Phoenix: Secure Computation in an Unstable Network with Dropouts and Comebacks

Ivan Damgård
Aarhus University, Denmark

Daniel Escudero
J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, New York, NY, USA

Antigoni Polychroniadou
J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, New York, NY, USA

Abstract

We consider the task of designing secure computation protocols in an unstable network where honest parties can drop out at any time, according to a schedule provided by the adversary. This type of setting, where even honest parties are prone to failures, is more realistic than traditional models, and has therefore gained a lot of attention recently. Our model, Phoenix, enables a new approach to secure multiparty computation with dropouts, allowing parties to drop out and re-enter the computation on an adversarially-chosen schedule and without assuming that these parties receive the messages that were sent to them while being offline - features that are not available in the existing models of Sleepy MPC (Guo et al., CRYPTO '19), Fluid MPC (Choudhuri et al., CRYPTO '21) and YOSO (Gentry et al. CRYPTO '21). Phoenix does assume an upper bound on the number of rounds that an honest party can be off-line – otherwise protocols in this setting cannot guarantee termination within a bounded number of rounds; however, if one settles for a weaker notion, namely guaranteed output delivery only for honest parties who stay on-line long enough, this requirement is not necessary.

In this work, we study the settings of perfect, statistical and computational security and design MPC protocols in each of these scenarios. We assume that the intersection of online-and-honest parties from one round to the next is at least $2t + 1$, $t + 1$ and 1 respectively, where t is the number of (actively) corrupt parties. We show the intersection requirements to be optimal. Our (positive) results are obtained in a way that may be of independent interest: we implement a traditional stable network on top of the unstable one, which allows us to plug in *any* MPC protocol on top. This approach adds a necessary overhead to the round count of the protocols, which is related to the maximal number of rounds an honest party can be offline. We also present a novel, perfectly secure MPC protocol in the preprocessing model that avoids this overhead by following a more "direct" approach rather than first building a stable network and then using existing protocols. We introduce our network model in the UC-framework, show that the composition theorem still holds, and prove the security of our protocols within this setting.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases Secure Multiparty Computation, Unstable Networks

Digital Object Identifier 10.4230/LIPIcs.ITC.2023.7

Related Version *Full Version*: `https://eprint.iacr.org/2021/1376`

Acknowledgements This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of

1 Introduction

Secure Multiparty Computation (MPC) is a technique that allows multiple mutually distrustful parties to compute a function of their inputs without leaking anything else beyond the output of the computation. Most protocols in the MPC literature assume that the parties communicate over a *synchronous network*, that is, all the parties have access to a global clock. This allows the parties to follow the protocol specification based on time. A protocol under such network model proceeds in *communication rounds*, each of which has a fixed duration and where each party can send a message to each other party.

Synchronous networks are natural for describing protocols and may make sense in many contexts, but the model is not resilient to sudden slowdowns: if a party fails to send a message within the allocated time for a specific round, this message will not be taken into account, and what is worse, in the context of an active adversary this will be considered a deviation from the protocol specification. Hence an honest party who accidentally misses a deadline will be classified as corrupt. The first problem with this is that an MPC protocol can only tolerate a certain maximal number of corruptions. Tagging parties as corrupt because of natural network issues that may appear in practice leaves little room for real corruptions. For instance, MPC over unstable mobile network connections or denial of service attacks might consume all the corruptions we can handle. The second problem is that once a party is tagged as corrupt, the protocol may now reveal her secret inputs, which seems unfair if the party was actually honest but suffered a random network delay. An alternative model is an *asynchronous network*, where the parties are not assumed to have a clock anymore. This modeling is more resilient to the type of attacks described above since the communication network allows for parties to be slow and no deadlines are set. However, this model comes with its own set of issues since, when dealing with an active adversary, the parties cannot distinguish a delayed message sent by a slow party, from a message that an actively corrupt party decided not to send in the first place. As a result asynchronous protocols tend to tolerate a smaller number of corruptions [3], and, what is worse, an asynchronous protocol cannot guarantee that all honest parties get to contribute inputs to the computation.

Therefore, it seems to be a better approach of considering an imperfect synchronous network where the adversary is allowed to cause some parties to go offline temporarily, and require protocols to not classify such parties as corrupt. In such a setting we may still hope to get (1) optimal corruption thresholds, (2) allow all parties to contribute input, and (3) guarantee termination at a certain time. A series of works has studied MPC in different variant of this model, see Section 1.3 and also the Full Version of this work for a detailed comparison of prior works. However, it is still an open question whether we can have MPC protocols with optimal security and corruption thresholds in the most adversarial, but also most realistic setting, that we call an *unstable network* in this paper. In such a network parties go offline and come back according to an adversarially chosen schedule (not a schedule prescribed by the protocol specifications), and parties are not assumed to receive messages sent while they were offline. Not receiving messages while being offline introduces more challenges since one can only rely on the parties that are online in the current round and were also online in the previous round.

1.1 Unstable Networks

As we have mentioned, there are multiple attempts in the literature to model what a realistic network where parties can dropout and return should represent concretely. In this work we are interested in studying the setting of MPC over an *unstable network*, which is a type of synchronous network we introduce where, in contrast to a *stable network* (*i.e.* a standard synchronous network), the adversary can choose in each round a subset of parties that will be offline in that specific round, and hence may not be able to send or receive messages. This models honest parties dropping out in that specific round, possibly due to network errors or malicious attacks, which serves to represent certain failures like weak mobile connections or DDoS attacks. We remark that our "timing model" is still synchronous in that the parties have a synchronized clock and know which current protocol step is being run, but crucially, they may drop and re-join in every round.

Given that over an unstable network the set of offline parties can be different in every round, an MPC protocol in such setting must allow parties to rejoin the computation after being offline. Furthermore, these parties may not know they are under network attack, so a missing message can mean that either (1) they are under attack, (2) the sender is under attack, or (3) the sender is malicious. This ambiguity is crucial to maintain a strong and realistic model, but it turns out to heavily complicate protocol design. This is further accentuated by the fact that, in an unstable network – and in stark contrast with previous networking models for tolerating dropouts – parties who rejoin the computation do not necessarily receive the messages sent to them while being offline, which is an important property to model settings like peer-to-peer networks where the parties do not count on "always-running" servers that can queue messages for them. This is an important scenario to consider in practice, since one might argue that counting on communication servers that never fail can be equivalent to assuming parties who never drop.

1.2 Our Contribution

In this work we formally introduce the notion of an *unstable network*, which we believe to be an appropriate communication model to capture realistic settings where parties join and leave an ongoing computation according to a potentially adversarial schedule. Our first contribution lies in the formal definition of this novel networking model, and we present a rigorous treatment of this notion within the confines of the UC framework, which in particular involves re-proving the UC theorem to ensure that composability still holds in this new setting.

Our second contribution – and where most of our work is devoted – consists of a full characterization of what types of security properties (*i.e.* perfect, statistical or computational) can be achieved by MPC protocols over unstable networks in terms of the underlying adversarial schedule. More precisely, we show that the minimum amount of honest parties that remain online from one round to the next is the crucial metric that determines whether a given level of security is attainable or not, and we show both impossibility and correspondingly matching feasibility results for each one of the three security notions: computational, statistical and perfect security. We believe our novel model and initial set of results open an exciting and interesting research direction on the design of MPC protocols over realistic networks.

In order to discuss what the characterizations above are in detail, let us introduce some notation. Let n be the number of parties and let t be the number of corrupt parties. Let $\mathcal{O}_r$ denote the set of online parties in round r, and let $\mathcal{H}$ denote the set of honest parties. Our goal is to determine if we can construct MPC protocols for an unstable network which enjoy

	Perfect security	Statistical security	Computational security
Passive adversary $\lvert\mathcal{O}_r \cap \mathcal{O}_{r+1}\rvert \geq$	$t+1$	$t+1$	1
Active adversary $\lvert\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}\rvert \geq$	$2t+1$	$t+1$	1

Figure 1 Overview of the required intersection sizes for each setting considered in this paper. The result for statistical and passive security follows from the one for perfect and passive security.

the same security guarantees as protocols over a stable network and if so, what constraints we must assume on the unstable network to make this happen. To be able to talk more concretely about this, we will say that two protocols π, π' are *equivalent* if they tolerate the same number of corruptions, achieve the same type of security (computational/statistical/perfect) and the same security guarantee (security with abort/fairness/guaranteed output delivery). Our first set of results is as follows:

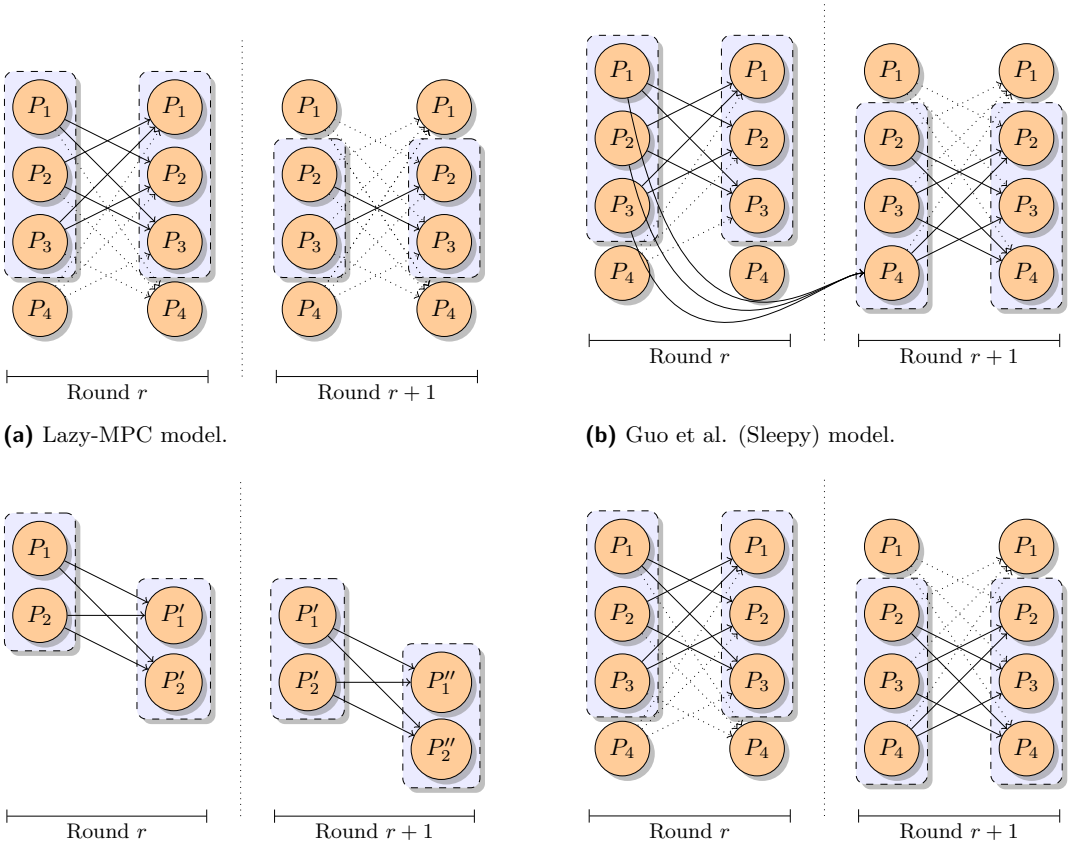
Perfect security. (Section 2) Given any perfectly secure synchronous MPC protocol against t corruptions, we construct an equivalent protocol over an unstable network, assuming that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for all $r > 0$. Furthermore, this condition is required for any MPC protocol with perfect security to exist over an unstable network.

Statistical security. (Section 3) Given any statistically secure synchronous MPC protocol against t corruptions, we construct an equivalent protocol over an unstable network, assuming that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ for all $r > 0$. This condition is required for any MPC protocol with statistical security to exist over an unstable network.

Computational security. (Full Version) Given any computationally secure synchronous MPC protocol secure against t corruptions, we construct an equivalent protocol over an unstable network, assuming that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for all $r > 0$ (and, for malicious security, assuming a PKI and public key encryption). The intersection condition is required for any computationally secure MPC protocol to exist over an unstable network.

An overview of the intersection sizes required in each of the settings considered in our work is presented in Fig 1. Notice that our results imply a necessary tradeoff between instability and corruptions: taking perfect security as an example, it is well known that we must have $n \geq 3t+1$ to have perfect security at all. So for a maximal value of t, we have only $2t+1$ honest parties, and the result above then says that all honest parties must stay online all the time. On the other hand, as we increase n above $3t+1$, an increasing number of honest players can be sent offline. Also, note that even if the (minimal) assumptions in our results say that a minimum amount of parties must stay online from one round to the next, this does not imply that any *particular party* stays online for more than one round. This makes protocol design considerably difficult, as in particular, the following scenario may occur: a given party can be offline for a while, not receiving any messages, then it is set to be online in a given round, but the scheduling[1] is such that this party only receives messages in this round after he or she has sent their own message, so this message can only depend on outdated information this party learned before going offline. Furthermore, this party may be

[1] As in the standard synchronous network, the adversary is allowed to choose the ordering of the messages received by honest parties.



(a) Lazy-MPC model.

(b) Guo et al. (Sleepy) model.

(c) Fluid MPC model.

(d) Our model.

Figure 2 Our model compared to other models in the literature. Parties inside the marked region are online, and messages represented by dashed arrows are dropped. In Lazy-MPC, Fig. 2a, the parties cannot return. In the model of Guo et al., Fig. 2b, the parties can return but it is assumed they receive the messages sent to them while they were offline. In the Fluid-MPC model, Fig. 2c, in each round the set of parties who send messages may differ from the set of parties who receive these messages, but the identities of these parties must be known by the protocol. In our model, Fig. 2d, the parties can return to the computation and it is not assumed that they receive the messages sent to them while they were offline.

set to be offline for the next round immediately after sending their message, which makes the contribution of this party to the protocol meaningless. The honest parties in $\mathcal{O}_r \cap \mathcal{O}_{r+1}$ are these who are able to receive the messages in round r, and simultaneously are able to send a derived message in round $r+1$, so having enough honest parties in this intersection is what enables us to design MPC protocols in this difficult networking setting.

1.3 Related Work

In what follows we discuss some of the works that study a similar problem to the one we address in this work. The description in this section is relatively lightweight, and we defer a more detailed analysis to the Full Version.

Fail-stop adversaries that may cause some parties to stop during a computation were considered for the first time in [5], but this and subsequent works assume parties know when a given party fail-stopped, plus these parties are not able to return the computation. A recent

model in [1] considers an adversary that can set parties to be offline at any round, but as before these parties cannot return the computation, plus that work focuses on computational assumptions, making use of strong homomorphic encryption tools. In the "sleepy model" of [8] parties who drop can return. However, a crucial difference with our model is that, in our case, parties who return after being offline may not receive the messages sent to them before becoming onine, while in [8] these parties (who are not "offline" but "slow") do receive these messages. This makes the problem considerably easier, plus the authors consider only computational assumptions. Finally, in [2, 12] a new model is considered where the set of parties can change dynamically from one round to the next. In that work, the set of "online" parties in a given round is not adversarially chosen, but rather set in advance and used in the design of the protocol. As a result, this work may not model adversarial attacks to the underlying network, and may be less realistic in these settings. Furthermore, the protocol in [2], although statistically secure, only achieves security with abort. Our compilation-based techniques allows us to transfer any result in the standard synchronous setting (*e.g.* protocols with guaranteed output delivery) to the unstable networking setting.

The "You Only Speak Once" (YOSO) model for MPC is introduced in [7]. Our model assumes a somewhat less powerful adversary who must allow a physical party to come back after being offline, while in [7] this adversary can take a party down as soon as they speak, and progress is guaranteed by means of assigning roles "on-the-fly" in certain randomized fashion. Their model does not allow for perfect security, while in our case, on top of achieving much easier protocol design, we can obtain information theoretic security based only on point-to-point secure channels, and we allow for termination such that all parties can provide input and get output. Finally, the "constrained parties" and "full-omission parties" from [10] and [13] are such that whose messages are selectively blocked by the adversary, as in our setting. However, in these works the adversary choses the subset of offline parties at the beginning of the protocol execution, while in our case this subset can change adaptively as the protocol is run. This is in fact one of the main sources of difficulties when designing protocols in our setting, since a party who is "full-omission-corrupt" can stop being so, and non-corrupted parties can later on become full-omission-corrupt. We remind the reader to visit the Full Version for a more detailed discussion on related work.

We present in Figure 2 a more graphical comparison of our model with respect to the works of [1, 8, 2].

1.4 Preliminaries and Organization

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of all parties, and $\mathcal{H}$ be the set of honest parties. We assume that the adversary corrupts t out of the n parties. Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| > n$. Due to space limitations we assume background on Shamir secret-sharing, with details given in Section A and in the Full Version. For our results in the computational setting, we assume the existence of a CPA-secure public key encryption scheme $(\mathsf{enc}, \mathsf{dec})$, and a EUF-CMA signature scheme $(\mathsf{sign}, \mathsf{verify})$. The formal definitions of these primitives and their security is standard and can be found in any modern book in Cryptography (e.g. [9]).

Unstable Networks

Now we provide the different functionalities we will make use of in our work. More thorough definitions and considerations, including the proof of the composition theorem, are given in the Full Version. Our timing model is synchronous, meaning there parties have a global clock and there is a known upper bound Δ on the time it takes for a message to be transmitted

between any pair of parties. The communication pattern proceeds in rounds, identified with integers $1, 2, 3, \ldots$, each taking Δ time and consisting of all parties sending messages to each other at the beginning of each round, and receiving some of these messages in a way we will specify later before the end of that round. We use $\mathcal{F}_{\mathsf{StableNet}}$ to denote the functionality that models a stable network in which all of the messages between honest parties are always delivered. We also consider a family of functionalities $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^{n}$ that models a synchronous channel from P_i to P_j only. In this work we take the following approach in order to obtain MPC over unstable networks: first, we instantiate the $\mathcal{F}_{\mathsf{StableNet}}$ functionality on top of an unstable network, that is, we design a way for each pair of parties to communicate reliably over an unstable network. Then, we take off-the-shelf MPC protocols set in the stable/synchronous model and compose them with our protocol for emulating the stable network, to get MPC protocols that are set in the unstable networking model. In the Full Version we elaborate on which protocols we use, and on why the modular approach sketched above works via the composition theorem. In this version we focus on instantiating the stable networking model only.

An unstable network is formalized as a functionality, that we denote by $\mathcal{F}_{\mathsf{UnstableNet}}$. In each round, the functionality proceeds as follows: (1) At the beginning of the round the environment, denoted by $\mathcal{Z}$, specifies a subset of parties $\mathcal{O}_r \subseteq \mathcal{P}$; (2) For every $P_i, P_j \in \mathcal{O}_r \cap \mathcal{H}$, the functionality delivers messages sent from P_i to P_j in the given round; (3) For every P_i and P_j with either one of the two parties in $(\mathcal{O}_r)^c \cap \mathcal{H}$, the environment can choose whether to drop the message sent from P_i to P_j in the given round.

If the adversary is allowed to set a given party P_i as offline forever, it is obvious that no stable channel to or from P_i could be instantiated. To address this we introduce the *B-assumption*, which states that the maximum amount of consecutive rounds that a party can be offline is B. The protocols we present here require this assumption in order to produce output, but in the Full Version we discuss alternative protocols that do not require this during the entire computation.

2 Instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with Perfect Security

In this section we take care of instantiating the functionality for a stable network with perfect security. First, in Section 2.1 we discuss the simplest setting of passive security. Then, in Section 2.2 we extend this to active security, while retaining perfect simulation.

2.1 Passive Security

Assuming a passive adversary, and assuming that $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t + 1$ for all $r > 0$, our protocol to instantiate $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security is obtained as follows. At every round, P_S tries to secret-share its message m towards all the parties, which succeeds in the round in which P_S comes online. In the following rounds, the parties try to send their shares of m to P_R, who is able to get them when it comes online, and hence is able to reconstruct m. The only missing step is that, when P_S secret-shares m, only the parties online in the current round are able to receive the shares. To alleviate this issue, the parties in each round "transfer" the shared secret to the parties that are online in the next round. This is done via a simple resharing protocol. Details are in Protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf},\mathsf{passive}}(P_S, P_R, m)$.

We remark that, although it is not explicitly written in the protocol description, whenever it is written that P_i sends a message to P_j, this is done by invoking the $\mathcal{F}_{\mathsf{UnstableNet}}$ functionality.

Protocol $\Pi^{\mathsf{perf,passive}}_{\mathsf{StableNet}}(P_S, P_R, m)$

- On input (m), P_S samples random elements $c_{ij} \in \mathbb{F}$ for $i, j = 0, \ldots, t$, subject to $c_{0,0} = m$ and $c_{ij} = c_{ji}$, and lets $f(x, y) = \sum_{i,j=0}^{t} c_{ij} x^i y^j$. Then, in rounds $1, \ldots, B$, P_S sends $f(x, i)$ to each party P_i.
- Every party P_i initializes a variable $\mathtt{f}_i = \perp$. In rounds $1, \ldots, 2B$, P_i does the following:
  - If $\mathtt{f}_i$ is not set already:
    - If P_i receives a polynomial $f_i(x) = f(x, i)$ from P_S, then P_i sets $\mathtt{f}_i = f_i$.
    - Else, if P_i receives messages $m_j \in \mathbb{F}$ from at least $t+1$ parties P_j, then P_i sets $\mathtt{f}_i$ to be the polynomial $f_i(x)$ such that $f_i(j) = m_j$ for the first $t+1$ messages m_j.
  - If $\mathtt{f}_i \neq \perp$, then P_i sends $\mathtt{f}_i(j)$ to each party P_j and $\mathtt{f}_i(0)$ to P_R.
- In rounds $B+1, \ldots, 2B$, P_R does the following: If P_R receives messages $m_j \in \mathbb{F}$ from at least $t+1$ parties P_j, then P_R computes the polynomial $f_0(x)$ such that $f_0(j) = m_j$ for the first $t+1$ messages m_j, and outputs $m = f_0(0)$.

▶ **Theorem 1.** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$ for every $r > 0$. Then, protocol $\Pi^{\mathsf{perf,passive}}_{\mathsf{StableNet}}(P_R, P_S)$ instantiates the functionality $\mathcal{F}^{P_R \to P_S}_{\mathsf{StableNet}}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model with perfect security against an adversary passively corrupting $t < n$ parties.*

Proof. We claim that, in an execution of protocol $\Pi^{\mathsf{perf,passive}}_{\mathsf{StableNet}}(P_R, P_S)$, P_R learns the value of m at the end of the interaction, and the adversary does not learn the value of m, unless P_S or P_R are passively corrupt.

To see this, let $r_S \in \{1, \ldots, B\}$ be the smallest value such that $P_S \in \mathcal{O}_{r_S}$, which exists due to the B-assumption. We claim the following invariant: at the end of every round r with $r_S \leq r \leq 2B$, each $P_i \in \mathcal{O}_r$ has $\mathtt{f}_i \neq \perp$, and these polynomials satisfy that $\mathtt{f}_i(x) = f(x, i)$, where $f(x, y)$ is the polynomial sampled by P_S at the beginning of the protocol. To see this we argue inductively. First, notice that the invariant holds for $r = r_S$ given that parties $P_i \in \mathcal{O}_{r_S}$ receive this directly from P_S. For the inductive step assume that the invariant holds for some round r, that is, each party $P_i \in \mathcal{O}_r$ has set its variable $\mathtt{f}_i$, and $\mathtt{f}_i(x) = f(x, i)$. In particular, this is held by the parties in $\mathcal{O}_r \cap \mathcal{O}_{r+1}$, so each party P_i in this set sends $\mathtt{f}_i(j)$ to every other party P_j in round $r+1$, which is received by the parties in $\mathcal{O}_{r+1}$. Since $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$, we see that each party $P_j \in \mathcal{O}_{r+1}$ receives at least $t+1$ values $\mathtt{f}_i(j) = f(j, i) = f(i, j)$, which enables P_j to interpolate $f(x, j)$, which is set to $\mathtt{f}_j$. We see then that the invariant is preserved.

Finally, let $r_R \in \{B+1, \ldots, 2B\}$ be a round in which $P_R \in \mathcal{O}_{r_R}$, which is guaranteed from the B-assumption. By the invariant, the parties in $\mathcal{O}_{r_R - 1}$ have set their variables $\mathtt{f}_i$ at the end of round $r_R - 1$ correctly, so in particular the parties in $\mathcal{O}_{r_R-1} \cap \mathcal{O}_{r_R}$ will send $\mathtt{f}_i(0) = f(0, i)$ to P_R in round $\mathcal{O}_{r_R}$. Since there are at least $t+1$ such parties, this means that P_R gets at least $t+1$ values $f(0, i)$, which allows P_R to interpolate $m = f(0, 0)$.

The fact that the adversary does not learn anything if both P_S and P_R are honest follows from the fact that its view is limited to t polynomials of the form $f(x, i)$, which look uniformly random. We remark that with the analysis above, it is straightforward to set up a simulator $\mathcal{S}$ for the proof. ◀

Optimality of $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$

Now we show that, in order to instantiate $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ with perfect security against a passive adversary, the assumption that the adversary's schedule satisfies $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$ in every round r is necessary. However, we have to be careful about what this should actually mean:

consider an adversary who respects the B-assumption and breaks the intersection condition in one, or some finite number of rounds. Now, if the sender happens to start our protocol for sending a message after the last bad round, it will clearly succeed. So we cannot hope to show that communication between sender and receiver is impossible, unless we consider an adversary who keeps breaking the intersection condition "for ever". So we construct below an adversary that breaks this condition once every B rounds, and by doing so it is able to learn the message sent by an honest sender using *any* instantiation of $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$.

Assume the existence of an implementation of $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security that tolerates an adversary that schedules the parties as follows: (1) The adversary chooses a set $A_1 \subset \mathcal{P}$ such that $|A_1| = t+1$, $P_S \in A_1$ and $\mathcal{O}_{k \cdot B} = A_1$ for $k > 0$, and (2) the adversary chooses a set A_2 such that $A_1 \cup A_2 = \mathcal{P}$ and $|A_1 \cap A_2| \leq t$ such that $P_R \in A_2$, $P_S \notin A_2$ and $\mathcal{O}_r = A_2$ for every r that is not of the form $k \cdot B$. Notice that this scheduling respects the B-assumption. Now, suppose that P_R learns the output in round $r_R = k \cdot B + \ell$ for some k and ℓ with $1 \leq \ell \leq B$. Since during the whole protocol P_R only hears from the parties in A_2, this means that these parties together had enough information to reconstruct the secret in round r_R. However, these parties only hear from P_S through $A_1 \cap A_2$, which means that at a given point in the protocol this set had enough information to reconstruct the secret. This is a contradiction since $|A_1 \cap A_2| \leq t$ and $P_S, P_R \notin A_1 \cap A_2$, and due to privacy no set of at most t parties that does not contain the sender nor the receiver can reconstruct the message.

We remark that this lower bound rules out general MPC over unstable networks when $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \leq t$, since $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ is a particular case of general MPC. This can be seen even more clearly since what the lower bound actually shows is that, if the minimum intersection size is not met, then the "state" of the computation is either leaked, or lost, which rules out general MPC. Indeed, our perfectly secure protocol from Section B, which does not use $\mathcal{F}_{\mathsf{StableNet}}$ directly, still requires $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$ to hold for every round.

2.2 Active Security

The construction we presented in the previous section does not carry over to the actively secure setting, given that a corrupted party P_i is not forced to send correct evaluations $\mathtt{f}_i(j)$. In this section we show an extension of this protocol that rules out this case. We assume that, for every r, $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$, which should be contrasted with the weaker condition in the passively secure setting of $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$. The use of a larger threshold allows us to make use of *error correction*, which allows the parties to reconstruct the right polynomials at each step of the protocol regardless of any incorrect value sent by corrupt parties.

The protocol for active security, Protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R, m)$, is similar to Protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf,passive}}(P_S, P_R, m)$, except for the following crucial change: when each P_i collects the messages $m_j \in \mathbb{F}$ for P_j received in a given round, only if there are at least $2t+1$ such messages, P_i performs error correction on these to reconstruct a polynomial $f_i(x)$ such that $f_i(j) = m_j$ for every received message m_j, and if this succeeds, then P_i sets $\mathtt{f}_i = f_i$. Similarly, only if P_R receives at least $2t+1$ messages $\{m_j\}_j$, then P_R performs error correction to recover a polynomial $f_0(x)$ such that $f_0(j) = m_j$ for every received message m_j, and if this succeeds then P_R outputs $m = f_0(0)$.

▶ **Theorem 2.** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every $r > 0$. Then, protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_R, P_S)$ instantiates the functionality $\mathcal{F}_{\mathsf{StableNet}}^{P_R \to P_S}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model with perfect security against an adversary actively corrupting $t < n/3$ parties.*[2]

[2] In principle the restriction is simply $t < n$, but we have that $n - t = |\mathcal{H}| \geq |\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$, so $n \geq 3t+1$.

Proof. We claim that, in an execution of protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_R, P_S)$, P_R learns the value of m at the end of the interaction, and, if P_R and P_S are honest, the adversary does not learn the value of m.

To see this, let $r_S \in \{1, \ldots, B\}$ be the smallest value such that $P_S \in \mathcal{O}_{r_S}$. We claim the following invariant: at the end of every round r with $r_S \leq r \leq 2B$, each $P_i \in \mathcal{O}_r \cap \mathcal{H}$ has $\mathtt{f}_i \neq \perp$, and these polynomials satisfy that $\mathtt{f}_i(x) = f(x, i)$, where $f(x, y)$ is the polynomial sampled by P_S at the beginning of the protocol. We use induction in order to show that the invariant holds. First, notice that the invariant is true for $r = r_S$ given that parties $P_i \in \mathcal{O}_{r_S} \cap \mathcal{H}$ receive the polynomial directly from P_S. For the inductive step assume that the invariant holds for some round r, and we show that it holds for round $r + 1$. By the hypothesis assumption each party $P_i \in \mathcal{O}_r \cap \mathcal{H}$ has set its variable $\mathtt{f}_i$, and $\mathtt{f}_i(x) = f(x, i)$. In particular, this holds for the parties in $\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}$, which means that each party P_i in this set sends $\mathtt{f}_i(j)$ to every other party P_j in round $r + 1$, which is received by the parties in $\mathcal{O}_{r+1}$. Since $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t + 1$, each party $P_j \in \mathcal{O}_{r+1} \cap \mathcal{H}$ receives at least $2t + 1$ correct values $\mathtt{f}_i(j) = f(j, i) = f(i, j)$. Even if P_j receives more shares, some of them potentially incorrect, P_j can still recover $f(x, j)$ via error correction, as instructed by the protocol. We see then that for P_j $\mathtt{f}_j = f(x, j)$, so the invariant is preserved.

Now, let $r_R \in \{B + 1, \ldots, 2B\}$ be a round in which $P_R \in \mathcal{O}_{r_R}$. By the invariant, the parties in $\mathcal{O}_{r_R - 1}$ have set their variables $\mathtt{f}_i$ at the end of round $r_R - 1$ correctly, so in particular the parties in $\mathcal{O}_{r_R - 1} \cap \mathcal{O}_{r_R} \cap \mathcal{H}$ will send $\mathtt{f}_i(0) = f(0, i)$ to P_R in round $\mathcal{O}_{r_R}$. Since there are at least $2t + 1$ such parties, this means that P_R gets at least $2t + 1$ correct values $f(0, i)$, which allows P_R to error-correct $m = f(0, 0)$. The fact that the adversary does not learn anything if both P_S and P_R are honest follows as in the proof of Theorem 1.

As with the case with passive security, the analysis above enables the construction of a simulator $\mathcal{S}$ for the proof in a straightforward manner. The main complication with the actively secure setting in contrast to the scenario with passive security is that a corrupt P_S may send inconsistent shares in the first round in which it becomes online. However, in this case, $\mathcal{S}$ can simply emulate the protocol exactly as the honest parties would do, and check if the receiver would be able to error-correct or not at the end of the execution. Only if this is the case, $\mathcal{S}$ would make use of the change command in the $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ functionality to set P_S's message to be the one that is recovered by P_R, and then it would clock-out P_R if P_R is honest. ◀

Optimality of $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t + 1$

As in Section 2.1, we show that the bound $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ is necessary for essentially all rounds by presenting an adversary that breaks the correctness of any perfectly secure implementation of $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ against active adversaries, by using a scheduling that breaks the condition above while still respecting the B-assumption.

The adversary's scheduling is as follows. For simplicity let us assume that $n = 5$ and $t = 1$, although the argument can be extended easily to any number of parties. Assume that P_1 is the sender, P_5 is the receiver.

- Let $\mathcal{O}_{k \cdot B} = \{P_1, P_2, P_3, P_4\}$ for $k = 0, 1, \ldots$.
- Let $\mathcal{O}_r = \{P_2, P_3, P_4, P_5\}$ for every r that is not of the form $r_0 + k \cdot B$. Notice that $|\mathcal{O}_{k \cdot B} \cap \mathcal{O}_{k \cdot B + 1} \cap \mathcal{H}| = |\{P_3, P_4\}| = 2 = 2t$ where $\mathcal{O}_{k \cdot B} \cap \mathcal{O}_{k \cdot B + 1} = \{P_2, P_3, P_4\}$.

Notice that this scheduling respects the B-assumption. Suppose that there is a protocol that instantiates $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ with perfect security against an active adversary, supporting the scheduling above. We will show a contradiction arising from the fact that the adversary can actively cheat.

Suppose that P_R learns the output in round $r_R = k_0 \cdot B + \ell$ for some k_0 and ℓ with $1 \leq \ell \leq B$. Consider two different messages $m \neq m'$, and let M_j and M'_j for $j = 2, 3, 4$ be the concatenation of the messages sent by P_j in round $k \cdot B$ to the parties in $\mathcal{O}_{k \cdot B} \cap \mathcal{O}_{k \cdot B+1} = \{P_2, P_3, P_4\}$ for $k = 0, \ldots, k_0$, when the inputs of P_S to the protocol are m and m' respectively.

First, we claim that the messages (M_2, M_3, M_4) (resp. (M'_2, M'_3, M'_4)) must uniquely determine the secret m (resp. m'). To see why this is the case, observe that the receiver, P_5, only ever hears from the parties P_2, P_3, P_4, but these in turn only hear from the sender, P_1, through the messages (M_2, M_3, M_4) (resp. (M'_2, M'_3, M'_4)), so these messages have to carry enough information to determine the secret.

Now, due to privacy, no single party must be able to determine whether the message sent is m or m'. If P_3 was corrupt and if $M_3 \neq M'_3$ for all possible initialization of all random tapes, then the adversary would be able to distinguish the message by simply looking at whether M_3 or M'_3 is being sent by P_3. Hence, we see that there must exist an initial random tape for which $M_3 = M'_3$. For the rest of the attack we assume this is the case.

With the observations we have seen so far, a corrupt party P_2 can mount the following attack: If P_2 sees it needs to send M_2, it will send M'_2 instead. Since the protocol withstands an active attack, the transcript (M_2, M_3, M_4), which would be transformed to (M'_2, M_3, M_4) after the attack, would uniquely determine m. On the other hand, the very same transcript can arise from an actively corrupt P_4 that modifies the message M'_4 when the message is m' to M_4 (recall that $M'_3 = M_3$). In this case, due to the resilience of the protocol against one active attack, (M'_2, M_3, M_4) should reconstruct to the same message as (M'_2, M'_3, M'_4), which is m'. This is, however, a contradiction, since the same transcript cannot lead to two different messages.

3 Instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with Statistical Security

The goal of this section is to develop an information-theoretic protocol that instantiates $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ against *active* adversaries, but replacing the condition $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t + 1$ from Section 2.2 with $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t + 1$. As shown in Section 2.2, perfect security cannot be achieved in this setting, so we settle with statistical security.

Our construction at a high level works as follows. First, we design a pair of functions $f(m) = (m_1, \ldots, m_n)$ and $g(m'_1, \ldots, m'_n) = m'$ such that, if $m'_i = m_i$ for at least $t + 1$ (unknown) indices, then $m' = m$. Also, it should hold that no set of at most t values m_i leaks anything about m. Assuming the existence of such pair of functions, we can envision a simple construction of a protocol $\Pi_1(P_S, P_R, m)$ that guarantees that a receiver P_R gets the message m sent by a sender P_S, as long as P_R comes online either in the same round where P_S is, or in the next one. This operates as follows: P_S computes $(m_1, \ldots, m_n) = f(m)$, and, in every round, P_S sends m_i to party P_i, as well as m to P_R. Once a party P_i receives m_i, it sends this value to P_R in the next round. Let $m'_1, \ldots, m'_n$ be the values received by P_R when it comes online, where $m'_i = \perp$ if P_R does not receive a message from P_i (notice that m'_i could differ from m_i if P_i is actively corrupt). Since $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t + 1$, we see that at least $t + 1$ of the m'_i are equal to m_i, so P_R can output $m = g(m'_1, \ldots, m'_n)$.

Now, we would like to "bootstrap" the protocol Π_1 into a protocol $\Pi_2(P_S, P_R, m)$ that guarantees that a receiver P_R gets the message m sent by a sender P_S, as long as P_R comes online either in the same round where P_S is, in the next one, *or in the one after that.* To this end, the parties run $\Pi_1(P_S, P_R, m)$, which guarantees that P_R gets m if it comes online in the same round as P_S, or at most in the round after. However, to deal with the case in which P_R comes online two rounds after P_S, the parties also execute the following *in parallel*: P_S

computes $(m_1, \ldots, m_n) = f(m)$ and executes $\Pi_1(P_S, P_R, m_i)$ for $i = 1, \ldots, n$. This ensures that every $P_i \in \mathcal{O}_2$ will get m_i, and at this point, the parties in $\mathcal{O}_3 \cap \mathcal{O}_2$ can send these to P_R in the third round. Upon receiving m'_i, P_R outputs $m = g(m'_1, \ldots, m'_n)$.

To analyze the protocol Π_2, assume for simplicity that $P_S \in \mathcal{O}_1$. We first observe that if $P_R \in \mathcal{O}_1 \cup \mathcal{O}_2$, then P_R gets m as $\Pi_1(P_S, P_R, m)$ is being executed. If, on the other hand, $P_R \in \mathcal{O}_3$, P_R gets m as $g(m_1, \ldots, m_n)$ since the parties $P_i \in \mathcal{O}_2$ get m_i from $\Pi_1(P_S, P_R, m_i)$. This idea can be iterated to obtain protocols that deliver messages as long as P_R comes online at most k rounds after P_S comes online.

In what follows we present the tools necessary to formalize this idea, and later discuss the actual protocols for instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$.

3.1 Robust Secret Sharing

The functions f and g discussed above are instantiated using robust secret-sharing, which are techniques that enables a dealer to distribute a secret among multiple nodes in such a way that (1) no subset of at most t nodes learn the secret and (2) if each node sends its share to a receiver, no subset of at most t corrupt nodes can stop the receiver from learning the correct secret.

The definition we consider here is more general than standard definitions from the literature since, at reconstruction time, we allow for missing shares, and if there are many of these we allow the reconstruction algorithm to output an error signal $\perp$. However, if there are enough honest non-missing shares, then reconstruction of the correct message must be guaranteed. This is needed since, in our protocols, there are some rounds in which parties may not receive enough shares to reconstruct the right secret, and they must be able to detect this is the case to wait for subsequent rounds where more shares are available.

▶ **Definition 3.** *Let $A \subseteq \{1, \ldots, n\}$ with $|A| \leq t$. A robust secret-sharing (RSS) scheme with deletions having message space M and share space S is made up of two randomized polytime functions, $\mathsf{share} : \mathsf{M} \to \mathsf{S}^n$ and $\mathsf{rec} : \mathsf{S}^n \to \mathsf{M}$, satisfying the properties below for any not-necessarily-polytime algorithm $\mathcal{A}$. Let $(s_1, \ldots, s_n) = \mathsf{share}(m)$. Let $B^c = \mathcal{A}(\mathsf{missing}, \{s_j\}_{j \in A}) \subseteq \mathcal{P}$ denote a set chosen by $\mathcal{A}$ of shares to be deleted. Let $(s'_1, \ldots, s'_n)$ be defined as follows: $s'_i = \perp$ for $i \in B^c$, $s'_i = \mathcal{A}(i, \{s_j\}_{j \in A}) \in \mathsf{S}$ for $i \in A \cap B$ and $s'_i = s_i$ for $i \in A^c \cap B$.*

- ***Privacy.*** *The distribution of $\{s_i\}_{i \in A}$ is independent of m.*
- ***Error detection.*** *With probability $1 - \mathsf{negl}(\kappa)$, $\mathsf{rec}(s'_1, \ldots, s'_n)$ outputs either m or $\perp$.*
- ***Guaranteed reconstruction.*** *If $|A^c \cap B| > t$ then, with probability $1 - \mathsf{negl}(\kappa)$, it holds that $m = \mathsf{rec}(s'_1, \ldots, s'_n)$.*

Several robust secret-sharing constructions can be found in the literature. However, since we consider a non-standard version of robust secret-sharing, we present below a concrete construction that fits Definition 3, which is motivated on the so-called information-checking signatures from [11]. We remark that *any* instantiation of Definition 3 will suffice for our stable network construction, with better parameters such as share length of computational complexity directly leading to direct improvements on our protocols.

The following proposition shows that the scheme $(\mathsf{share}, \mathsf{rec})$ is an RSS scheme with error detection.

▶ **Proposition 4.** *The construction $(\mathsf{share}, \mathsf{rec})$ from above is an RSS scheme with deletions.*

RSS scheme with deletions: $(\mathsf{share}, \mathsf{rec})$

$\mathsf{share}(m)$: Compute Shamir shares $m_1, \ldots, m_n$ of m. For each $i \in \{1, \ldots, n\}$, sample $(\alpha_i, \{\beta_{ij}\}_{j=1}^n)$, and let, for every $i, j \in \{1, \ldots, n\}$, $\tau_{ij} = \alpha_j m_i + \beta_{ji}$. Return $(s_1, \ldots, s_n)$, with $s_i = (m_i, (\alpha_i, \{\beta_{ij}\}_{j=1}^n, \{\tau_{ij}\}_{j=1}^n)$.

$\mathsf{rec}(s'_1, \ldots, s'_n)$. Let $B = \{i : s'_i \neq \perp\}$. Parse each s'_i for $i \in B$ as $(m'_i, (\alpha'_i, \{\beta'_{ij}\}_{j=1}^n, \{\tau'_{ij}\}_{j=1}^n)$. Then proceed as follows:

1. If $|B| \geq t+1$: for every $i \in B$ do the following. If $\alpha'_j m'_i + \beta'_{ji} \stackrel{?}{=} \tau'_{ij}$ does not hold for at least $t+1$ values of $j \in B$, then set $m'_i = \perp$.[a]
2. After this process, if $|\{m'_i : m'_i \neq \perp\}| > t$, then using any subset of this set of size $t+1$ to interpolate a polynomial $f(x)$ of degree at most t, and output $m = f(0)$. Else, output $\perp$.

[a] In particular, if $0 \leq |B| \leq t$ then all m'_i would be set to $\perp$ as the check would always fail.

Proof. Let $\mathsf{share}(m) = (s_1, \ldots, s_n)$ with $s_i = (m_i, (\alpha_i, \{\beta_{ij}\}_{j=1}^n), \{\tau_{ij} = \alpha_j m_i + \beta_{ji}\}_{j=1}^n)$. First we argue privacy. It is clear that the n Shamir shares $m_1, \ldots, m_n$ do not leak anything about the secret m towards the adversary. Additionally, the keys $(\alpha_i, \{\beta_{ij}\}_{j=1}^n)$ are simply random values, which do not leak anything either. Finally, each P_i receives $\{\tau_{ij} = \alpha_j m_i + \beta_{ji}\}_{j=1}^n$, but these only involve m_i, which is already known by P_i. Notice that, since β_{ji} is uniformly random and unknown to P_i (if $j \neq i$), P_i learns no information about α_j. This will be crucial since, as we show below, α_j is used to prevent P_i from changing their share.

Now, to see the guaranteed reconstruction property, let $(s'_1, \ldots, s'_n)$ be as in Definition 3. Assume that $|A^c \cap B| > t$, we want to show that $\mathsf{rec}(s'_1, \ldots, s'_n)$ outputs m in this case. Let us write each s'_i for $i \in A \cap B$ as $s'_i = (m'_i, (\alpha'_i, \{\beta'_{ij}\}_{j=1}^n), \{\tau'_{ij}\}_{j=1}^n)$. We claim that if $m'_i = m_i + \delta_i$ with $\delta_i \neq 0$, then $\tau'_{ij} = \alpha_j m'_i + \beta_{ji}$ for at least $j \in A^c \cap B$ can only happen with negligible probability. To see why this holds, let us write $\tau'_{ij} = \tau_{ij} + \epsilon_{ij}$, so $\tau'_{ij} = (\alpha_j m_i + \beta_{ji}) + \epsilon_{ij} = (\alpha_j m'_i + \beta_{ji}) - \alpha_j \delta_i + \epsilon_{ij}$. For this to be equal to $\alpha_j m'_i + \beta_{ji}$, it has to hold that $\alpha_j = \delta_i^{-1} \epsilon_{ij}$. However, δ_i and ϵ_{ij} are functions of $\{s_\ell\}_{\ell \in A}$, so they are computed independently of the uniformly random value α_j since $j \notin A$. This shows that the equation $\alpha_j = \delta_i^{-1} \epsilon_{ij}$ for at least $j \in A^c \cap B$ can only hold with probability at most $1/|\mathbb{F}| = \mathsf{negl}(\kappa)$, so in particular the claim above holds (recall that $n = \mathsf{poly}(\kappa)$).

From the above we see that if $m'_i \neq m_i$ then, with overwhelming probability, $\tau'_{ij} \neq \alpha_j m'_i + \beta_{ji}$ for every $j \in A^c \cap B$, so in particular $\tau'_{ij} = \alpha_j m'_i + \beta_{ji}$ can only be satisfied for $j \in A \cap B$, but since $|A \cap B| \leq t$, we see that m'_i would be set to $\perp$ from the definition of $\mathsf{rec}(\cdot)$. As a result, only values with $m'_i = m_i$ remain, and since there are at least $|A^c \cap B| > t$ of these, we see that $\mathsf{rec}(\cdot)$ outputs m correctly in this case.

The argument above also shows the error detection property: the extra assumption $|A^c \cap B| > t$ was only used at the end to show that the set $\{m'_i : m'_i \neq \perp\}$ will have at least $t+1$ elements, in which case the correct m could be reconstructed. If this does not hold, then $\mathsf{rec}(\cdot)$ outputs $\perp$. ◀

3.2 Delivering within 2 rounds

Let $(\mathsf{share}, \mathsf{rec})$ be a robust secret-sharing scheme with deletions. We begin by presenting a protocol $\Pi_1(P_S, P_R, m)$ that guarantees that P_R gets the message m sent by P_S as long as P_R comes online either in the same round as P_S, or at most one round later. First, we define the concept of k-delivery, which formalizes and generalizes this notion.

Protocol $\Pi_1(P_S, P_R, m)$

P_S does the following:
- Let $(s_1, \ldots, s_n) = \mathsf{share}(m)$. Send s_i to P_i in every round.
- Send m to P_R.

Every party P_i does the following:
- P_i sets an internal variable $\mathtt{s}_i = \perp$. In every round, if P_i receives s_i from P_i, then it sets $\mathtt{s}_i = s_i$.
- In every round, if $\mathtt{s}_i \neq \perp$, then P_i sends $\mathtt{s}_i$ to P_R.

P_R does the following in every round:
- If P_R receives m from P_S, then P_R outputs m.
- Let s'_i be the message P_R receives from P_i, setting $s'_i = \perp$ if no such message arrives. If $\mathsf{rec}(s'_1, \ldots, s'_n) \neq \perp$, then P_R outputs this value.

▶ **Definition 5** (k-delivery)**.** *A protocol Π is said to satisfy k-delivery if it instantiates the functionality $\mathcal{F}_{\mathsf{StableNet}}^{P_S,P_R}$ (with statistical security), modified so that P_R is only guaranteed to receive the message sent by P_S if $P_R \in \bigcup_{r=0}^{k} \mathcal{O}_{r_S+r}$, where r_S is the first round in which $P_S \in \mathcal{O}_{r_S}$. If $P_R \notin \bigcup_{r=0}^{k} \mathcal{O}_{r_S+r}$, then P_R cannot output an incorrect message.*

▶ **Proposition 6.** *$\Pi_1(P_R, P_S, m)$ satisfies 1-delivery.*

Proof. Privacy holds from the privacy of the robust secret-sharing scheme.

Now, assume that $P_R \in \mathcal{O}_{r_S} \cup \mathcal{O}_{r_S+1}$. If $P_R \in \mathcal{O}_{r_S}$, then P_R gets m as it is being sent by P_S directly. On the other hand, if $P_R \in \mathcal{O}_{r_S+1}$, the argument is the following. First, each $P_i \in \mathcal{O}_{r_S}$ receves s_i from P_S, which in particular means that the parties in $\mathcal{O}_{r_S} \cap \mathcal{O}_{r_S+1} \cap \mathcal{H}$ send the correct s_i to P_R. P_R receives at least $t+1$ correct shares s_i and at most t incorrect ones, hence, by the guaranteed reconstruction property of the RSS, P_R obtains s from these shares.

Finally, the fact that if $P_S \notin \mathcal{O}_{r_S} \cup \mathcal{O}_{r_S+1}$ then P_S does not output an incorrect message follows from the error detection property of $(\mathsf{share}, \mathsf{rec})$. ◀

3.3 From $(k-1)$-delivery to k-delivery

Now we show that, given a protocol $\Pi_{\mathsf{k}-1}(P_R, P_S, \cdot)$ that achieves $(k-1)$-delivery, one can obtain a protocol that achieves k-delivery. This is achieved by Protocol $\Pi_{\mathsf{k}}(P_R, P_S, m)$.

Protocol $\Pi_{\mathsf{k}}(P_R, P_S, m)$

In the following, multiple protocols will be executed in parallel. We assume that messages are tagged with special identifiers so that they can be effectively distinguished.

The parties execute $\Pi_{\mathsf{k}-1}(P_S, P_R, m)$. In parallel, they execute the following.
- Let $(s_1, \ldots, s_n) = \mathsf{share}(m)$. The parties run n protocol instances $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$ for $i = 1, \ldots, n$.
- Each P_i, upon outputting s_i from $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$, send (s_i) to P_R in all subsequent rounds.
- P_R initializes variables $\mathtt{s}_1, \ldots, \mathtt{s}_n = \perp$. Then P_R does the following in every round:
  - Upon outputting s_i from some execution $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$, P_R sets $\mathtt{s}_i = s_i$.
  - Upon receiving s'_i from some party, sets $\mathtt{s}_i = s'_i$.
  - P_R outputs $\mathsf{rec}(\mathtt{s}_1, \ldots, \mathtt{s}_n)$ if this value is not $\perp$.

▶ **Proposition 7.** *Protocol* $\Pi_{\mathsf{k}}(P_S, P_R, m)$ *achieves* k*-delivery.*

Proof. Let r_S be the first round in which $P_S \in \mathcal{O}_{r_S}$, and assume that $P_R \in \bigcup_{r=0}^{k} \mathcal{O}_{r_S+r}$. If $P_R \in \bigcup_{r=0}^{k-1} \mathcal{O}_{r_S+r}$, then P_R would receive m correctly from the properties of $\Pi_{\mathsf{k}-1}$.

Given the above, it remains to analyze the case in which $P_R \in \mathcal{O}_{r_S+k}$. From the properties of $\Pi_{\mathsf{k}-1}$, every party $P_i \in \mathcal{O}_{r_S+(k-1)}$ receives s_i from P_S in round $r_S + (k-1)$. In particular, each party $P_i \in \mathcal{O}_{r_S+(k-1)} \cap \mathcal{O}_{r_S+k}$ sends s_i to P_R in round $r_S + k$. An analysis similar to the one in the proof of Proposition 6 shows that P_R is able to recover m from this information, and it also shows that if $P_R \notin \bigcup_{r=0}^{k} \mathcal{O}_{r_S+r}$, then P_R cannot be fooled into reconstructing an incorrect message. ◀

Combining Propositions 6 and 7, we obtain the following corollary:

▶ **Corollary 8.** *For every* k*, there exists a protocol* Π_{k} *satisfying* k*-delivery.*

Now, recalling that the B-assumption implies that there is one round among $1, \ldots, B$ in which P_S will come online, and a round among $B+1, \ldots, 2B$ in which P_R is online as well, we obtain the following theorem as a corollary.

▶ **Theorem 9.** *Assume that* $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ *for every* $r > 0$*. Then, protocol* $\Pi_{\mathsf{2B}}(P_R, P_S, \cdot)$ *instantiates the functionality* $\mathcal{F}_{\mathsf{StableNet}}^{P_R \to P_S}$ *in the* $\mathcal{F}_{\mathsf{UnstableNet}}$*-hybrid model with statistical security against an adversary actively corrupting* $t < n/2$ *parties.*[3]

▶ Remark 10. The communication complexity of Π_{k} is $\Theta(n^k)$. This is because, in the execution of Π_{k}, P_S must use $\Pi_{\mathsf{k}-1}$ to communicate a share to each single party, adding a factor of n with respect to the communication complexity of this protocol. This is too inefficient for large values of k. We leave is an open problem the challenging task of obtaining instantiations of $\mathcal{F}_{\mathsf{StableNet}}^{P_S,P_R}$ with statistical security in the setting in which $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ having communication complexity that is polynomial in the bound B.

References

1. Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: Laziness leads to GOD. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 120–150. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-64840-4_5.
2. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid mpc: Secure multiparty computation with dynamic participants. In *Annual International Cryptology Conference*, pages 94–123. Springer, 2021.
3. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, Heidelberg, March 2009. doi:10.1007/978-3-642-00468-1_10.
4. Ivan Damgård, Daniel Escudero, and Divya Ravi. Information-theoretically secure mpc against mixed dynamic adversaries. Thheory of Cryptography Conference, 2021.
5. Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 121–136. Springer, Heidelberg, August 1998. doi:10.1007/BFb0055724.

[3] As with Theorem 2, in principle the restriction is simply $t < n$, but we have that $n - t = |\mathcal{H}| \geq |\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$, so $n \geq 2t+1$.

6 Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information processing letters*, 43(4):169–174, 1992.

7 Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO 2021*, 2021.

8 Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_18.

9 Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

10 Chiu-Yuen Koo. Secure computation with partial message loss. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 502–521. Springer, Heidelberg, March 2006. doi:10.1007/11681878_26.

11 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989. doi:10.1145/73007.73014.

12 Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. Cryptology ePrint Archive, Paper 2021/1579, 2021. URL: https://eprint.iacr.org/2021/1579.

13 Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multi-party computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 274–293. Springer, Heidelberg, March 2009. doi:10.1007/978-3-642-00457-5_17.

A Shamir Secret Sharing

Throughout this work we will make use of Shamir secret sharing in order to distribute data among different parties. To secret-share a value $s \in \mathbb{F}$ among the n parties $P_1, \ldots, P_n$ using threshold t, a dealer proceeds as follows: (1) sample a uniformly random polynomial $f(x) \in \mathbb{F}[x]$ of degree at most t, subject to $f(0) = s$, and (2) send to P_i its share $s_i := f(i)$. It is well known that for every set of $t+1$ points (i, s_i) there exists a unique polynomial $f(x)$ of degree at most t such that $f(i) = s_i$ for all i, which implies that any set of at least $t+1$ shares can recover the secret, and any set of t shares does not reveal anything about the secret.

Bivariate sharings

Sometimes we will make use of *bivariate sharings*, in which the dealer, to distribute a secret $s \in \mathbb{F}$, samples a random symmetric bivariate polynomial $f(x, y)$ of degree at most t in each variable subject to $f(0, 0) = s$, and sends the polynomial $f(x, i)$ to P_i. As before, given at most t of these polynomials nothing is leaked about the secret s since any secret could be chosen so that it looks consistent with the given polynomials.

Error-detection and error-correction

Given m shares among which at most t can be incorrect, then the parties output $f(0)$ as the secret, where $f(x)$ is the reconstructed polynomial. Given m shares $\{s_i\}$ among which at most t are incorrect we have the following two possibilities:

- If at least $t+1$ are guaranteed to be correct, *error-detection* can be performed by checking if these shares all lies in a polynomial of degree at most t, and if this is the case, the reconstructed polynomial is guaranteed to be correct since it is determined by the $t+1$ correct shares.

- If at least $2t+1$ are guaranteed to be correct, *error-correction* is possible by looping through all possible subsets of these shares of size $2t+1$ and checking if all shares in the given subset are consistent with a polynomial of degree at most t. The subset used for reconstructing this polynomial has $2t+1$ points among which at least $t+1$ are correct (since at most t shares are assumed to be incorrect), which guarantees that the reconstructed polynomial is the correct one. Although the process of looping through all subsets of size $2t+1$ can be too inefficient if m is much larger than $2t+1$, this can be made polynomial in m by using error-detection algorithms like Berlekamp-Welch [6].

In some of our protocols we will need a version of error-correction, which we call *enhanced error-correction*, in which the correct polynomial is recovered if there are enough correct shares, and else an error is output. To this end, given $m \geq 2t+1$ shares as above among which at most t are incorrect, all possible subsets of $2t+1$ shares are inspected, checking if all these shares are consistent with a polynomial of degree at most t. If one such subset is found, then its corresponding polynomial is output, and else, an error $\perp$ is produced as the result. By the same analysis as above, this either results in the correct polynomial or an error. The main complication is that error-correcting algorithms like Berlekamp-Welch are not designed to handle this setting in which not enough correct shares may be available, but one can easily modify this algorithm to handle this case (see for example [4]).

B A More Efficient Protocol with Perfect Security

Recall that in Section 2.2 we presented a protocol to instantiate the functionality $\mathcal{F}_{\mathsf{StableNet}}$, which is intended to represent a traditional stable and secure network among the n parties. This is the typical communication model used in several MPC protocols, and, assuming $t < n/3$, we can find perfectly secure protocols in this model which can be used together with our protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_S, P_R)$ from Section 2.2 to obtain a perfectly secure protocol over an unstable network.

In order to instantiate the functionality $\mathcal{F}_{\mathsf{StableNet}}$, we required that the scheduling the adversary provides allows each party to come online at least *once* within certain amount of rounds, say B. This is necessary since $\mathcal{F}_{\mathsf{StableNet}}$ requires each message between honest parties to be delivered, and if the receiver never comes online such guarantee cannot hold. Unfortunately, our protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_S, P_R)$ requires $2B$ rounds to deliver a message between a sender and a receiver, which ultimately means that the final protocol after composing $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_S, P_R)$ with an existing perfectly secure protocol would lead to a multiplicative overhead of $2B$ in the number of rounds.

Round-count is a very sensitive metric in distributed protocols, especially in high-latency scenarios where every communication trip incurs in a noticeable waiting time. Furthermore, the $\theta(B)$ overhead may not be so noticeable if the higher level protocol has a low round count, but unfortunately, it is a well-known open problem to achieve constant round protocols with *perfect security* for functionalities outside NC^1 while achieving polynomial computation and communication complexity. Motivated by this, we develop in this section a perfectly secure protocol over an unstable network whose number of rounds corresponds to the depth of the circuit being computed plus a term that depends on B, but is independent of the size of the circuit, matching the round complexity of existing protocols over stable networks. Furthermore, after the inputs have been provided, our protocol does not require anymore the

assumption that each party has to be online at least once every B rounds.[4] This is because, as we will see, our protocol only relies on the assumption that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every round r in order to transmit and advance the *secret-shared state* of the computation from one round to the next. Intuitively, it is irrelevant if certain specific parties become online at certain points of the protocol, and the only thing that matters is that *enough* parties remain online from one round to the next one, irrespectively of their identities.

B.1 Bivariate Sharings and Transition of Shares

We describe the input and preprocessing phases of our protocol in Section B.2, and in Section B.3 we describe its computation phase. However, before we dive into the protocols themselves, we need to present certain primitives that will be useful for these constructions. These are bivariate sharings, together with methods for transmitting bivariate shared values from one round to the next. This will allow the parties to "transmit" the state of the computation from the parties that are online in a given round, to these online in the next one, making progress in one layer of the circuit at the same time.

We say that the parties have bivariate shares of a value s if there exists a symmetric bivariate polynomial $f(x,y)$ of degree at most t in both variables such that (1) each party $P_i \in \mathcal{P}$ has $f(x,i)$ and (2) it holds that $f(0,0) = s$. We denote this by $\langle s \rangle$. Observe that this scheme is linear, i.e. parties can locally compute additions of secret shared values, which is denoted by $\langle x+y \rangle \leftarrow \langle x \rangle + \langle y \rangle$.

Bivariate sharings were used indirectly in Section 2.2 to instantiate $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security against an active adversary. This type of sharings proved useful in Protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R)$ to "transfer" a state between a set of parties to another one, and this is the purpose of this primitive in this section as well. In a bit more detail, during the execution of our protocol it will not hold that all parties have shares of certain given values, but rather only specific subsets corresponding to online parties will do. Since the set of online parties potentially changes from round to round, a crucial primitive our protocol relies on is what we call *transition of shares*, which takes care of transmitting the shared state from one set of parties to another.

We first formalize the notion that only (part of) the online parties hold shares of a given value. We say that the parties have a bivariate-shared value s *in round* r if there exists a symmetric bivariate polynomial $f(x,y)$ of degree at most t in both variables such that (1) there exists a subset $\mathcal{S}_r \subseteq \mathcal{O}_r \cap \mathcal{H}$ with $|\mathcal{S}_r| \geq 2t+1$ such that each $P_i \in \mathcal{S}_r$ has $f(x,i)$, (2) each $P_i \in (\mathcal{O}_r \cap \mathcal{H}) \setminus \mathcal{S}_r$ has set their share to either $f(x,i)$, or a predefined value $\perp$, and (3) it holds that $f(0,0) = s$. This is denoted by $\langle s \rangle^{\mathcal{O}_r}$. Observe that nothing is required from parties outside $\mathcal{O}_r \cap \mathcal{H}$. Also, notice that if all the parties have bivariate shares of a value s, which we denote by $\langle s \rangle$, then it holds that $\langle s \rangle^{\mathcal{O}_r}$ for every r.

A protocol for transition of shares is a one-round protocol in which the parties start with $\langle s \rangle^{\mathcal{O}_r}$ in round r, and they obtain $\langle s \rangle^{\mathcal{O}_{r+1}}$ in the next round $r+1$. In what follows we present a protocol for transition of shares, which is motivated in the perfectly secure protocol for instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ from Section 2.

[4] However, the output will be received only by the parties who happen to be online at the output phase.

Protocol Π_{transfer}

Input: $\langle s \rangle^{\mathcal{O}_r}$ in round r
Output: $\langle s \rangle^{\mathcal{O}_{r+1}}$ in round $r+1$.

Parties do the following:

1. For each $i = 1, \ldots, n$, if P_i has a share $f(x, i)$ of $\langle s \rangle^{\mathcal{O}_{r+1}}$ (different to $\perp$), then P_i sends $f(j, i)$ to P_j for $j = 1, \ldots, n$.
2. For each $j = 1, \ldots, n$, if P_j receives at least $2t + 1$ messages $\{f(j, i)\}_i$, then P_j performs *enhanced* error correction (see Section A) to either recover $f(j, x)$ or output an error $\perp$.

▶ **Theorem 11.** *If executed in round r, protocol Π_{transfer} guarantees that the parties get sharings $\langle s \rangle^{\mathcal{O}_{r+1}}$.*

Proof. Let $\mathcal{S}_r \subseteq \mathcal{O}_r \cap \mathcal{H}$ with $|\mathcal{S}_r| \geq 2t + 1$ be the set of honest parties P_i having $f(x, i)$, guaranteed from the definition of bivariate sharings. Since the protocol above is executed in round r, each party $P_i \in \mathcal{S}_r$ will send $f(j, i)$ to each other party P_j, which in particular is received by the parties $P_j \in \mathcal{O}_{r+1} \cap \mathcal{O}_r \cap \mathcal{H}$, and given that $|\mathcal{S}_r| \geq 2t + 1$, the enhanced error-correction algorithm executed by P_j will result in P_j recovering $f(j, x)$, which is equal to $f(x, j)$. Let $\mathcal{S}_{r+1} := \mathcal{O}_{r+1} \cap \mathcal{O}_r \cap \mathcal{H}$ and note that (1) $|\mathcal{S}_{r+1}| \geq 2t + 1$ and also each $P_j \in \mathcal{S}_{r+1}$ has $f(x, j)$, (2) each $P_j \in (\mathcal{O}_{r+1} \cap \mathcal{H}) \setminus \mathcal{S}_{r+1}$ set their share to either $f(x, j)$ or $\perp$ due to the properties of the enhance error-correction mechanism, and (3) it (still) holds that $f(0, 0) = s$. From the definition of bivariate sharings, it holds that $\langle s \rangle^{\mathcal{O}_{r+1}}$. ◀

Transitioned Reconstruction

Another primitive that we will need in our protocol, besides transferring shares from one set of parties to another, consists of reconstructing a bivariate-shared value. Assume that the parties in round r have $\langle s \rangle^{\mathcal{O}_r}$. If all parties in round r send their shares $\{f(0, j)\}_j$ to all other parties, they can perform (enhanced) error correction to reconstruct $s = f(0, 0)$. In this way, the parties in $\mathcal{O}_r \cap \mathcal{H}$ are guaranteed to learn s. In particular, s is known by the parties in $\mathcal{O}_{r+1} \cap \mathcal{O}_r \cap \mathcal{H}$, which contains at least $2t + 1$ parties. This protocol is denoted by $s \leftarrow \Pi_{\mathsf{rec}}(\langle s \rangle^{\mathcal{O}_r})$.

▶ Remark 12. An important fact about the proof of Theorem 11 is that, it holds that $\mathcal{S}_{r+1} \subseteq \mathcal{O}_{r+1} \cap \mathcal{O}_r \cap \mathcal{H}$. In addition, the reconstruction protocol from above ensures that the parties in $\mathcal{O}_{r+1} \cap \mathcal{O}_r \cap \mathcal{H}$, so in particular the parties in $\mathcal{S}_{r+1}$, learn the secret. This will be important in our main protocol in Section B.3.

B.2 Preprocessing and Input Phases

We assume that the functionality to be computed is given by a layered circuit $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$. Considering layered circuits, in contrast to more general circuits, is useful for our construction since in this case the values in a given layer completely determine the current *state* of the computation, that is, the next layer, and in particular the remainder of the computation, is fully determined by these values. This is important since, as we will see, at the heart of our construction lies the possibility of a given set of online parties to transmit their shared state to the online parties in the next round, and, from the structure of the protocol, this state is comprised by the shared values in a given layer.

For our main protocol, we assume that *all* the parties have certain bivariate-shared multiplication triples (as specified below), plus bivariate shares of the inputs of the computation. By making use of the B-assumption, these shares can be computed by using *any* generic

MPC protocol for these tasks, together with our compiler from Section 2.2. This would incur a multiplicative overhead of B in the number of rounds, however, the circuit representing this computation is constant-depth, so this does not affect the overall result of this section. Notice that this does not require all the parties to be online during the computation of these sharings, but instead, the B-assumption, that requires every honest party to come online once every B rounds, suffices.

The correlation required for the computation consists of secret-shared values $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, one tuple for every multiplication gate in the circuit, where $a, b \in_R \mathbb{F}$ and $c = a \cdot b$.

B.3 Computation Phase

With the primitives described above, the protocol for computing the given functionality F is relatively straightforward: by making use of the Π_{transfer} and Π_{rec} protocols, the parties can use the standard approach to secure computation based on multiplication triples, making progress from round to round depending on the set of parties that is online. This is possible since, at the end of the execution of the method described in Section B.2, *all* the parties hold the preprocessing material and shares of the inputs (even if some parties were offline during certain parts of the execution), together with the fact that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every round r, which enables share transfer and reconstruction. The protocol is described in detail below. The security proof follows straightforwardly from existing techniques, together with the properties proven in Section B.1, and a sketch of this proof can be found towards the end of this section. Observe that the protocol requires only L rounds, which, added to the $O(1)$ rounds from the preprocessing and input phases, leads to a protocol with comparable round efficiency to protocols in the stable (i.e. traditional) model.

Protocol Π_{MPC}

Input: Secret-shared inputs $\langle x_1^{(0)}\rangle, \ldots, \langle x_{\ell_0}^{(0)}\rangle$, where ℓ_0 is the number of input wires.
Preprocessing: A multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c = a \cdot b\rangle)$ for every multiplication gate in the circuit.
Output: Let L be the final round of the protocol. The parties have $\langle x_1^{(L)}\rangle^{\mathcal{O}_L}, \ldots, \langle x_{\ell_L}^{(L)}\rangle^{\mathcal{O}_L}$ in round L, where $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$.

For rounds $r = 1, \ldots, L$:

- The parties in round $r-1$ already have shares $\langle x_1^{(r-1)}\rangle^{\mathcal{O}_{r-1}}, \ldots, \langle x_{\ell_{r-1}}^{(r-1)}\rangle^{\mathcal{O}_{r-1}}$.
- The parties in round r obtain shares $\langle x_1^{(r)}\rangle^{\mathcal{O}_r}, \ldots, \langle x_{\ell_r}^{(r)}\rangle^{\mathcal{O}_r}$ as follows:
  1. For every addition gate with inputs $\langle x\rangle^{\mathcal{O}_{r-1}}$ and $\langle y\rangle^{\mathcal{O}_{r-1}}$, the parties locally obtain $\langle x+y\rangle^{\mathcal{O}_{r-1}}$ and call $\langle x+y\rangle^{\mathcal{O}_r} \leftarrow \Pi_{\mathsf{transfer}}(\langle x+y\rangle^{\mathcal{O}_{r-1}})$.
  2. For every multiplication gate with inputs $\langle x\rangle^{\mathcal{O}_{r-1}}$ and $\langle y\rangle^{\mathcal{O}_{r-1}}$, the parties proceed as follows:
     a. Let $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ be the next available multiplication triple. The parties in round $r-1$ locally compute $\langle d\rangle^{\mathcal{O}_{r-1}} = \langle x\rangle^{\mathcal{O}_{r-1}} - \langle a\rangle^{\mathcal{O}_{r-1}}$ and $\langle e\rangle^{\mathcal{O}_{r-1}} = \langle y\rangle^{\mathcal{O}_{r-1}} - \langle b\rangle^{\mathcal{O}_{r-1}}$.
     b. The parties in round r learn d and e by calling $d \leftarrow \Pi_{\mathsf{rec}}(\langle d\rangle^{\mathcal{O}_{r-1}})$ and $e \leftarrow \Pi_{\mathsf{rec}}(\langle e\rangle^{\mathcal{O}_{r-1}})$.
     c. The parties in round r compute $\langle x \cdot y\rangle^{\mathcal{O}_r}$ as $d \cdot \langle b\rangle^{\mathcal{O}_r} + e \cdot \langle a\rangle^{\mathcal{O}_r} + \langle c\rangle^{\mathcal{O}_r} + d \cdot e$.[a]
  3. For every identity gate with input $\langle x\rangle^{\mathcal{O}_{r-1}}$ the parties call $\langle x\rangle^{\mathcal{O}_r} \leftarrow \Pi_{\mathsf{transfer}}(\langle x\rangle^{\mathcal{O}_{r-1}})$.

[a] Here is where Remark 12 becomes relevant: parties in $\mathcal{O}_r$ (or rather $\mathcal{S}_r$) can compute the linear combination defining $\langle x \cdot y\rangle^{\mathcal{O}_r}$ since both the constants and the sharings are known to the parties in $\mathcal{S}_r$.

▶ Remark 13 (About the output). In our protocol above, the parties in $\mathcal{O}_L$ obtain shares $\langle x_1^{(L)}\rangle^{\mathcal{O}_L}, \ldots, \langle x_{\ell_L}^{(L)}\rangle^{\mathcal{O}_L}$ in round L, where $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$ is the result of the computation. This output can be dealt with in multiple different ways:

- The parties in $\mathcal{O}_L$ can reconstruct the output to each other. This way, the parties in $\mathcal{O}_L$ are guaranteed to learn the output, but parties outside this set may not satisfy this.
- If the B-assumption holds for some B, the parties can reconstruct and transfer this sharing for B more rounds so that all parties learn the output.

B.4 Security Analysis

Now we provide a *sketch* of the security properties of protocol Π_{MPC} from Section B.3. Recall that the function to be computed is assumed to be given by a layered circuit $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$. Furthermore, it is assumed that the parties have bivariate shares of the inputs $\langle x_1^{(0)}\rangle, \ldots, \langle x_{\ell_0}^{(0)}\rangle$, and also, for every multiplication gate, a triple $(\langle a\rangle, \langle b\rangle, \langle c = a \cdot b\rangle)$ with a, b uniformly random in $\mathbb{F}$.[5] Recall that $\langle s\rangle^{\mathcal{O}_r}$ means that there is a large enough subset $\mathcal{S}_r \subseteq \mathcal{O}_r \cap \mathcal{H}$ such that every party $P_i \in \mathcal{S}_r$ has $f(x, i)$ such that $f(0, 0) = s$, and parties in $(\mathcal{O}_r \cap \mathcal{H}) \setminus \mathcal{S}_r$ either have $f(x, i)$ or a special symbol $\perp$.

Assume the protocol starts in round 0. We claim that the following invariant holds: In round r, the parties in $\mathcal{O}_r$ have shares of the intermediate results in layer r, namely $\langle x_1^{(r)}\rangle^{\mathcal{O}_r}, \ldots, \langle x_{\ell_r}^{(r)}\rangle^{\mathcal{O}_r}$. To see this we argue inductively. For $r = 0$ this follows trivially as we assumed that the parties start with shares $\langle x_1^{(0)}\rangle, \ldots, \langle x_{\ell_0}^{(0)}\rangle$, which in particular means they have shares $\langle x_1^{(0)}\rangle^{\mathcal{O}_0}, \ldots, \langle x_{\ell_0}^{(0)}\rangle^{\mathcal{O}_0}$.

Assume the invariant holds for r, and let us show it also holds for $r + 1$. Let $k \in \{1, \ldots, \ell_{r+1}\}$. From the definition of a layered circuit, the value $x_k^{(r+1)}$ can be computed in either one of three ways:

- *Identity gate* $x_k^{(r+1)} = x_i^{(r)}$. In this case the protocol instructs that the parties must call $\langle x_k^{(r+1)}\rangle^{\mathcal{O}_{r+1}} \leftarrow \Pi_{\mathsf{transfer}}(\langle x_i^{(r)}\rangle^{\mathcal{O}_r})$.
- *Addition gate* $x_k^{(r+1)} = x_i^{(r)} + x_j^{(r)}$. In this case the protocol dictates the parties to compute $\langle x_k^{(r)}\rangle^{\mathcal{O}_r} = \langle x_i^{(r)}\rangle^{\mathcal{O}_r} + \langle x_j^{(r)}\rangle^{\mathcal{O}_r}$, followed by $\langle x_k^{(r+1)}\rangle^{\mathcal{O}_{r+1}} \leftarrow \Pi_{\mathsf{transfer}}(\langle x_k^{(r+1)}\rangle^{\mathcal{O}_r})$.
- *Multiplication gate* $x_k^{(r+1)} = x_i^{(r)} \cdot x_j^{(r)}$. Here, the parties in $\mathcal{O}_r$ first compute locally $\langle d\rangle^{\mathcal{O}_r} = \langle x_i^{(r)}\rangle^{\mathcal{O}_r} - \langle a\rangle^{\mathcal{O}_r}$ and $\langle e\rangle^{\mathcal{O}_r} = \langle x_j^{(r)}\rangle^{\mathcal{O}_r} - \langle b\rangle^{\mathcal{O}_r}$, and call $d \leftarrow \Pi_{\mathsf{rec}}(\langle d\rangle^{\mathcal{O}_r})$ and $e \leftarrow \Pi_{\mathsf{rec}}(\langle e\rangle^{\mathcal{O}_{r-1}})$, which enables the parties in $\mathcal{O}_r \cap \mathcal{H}$, which include $\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}$, to learn d and e. Observe that this does not reveal anthing about $x_i^{(r)}$ and $x_j^{(r)}$ to the adversary since a and b are assumed to be uniformly random and unknown to the adversary. Finally, these parties, which define the set $\mathcal{S}_{r+1}$, compute $d \cdot \langle b\rangle^{\mathcal{O}_{r+1}} + e \cdot \langle a\rangle^{\mathcal{O}_{r+1}} + \langle c\rangle^{\mathcal{O}_{r+1}} + d \cdot e$, which can be easily checked to be equal to $\langle x_i^{(r)} \cdot x_j^{(r)}\rangle^{\mathcal{O}_{r+1}}$, which is the same as $\langle x_k^{(r+1)}\rangle^{\mathcal{O}_{r+1}}$.

Since the invariant holds for every layer, in particular it holds for $r = L$, which shows that, after L rounds, the parties obtain $\langle x_1^{(L)}\rangle^{\mathcal{O}_L}, \ldots, \langle x_{\ell_L}^{(L)}\rangle^{\mathcal{O}_L}$. As mentioned in Remark 13 in Section B.3, these shared outputs can be handled in different ways, depending on the application under consideration.

[5] A simple "optimization" is that these shares do not need to be held by *all* the parties, but rather by these that will make use of these sharings in each corresponding round.