triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, then all parties will be in a position to receive the output. Thus, the simulator can simply run the adversary to determine whether it has decided to enable parties to obtain the output, in which the simulator would ask the trusted party to continue, or not, in which case the simulator would ask the trusted party to abort.

5.2 Protocol

We now present the protocol for preprocessing fair secure computation in the $(\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model.

STAGE I: PREPROCESSING

Preliminaries. P_a and P_b are two parties for $a, b \in [N]$ with $a < b$, where N is some universal upper bound on the number of parties; $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ represents the instantiation of the $\mathcal{F}_{\mathsf{SyX}}$ functionality used by parties P_a, P_b with time out round numbers $\mathsf{INPUT\_TIMEOUT} = \infty$ and $\mathsf{TRIGGER\_TIMEOUT} = \infty$; $(\mathsf{Com}, \mathsf{Open})$ is a commitment scheme; $H : \{0,1\}^* \to \{0,1\}^{L(\lambda)}$[22] is a random oracle; r denotes the current round number.

Protocol. The protocol $\Pi_{\mathsf{Preprocess}}$ proceeds as follows:

- Define the following functions.
  - Let $f_1^{a,b}$ be the function that takes no input, samples two strings $v_{a,b}, v_{b,a} \xleftarrow{\$} \{0,1\}^\lambda$ and computes and stores
    $$V_{a,b} = V_{b,a} = v_{a,b} \oplus v_{b,a}$$
    It also computes
    $$(c^v_{a,b}, \omega^v_{a,b}) \xleftarrow{\$} \mathsf{Com}(v_{a,b})$$
    and
    $$(c^v_{b,a}, \omega^v_{b,a}) \xleftarrow{\$} \mathsf{Com}(v_{b,a})$$
    Finally, it outputs $(v_{a,b}, c^v_{a,b}, \omega^v_{a,b}, c^v_{b,a})$ to party P_a and $(v_{b,a}, c^v_{a,b}, c^v_{b,a}, \omega^v_{b,a})$ to party P_b.
  - Let $\phi_1^{a,b}$ be the function that takes as input a witness w and parses
    $$w = \left(\mathsf{id}, t, \overrightarrow{c^\pi}, \overrightarrow{\omega^\pi}, \overrightarrow{\pi}, \overrightarrow{\mathsf{ind}}\right)$$
    It checks that:
    * $\mathsf{id} \in \{0,1\}^\lambda$
    * t is a valid round number and $t < r$
    * $\left|\overrightarrow{\mathsf{ind}}\right| = \left|\overrightarrow{\omega^\pi}\right| = |\overrightarrow{\pi}| = r - t$
    * $\overrightarrow{\mathsf{ind}}$ consists of distinct indices in $\left[|\overrightarrow{c^\pi}|\right]$.
    * $\mathsf{Open}\left(c^\pi_{\mathsf{ind}_j}, \omega^\pi_j, \pi_j\right) = 1$ for every $j \in [r-t]$.

    If all of these checks pass, then $\phi_1^{a,b}$ outputs 1 and otherwise it outputs 0.
  - Let $\phi_2^{a,b}$ be identical to $\phi_1^{a,b}$.

[22] The following type-check must and can be performed: $L(\lambda)$ is the length of the key that will be used to encrypt shares of a key that will be used to encrypt the output of the secure function evaluations to be performed in the future.

– Let $f_2^{a,b}$ be the function that takes as input w where w is as above, and outputs

$$h_{\mathsf{id},a,b} = h_{\mathsf{id},b,a} = H\left(V_{a,b} \| \mathsf{id} \| t \| \overrightarrow{c^{\pi}}\right)$$

- Parties P_a, P_b run the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$, providing the same input $(\perp, f)$, where

$$f = \left(f_1^{a,b}, f_2^{a,b}, \phi_1^{a,b}, \phi_2^{a,b}\right)$$

If parties P_a and P_b receive their outputs $v_{a,b}$ and $v_{b,a}$ respectively, we say that their preprocessing phase has been successfully completed.

STAGE II: FAIR SECURE FUNCTION EVALUATION

Preliminaries. $S \subseteq [N]$, where n is some universal upper bound on the number of parties, and $|S| = n$; F is the n-input n-output functionality to be computed; x_i is the input of party P_i for $i \in S$; $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ represents the instantiation of the $\mathcal{F}_{\mathsf{SyX}}$ functionality used by parties P_a, P_b with time out round numbers $\mathsf{INPUT\_TIMEOUT} = \infty$ and $\mathsf{TRIGGER\_TIMEOUT} = \infty$ for $a < b$, where $a, b \in S$; $(\mathsf{Com}, \mathsf{Open})$ is an commitment scheme; $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is non-interactive non-committing encryption scheme; $H : \{0,1\}^* \to \{0,1\}^{L(\lambda)}$[23] is a random oracle; r denotes the current round number.

Protocol. The protocol $\Pi_{\mathsf{FMPC\text{-}preprocess}}$ proceeds as follows:

- Party P_i for every $i \in S$ ensures that its preprocessing phases with every other party P_j for $j \in S \setminus \{i\}$ have been successfully completed. If not, party P_i aborts. Otherwise, it has obtained values $\left\{v_{i,j}, c_{i,j}^{v,i}, \omega_{i,j}^{v,i}, c_{j,i}^{v,i}\right\}_{j \in S \setminus \{i\}}$ as outputs from its preprocessing phases with every other party.
- Define F' to the be the following n-input n-output functionality: On input $\overrightarrow{X} = (X_1, \ldots, X_n)$:
  - Parse
$$X_i = \left(x_i, t_i, \left\{v_{i,j}, c_{i,j}^{v,i}, \omega_{i,j}^{v,i}, c_{j,i}^{v,i}\right\}_{j \in S \setminus \{i\}}\right)$$
  - Check that $t = t_i = t_j$ for all $i, j \in S$. If not, abort.
  - Check that $c_{i,j}^{v,i} = c_{i,j}^{v,j}$ and that $\mathsf{Open}(c_{i,j}^{v,i}, \omega_{i,j}^{v,i}, v_{i,j}) = 1$ for all $i, j \in S$ with $i \neq j$. If not, abort.
  - Sample a random identifier $\mathsf{id} \in \{0,1\}^{\lambda}$ for this instance of fair secure function evaluation.
  - Let $(y_1, \ldots, y_n) = F(x_1, \ldots, x_n)$ and let
$$y = y_1 \| \ldots \| y_n$$

[23]The following type-check must and can be performed: $L(\lambda)$ is the length of the randomness needed to generate a key long enough to encrypt shares of a key long enough to encrypt the output of the secure function evaluations to be performed in the future.

Sample random strings $\alpha_i \xleftarrow{\$} \{0,1\}^*$ such that $|\alpha_i| = |y_i|$ for each $i \in [n]$. Let

$$\alpha = \alpha_1 \| \ldots \| \alpha_n$$

Let $z = y \oplus \alpha$.

- Sample a random encryption key-pair $(\mathsf{pk}, \mathsf{sk})$ by invoking $\mathsf{Gen}(1^\lambda)$. Compute an encryption of the output

$$\mathsf{ct} = \mathsf{Enc}(z; \mathsf{pk})$$

- Sample n random additive n-out-of-n secret sharings $\{k_{1,j}\}_{j \in S}, \ldots, \{k_{n,j}\}_{j \in S}$ of sk such that

$$\mathsf{sk} = \bigoplus_{j \in [n]} k_{i,j}$$

for every $i \in S$.

- Sample random proof values $\pi_1, \ldots, \pi_n \xleftarrow{\$} \{0,1\}^\lambda$. Compute commitments along with their openings $(c_i^\pi, \omega_i^\pi) \xleftarrow{\$} \mathsf{Com}(\pi_i)$ to each of the proof values π_i for each $i \in [n]$. Let

$$\overrightarrow{c^\pi} = (c_1^\pi, \ldots, c_n^\pi)$$

- Compute

$$h_{i,j} = h_{j,i} = H\left(V_{i,j} \| \mathsf{id} \| t \| \overrightarrow{c^\pi}\right)$$

where

$$V_{i,j} = V_{j,i} = v_{i,j} \oplus v_{j,i}$$

for every $i, j \in [n]$ with $i \neq j$.

- Sample encryption key-pairs $(\mathsf{pk}_{i,j}, \mathsf{sk}_{i,j})$ by invoking $\mathsf{Gen}(1^\lambda; h_{i,j})$ for every $i, j \in [n]$ with $i \neq j$. Compute

$$\mathsf{ct}_{i,j} = \mathsf{Enc}(k_{i,j}; \mathsf{pk}_{i,j})$$

for every $i, j \in [n]$ with $i \neq j$.

- In the first stage of output delivery[24], party P_i receives output

$$\left(\alpha_i, \mathsf{id}, t, \mathsf{ct}, \{\mathsf{ct}_{i,j}\}_{j \in S \setminus \{i\}}, k_{i,i}, \omega_i^\pi, \pi_i\right)$$

for each $i \in S$.

- In the second stage of output delivery, party P_i receives output $\overrightarrow{c^\pi}$ for each $i \in S$.
- Finally party P_i receives the output z for each $i \in S$.

[24]While this would require setting up additional notation, for ease of presentation, we suppress formally defining security of multi-stage primitives. In our case, the functionality F' is a two-stage functionality. While defining security with abort for F', which is all we will need for this work, we consider security with abort for each stage. That is, the adversary obtains its output for the first stage and then decides whether the honest parties may receive their output for the first stage. If not, neither the adversary nor the honest parties obtain their output for the second stage. If yes, the honest parties receive their output for the first stage and the adversary receives its output for the second stage, following which it decides if the honest parties may receive their output for the second stage.

- For each $i \in S$, party P_i estimates the round number t_i when the secure function evaluation of F' using the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ will be complete. It then constructs

$$X_i = \left(x_i, t_i, \left\{v_{i,j}, c_{i,j}^{v,i}, \omega_{i,j}^{v,i}, c_{j,i}^{v,i}\right\}_{j \in S \setminus \{i\}}\right)$$

- The parties invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ with inputs $\{(X_i, F')\}_{i \in S}$. If the ideal functionality returns $\perp$ after the first stage to party P_i, then P_i aborts for any $i \in [n]$[25]. Otherwise, party P_i receives output

$$\left(\alpha_i, \mathsf{id}, t, \mathsf{ct}, \{\mathsf{ct}_{i,j}\}_{j \in S \setminus \{i\}}, k_{i,i}, \omega_i^\pi, \pi_i\right)$$

for each $i \in S$. If the ideal functionality returns z after the third stage, in round t, to party P_i for any $i \in S$, then P_i parses z as $z_1 \| \ldots \| z_n$ where $|z_j| = |y_j|$ for all $j \in [n]$ and computes $y_i = z_{\mathsf{ind}_i} \oplus \alpha_i$ to obtain the output of the computation, where ind_i is the index of i in S. If the ideal functionality returned $\perp$ after the third stage but did not return $\perp$ after the second stage to party P_i, then party P_i also receives $\overrightarrow{c^\pi}$ by round t. Then, in round $r = t+1$, P_i uses the witness

$$w = \left(\mathsf{id}, t, \overrightarrow{c^\pi}, \omega_i^\pi, \pi_i, i\right)$$

to invoke the trigger phase of each instance of $\mathcal{F}_{\mathsf{SyX}}$ it is involved in. Once all such instances of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_i have been triggered, P_i obtains $h_{\mathsf{id},i,j}$ for every $j \in S \setminus \{i\}$. Using these values, it computes encryption key-pairs $(\mathsf{pk}_{i,j}, \mathsf{sk}_{i,j})$ by invoking $\mathsf{Gen}(1^\lambda; h_{\mathsf{id},i,j})$ for every $j \in S \setminus \{i\}$. Then, it computes

$$\mathsf{k}_{i,j} = \mathsf{Dec}(\mathsf{ct}_{i,j}; \mathsf{sk}_{i,j})$$

for every $j \in S \setminus \{i\}$. Then, it computes

$$\mathsf{sk} = \bigoplus_{j \in S} k_{i,j}$$

and

$$z = \mathsf{Dec}(\mathsf{ct}; \mathsf{sk})$$

Finally, P_i parses z as $z_1 \| \ldots \| z_n$ where $|z_j| = |y_j|$ for all $j \in [n]$ and computes the value $y_i = z_{\mathsf{ind}_i} \oplus \alpha_i$ to obtain its output, where ind_i is the index of i in S. Otherwise, if the ideal functionality returned $\perp$ after both the second and third stages, the protocol proceeds as follows.

- If $r > t + n$, abort. Otherwise, wait until $r > t$. While $t + 1 \leq r \leq t + n$,

[25] In the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, let $\pi_{F'}$ denote the protocol for the functionality F' defined in Lemma 8. The parties execute $\pi_{F'}$. If the execution of $\pi_{F'}$ aborts, we are assuming that all (honest) parties are aware of the round when the execution of $\pi_{F'}$ aborts, that is, when the adversary has decided to abort the execution of $\pi_{F'}$. Since we are working in the $\mathcal{F}_{\mathsf{MPC}}$-hybrid model, we know that in the ideal model, this is the case when the honest parties receive $\perp$ as their output. If we assume that in the case when the the adversary decides to let the honest parties obtain their outputs, no honest party ever receives $\perp$, this could be used to identify the scenario when the adversary has decided to abort the execution of $\pi_{F'}$. Thus, we could, in principle, replace this instruction with: If party P_i receives $\perp$ as its output, it aborts. Furthermore, since we are considering the case of *unanimous abort*, if the adversary has decided to abort the execution of $\pi_{F'}$, all honest parties abort the protocol.

– If party P_a for $a \in S$ has not received the output of the computation and an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a is first triggered in round $t+1 \leq r \leq t+n-1$ using a "good" witness, it triggers each instance of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in during round $r+1$ using the output out it receives from the instance of $\mathcal{F}_{\mathsf{SyX}}$ as follows:

  * If
  $$\mathsf{out}_1 = \left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \overrightarrow{\omega^{\pi}}, \overrightarrow{\pi}, \overrightarrow{\mathsf{ind}}\right)$$
  it prepares the witness
  $$w = \left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \overrightarrow{\omega^{\pi}}', \overrightarrow{\pi}', \overrightarrow{\mathsf{ind}}'\right)$$
  where
    · $|\overrightarrow{\pi}'| = r+1,\ \overrightarrow{\pi}'|_{[r]} = \overrightarrow{\pi}|_{[r]},\ \pi'_{r+1} = \pi_a$
    · $\left|\overrightarrow{\omega^{\pi}}'\right| = r+1,\ \overrightarrow{\omega^{\pi}}'\Big|_{[r]} = \overrightarrow{\omega^{\pi}}\Big|_{[r]},\ {\omega^{\pi}_{r+1}}' = \omega^{\pi}_a$
    · $\left|\overrightarrow{\mathsf{ind}}'\right| = r+1,\ \overrightarrow{\mathsf{ind}}'\Big|_{[r]} = \overrightarrow{\mathsf{ind}}\Big|_{[r]},\ \mathsf{ind}'_{r+1} = a$

  Once all instances of $\mathcal{F}_{\mathsf{SyX}}$ involving party P_a have been triggered, P_a obtains $h_{\mathsf{id},a,b}$ for every $b \in S \setminus \{a\}$. Using these values, it computes encryption key-pairs $(\mathsf{pk}_{a,b}, \mathsf{sk}_{a,b})$ by invoking $\mathsf{Gen}(1^{\lambda}; h_{\mathsf{id},a,b})$ for every $b \in S \setminus \{a\}$. Then, it computes
  $$\mathsf{k}_{a,b} = \mathsf{Dec}(\mathsf{ct}_{a,b}; \mathsf{sk}_{a,b})$$
  for every $b \in S \setminus \{a\}$. Then, it computes
  $$\mathsf{sk} = \bigoplus_{b \in S} k_{a,b}$$
  and
  $$z = \mathsf{Dec}(\mathsf{ct}; \mathsf{sk})$$
  Finally, P_a parses z as $z_1 \| \ldots \| z_n$ where $|z_j| = |y_j|$ for all $j \in [n]$ and computes the value $y_a = z_{\mathsf{ind}_a} \oplus \alpha_a$ to obtain its output, where ind_a is the index of a in S.

5.3 Proof sketch of Security

We sketch the proof of security of the above protocol. The correctness of the computation of the functionality F' follows by definition from the correctness of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Furthermore, we have that at the end of the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, either all honest parties *unanimously abort* or all honest parties *unanimously continue*. Thus, assuming that $\mathcal{F}_{\mathsf{MPC}}$ did not abort, and that all honest parties continue, every party receives the output of the first stage of F'. If the honest parties also receive the output of the third stage of F', there is nothing to prove and correctness is obvious. This is the optimistic setting. We now consider the case that all honest parties obtained the output of the first and second stages of F' but not the third stage of F'. Note that a valid witness to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ in round $t+1 \leq r \leq t+n$ consists of, apart from id and t, a set of proof values and openings $\left(\overrightarrow{\omega^{\pi}}, \overrightarrow{\pi}\right)$ of size $r-t$. By valid, we mean that the values of $\left(\mathsf{id}, t, \overrightarrow{c^{\pi}}\right)$ are the ones picked during the execution of F'. Triggering the instances of $\mathcal{F}_{\mathsf{SyX}}$ with other values will produce some output but will not be useful because the

values $h_{i,j}$ are computed by hashing $V_{i,j}$ along with $\left(\mathsf{id}, t, \overrightarrow{c^{\pi}}\right)$. Also note that if and only if a party does manage to have all the instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in triggered with valid witnesses during the course of the protocol, it is able to reconstruct its output correctly. If the honest party P_i obtained the output of the second stage of F', then P_i is able to prepare the valid witness $w = \left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \omega_i^{\pi}, \pi_i, i\right)$ and trigger all instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round $r = t + 1$, and is thus able to reconstruct its output correctly. Suppose the honest parties receive the output of the first stage but not the second or third stages of F'. It is now possible that the adversary has received its output corresponding to the first and second stages of F'. At this point, if the adversary wishes to learn the output, it must do so by obtaining $k_{i,j}$s for $i \in \mathcal{I}$ and $j \in S \setminus \mathcal{I}$. For this, it must be the case that for some corrupt party P_i with $i \in \mathcal{I}$, all instances of $\mathcal{F}_{\mathsf{SyX}}$ involving P_i and an honest party must be triggered, either by the honest parties or by P_i. Since none of the honest parties possess valid witnesses to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ (as they do not have $\overrightarrow{c^{\pi}}$), the corrupt party must be the first to trigger. P_i would only have at most $|\mathcal{I}|$ openings to commitments in $\overrightarrow{c^{\pi}}$ and thus, if the adversary learns the output, it must be the case that P_i triggers an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an honest party P_j for some $j \in S \setminus \mathcal{I}$ in round $r \leq t + |\mathcal{I}| \leq t + n - 1$. At this point, P_j learns $\overrightarrow{c^{\pi}}$ and a set of openings to commitments in $\overrightarrow{c^{\pi}}$ corresponding to some set of $r - t$ indices $\overrightarrow{\mathsf{ind}}$ in $S \setminus \{j\}$. Combining this information with id, t and the opening $\left(\omega_j^{\pi}, \pi_j\right)$ received from the output of the first stage of F', P_j is able to prepare a valid witness to trigger all instances of $\mathcal{F}_{\mathsf{SyX}}$ that it is involved in in round $r + 1 \leq t + n$ and is thus able to reconstruct its output correctly. Finally, consider any honest party P_j for $j \in S \setminus \mathcal{I}$. If no instance of $\mathcal{F}_{\mathsf{SyX}}$ involving P_j is triggered in a round $r \leq t + n - 1$, then for all $i \neq j$, party P_i does not learn $h_{i,j}$ and hence, does not learn $k_{i,j}$. Without $k_{i,j}$, P_i does not learn sk and hence does not learn its output. In particular, this means that the adversary does not learn its output. Thus, if the adversary learns its output, that for all honest parties P_j for $j \in S \setminus \mathcal{I}$, some instance of $\mathcal{F}_{\mathsf{SyX}}$ involving P_j is triggered in a round $r \leq t + n - 1$ which in turns means that P_j learns its output as well. This completes the proof of security.

5.4 Security

We now prove the following lemma.

Lemma 10. *If* $(\mathsf{Com}, \mathsf{Open})$ *is a commitment scheme,* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a non-interactive non-committing encryption scheme and* H *a random oracle, then the protocols* $\Pi_{\mathsf{Preprocess}}, \Pi_{\mathsf{FMPC\text{-}preprocess}}$ *securely preprocess for and compute an arbitrary (polynomial) number of instances of* $\mathcal{F}_{\mathsf{MPC}}$ *with fairness in the* $(\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$*-hybrid model.*

Proof. Let $\mathcal{A}$ be an adversary attacking the execution of the protocols described in Section 5.2 in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model. We construct an ideal-model adversary $\mathcal{S}$ in the ideal model of type fair. Let $\mathcal{I}$ be the set of corrupted parties. If $\mathcal{I}$ is empty, then there is nothing to simulate. $\mathcal{S}$ begins by simulating the first stage, namely, preprocessing.

STAGE I: PREPROCESSING

$\mathcal{S}$ has to simulate the invocations of the load phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that involve corrupt parties. Here, $\mathcal{S}$ behaves as the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$. Recall that

the type of $\mathcal{F}_{\mathsf{SyX}}$ is $\mathsf{g.d.}$. For any $a, b \in [n]$ with $a < b$ and $a \in \mathcal{I}$ and $b \in [n] \setminus \mathcal{I}$, if $\mathcal{A}$ instructs P_a to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ with no input, $\mathcal{S}$ executes $f_1^{a,b}$ as defined in $\Pi_{\mathsf{Preprocess}}$. $\mathcal{S}$ picks two strings $v_{a,b}, v_{b,a} \xleftarrow{\$} \{0,1\}^\lambda$ and computes and stores

$$V_{a,b} = V_{b,a} = v_{a,b} \oplus v_{b,a}$$

It also computes

$$(c_{a,b}^v, \omega_{a,b}^v) \xleftarrow{\$} \mathsf{Com}(v_{a,b})$$

and

$$(c_{b,a}^v, \omega_{b,a}^v) \xleftarrow{\$} \mathsf{Com}(v_{b,a})$$

Finally, it forwards $(v_{a,b}, c_{a,b}^v, \omega_{a,b}^v, c_{b,a}^v)$ to the adversary. $\mathcal{S}$ behaves symmetrically if for any $a, b \in [n]$ with $a < b$ and $b \in \mathcal{I}$ and $a \in [n] \setminus \mathcal{I}$, if $\mathcal{A}$ instructs P_b to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$. The final case to consider is if for any $a, b \in [n]$ with $a < b$ and $a, b \in \mathcal{I}$, if $\mathcal{A}$ instructs P_a, P_b to invoke the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$ with no input, $\mathcal{S}$ executes $f_1^{a,b}$ as defined in $\Pi_{\mathsf{Preprocess}}$. $\mathcal{S}$ picks two strings $v_{a,b}, v_{b,a} \xleftarrow{\$} \{0,1\}^\lambda$ and computes and stores

$$V_{a,b} = V_{b,a} = v_{a,b} \oplus v_{b,a}$$

It also computes

$$(c_{a,b}^v, \omega_{a,b}^v) \xleftarrow{\$} \mathsf{Com}(v_{a,b})$$

and

$$(c_{b,a}^v, \omega_{b,a}^v) \xleftarrow{\$} \mathsf{Com}(v_{b,a})$$

Finally, it forwards $v_{a,b}, v_{b,a}, c_{a,b}^v, \omega_{a,b}^v, c_{b,a}^v, \omega_{b,a}^v$ to the adversary. If in any of the invocations, the adversary instructs a corrupt party to invoke the load phase of an instance of $\mathcal{F}_{\mathsf{SyX}}$ incorrectly, $\mathcal{S}$ simply aborts simulating the load phase of this particular instance of $\mathcal{F}_{\mathsf{SyX}}$. At the end of this phase, let $\mathsf{PreprocessSuccess}$ denote the set of all $\{a, b\}$ with $a < b$ such that P_a and P_b successfully invoked the load phase of $\mathcal{F}_{\mathsf{SyX}}^{a,b}$.

STAGE II: FAIR SECURE FUNCTION EVALUATION

Let F be the n-input n-output functionality to be computed. Let S be the set of n parties that are participating in this instance of fair secure function evaluation. If $S \cap \mathcal{I} = \phi$, then there is nothing to simulate. $\mathcal{S}$ then checks that for every $a, b \in S$ with $a < b$ and either $a \in [N] \setminus \mathcal{I}$ or $b \in [N] \setminus \mathcal{I}$, $\{a, b\} \in \mathsf{PreprocessSuccess}$. If not, then $\mathcal{S}$ aborts this instance of fair secure function evaluation as the honest parties would have done so in the real execution as well. $\mathcal{S}$ then begins by simulating the first step of the protocol, namely, the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Here, $\mathcal{S}$ behaves as the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. Recall that the type of $\mathcal{F}_{\mathsf{MPC}}$ is abort. $\mathcal{S}$ obtains the inputs $\{(X_i, f_i)\}_{i \in \mathcal{I}}$ of the corrupted parties from $\mathcal{A}$. If $(X_i, f_i) = \mathsf{abort}$ for any $i \in \mathcal{I}$, $\mathcal{S}$ forwards $\{(X_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, receives $\perp$ as the output of all parties, which it forwards $\mathcal{A}$. Suppose $(X_i, f_i) \neq \mathsf{abort}$ for all $i \in \mathcal{I}$. If there exists a $j \in \mathcal{I}$ such that $f_j \neq F'$ as defined in protocol $\Pi_{\mathsf{FMPC\text{-}preprocess}}$, $\mathcal{S}$ forwards $\{(X_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, which aborts, and then aborts this instance of fair secure function evaluation. If there exists a $j \in \mathcal{I}$ such that (X_j, f_j) is not of the specified format, $\mathcal{S}$ replaces

(X_j, f_j) with a default value. Going forward, we assume that for all $i \in \mathcal{I}$, (X_i, f_i) is well-formed, that is,

$$X_i = \left(x_i, t_i, \left\{v^*_{i,j}, c^{*,v,i}_{i,j}, \omega^{*,v,i}_{i,j}, c^{*,v,i}_{j,i}\right\}_{j \in S \setminus \{i\}}\right)$$

$$X_i = \left(x_i, t_i, \{v^*_{i,j}\}_{j \in S \setminus \{i\}}\right)$$

and that $f_i = F'$ as defined in $\Pi_{\mathsf{FMPC\text{-}preprocess}}$. Note that $v^*_{i,j}$ may not equal $v_{i,j}$ as picked by $\mathcal{S}$ in the simulation of the preprocessing stage, $c^{*,v,i}_{i,j}$ may not equal $c^{v,i}_{i,j}$, and so on.

$\mathcal{S}$ now needs to simulate the outputs received by the corrupted parties from the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. $\mathcal{S}$ estimates the round number t when the secure function evaluation of F' using the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ will be complete. It checks that $t = t_i$ for all $i \in \mathcal{I}$ and aborts this instance of fair secure function evaluation otherwise. It then checks that $c^{*,v,i}_{i,j} = c^{v,i}_{i,j}$, $c^{*,v,i}_{j,i} = c^{v,i}_{j,i}$ and that $\mathsf{Open}\left(c^{v,i}_{i,j}, \omega^{*,v,i}_{i,j}, v^*_{i,j}\right) = 1$ for all $i \in \mathcal{I}$ and $j \in S \setminus \{i\}$. If not, it aborts this instance of fair secure function evaluation. Note that if these checks pass, we also have that $v^*_{i,j} = v_{i,j}$ and $\omega^{*,v,i}_{i,j} = \omega^{v,i}_{i,j}$ for all $i \in \mathcal{I}$ and $j \in S \setminus \{i\}$. It then samples a random identifier $\mathsf{id} \in \{0,1\}^\lambda$ for this instance of secure function evaluation. For each $i \in S$, $\mathcal{S}$ samples a random string $\alpha_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the length of the ith output of F. Let

$$\alpha = \alpha_1 \| \ldots \| \alpha_n$$

$\mathcal{S}$ samples a random encryption key-pair $(\mathsf{pk}, \mathsf{sk})$ by invoking $\mathsf{Gen}(1^\lambda)$ and a ciphertext ct representing the encryption of the output z under a secret key sk. Note that z is not known to $\mathcal{S}$ at this point. However, since $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a non-committing encryption scheme, $\mathcal{S}$ can sample ct and later *equivocate* it. For each $i \in \mathcal{I}$, $\mathcal{S}$ samples random additive n-out-of-n secret sharings $k_{i,1}, \ldots, k_{i,n}$ of sk such that

$$\mathsf{sk} = \bigoplus_{j \in [n]} k_{i,j}$$

$\mathcal{S}$ samples random proof values $\pi_1, \ldots, \pi_n \xleftarrow{\$} \{0,1\}^\lambda$ and compute commitments along with their openings $(c^\pi_i, \omega^\pi_i) \xleftarrow{\$} \mathsf{Com}(\pi_i)$ to each of the proof values π_i. Let

$$\overrightarrow{c^\pi} = (c^\pi_1, \ldots, c^\pi_n)$$

and

$$\overrightarrow{\omega^\pi} = (\omega^\pi_1, \ldots, \omega^\pi_n)$$

and

$$\overrightarrow{\pi} = (\pi_1, \ldots, \pi_n)$$

$\mathcal{S}$ then computes

$$h_{i,j} = h_{j,i} = H\left(V_{i,j} \| \mathsf{id} \| t \| \overrightarrow{c^\pi}\right)$$

where

$$V_{i,j} = V_{j,i} = v_{i,j} \oplus v_{j,i}$$

for every $i \in \mathcal{I}$ and $j \in S$ with $i \neq j$. $\mathcal{S}$ then samples encryption key-pairs $(\mathsf{pk}_{i,j}, \mathsf{sk}_{i,j})$ by invoking $\mathsf{Gen}(1^\lambda; h_{i,j})$ for every $i \in \mathcal{I}$ and $j \in S$ with $i \neq j$, and computes

$$\mathsf{ct}_{i,j} = \mathsf{Enc}(k_{i,j}; \mathsf{pk}_{i,j})$$

for every $i \in \mathcal{I}$ and $j \in S$ with $i \neq j$. Thus, the simulator constructs the first stage output

$$\left(\alpha_i, \mathsf{id}, t, \mathsf{ct}, \{\mathsf{ct}_{i,j}\}_{j \in S \setminus \{i\}}, k_{i,i}, \omega_i^{\pi}, \pi_i\right)$$

for each $i \in \mathcal{I}$ and forwards it to $\mathcal{A}$. If $\mathcal{A}$ then sends abort, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, which it forwards $\mathcal{A}$. Otherwise, $\mathcal{A}$ responds with $\mathsf{continue}$. $\mathcal{S}$ then forwards $\overrightarrow{c^{\pi}}$ to $\mathcal{A}$.

Case A. The adversary lets the honest parties obtain the output of the second stage of F'. In this case, $\mathcal{A}$ has responded with $\mathsf{continue}$ after receiving $\overrightarrow{c^{\pi}}$. At this point, all parties are going to obtain their outputs. $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness. It receives the corrupt parties outputs, namely, $\{y_i\}_{i \in \mathcal{I}}$. $\mathcal{S}$ chooses the outputs of the honest party completely at random, that is, it samples random strings $y_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the length of the ith output of F, for $i \in [n] \setminus \mathcal{I}$. $\mathcal{S}$ then constructs

$$y = y_1 \| \ldots \| y_n$$

It then defines

$$z = y \oplus \alpha$$

$\mathcal{S}$ now ensures that $\mathsf{Dec}(\mathsf{ct}; \mathsf{sk}) = z$. $\mathcal{S}$ then sends the outputs of the corrupt parties, namely, $\{y_i\}_{i \in \mathcal{I}}$, to $\mathcal{A}$. At this point, $\mathcal{S}$ has completed simulating the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. If $\mathcal{A}$ responds with $\mathsf{continue}$, then $\mathcal{S}$ simply terminates. Otherwise, in round $r = t + 1$, it simulates the honest parties triggering all the instances of $\mathcal{F}_{\mathsf{SyX}}$ they are involved in with the corrupt parties and hence for every $i \in \mathcal{I}$ and $j \in S \setminus \mathcal{I}$, $\mathcal{S}$ sends P_i (the adversary $\mathcal{A}$) the set of values $\left(\left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \omega_j^{\pi}, \pi_j, j\right), h_{i,j}\right)$.

Case B. The adversary does not let the honest parties obtain the output of the second stage of F'. In this case, $\mathcal{A}$ has responded with abort after receiving $\overrightarrow{c^{\pi}}$. At this point, $\mathcal{S}$ has completed simulating the invocation of the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$. We first discuss how $\mathcal{S}$ simulates certain invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that the adversary instructs the corrupt parties to trigger.

- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving another corrupt party, say P_j for $j \in \mathcal{I}$, with a *valid* witness. $\mathcal{S}$ sends $(w, h_{i,j})$ to parties P_i and P_j.
- Suppose the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an *invalid* witness. $\mathcal{S}$ simply sends no response.

Suppose the adversary does not instruct a corrupt party, say P_i for some $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_j for some $j \in S \setminus \mathcal{I}$, with a *valid* witness and the round counter exceeds $t + n$, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i \in \mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness, with (x_j, f_j) replaced with abort for some $j \in \mathcal{I}$, receives $\perp$ as the output of all parties, and aborts itself. Otherwise, at the first instant $r \leq t + n - 1$ that the adversary instructs a corrupt party P_i for $i \in \mathcal{I}$ to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party P_j for $j \in S \setminus \mathcal{I}$ with

a *valid* witness $w = \left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \overrightarrow{\omega^{\pi}}, \overrightarrow{\pi}, \overrightarrow{\mathsf{ind}}\right)$, $\mathcal{S}$ forwards $\{(x_i, f_i)\}_{i\in\mathcal{I}}$ to the trusted party computing $\mathcal{F}_{\mathsf{MPC}}$ with fairness. It receives the corrupt parties outputs, namely, $\{y_i\}_{i\in\mathcal{I}}$. $\mathcal{S}$ chooses the outputs of the honest party completely at random, that is, it samples random strings $y_i \xleftarrow{\$} \{0,1\}^*$ of length equal to the length of the ith output of F, for $i \in [n] \setminus \mathcal{I}$. $\mathcal{S}$ then constructs

$$y = y_1 \| \ldots \| y_n$$

It then defines

$$z = y \oplus \alpha$$

$\mathcal{S}$ now ensures that $\mathsf{Dec}(\mathsf{ct}; \mathsf{sk}) = z$. $\mathcal{S}$ then sends $(w, h_{i,j})$ to P_i. In round $r+1$, it simulates the P_j triggering all the instances of $\mathcal{F}_{\mathsf{SyX}}$ they are involved in with the corrupt parties and hence for every $i \in \mathcal{I}$, $\mathcal{S}$ sends P_i the set of values

$$\left(\left(\mathsf{id}, t, \overrightarrow{c^{\pi}}, \overrightarrow{\omega^{\pi}}', \overrightarrow{\pi}', \overrightarrow{\mathsf{ind}'}\right), h_{i,j}\right)$$

where

- $|\overrightarrow{\pi}'| = r+1,\ \overrightarrow{\pi}'|_{[r]} = \overrightarrow{\pi}|_{[r]},\ \pi'_{r+1} = \pi_j$
- $\left|\overrightarrow{\omega^{\pi}}'\right| = r+1,\ \overrightarrow{\omega^{\pi}}'\Big|_{[r]} = \overrightarrow{\omega^{\pi}}\Big|_{[r]},\ {\omega^{\pi}_{r+1}}' = \omega^{\pi}_j$
- $\left|\overrightarrow{\mathsf{ind}'}\right| = r+1,\ \overrightarrow{\mathsf{ind}'}\Big|_{[r]} = \overrightarrow{\mathsf{ind}}\Big|_{[r]},\ \mathsf{ind}'_{r+1} = j$

Finally, for every $k \in S \setminus \mathcal{I}$ such that P_k did not have an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving itself and some corrupt party triggered in round r, $\mathcal{S}$ simulates P_k triggering all instances of $\mathcal{F}_{\mathsf{SyX}}$ involving P_k and every corrupt party in round $r+2$. Note that by the existence of j, k, $|\mathcal{I}| \leq n-2$ and hence $r \leq t+n-2$, or, $r+2 \leq t+n$. To simulate the triggers, $\mathcal{S}$ sends along the appropriate valid witnesses and $h_{i,k}$s. Note that this is possible to do as by this point, $\mathcal{S}$ has all the values it will ever need in the simulation. Going forward, $\mathcal{S}$ simulates invocations of the trigger phases of the instances of the ideal functionality $\mathcal{F}_{\mathsf{SyX}}$ that the adversary instructs corrupt parties to trigger as follows.

- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving another corrupt party, say P_j for $j \in \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, h_{i,j})$ to parties P_i and P_j.
- Suppose the adversary instructs a corrupt party, say P_i for $i \in \mathcal{I}$, to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ involving an honest party, say P_j for $j \in S \setminus \mathcal{I}$, with a *valid* witness w, $\mathcal{S}$ sends $(w, h_{i,j})$ to P_i.
- Suppose the adversary instructs a corrupt party to trigger an instance of $\mathcal{F}_{\mathsf{SyX}}$ with an *invalid* witness. $\mathcal{S}$ simply sends no response.

Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs. It is easy to see that the view of $\mathcal{A}$ is indistinguishable in the execution of the protocols $\Pi_{\mathsf{Preprocess}}, \Pi_{\mathsf{FMPC\text{-}preprocess}}$ and the simulation with $\mathcal{S}$, if $(\mathsf{Com}, \mathsf{Open})$ is a commitment scheme$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a non-interactive non-committing encryption scheme and H a random oracle. We therefore conclude that the protocols $\Pi_{\mathsf{Preprocess}}, \Pi_{\mathsf{FMPC\text{-}preprocess}}$ securely preprocess for and compute an arbitrary (polynomial) number of instances of $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the $(\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model, as required. □

Remark. In the proof of Lemma 9, we ignore some annoying technicalities. For instance, the adversary may cause the honest parties to abort, will be unable to obtain its output but still pointlessly interact with some of the ideal functionalities. In the proof, however, the simulator would have aborted. We note that these details are not particularly enlightening and are of no consequence. One can deal with these sorts of attacks by asking the simulator to wait in these scenarios until the adversary says that it is done and then finally abort if it has to. Thus, we assume, for the purpose of the proof, that if the adversary forces the honest parties to abort in a situation where it will be unable to obtain its output, without loss of generality, it halts. Other examples of such technicalities are when the adversary sends "unexpected" messages, "incomplete" messages, etc. Note that such messages can be easily detected and ignored, and do not affect the protocol in any way.

5.5 Getting to the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model

Combining Lemmas 1, 8 and 10, we obtain the following theorem.

Theorem 3. *Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, assuming the existence of one-way permutations, there exists a protocol π in the programmable random oracle model which securely preprocesses for and computes an arbitrary (polynomial) number of instances of $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the presence of t-threshold adversaries for any $0 \leq t < n$ in the $(\mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{SyX}})$-hybrid model.*

As discussed in Section 3, $\mathcal{F}_{\mathsf{2PC}}$, and hence $\mathcal{F}_{\mathsf{OT}}$, can be realized in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model. We thus have the following theorem.

Theorem 4. *Consider n parties $P_1, \ldots, P_n$ in the* point-to-point *model. Then, assuming the existence of one-way permutations, there exists a protocol π in the programmable random oracle model which securely preprocesses for and computes an arbitrary (polynomial) number of instances of $\mathcal{F}_{\mathsf{MPC}}$ with fairness in the presence of t-threshold adversaries for any $0 \leq t < n$ in the $\mathcal{F}_{\mathsf{SyX}}$-hybrid model.*

References

[ABMO15] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete characterization of fairness in secure two-party computation of boolean functions. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 199–228, 2015.

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In *Financial Cryptography and Data Security - FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*, pages 105–121, 2014.

[ADMM16] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. *Commun. ACM*, 59(4):76–84, 2016.

[Ash14] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 291–316, 2014.

[ASW97] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997.*, pages 7–17, 1997.

[ASW00] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 421–439, 2014.

[BLOO11] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 277–296, 2011.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 236–254, 2000.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19, 1988.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 19–40, 2001.

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 719–728, 2017.

[CHK05] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 150–168, 2005.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 565–582, 2003.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *J. Cryptology*, 30(4):1157–1186, 2017.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.

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

[FGH+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam D. Smith. Detectable byzantine agreement secure against faulty majorities. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002*, pages 118–126, 2002.

[FGMO05] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[FGMvR02] Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, pages 482–501, 2002.

[GHKL11] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. ACM*, 58(6):24:1–24:37, 2011.

[GIM+10] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *Theory of Cryptography, 7th Theory of Cryptography*

Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings, pages 91–108, 2010.

[GJ02] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*, pages 168–182, 2002.

[GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 19–35, 2009.

[GK10] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, pages 157–176, 2010.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 179–186, 2011.

[GMPY11] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *J. Cryptology*, 24(4):615–658, 2011.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GP03] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*, pages 190–207, 2003.

[GV87] Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 73–86, 1987.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 572–591, 2008.

[Kat07] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 11–20, 2007.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31, 1988.

[KRS16] Ranjit Kumaresan, Srinivasan Raghuraman, and Adam Sealfon. Network oblivious transfer. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 366–396, 2016.

[KVV16] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 406–417, 2016.

[LMPS04] Matt Lepinski, Silvio Micali, Chris Peikert, and Abhi Shelat. Completely fair SFE and coalition-safe cheap talk. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 1–10, 2004.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 111–126, 2002.

[Pin03] Benny Pinkas. Fair secure two-party computation. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 87–105, 2003.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 260–289, 2017.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85, 1989.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 387–394, 1990.

[SGK19] Rohit Sinha, Sivanarayana Gaddam, and Ranjit Kumaresan. Luciditee: Policy-based fair computing at scale. *IACR Cryptology ePrint Archive*, 2019:178, 2019.

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 222–232, 2006.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.