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

Ivan Damgård[1]*, Daniel Escudero[2]**, Antigoni Polychroniadou[2]***

[1] Aarhus University
[2] J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE

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.

* ivan@cs.au.dk
** daniel.escudero@protonmail.com
*** antigonipoly@gmail.com

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 [10], 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 A 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—and as we expand in Section A—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.[3] 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 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 3) 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 4) 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. (Section E in the Supplementary Material) 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

[3] We consider active corruptions in this section, but later in the paper we also present results for passive security.

	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

Fig. 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.

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[4] 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 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.

On guaranteeing output and input provision. In our model, parties can leave the computation and never return, but in that case, naturally, there is no way to guarantee their input will be considered, or that they will receive any output. Since the goal of this work is to design protocols over an unstable network with comparable properties to these over stable networks, including input provision and guaranteed output delivery for honest parties, we address this technicality by explicitly considering a bound B such that an honest party is never offline for more than B rounds, and we assume throughout most of the paper that $B < \infty$, that is, every party eventually rejoins the computation within a bounded amount of rounds. Without this requirement, our protocols may stall indefinitely. Nevertheless, very importantly, we show that the requirement that the parties eventually return to the computation is *not* necessary, if one ignores the properties of input provision and guaranteed output. Indeed, in Section 5, we present a protocol for the case of perfect security that *does not require parties*

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

to return (assuming certain distributed preprocessing data), but in that case not all honest parties are ensured to provide input and receive output. However, as we elaborate later in Section 1.3 when we present an overview of our techniques, this protocol has the advantage of having a considerably smaller round count.

1.3 Technical Overview

At a high level, our constructions are obtained via a generic and modular approach in which we emulate a stable synchronous network on top of the unstable network which then, coupled with the composition theorem we prove in our work, allows us to compile any existing synchronous protocol to the unstable network setting. In a bit more detail, we first define a functionality $\mathcal{F}_{\mathsf{StableNet}}$ that represents a secure and stable network: it allows parties to send and receive messages, and if both sender and receiver are honest then the functionality guarantees that the message is eventually transmitted securely (without any eavesdropping or modification) and reliably (the transmission cannot be stopped). Then, to achieve the general "feasibility" results presented in Section 1.2, we take a very general approach that may be of independent interest: instead of developing full-fledged MPC protocols for general functionalities, we focus on developing protocols to instantiate the simpler primitive $\mathcal{F}_{\mathsf{StableNet}}$ assuming a functionality that models an unstable network. Once this is done, due to the composability of the UC framework, *any* MPC protocol that is computational/statistically/perfectly secure in the $\mathcal{F}_{\mathsf{StableNet}}$-hybrid model composed with our instantiations results in an equivalent protocol over an unstable network.

Approaching cryptographic problems in a modular way by splitting them into separate components and approaching each separately is at the crux of many major results in our field, and it constitutes the main reason of existence of the UC framework (which we show to maintain its composability properties in our setting). This elegant approach has several advantages, and our modular approach in particular also inherits these: First, the compiled network can be used for other use-cases beyond MPC that may require stability. Second, using off-the-shelf MPC protocols allows us to directly translate improvements on the "traditional" MPC setting to improvements over the unstable setting. Finally, it allows us to focus on the hardest part of dealing with unstable networks, which has to do with message transmission, while ignoring other "extra" complexities like secure additions or multiplications. One drawback of our approach, however, is that there is an (inherent) overhead on the round count. As we discuss below, we show how to overcome this in some concrete settings by building MPC protocols directly over the unstable network, instead of compiling the network first. This shows the potential of the unstable network as a meaningful and *efficient* model for realistic communication.

High-level ideas for instantiating $\mathcal{F}_{\mathsf{StableNet}}$. We present instantiations of $\mathcal{F}_{\mathsf{StableNet}}$ with perfect, statistical and computational security, assuming that for every round r it holds that $|\mathcal{O}_r \cap \mathcal{O}_{r+1}|$ is larger than $2t+1$, $t+1$ and 1, respectively, and we show these bounds to be optimal.

The instantiation for computational security is straightforward: the sender P_S creates a signed and encrypted message intended for the receiver. In the next B rounds, P_S attempts to send it to all other parties, as this guarantees he will be heard at least once, by the B-round assumption. Now for B rounds, parties echo what they heard in the previous round. Again by the B-round assumption, the sender is guaranteed to come on-line before this is over, and can pick up the message. This simple construction is described in detail in Section E in the Supplementary Material, where we also prove security in our extended UC framework, defined in detail in Section D in the Supplementary Material.

Of course, this approach does not work for information theoretic security, and so the solutions for perfect and statistical security are more involved. Some of the primitives we use for unconditional security have been used many times before in secure computation, our technical contribution is to use them in new ways and for different purposes, as we now explain.

For perfect security, the idea is to make use of the method for secret-sharing using bi-variate polynomials that was first used for verifiable secret sharing in [4]. It can be thought of as a redundant version of Shamir secret-sharing where each party P_i receives a polynomial $f_i(x)$ rater than a single field element. This redundancy was originally used in [4] to allow honest parties to detect inconsistencies introduced by a corrupt dealer. Our observation is that it can also be used to "keep a sharing alive" over several rounds, even if different parties may be on-line in different rounds. We do this by asking each party P_i to send $f_i(j)$ to each P_j. If P_j is on-line in the next round and has received enough values, she can recover her polynomial $f_j(x)$ using error correction. Hence, in every round, enough honest parties will have shares they can send to the receiver, so once she comes online, she will receive the original secret message. This protocol is described in full detail in Section 3.

For statistical security, the solution is much more involved, and is presented in Section 4. Intuitively, one reason why this setting is much more difficult than the one with perfect security is that the number of honest parties that remain online from one round to the next is not large enough to enable error-correction, but instead only ensure error-detection. To overcome this issue, we introduce the notion of *robust secret-sharing with deletions* (RSSD), an extension of the known notion of robust secret-sharing. We show that by using RSSD parties can communicate, assuming that the rounds in which they are online are not too far from each other. It happens to be the case that RSSD can be instantiated using the well-known technique where Shamir-shares are authenticated using unconditionally secure message authentication codes. But we want to emphasize that the technical contribution is the insight that RSSD is the right notion, rather than the instantiation. Finally, we devise a novel and non-trivial recursive construction that leverages the method using RSSD to communicate between parties that come online at potentially very different times.[5]

[5] This solution has a communication overhead that is exponential in the bound B. We leave it as an open problem to design a more efficient solution, however we show a

We emphasize that the task we are dealing with here is fundamentally different from Secure Message Transmission (SMT) (see e.g. [23]), where two parties want to send a message to each other over n channels among which an unknown subset can be compromised. In SMT these channels are known in advance, and the two communicating parties are connected "directly" to them. In our case, what could be regarded as the "channels" is the set of online parties at a given time, which is unknown to any party and can change arbitrarily in every round. This means the two communicating parties are not "connected directly" to the channels, and instead, mechanisms for *transferring* messages from one set of channels to the next is required. This is not a concern in SMT.

Better bounds for IT-security with pre-shared keys. Interestingly, for the case of information-theoretic security we can allow the intersection $\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}$ to be smaller, if we assume the parties run first a setup phase where each pair of parties gets access to a large enough common random key. It turns out that with this set-up assumption, we can emulate a stable network on top of the unstable one assuming $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for statistical security and assuming $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ for perfect security. This also implies protocols without set-up assumptions: namely, to generate the shared keys, the parties first use the generic network compilation approach to send keys secretly between each pair of players. Then, to run the actual MPC protocol, they use the alternative network emulation using shared keys. In practice, this can be an advantage since the stricter condition on the number of honest players surviving from one round to the next, only has to be satisfied during the (short) preprocessing phase where shared keys are exchanged. The reason why results are different with preshared keys is that a sender can one-time-pad-encrypt his or her so that privacy is not a concern anymore, and only integrity must be taken care of. This way, the problem is simply getting the encrypted message unchanged to the receiver by relaying the message in every round. See Section G in the Supplementary Material for details.

Less rounds in the preprocessing model (for perfect security). The modular approach we have taken of first instantiating $\mathcal{F}_{\mathsf{StableNet}}$ over an unstable network is clean and powerful as it allows, in principle, the deployment of *any* synchronous interactive protocol over an unstable network where parties may drop and return according to an adversarially-chosen schedule. However, this comes at a price: the round complexity of the new MPC protocol is a factor $\theta(B)$ larger than that of the underlying protocol (recall that B is the maximal number of rounds an honest party can stay offline). This is unavoidable and stems from the fact that communication between two parties can be delayed by an amount of $\Omega(B)$ rounds. To address this, we also consider the construction of more efficient MPC protocols *directly* on top of an unstable network, without first compiling a stable network and without assuming parties eventually rejoin the computation, and we

more efficient protocol (in the statistical setting) by generating pre-shared secret key material, see Section G.

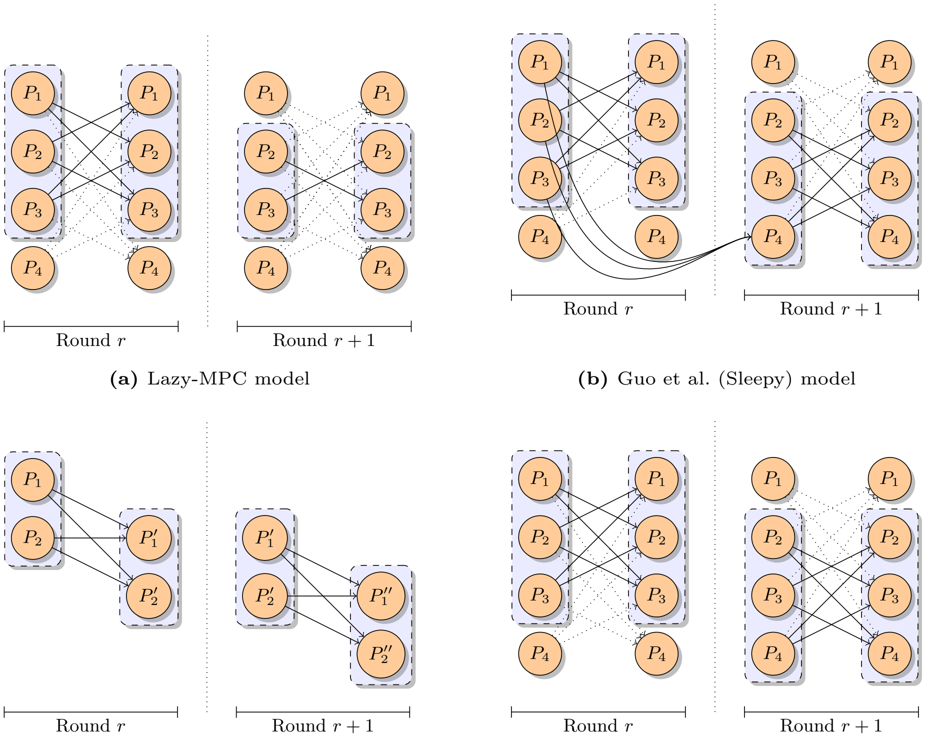
present a concrete construction in the perfect security setting in Section 5. The resulting protocol requires certain preprocessed material which can be generated in a preprocessing phase by running the existing techniques we have discussed so far. Once this is set, the protocol makes progress regardless of whether parties eventually rejoin the computation, and all the parties who are online at the end of the protocol are guaranteed to obtain output.

To design our "direct" perfectly secure protocol in the preprocessing model, we start from the standard idea of preprocessed multiplication triples. We then go through the circuit in the usual way, spending a triple for every multiplication gate. We use sharing by bivariate polynomials to transfer state from one round to the next, so the protocol can proceed despite the fact that different sets of honest parties may be online. At a high level, we reuse the technique sketched before in which each party has a "share" $f(x, i)$ under a symmetric polynomial $f(x, y)$, but this time the underlying secret is an intermediate value of the computation. Using the same "transition" mechanism as before, the parties can transfer the shared state to the next set of online parties, which, coupled with a method to open masked shared values for Beaver-based multiplication towards this upcoming set of parties, enables computation to make progress in a "layer-by-layer" fashion. This leads to a protocol where the computation phase has essentially the same number of rounds as that for a stable network. This compares favourably to our generic compilation where the round complexity is multiplied by $2B$. We present our protocol in Section 5.

1.4 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 in Section A in the Supplementary Material.

Fail-stop adversaries that may cause some parties to stop during a computation were considered for the first time in [12], 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 [16] 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 [16] 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 [8,22] 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 [8], although statistically secure, only achieves security with



(a) Lazy-MPC model

(b) Guo et al. (Sleepy) model

(c) Fluid MPC model

(d) Our model

Fig. 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.

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 [14]. Our model assumes a somewhat less powerful adversary who must allow a physical party to come back after being offline, while in [14] 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 [19] and [24] 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 Section A in the Supplementary Material for an even more detailed discussion on related work.

We present in Figure 2 a more graphical comparison of our model with respect to the works of [1,16,8].

1.5 Preliminaries and Organization

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ 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 B in the Supplementary Material. 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. [17]).

Regarding organization, we present in Section 2 the core ideas of our *unstable network*, together with a proof of the composition theorem in our new networking model. A more complete and formal description is given in Section D in the Supplementary Material. Then, in Section 3 we present our first instantiation of the functionality for a stable network, $\mathcal{F}_{\mathsf{StableNet}}$, in the setting with perfect security. This also includes the impossibility results that shows that the bound $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t + 1$ is optimal. At that point, we believe the reader has grasped the main ideas behind our model and our constructions. Past the 15th page, in Section 4 we present our instantiation of $\mathcal{F}_{\mathsf{StableNet}}$ for statistical security, which uses the perfectly secure construction as a motivating starting point but has crucial differences due to the lack of redundancy to perform error correction. Then, in Section 5 we revisit the perfectly secure case by presenting a direct MPC construction without compiling the stable network first. This cleverly reuses some ideas from Section 3.

2 Networking Model

In this section we provide the different functionalities we will make use of in our work, together with a sketch of how to model the type of adversaries we consider in the UC framework [7]. Importantly, we provide a proof of the composition theorem in our model in Section 2.3. This is central in our work since this shows that our overall approach of first compiling the network from unstable to stable, and *then* using a generic MPC protocol on top, does result in a secure protocol overall. A much more detailed description of our networking model is presented in Section D in the Supplementary Material.

Our starting point is a synchronous network, where an upper bound $\varDelta$ on the time it takes for a message to be transmitted between any pair of parties is known. The communication pattern proceeds in rounds, identified with integers $1, 2, 3, \ldots$, each taking $\varDelta$ time and consisting of all parties sending messages to each other at the beginning of each round. Since each round r takes $\varDelta$ time, it is guaranteed that all the messages sent at the beginning of round r will be delivered within the same round r.

A synchronous network as described above is modeled by a functionality that we denote by $\mathcal{F}_{\mathsf{StableNet}}$ (described in detail in Section D.1 in the Supplementary Material). Jumping ahead, it is this functionality the one we will implement in a secure fashion on top of the unstable network we will describe next. Before we proceed to defining an unstable network, however, we remark that we consider a family of functionalities $\{\mathcal{F}^{P_i \to P_j}_{\mathsf{StableNet}}\}^n_{i,j=1}$ that models a synchronous channel from P_i to P_j only. It is obvious that $\mathcal{F}_{\mathsf{StableNet}}$ can be securely instantiated in the $\{\mathcal{F}^{P_i \to P_j}_{\mathsf{StableNet}}\}^n_{i,j=1}$-hybrid model, so to instantiate $\mathcal{F}_{\mathsf{StableNet}}$, it suffices to provide an instantiation of all directed channels between each pair of parties. This is the approach we take in this work.

2.1 Unstable Networks

An unstable network is formalized as a functionality, that we denote by $\mathcal{F}_{\mathsf{UnstableNet}}$. In each round, the functionality proceeds as follows (more details are given in Section D.2 in the Supplementary Material):

- At the beginning of the round the environment, denoted by $\mathcal{Z}$, specifies a subset of parties $\mathcal{O}_r \subseteq \mathcal{P}$. This is intended to represent the *online* parties in round r.
- 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.
- 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.

2.2 A Stable Network on Top of an Unstable Network

As we have mentioned already, our approach is to instantiate $\mathcal{F}_{\mathsf{StableNet}}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model. In this model, and considering an active adverary, there exist computationally secure protocols with $t < n$ (e.g. [6]), statistically secure protocols with $t < n/2$ (e.g. [15,5])[6], and perfectly secure protocols with $t < n/3$

[6] These protocols require an additional *broadcast channel* which, unlike in the other two settings, cannot be instantiated from point-to-point channels. Since such channel must be assumed anyway, we do not bother with instantiating it in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model. We elaborate in Section D.4 in the Supplementary Material.

(e.g. [3]). It turns out that composability still holds even after extending the UC framework with the environments from previous sections, a fact that we prove in Section 2.3 in the Supplementary Material. As a result, a protocol that instantiates $\mathcal{F}_{\mathsf{StableNet}}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model would carry the results above from the $\mathcal{F}_{\mathsf{StableNet}}$ networking setting to $\mathcal{F}_{\mathsf{UnstableNet}}$, effectively enabling secure MPC over an unstable network. Furthermore, we remark that, as we have already hinted, to instantiate $\mathcal{F}_{\mathsf{StableNet}}$ it suffices to provide instantiations to the individual channels $\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}$ for $i, j = 1, \ldots, n$.

B-termination assumption. 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 assume that the adversary, or rather, the environment, enables parties to become online "every once in a while". This is captured by the B-assumption, defined next.

Definition 1. *Let B be a positive integer. We say that an adversary respects the B-assumption if, for every party P_i and for every non-negative multiple of B, $r \cdot B$, there exists $1 \leq k \leq B$ such that $P_i \in \mathcal{O}_{r \cdot B + k}$.*

Consider a sender P_S who wishes to send a message to a receiver P_R. If it is the adversary's goal to delay this delivery as much as possible, while still respecting the B-assumption, then a possible scheduling could consist of the following: among the rounds $r = 1, \ldots, B$, only set P_S online in round B, and P_R in round 1; among the rounds $r = B+1, \ldots, 2B$, only set P_R online in round $2B$. With this scheduling, we see that P_R cannot get the message until round $2B$, because it was only online in two rounds, 1 and $2B$, but it cannot receive the message on round 1 since up to that point P_S has not been online in order to send the message. Our protocols from Sections E, 3 and 4 guarantee that each message is delivered within $2B$ rounds, which is optimal according to the reasoning above.

Finally, we stress that the B-assumption is only needed to ensure all honest parties are guaranteed to provide input and receive output, and we recall that, in Section 5, we present a perfectly secure MPC protocol (not an instantiation of $\mathcal{F}_{\mathsf{Stable}}$) that does not require the B-assumption and is able to make progress just assuming that the intersection of online parties from round to round is large enough. Furthermore, the $2B$ lower bound only applies to the instantiation of $\mathcal{F}_{\mathsf{StableNet}}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model. For general MPC, as we show in Section 5 for the case of perfect security, the parties can make progress on the computation directly over the unstable network without paying a penalty in the round complexity, spending one round per multiplication layer in the circuit. The $2B$-round overhead will only be apparent in the output phase when requiring all parties to receive output.

2.3 The Composition Theorem for Unstable Networks

Recall that our main goal is to instantiate the $\mathcal{F}_{\mathsf{StableNet}}$ functionality in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model, with the ultimate goal of, as we present in Section D.3 in the Supplementary Material, leveraging the existence of different MPC protocols over traditional stable networks to obtain equivalent protocols over unstable networks. This, however, requires the *composition theorem* to hold, which states that composing protocols that are proven secure individually leads to a secure protocol. This is known to hold in the “standard” UC framework (as suggested by the naming Universal Composability), but it is crucial to argue that the changes we have introduced in order to model instability do not affect this result. This is indeed the case, as we now show.

The Composition Theorem, as stated (and proven) in [9], shows that composition holds as long as the set of environments involved constitutes what is called an *environment class*. Our approach consists then of simply showing that the set of environments we have defined in our modeling of an unstable network forms indeed an environment class. We argue this in what follows.

Environment classes. We first introduce the notion of an *environment class*, presented as Definition 4.17 in [9]. A set of environments Env forms an environment class if for every $\mathcal{Z} \in \mathsf{Env}$, every protocol Π and every simulator $\mathcal{S}$, the composed automatas $\mathcal{Z} \diamond \Pi$ and $\mathcal{Z} \diamond \mathcal{S}$ both belong to Env. We will see below where the notion of an environment class becomes relevant in the proof of the composition theorem.

The composition theorem. Now we discuss the composition theorem together with its proof. We remark that this discussion is kept at an intuitive level, and its main goal is to convince the reader that, for the composition theorem to hold, it suffices to show that the set of environments under consideration constitutes an environment class. For all the details that are left out of the discussion we refer the reader to [9].

Theorem 1 (Composition Theorem, Thm. 4.20 in [9]). *Let Env be an environment class. Let $\mathcal{F}$, $\mathcal{H}$ and $\mathcal{R}$ be functionalities. Let $\Pi_{\mathcal{F}}$ be a protocol that securely instantiates $\mathcal{F}$ in the $\mathcal{R}$-hybrid model, and let $\Pi_{\mathcal{R}}$ be a protocol that securely instantiates $\mathcal{R}$ in the $\mathcal{H}$-hybrid model. Then the composed protocol $\Pi_{\mathcal{R}} \diamond \Pi_{\mathcal{F}}$ securely instantiates $\mathcal{F}$ in the $\mathcal{H}$-hybrid model.*

Proof (Sketch). For the sake of illustrating where the assumption that Env is an environment class plays in, we present the intuition behind the proof of the composition theorem. Since $\Pi_{\mathcal{F}}$ securely instantiates $\mathcal{F}$ in the $\mathcal{R}$-hybrid model, we have that there exists a simulator $\mathcal{S}_{\mathcal{F}}$ such that for any $\mathcal{Z}_{\mathcal{F}} \in \mathsf{Env}$, the random variables $\mathcal{Z}_{\mathcal{F}} \diamond \Pi_{\mathcal{F}} \diamond \mathcal{R}$ and $\mathcal{Z}_{\mathcal{F}} \diamond \mathcal{S}_{\mathcal{F}} \diamond \mathcal{F}$ have a “small” statistical distance, a situation we denote by $\mathcal{Z}_{\mathcal{F}} \diamond \Pi_{\mathcal{F}} \diamond \mathcal{R} \equiv \mathcal{Z}_{\mathcal{F}} \diamond \mathcal{S}_{\mathcal{F}} \diamond \mathcal{F}$. Similarly, since $\Pi_{\mathcal{R}}$ securely instantiates $\mathcal{R}$ in the $\mathcal{H}$-hybrid model, there exists a simulator $\mathcal{S}_{\mathcal{R}}$ such that for any $\mathcal{Z}_{\mathcal{R}} \in \mathsf{Env}$, it holds that $\mathcal{Z}_{\mathcal{R}} \diamond \Pi_{\mathcal{R}} \diamond \mathcal{H} \equiv \mathcal{Z}_{\mathcal{R}} \diamond \mathcal{S}_{\mathcal{R}} \diamond \mathcal{R}$.

We claim that the composed automata $\mathcal{S}_{\mathcal{F}} \diamond \mathcal{S}_{\mathcal{R}}$ is a simulator for the protocol $\Pi_{\mathcal{F}} \diamond \Pi_{\mathcal{R}}$, that is, for any $\mathcal{Z} \in \mathsf{Env}$ it holds that $\mathcal{Z} \diamond (\Pi_{\mathcal{F}} \diamond \Pi_{\mathcal{R}}) \diamond \mathcal{H} \equiv \mathcal{Z} \diamond (\mathcal{S}_{\mathcal{F}} \diamond \mathcal{S}_{\mathcal{R}}) \diamond \mathcal{F}$. To see this, first we notice that, since Env is an environment class, we have that

$\mathcal{Z}_{\mathcal{F}} := \mathcal{Z} \diamond \mathcal{S}_{\mathcal{R}}$ and $\mathcal{Z}_{\mathcal{R}} := \mathcal{Z} \diamond \mathit{\Pi}_{\mathcal{F}}$ both belong to Env, so using the two expressions from before:

$$\begin{aligned}\mathcal{Z} \diamond (\mathit{\Pi}_{\mathcal{F}} \diamond \mathit{\Pi}_{\mathcal{R}}) \diamond \mathcal{H} &\equiv \overbrace{(\mathcal{Z} \diamond \mathit{\Pi}_{\mathcal{F}})}^{\mathcal{Z}_{\mathcal{R}}} \diamond \mathit{\Pi}_{\mathcal{R}} \diamond \mathcal{H} \equiv \overbrace{(\mathcal{Z} \diamond \mathit{\Pi}_{\mathcal{F}})}^{\mathcal{Z}_{\mathcal{R}}} \diamond \mathcal{S}_{\mathcal{R}} \diamond \mathcal{R} \\ &\equiv \underbrace{(\mathcal{Z} \diamond \mathcal{S}_{\mathcal{R}})}_{\mathcal{Z}_{\mathcal{F}}} \diamond \mathit{\Pi}_{\mathcal{F}} \diamond \mathcal{R} \equiv \underbrace{(\mathcal{Z} \diamond \mathcal{S}_{\mathcal{R}})}_{\mathcal{Z}_{\mathcal{F}}} \diamond \mathcal{S}_{\mathcal{F}} \diamond \mathcal{F} \equiv \mathcal{Z} \diamond (\mathcal{S}_{\mathcal{F}} \diamond \mathcal{S}_{\mathcal{R}}) \diamond \mathcal{F}.\end{aligned}$$

□

"Unstable" environments form an environment class. Given what we have seen so far, in order to show that the composition theorem holds in our new unstable networking model, it suffices to show that our custom environments, as defined formally in Section D.2 in the Supplementary Material, constitute an environment class Env. Consider $\mathit{\Pi}$ a protocol and $\mathcal{S}$ a simulator, and let $\mathcal{Z} \in \mathsf{Env}$. Our goal is to show that both $\mathcal{Z} \diamond \mathit{\Pi}$ and $\mathcal{Z} \diamond \mathcal{S}$ are in Env.

$\mathcal{Z} \diamond \mathit{\Pi} \in \mathsf{Env}$. We first show that the composition of $\mathcal{Z}$ with a protocol $\mathit{\Pi}$ lies again in Env. This is relatively simple, given that the composition of $\mathcal{Z}$ with a protocol does not affect the ports the environment has with a given functionality, so if the environment sends the commands such as erase or schedule, it will keep doing so after composition with $\mathit{\Pi}$. This shows that $\mathcal{Z} \diamond \mathit{\Pi} \in \mathsf{Env}$.

$\mathcal{Z} \diamond \mathcal{S} \in \mathsf{Env}$. The composition of an environment with a simulator is slightly more difficult to tackle, mainly because the simulator connects to $\mathcal{Z}$ via the ports where the latter would communicate the unstable-network-related commands. To fix this, we simply expand the definition of the simulators so that they are required to forward all the unstable-network-related commands that $\mathcal{Z}$ issues to the functionality at hand. This way, the composed automata $\mathcal{Z} \diamond \mathcal{S}$ would provide the same interface as $\mathcal{Z}$, which implies that $\mathcal{Z} \diamond \mathcal{S} \in \mathsf{Env}$.

This modification can be regarded as an artifact to make the formal model work, and it is analogous to (and in fact, must be performed in addition to) the *clock-preserving* property presented in [9], which requires the simulator to forward the clockin and clockout commands that model synchrony from the environment to the functionality under consideration. However, notice that in the case of synchrony, it is natural that the functionality at hand knows how to handle these synchrony-related commands (e.g. an OT functionality may use the clock commands to determine when to receive inputs and when to send outputs), but in the case of an unstable network, commands like erase or schedule may not be of any use to many functionalities. For example, $\mathcal{F}_{\mathsf{StableNet}}$ certainly does not know how to handle these commands, since they only make sense in the context of an unstable network. Nevertheless, we can safely assume this type of functionalities ignore these unstable-network-related commands.

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

In this section we take care of instantiating the functionality for a stable network with perfect security. First, in Section 3.1 we discuss the simplest setting of passive security. Then, in Section 3.2 we extend this to active security, while retaining perfect simulation.

3.1 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 $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ 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.

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 $\mathtt{f}_i = \bot$. 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 $\mathtt{f}_i \neq \bot$, 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 $m = f_0(0)$.

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.

Theorem 2. *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 $r_S \in \{1, \dots, B\}$ 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 $r_R \in \{B+1, \dots, 2B\}$ 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 $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$.

Assume the existence of an implementation of $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ 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 5, 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.

3.2 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 $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R, m)$, is similar to Protocol $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,passive}}(P_S, P_R, m)$, 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 3. *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every $r > 0$. Then, protocol $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_R, P_S)$ 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.[7]

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 $r_S \in \{1, \dots, B\}$ 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 $\mathtt{f}_i(j) = f(j, i) = f(i, j)$. As discussed in Section B, 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 $r_R \in \{B + 1, \dots, 2B\}$ 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 2.

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. As with the proof of Theorem 8, 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 3.1, 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

[7] 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 $n \geq 3t + 1$.

of $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ 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 $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ 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.

4 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 3.2 with $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$. As shown in Section 3.2, 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 $f(m) = (m_1, \dots, m_n)$ and $g(m'_1, \dots, m'_n) = m'$ 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 $(m_1, \dots, m_n) = f(m)$, 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 $m'_1, \dots, m'_n$ be the values received by P_R when it comes online, where $m'_i = \bot$ 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 $m = g(m'_1, \dots, m'_n)$.

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 $(m_1, \dots, m_n) = f(m)$ and executes $\Pi_1(P_S, P_R, m_i)$ for $i = 1, \dots, n$. 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 $m = g(m'_1, \dots, m'_n)$.

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 $g(m_1, \dots, m_n)$ 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 $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$.

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 2. *Let $A \subseteq \{1, \dots, n\}$ 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 $(s_1, \dots, s_n) = \mathsf{share}(m)$. 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 $(s'_1, \dots, s'_n)$ 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 $1 - \mathsf{negl}(\kappa)$, $\mathsf{rec}(s'_1, \dots, s'_n)$ outputs either m or $\perp$.*
- ***Guaranteed reconstruction.*** *If $|A^c \cap B| > t$ then, with probability $1 - \mathsf{negl}(\kappa)$, it holds that $m = \mathsf{rec}(s'_1, \dots, s'_n)$.*

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 2, which is motivated on the so-called information-checking signatures from [21]. We remark that *any* instantiation of Definition 2 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 1. *The construction $(\mathsf{share}, \mathsf{rec})$ from above is an RSS scheme with deletions.*

Proof. Let $\mathsf{share}(m) = (s_1, \dots, s_n)$ 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 $m_1, \dots, m_n$ 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 $(s'_1, \dots, s'_n)$ be as in Definition 2. Assume that $|A^c \cap B| > t$, we want to show that $\mathsf{rec}(s'_1, \dots, s'_n)$ 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

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 $B = \{i : s'_i \neq \bot\}$. 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 $\alpha'_j m'_i + \beta'_{ji} \overset{?}{=} \tau'_{ij}$ does not hold for at least $t+1$ values of $j \in B$, then set $m'_i = \bot$.[a]
2. After this process, if $|\{m'_i : m'_i \neq \bot\}| > t$, 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 $\bot$.

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

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 $\bot$ 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 $\{m'_i : m'_i \neq \bot\}$ 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 $\bot$. □

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.

Definition 3 (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.*

The following protocol satisfies 1-delivery:

Protocol $\mathit{\Pi}_1(P_S, P_R, m)$

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 $\mathsf{s}_i = \perp$. In every round, if P_i receives s_i from P_i, then it sets $\mathsf{s}_i = s_i$.
- In every round, if $\mathsf{s}_i \neq \perp$, then P_i sends 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.

Proposition 2. *$\mathit{\Pi}_1(P_R, P_S, m)$ 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 $(\mathsf{share}, \mathsf{rec})$. □

From $(k-1)$-delivery to k-delivery. Now we show that, given a protocol $\mathit{\Pi}_{\mathsf{k-1}}(P_R, P_S, \cdot)$ that achieves $(k-1)$-delivery, one can obtain a protocol that achieves k-delivery.

Proposition 3. *Protocol $\mathit{\Pi}_{\mathsf{k}}(P_S, P_R, m)$ 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 $\mathit{\Pi}_{\mathsf{k-1}}$.

Protocol $\mathit{\Pi}_{\mathsf{k}}(P_R, P_S, m)$

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 $\mathit{\Pi}_{\mathsf{k-1}}(P_S, P_R, m)$. In parallel, they execute the following.

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

Given the above, it remains to analyze the case in which $P_R \in \mathcal{O}_{r_S+k}$. From the properties of $\mathit{\Pi}_{\mathsf{k-1}}$, 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 2 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. This concludes the proof. □

Combining Propositions 2 and 3, we obtain the following corollary:

Corollary 1. *For every k, there exists a protocol $\mathit{\Pi}_{\mathsf{k}}$ 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 4. *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ for every $r > 0$. Then, protocol $\mathit{\Pi}_{\mathsf{2B}}(P_R, P_S, \cdot)$ 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.*[8]

Remark 1. The communication complexity of $\mathit{\Pi}_{\mathsf{k}}$ is $\mathit{\Theta}(n^k)$. This is because, in the execution of $\mathit{\Pi}_{\mathsf{k}}$, P_S must use $\mathit{\Pi}_{\mathsf{k-1}}$ to communicate a share to each single party, adding a factor of n with respect to the communication complexity of this protocol.

[8] As with Theorem 3, 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 $n \geq 2t + 1$.

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.

5 A More Efficient Protocol with Perfect Security

Recall that in Section 3.2 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 $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R)$ from Section 3.2 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 $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R)$ requires $2B$ rounds to deliver a message between a sender and a receiver, which ultimately means that the final protocol after composing $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R)$ 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.[9] 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.

Remark 2. (Low-round complexity in the computational setting) As mentioned above, if the high level protocol has a low/constant number of rounds then the

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

$\theta(B)$ overhead is less of a problem. In the computational setting constant round protocols can be designed, for example the early works based on garbled circuits [2] or on threshold variants of fully homomorphic encryption [20]. For instance, we could use the 3-round protocol from [1] together with $\Pi^{\mathsf{comp,active}}_{\mathsf{StableNet}}(P_R, P_S)$ from Section E.2 to obtain a computationally secure protocol in an unstable network using $3 \cdot (2B)$ rounds. The first $2B$ rounds would consist of the parties sending to each other certain parameters for an underlying threshold multi-key fully homomorphic encryption scheme and a non-interactive zero-knowledge protocol. In the second $2B$ rounds the parties send to each other encryptions of their inputs, and the remaining $2B$ rounds consist of the parties sending decryption shares to recover the output, after computing homomorphically on the ciphertexts received in the previous rounds.

5.1 Bivariate Sharings and Transition of Shares

We describe the input and preprocessing phases of our protocol in Section 5.2, and in Section 5.3 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, defined initially in Section B, 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 3.2 to instantiate $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ with perfect security against an active adversary. This type of sharings proved useful in Protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_S, P_R)$ 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 $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ from Section 3.

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 $i = 1, \dots, n$, 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 $j = 1, \dots, n$.
2. For each $j = 1, \dots, n$, if P_j receives at least $2t+1$ messages $\{f(j,i)\}_i$, then P_j performs *enhanced* error correction (see Section B) to either recover $f(j,x)$ or output an error $\perp$.

Theorem 5. *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 3. An important fact about the proof of Theorem 5 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 5.3.

5.2 Preprocessing and Input Phases

We assume that the functionality to be computed is given by a layered circuit $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$, as defined in Section C in the Supplementary Material. 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 from Section E, these shares can be computed by using *any* generic MPC protocol for these tasks, together with our compiler from Section 3.2. 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 $c = a \cdot b$.

5.3 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 5.2, *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 5.1, and a sketch of this proof can be found in Section F in the Supplementary Material.

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 $\mathit{\Pi}_{\mathsf{MPC}}$

Input: Secret-shared inputs $\langle x_1^{(0)} \rangle, \dots, \langle x_{\ell_0}^{(0)} \rangle$, 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 $\langle x_1^{(L)} \rangle^{\mathcal{O}_L}, \dots, \langle x_{\ell_L}^{(L)} \rangle^{\mathcal{O}_L}$ in round L, where $(x_1^{(L)}, \dots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \dots, x_{\ell_0}^{(0)})$.

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

- The parties in round $r-1$ already have shares $\langle x_1^{(r-1)} \rangle^{\mathcal{O}_{r-1}}, \dots, \langle x_{\ell_{r-1}}^{(r-1)} \rangle^{\mathcal{O}_{r-1}}$.
- The parties in round r obtain shares $\langle x_1^{(r)} \rangle^{\mathcal{O}_r}, \dots, \langle x_{\ell_r}^{(r)} \rangle^{\mathcal{O}_r}$ 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 $\langle x + y \rangle^{\mathcal{O}_r} \leftarrow \mathit{\Pi}_{\mathsf{transfer}}(\langle x + y \rangle^{\mathcal{O}_{r-1}})$.
  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 $d \leftarrow \mathit{\Pi}_{\mathsf{rec}}(\langle d \rangle^{\mathcal{O}_{r-1}})$ and $e \leftarrow \mathit{\Pi}_{\mathsf{rec}}(\langle e \rangle^{\mathcal{O}_{r-1}})$.
     (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 $\langle x \rangle^{\mathcal{O}_r} \leftarrow \mathit{\Pi}_{\mathsf{transfer}}(\langle x \rangle^{\mathcal{O}_{r-1}})$.

[a] Here is where Remark 3 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 4 (About the output). In our protocol above, the parties in $\mathcal{O}_L$ obtain shares $\langle x_1^{(L)} \rangle^{\mathcal{O}_L}, \dots, \langle x_{\ell_L}^{(L)} \rangle^{\mathcal{O}_L}$ in round L, where $(x_1^{(L)}, \dots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \dots, x_{\ell_0}^{(0)})$ 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.

References

1. S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. Secure MPC: Laziness leads to GOD. pages 120–150, 2020.
2. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). pages 503–513, 1990.
3. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. pages 213–230, 2008.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). pages 1–10, 1988.
5. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. pages 663–680, 2012.
6. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. pages 169–188, 2011.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.
8. A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid mpc: Secure multiparty computation with dynamic participants. In *Annual International Cryptology Conference*, pages 94–123. Springer, 2021.
9. R. Cramer, I. B. Damgård, and J. Nielsen. *Secure multiparty computation*. Cambridge University Press, 2015.
10. I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. pages 160–179, 2009.
11. I. Damgård, D. Escudero, and D. Ravi. Information-theoretically secure mpc against mixed dynamic adversaries. Thheory of Cryptography Conference, 2021.
12. M. Fitzi, M. Hirt, and U. M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). pages 121–136, 1998.
13. P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Information processing letters*, 43(4):169–174, 1992.
14. C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO 2021*, 2021.
15. V. Goyal, Y. Song, and C. Zhu. Guaranteed output delivery comes free in honest majority MPC. pages 618–646, 2020.
16. Y. Guo, R. Pass, and E. Shi. Synchronous, with a chance of partition tolerance. pages 499–529, 2019.
17. J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2020.
18. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. pages 477–498, 2013.
19. C.-Y. Koo. Secure computation with partial message loss. pages 502–521, 2006.
20. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. pages 1219–1234, 2012.
21. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). pages 73–85, 1989.

22. R. Rachuri and P. Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. Cryptology ePrint Archive, Paper 2021/1579, 2021. https://eprint.iacr.org/2021/1579.
23. G. Spini and G. Zémor. Perfectly secure message transmission in two rounds. In *Theory of Cryptography Conference*, pages 286–304. Springer, 2016.
24. V. Zikas, S. Hauser, and U. M. Maurer. Realistic failures in secure multi-party computation. pages 274–293, 2009.

Supplementary Material

A Related Work

Fail-stop adversaries. A series of works have studied the setting of MPC, where the adversary is allowed to not only corrupt some parties passively/actively, but also cause some parties to fail (e.g. [12] and subsequent works). However, their setting differs to ours in several aspects. First, in these works it is typically assumed that parties who are set to fail do *not* do so *silently*, i.e. all the other parties know when a given party failed. Second, and most crucially, once a party is set to fail by the adversary, it does not return to the computation.

LazyMPC. The work of [1] considers an adversary that can set parties to be offline at any round (called "honest but lazy" in that work). This work differs from ours in several places. First, the authors focus only on the case of computational security, making use of rather strong techniques such as multi-key fully homomorphic encryption. Second, just like the case of the fail-stop parties described above, once a party becomes offline, or "lazy", it is assumed not to come back. This has the impact that, in particular, honest parties who leave the computation do not receive output.

Synchronous but with partition tolerance. Recently, the work of [16] designed MPC protocol in the so-called "sleepy model", which enables some of the parties to lag behind the protocol execution, while not being marked as corrupt. This could be achieved with an asynchronous protocol, naturally, but the main result of [16] is obtaining such protocols without the strong threshold assumptions required to obtain asynchronous protocols. In particular, the authors obtain *computationally secure* constant-round protocols, assuming that the set of "fast"-and-honest parties in every round constitutes as majority, an assumption that is shown to be necessary.

The honest-and-fast-majority assumption implies the one we use in this work for the computational case: $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for every round r, so, in a way, the results in [16] (except for the constant-round aspect of their protocols) can be derived from our work as well. However, a crucial aspect of our protocols and our model which is not present in [16] is the following. In our setting, parties are set to drop out from the protocol execution, and they can rejoin at some point in the future. Importantly, we do not assume that parties receive the messages sent to them while they are offline, that is, after rejoining the computation, a given party has an outdated view of the protocol execution. In contrast, the model in [16] is not described as parties being "offline", but rather as simply being slow. In particular, once they return to "normal speed", they receive all the messages that were sent to them in previous rounds. This is explicitly used in the protocol from [16], and the fact that we are avoiding such assumption makes designing protocols in our setting a considerably harder task.

FluidMPC. In [8], the "fluid" model of MPC is introduced with a statistical secure protocol. In this model, instead of considering a fixed set of parties as done traditionally in MPC, the set of parties involved in the protocol execution can be different from one round to the next one. This setting is related to ours since, in a way, we can interpret our model of parties dropping out and rejoining the computation at given times as if the set of active parties in the protocol changed from round to round.

However, a feature that seems to be essential for the protocol in [8] is that the identities of the parties that are active in a given round are known beforehand. This is not the case in our model: the set of parties that are active in a given round are those that the adversary have not "taken out", and this set is not known to the parties in the protocol (in fact, a party does not even know if it is "offline" or not).

Finally, we also notice that the protocol in [8] satisfies security with abort. On the other hand, the protocol from our work that is somewhat comparable to that from [8] in terms of security setting, namely the statistically secure one from Section 4, satisfies a form of guaranteed output delivery, in the sense that parties who eventually return to the computation are guaranteed to receive output. It is not clear how to extend [8] to achieve such notion given that their protocol is based on certain checks that only enable the parties to detect that some errors have been introduced, without being able to somehow recover from these.

YOSO. In the recent work of Gentry et al. [14], the "You Only Speak Once" model for MPC is introduced. In this model, the basic assumption is that the adversary is able to take a party down as soon as that party sends a message – using, say, a denial of service attack. Although some number of parties are assumed to be alive and can receive messages, no particular party is guaranteed to come back (which is the major difference to our model). Instead, the YOSO model breaks the computation into small atomic pieces called *roles* where a role can be executed by sending only one message. The responsibility of executing each role is assigned to a physical party in a randomized fashion. The assumption is that this will prevent the adversary from targeting the relevant party until it sends its (single) message. This means that one should think of the entire set of parties as one "community" which as a whole is able to provide secure computation as a service. In a sense, YOSO aims to make progress and keep the computation alive without any guarantees for particular physical parties such as contributing inputs and receiving the output. This makes good sense in the context of a blockchain, for instance. On the other hand, the demand that the MPC protocol must be broken down into roles makes protocol design considerably harder, particularly for information theoretically secure protocols. An additional caveat with the YOSO model is that one can only have information theoretically or statistically secure protocols assuming that the role assignment mechanism is given as an ideal functionality, and an implementation of such a mechanism must inherently be only computationally secure. In comparison, our model assumes a somewhat less powerful adversary who must allow a physical party to come back after being offline. This allows for much easier protocol design, information theoretic security based only on point-to-point secure channels, and allows termination such that all parties can provide input and get output.

Omission-corruption model. The “constrained parties” and “full-omission parties” from [19] and [24], respectively, are closely related to the unstable parties in our model. A full-omission corrupted party is an honest party whose messages are selectively blocked by the adversary, as in our setting. However, a crucial difference of our model with respect to the one in [24] (which contains improved results with respect to [19]) is that, in [24], the adversary can full-omission-corrupt a given subset of the parties *at the beginning of the protocol execution*, whereas in our case this subset can change adaptively as the protocol is run.[10] 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.

In terms of results, the work of [24] shows that perfectly secure MPC in the full-omission setting is possible if and only if $3t + 2s < n$, where t and s are the number of active and full-omission corruptions, respectively. The part of our work that deals with perfect security produces this result as a particular case: the adversary chooses a set $\mathcal{O} \subseteq \mathcal{H}$ of $n - t - s$ honest parties that will *not* be full-omission corrupt, and when translated to our setting all of our sets $\mathcal{O}_r$ would be equal to $\mathcal{O}$. As a result, our optimality condition $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t + 1$ becomes $|\mathcal{O}| \geq 2t + 1$, which is equivalent to $n \geq 3t + 2s + 1$.

Finally, [24] also mentions in a closing remark that one can allow even honest players to lose some messages in each round. However, no details or proof is given, so we cannot do a meaningful comparison to our results. In any case, such a comparison would be relevant only for our network compilation result in the perfect case. Our direct construction for the perfect case is of a different nature (and much more efficient).

B 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

[10] We remark, however, that [24] is more fine-grained in that they consider different parties who can be blocked for sending and separately for receiving messages.

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 [13].

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 [11]).

C Layered Circuits

As described before, we present in Section 5 a *direct* construction of an MPC protocol in the perfectly secure setting. This construction will make use of the concept of a layered circuit [8, Definition 6]. A *layered circuit* is a circuit that can be decomposed into *layers*, indexed by integers $0,\dots,L$. Wires in layer 0 are input wires, and these in layer L are output wires. We denote by ℓ_i the number of wires in layer i, and we denote by $x_1^{(i)},\dots,x_{\ell_i}^{(i)}$ the values in these wires. For every $i = 1, \ldots, L$, every value $x_j^{(i)}$ is either the sum or product of two values $x_k^{(i-1)}$ and $x_h^{(i-1)}$, or it is equal to a value $x_k^{(i-1)}$. In other words, all wires in a given layer are a function of the wires in the immediate previous layer, only.

We assume that the function $f(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$ is given by a layered circuit with L-layers. This is not very restrictive, as it is shown in [8] that any arithmetic circuit over a field $\mathbb{F}$ with depth d and width w can be transformed into a layered circuit having $L = d$ layers and maximum width $2w$.

D Unstable Network with Dropouts and Comebacks

Our starting point is a synchronous network, where an upper bound Δ on the time it takes for a message to be transmitted between any pair of parties is known. 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. Since each round r takes Δ time, it is guaranteed that all the messages sent at the beginning of round r will be delivered within round r.

In an unstable network with dropouts and comebacks, the parties are allowed to drop from the computation at any given round, potentially missing some of the messages sent in that round, as well as failing to send some of their own messages. Furthermore, as clarified in more detail later on, a crucial aspect of our model is that the parties who return to the computation after dropping out for one or more rounds are not assumed to receive the messages that were sent to them during this offline period.

The set of parties who are set to go offline in each round is specified by the adversary. We denote by $\mathcal{O}_r$ the set of online parties in round r.[11] Although several dropouts and comebacks are likely to be caused by more "non-adversarial" events (e.g. a party running MPC from a phone entering a tunnel while on a train), allowing the adversary to control such scheduling makes our results stronger. We assume a rushing adversary, which in particular means that the adversary gets to decide which parties to set offline in a given round even after learning the messages that the honest parties send to the corrupt parties. Additionally, once the adversary has chosen which honest parties will be set to go offline in that round, the adversary can choose which messages from and to these parties are actually delivered.

Recall that an honest party does not know whether it was set to be offline in a particular round. For example, an honest party may fail to receive certain messages while still receiving others, and this could either be because the senders were offline, or because the receiving honest party was offline. This imposes a big challenge when designing protocols in an unstable network since it is not possible for the parties to selectively send messages depending on whether the receiver is online or not, for example.

[11] This notation is similar to the one in [16], except in that work $\mathcal{O}_r$ denotes the set of online *and honest* parties, which would correspond in our notation to $\mathcal{O}_r \cap \mathcal{H}$.

Another complication of working in an unstable network with dropouts and comebacks is that honest parties may not contribute to the computation anymore, even if they eventually rejoin the computation. For example, imagine an honest party that is offline for most of the computation, so it misses essentially all the messages. This party may rejoin after a while, and maybe in the round in which this party is online it manages to receive enough information to be able to contribute in the next round. However, the problem here is that there are no guarantees that this party will be online for contributing in the next round. In general, we do not assume that a party who returns to the computation stays for long enough time to receive the messages sent to it in the comeback round, as well as sending messages in the following round.[12]

In what follows we describe in detail our model for an unstable network with dropouts and comebacks.

D.1 UC Framework

The UC framework was initially introduced by Canetti [7]. However, different variants and alternatives have been proposed in the literature. In our work, we follow the definition of the universally-composable (UC) framework as defined in [9, Chapter 4], which we find conceptually simpler than other alternatives in the literature and lets us define more appropriately the concept of an unstable network. We first provide a high level overview of this UC model, before proceeding to our modifications for the unstable network in Section D.2.

We begin by discussing some basic concepts.

I/O automata. This is a recursive polytime machine (as defined in [9]) that has named *ports*, which are common message tapes that the machine can write to and read from. If different machines have the same named ports then the resource is shared. Finally, different automatas can be composed, meaning they form a larger system where the automatas connect to each other on their open ports with matching names. This operation is denoted by $\diamond$

Ideal functionalities. These are I/O automatas that model the way the parties can interact with each other. It has two connections to each of the parties, one to send and another to receive messages. An ideal functionality may simply model authenticated or secure channels, or it may model something more involved such as an oblivious transfer channel. It is also used to model other types of interaction like a complex computation done on the inputs received from the parties. An ideal functionality also connects to the environment to allow adversarial control, plus other low-level details like activations, which dictates when a given party "acts" in the protocol.

[12] If we require the adversary to let parties who return to the computation stay online for at least two rounds (that is, if $P_i \in \mathcal{O}_{r-1}^c \cap \mathcal{O}_r$ then $P_i \in \mathcal{O}_{r+1}$), then several of the obstacles we need to overcome in this work would not be present anymore. This assumption is not too unrealistic.

Communication resource. A particular type of functionalities that are of high relevance are communication resources. These functionalities model the underlying network over which a given protocol is run, and we will use them to model our stable and unstable networks.

Protocols. A protocol is a collection of I/O automatas $\{P_1, \ldots, P_n\}$ connected through a communication resource, and each connected to the environment.

Environment. This is an I/O automata $\mathcal{Z}$ that is connected to both the parties and an ideal functionality serving as the *communication resource*. It is in charge of several things, like providing inputs to the computation, orchestrating it by *activating*[13] the machines it is connected to in certain specific order, overriding the behavior of actively corrupt parties or manipulating the communication resource. Finally, it is also $\mathcal{Z}$ the machine that has to distinguish real/ideal executions, as described below.

Simulator. The goal of a protocol is to achieve the "same" behavior as some given ideal functionality. The simulator $\mathcal{S}$ is an I/O automata that "sits between" the corrupt parties (controlled by $\mathcal{Z}$) and the ideal functionality $\mathcal{F}$ that the protocol is supposed to instantiate. $\mathcal{S}$ connects to $\mathcal{Z}$ and $\mathcal{F}$, and $\mathcal{Z}$ executes the computation $\mathcal{S}$ just like it was doing it with the real parties. The goal of $\mathcal{S}$ is then to ensure $\mathcal{Z}$ cannot distinguish between an execution with real parties and one in which $\mathcal{S}$ is involved.

Corruptions. The environment is also in charge of executing corruptions. For the case of active corruptions, $\mathcal{Z}$ gets absolute control of the chosen t corrupt parties during the xecution of the protocol. For the case of passive corruptions, $\mathcal{Z}$ is only allowed to see the messages that the chosen t corrupt parties send and receive.

Real and Ideal Worlds. Consider a pair of functionalities $\mathcal{R}$ and $\mathcal{F}$, and consider a protocol Π that is intended to realize the functionality $\mathcal{F}$, making use of the resource $\mathcal{R}$. The real and ideal worlds are described as follows (a good graphical illustration of the real and ideal worlds is Figure 4.7 in [9]).

- In the *real world* the environment $\mathcal{Z}$, the resource $\mathcal{R}$ and the protocol Π are connected, which is denoted by $\mathcal{Z} \diamond \Pi \diamond \mathcal{R}$. The environment orchestrates the execution, plays the role of actively corrupt parties, sees the state of passively corrupt parties and provides inputs and receives outputs of honest parties. $\mathcal{Z}$ also has certain limited interaction with $\mathcal{R}$ (e.g. if $\mathcal{R}$ models a communication channel, then $\mathcal{Z}$ can typically learn when parties communicate and control the order in which messages are delivered).
- In the *ideal world* the environment $\mathcal{Z}$ provides the honest parties' inputs to the functionality $\mathcal{F}$ instead, and the corrupted parties controlled by $\mathcal{Z}$ interact with the simulator $\mathcal{S}$ instead, who is also in charge of emulating the resource $\mathcal{R}$, and gets to interact with the actual functionality $\mathcal{F}$ on behalf of the corrupt parties. This composed system is denoted by $\mathcal{Z} \diamond \mathcal{S} \diamond \mathcal{F}$.

[13] Only *active* parties are allowed to run at a given time. A party is activated by inputting a special activation token, which is returned upon termination of the current activation step.

Security. At a high level, a protocol Π for a given functionality $\mathcal{F}$ is secure if, for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that the statistical distance of the two random variables $\mathcal{Z} \diamond \Pi \diamond \mathcal{R}$ and $\mathcal{Z} \diamond \mathcal{S} \diamond \mathcal{F}$ is very small.[14] Computational security relates to the fact that such distance is a negligible function for each environment, but only polynomial-time environments are considered, while statistical security means that the environments are not restricted. Perfect security refers to the case in which this distance is exactly 0.

We are deliberately using intuitive language, leaving a lot of details out of our discussion. We remark that our intention is mostly to recap basic notions which are useful when we discuss the extension to unstable networks, and we refer the reader to the thorough description from, say [9], for full details.

Synchrony and Stable Networks. We take the approach from [9] of defining synchrony as a restriction of the way in which the environment activates the different parties in a protocol. This, in contrast to other approaches to defining synchronous communication in the UC framework such as the one from [18], fits much better our extension to an unstable network from Section D.2, and it is conceptually much simpler.

Synchronous protocols proceed by rounds. Each round allows the parties to send messages to the communication resource and hear back from it after it has proceessed all the messages. A *synchronous environment* activate honest and semi-honest parties within a round as dictated below. Actively corrupt parties can be activated at any point.

1. For every $P_i \in \mathcal{H} \cup \mathcal{SH}$, $\mathcal{Z}$ activates P_i and then sends $(\mathsf{clockin}, P_i)$ to the communication resource $\mathcal{R}$. Then $\mathcal{Z}$ activates $\mathcal{R}$. Overall, this allows P_i to send messages to the communication resource, which can then be processed. The parties can be chosen in any order.
2. The environment possibly interacts with $\mathcal{R}$.
3. For every $P_i \in \mathcal{H} \cup \mathcal{SH}$, $\mathcal{Z}$ sends $(\mathsf{clockout}, P_i)$ to $\mathcal{R}$ and activates $\mathcal{R}$. Then $\mathcal{Z}$ activates P_i. Overall, this allows P_i to receive messages from the communication resource. The parties can be chosen in any order.

Synchronous communication per se does not depend only on how $\mathcal{Z}$ schedules activations but also on how the communication resources manage messages. A crucial communication resource we will consider in this work is given by $\mathcal{F}_{\mathsf{StableNet}}$, which is intended to model a synchronous stable network (in contrast to an unstable network, discussed in Section D.2 below) in which parties send messages to each other in each round, and all of these messages are received within the same round. The purpose of having this communication resource is two-fold. First, it serves as a basis for our communication resource $\mathcal{F}_{\mathsf{UnstableNet}}$ modelling an unstable network with dropouts and comebacks. Second, it becomes the functionality that

[14] In both composed systems, at the end of their run, $\mathcal{Z}$ outputs a bit, which is what defines the respective random variables.

we wish to instantiate in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model in order to obtain MPC over an unstable network. We return to this discussion in Section D.3.

The functionality $\mathcal{F}_{\mathsf{StableNet}}$ is described below. It is inspired by the functionality F_{SC} from [9, Section 4.4].

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

Let $\mathcal{C}$ and $\mathcal{SH}$ be the set of actively corrupt and semi-honest parties, respectively. Upon activation, proceed as follows.

- On input $(\mathsf{clockin}, P_i)$, check for input from P_i and, if there is one, parse it as $(m_{i1}, m_{i2}, \ldots, m_{in})$. Store $(P_i, m_{i1}, \ldots, m_{in})$, and send $\{m_{ij}\}_{P_j \in \mathcal{C} \cup \mathcal{SH}}$ to $\mathcal{Z}$.
- On input $(\mathsf{change}, P_i, \{m_{ij}\}_{j=1}^n)$, where $P_i \in \mathcal{C}$, store $(P_i, m_{i1}, \ldots, m_{in})$, deleting any previous record of the same form. This will be important for the simulation.
- Upon receiving a message $(\mathsf{clockout}, P_i)$ from $\mathcal{Z}$, send $\{(P_j, m_{ji})\}_{j=1}^n$ to P_i, where $m_{ji} = \perp$ if there is not a recorded message of the form $(P_j, m_{j1}, \ldots, m_{jn})$.

The functionality $\mathcal{F}_{\mathsf{StableNet}}$, together with the restrictions of a synchronous environment described above, constitute what a synchronous protocol looks like: in every round, in the clockin phase each honest and semi-honest party gets the chance to send a message to $\mathcal{F}_{\mathsf{StableNet}}$, which is activated in order to process these messages. Then, in the clockout phase the functionality sends the messages back to the parties, which are activated in order to be able to retrieve them.

For simplicity in our protocols, we will not attempt to instantiate $\mathcal{F}_{\mathsf{StableNet}}$ directly, but rather, we will instantiate a set of functionalities $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^n$, where each $\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}$ is defined as $\mathcal{F}_{\mathsf{StableNet}}$, except that only P_i is clocked-in, only P_j is clocked-out, and the message that P_i sends has the form (m_{ij}) (rather than $(m_{i1}, \ldots, m_{in})$). Similarly, the message that $\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}$ sends to P_j is only comprised by the message that P_i sent, if there is any. This functionality models a channel from P_i to P_j only. It is obvious that $\mathcal{F}_{\mathsf{StableNet}}$ can be securely instantiated in the $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^n$-hybrid model.

D.2 Unstable Networks

In an unstable network with dropouts and comebacks, the guarantees from a stable network only need to hold for a subset $\mathcal{O}_r$ of parties specified by $\mathcal{Z}$ for the given round r. For parties outside this set, $\mathcal{Z}$ gets to choose who is allowed to send and receive messages.

This is captured by the following functionality.

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

Let $\mathcal{C}$ and $\mathcal{SH}$ be the set of actively corrupt and semi-honest parties, respectively. Upon activation, proceed as follows.

- On input $(\mathsf{clockin}, P_i)$, check for input from P_i and, if there is one, parse it as $(m_{i1}, m_{i2}, \dots, m_{in})$. Store $(P_i, m_{i1}, \dots, m_{in})$, and send $\{m_{ij}\}_{P_j \in \mathcal{C} \cup \mathcal{SH}}$ to $\mathcal{Z}$.
- On input $(\mathsf{change}, P_i, \{m_{ij}\}_{j=1}^{n})$, where $P_i \in \mathcal{C}$, store $(P_i, m_{i1}, \dots, m_{in})$, deleting any previous record of the same form.
- On input $(\mathsf{erase}, P_i, P_j)$, look for a record of the form $(P_i, m_{i1}, \dots, m_{in})$, and if there is one, replace m_{ij} with $\perp$. This allows $\mathcal{Z}$ to specify messages to be dropped.
- Upon receiving a message $(\mathsf{clockout}, P_i)$ from $\mathcal{Z}$, send $\{(P_j, m_{ji})\}_{j=1}^{n}$ to P_i, where $m_{ji} = \perp$ if there is not a recorded message of the form $(P_j, m_{j1}, \dots, m_{jn})$.

The environment $\mathcal{Z}$, on top of following the rules for synchrony described before, follows this rule: at every round, and after clocking-in the honest and semi-honest parties so that they send messages to $\mathcal{F}_{\mathsf{UnstableNet}}$, $\mathcal{Z}$ internally chooses a set $\mathcal{O}_r \subseteq \mathcal{P}$. We require then that, for every $P_i, P_j \in \mathcal{O}_r \cap \mathcal{H} \cap \mathcal{SH}$, $\mathcal{Z}$ does *not* send $(\mathsf{erase}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{UnstableNet}}$. Furthermore, for simplicity, we assume that $\mathcal{Z}$ sends $(\mathsf{schedule}, \mathcal{O}_r)$ to $\mathcal{F}_{\mathsf{UnstableNet}}$ after clocking-in the honest and semi-honest parties. Intuitively, $\mathcal{O}_r$ is the set of online parties in the given round r, which means that all the messages they send are guaranteed to be received by parties in this set. However, this only holds for honest and semi-honest parties, since actively corrupt parties may simply refrain from sending or receiving messages completely.

Notice that we do not place any restriction on $\mathcal{Z}$ besides ensuring a stable network among the parties in $\mathcal{O}_r$. For example, $\mathcal{Z}$ may let *some* of the parties outside $\mathcal{O}_r$ send some of their messages, and some others may receive only part of their intended messages, by making use of the erase command. Also, observe that this command completely deletes the stated message from $\mathcal{F}_{\mathsf{UnstableNet}}$'s state, which models the fact that a party that rejoins the computation at a later point does not get messages sent to it in previous rounds. If we wanted to model, say, the network in [16] in which parties who return to the computation get previous missed messages, we could modify $\mathcal{F}_{\mathsf{UnstableNet}}$ so that erased messages do not get completely deleted, but rather delayed.

D.3 MPC in the $\mathcal{F}_{\mathsf{StableNet}}$-Hybrid Model

Basically, $\mathcal{F}_{\mathsf{StableNet}}$ allows an honest sender to transmit a message to another party while ensuring confidentiality from the adversary, as well as guaranteeing that the message will be received at the other end, and furthermore without any alteration. Thus, this fuctionality effectively emulates a stable network with private and authenticated channels.

Fortunately, the study of MPC over such type of networks is very extensive. For instance, the following results can be obtained from existing works.

Theorem 6.
- *(e.g. [6]) Assume that $t < n$. Then there exists a computationally secure protocol with abort in the $\mathcal{F}_{\mathsf{StableNet}}$-hybrid model.*
- *(e.g. [15,5]) Assume that $t < n/2$. Then there exists a statistically secure protocol with guaranteed output delivery in the $\mathcal{F}_{\mathsf{StableNet}}$-hybrid model.*[15]
- *(e.g. [3]) Assume that $t < n/3$. Then there exists a perfectly secure protocol with guaranteed output delivery in the $\mathcal{F}_{\mathsf{StableNet}}$-hybrid model.*

As a consequence of this, and due to the composability of our model, we see that it suffices to develop protocols to instantiate the $\mathcal{F}_{\mathsf{StableNet}}$ functionality.

Intersections of online parties from round to round. Previous works, like [16], characterize the feasibility of MPC in dynamic settings dependending on the fraction of the online and honest parties on each round with respect to the total number of parties. For example, in [16] it is shown that the set of honest and online parties has to be at least $\frac{1}{2}n + 1$ in order for MPC to be possible in a dynamic setting with computational security.[16]

In this work we take a different approach and characterize the feasibility of MPC in an unstable network with dropouts and comebacks by measuring not the amount of online and honest parties in each round, but rather the amount of honest parties that are online *from one round to the next one*, that is, the size of the set $\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}$ (or $\mathcal{O}_r \cap \mathcal{O}_{r+1}$ for passive security). This is more flexible than a characterization in terms of the relative number of online and honest parties with respect to n, since in particular it could be the case that n is large and not so many parties are online in each round, as long as there are enough parties that are online from each round to the next. This also reflects the intuition that, in order to get MPC, we need to get enough "quorum" that transmits the states from one round to the next one, and this quorum is precisely the set of parties that were allowed to receive messages in one round and also send messages in the next one.

***B*-termination assumption.** Now we introduce an assumption that we will need throughout this work, which restricts the scheduling the adversary can make with the goal of guaranteeing that honest parties receive messages. Intuitively, no protocol can instantiate $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ if we allow the adversary to set parties offline forever, since the functionality requires honest parties to receive messages sent to them by other honest parties. Given this, we assume, in words, that every party gets the chance to be online "with certain regularity". This is quantified

[15] These protocols require *broadcast*. We elaborate on this in Section D.4.

[16] Recall, however, that [16] assumes that the parties who return to the computation get the messages sent to them while being offline. See Section A for details.

by requiring that every party should be online at least “once every B rounds”, which is captured in the following definition.

Definition 4 (Definition 1, re-stated). *Let B be a positive integer. We say that an adversary respects the B-assumption if, for every party P_i and for every non-negative multiple of B, $r \cdot B$, there exists $1 \leq k \leq B$ such that $P_i \in \mathcal{O}_{r \cdot B+k}$.*

Consider a sender P_S who wishes to send a message to a receiver P_R. If it is the adversary’s goal to delay this delivery as much as possible, while still respecting the B-assumption, then a possible scheduling could consist of the following: among the rounds $r = 1, \dots, B$, only set P_S online in round B, and P_R in round 1; among the rounds $r = B+1, \dots, 2B$, only set P_R online in round $2B$. With this scheduling, we see that P_R cannot get the message until round $2B$, because it was only online in two rounds, 1 and $2B$, but it cannot receive the message on round 1 since up to that point P_S has not been online in order to send the message. Our protocols from Sections E, 3 and 4 guarantee that each message is delivered within $2B$ rounds, which is optimal according to the reasoning above.

D.4 Broadcast in the Statistical Setting

When designing secure MPC protocols in the statistical setting with $t < n/2$, it is well known that a broadcast channel is required (for computing general functions), and furthermore, it cannot be instantiated from point-to-point channels alone. It is possible to define a reasonable notion of broadcast over an unstable network: parties that receive a broadcast message in a given round are guaranteed to receive the exact same message, which in the case the sender is honest, corresponds to the message this party intended to send. Unfortunately, this notion is insufficient to instantiate an actual “stable” broadcast functionality in which *all* honest parties (and not only these that are online at a given round) must agree on some value. This is because a corrupt sender may behave honestly during almost all rounds and then, before the last round, it changes its input towards the parties that are online in that given round. This way, all the other parties output the old value, while the parties online in the last round output the new, different value.

The above analysis is not intended to show an impossibility, but rather, to illustrate how highly non-trivial is the problem of instantiating “stable” broadcast over an unstable network, a problem which we believe is orthogonal to our results. To further support this claim, we notice that the work of [16] is devoted almost in its entirety to solving the problem of broadcast and agreement in a networking model that, as discussed in Section A in the introduction, is in a way stronger than ours.

In practice, even over a stable network, a statistically secure protocol for secure computation with $t < n/2$ must assume the existence of a broadcast channel. This can be instantiated, for example, using a bulletin board, and in such case this type of instantiations would also work, from a more pragmatic perspective, over an unstable network.

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

In this section we present protocols for instantiating the $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ functionality in the computational setting, with both passive and active security. We remark that, even though the protocols here are quite simple in nature, we provide full-fledged security proofs that make use of our detailed unstable network model from Section D.

E.1 Passive Security

Our protocol requires the existence of a PKI, which we model as a functionality $\mathcal{F}_{\mathsf{PKI}}^{P_S, P_R}$ that samples two secret/public key pairs $(\mathsf{sk}_R, \mathsf{pk}_R)$ and $(\mathsf{sk}_S, \mathsf{pk}_S)$ and sends $(\mathsf{sk}_R, \mathsf{pk}_R, \mathsf{pk}_S)$ to P_R, and $(\mathsf{sk}_S, \mathsf{pk}_S, \mathsf{pk}_R)$ to S.[17] This functionality is executed before the protocol starts. Observe that, since the environment $\mathcal{Z}$ follows the rules for synchronized computation from Section D.1, it in particular activates all the parties in every round, which means that the PKI is effectively distributed, regardless of dropouts and comebacks.

We begin by presenting an instantiation of $\mathcal{F}_{\mathsf{StableNet}}$ in the passively secure setting. In this case we assume that, for every round r, $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq 1$. The reason why this is necessary is rather simple: if the intersection $\mathcal{O}_r \cap \mathcal{O}_{r+1}$ is allowed to be empty, then the adversary could choose two disjoint sets $A_1, A_2 \subseteq \mathcal{P}$ and set $\mathcal{O}_{2k} = A_1$ and $\mathcal{O}_{2k+1} = A_2$ for every $k > 0$, which means that the parties in A_1 only talk among themselves, and same for the parties in A_2. In particular, a sender $P_S \in A_1$ could not deliver a message to a receiver $P_R \in A_2$.

The construction is also quite simple: essentially the sender sends its message on encrypted form to all other players, who then echo it to all others until we know that the receiver has had a chance to see it.

[17] This means that to instantiate $\mathcal{F}_{\mathsf{StableNet}}$ in the $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^{n}$-hybrid model, the parties need to call $\mathcal{F}_{\mathsf{PKI}}^{P_S, P_R}$ for every possible sender/receiver pair (P_S, P_R), which in turn implies that each party P_i gets a different public key for each other party. This can be avoided by having one single "global" functionality $\mathcal{F}_{\mathsf{PKI}}$ that assigns a single secret/public key pair to each party, and calling this inside each protocol execution $\Pi_{\mathsf{StableNet}}^{\mathsf{comp,passive}}(P_S, P_R)$ when instantiating $\mathcal{F}_{\mathsf{StableNet}}$.

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

Setup: The parties call $\mathcal{F}^{P_S,P_R}_{\mathsf{PKI}}$, so each P_R gets $(\mathsf{sk}_R, \mathsf{pk}_R, \mathsf{pk}_S)$ and P_S gets $(\mathsf{sk}_S, \mathsf{pk}_S, \mathsf{pk}_R)$.

- On input (m) from $\mathcal{Z}$, P_S does the following: In rounds $1, \ldots, B$, P_S sends $(c, \ldots, c)$ to $\mathcal{F}_{\mathsf{UnstableNet}}$, where $c = \mathsf{enc}_{\mathsf{pk}_R}(m)$.
- Every party P_i initializes a variable $\mathtt{msg}_i = \bot$. In rounds $1, \ldots, 2B$, P_i does the following:
  - If P_i receives a message $\{(P_j, c_j)\}_{j=1}^{n}$ from $\mathcal{F}_{\mathsf{UnstableNet}}$, then P_i sets $\mathtt{msg}_i$ to be equal to c_{j_0}, where j_0 is the smallest index such that $c_{j_0} \neq \bot$.
  - If $\mathtt{msg}_i \neq \bot$, then P_i sends $(\mathtt{msg}_i, \ldots, \mathtt{msg}_i)$ to $\mathcal{F}_{\mathsf{UnstableNet}}$.
- In rounds $B+1, \ldots, 2B$, P_R does the following: If P_R receives a message $\{(P_j, c_j)\}_{j=1}^{n}$ from $\mathcal{F}_{\mathsf{UnstableNet}}$, then P_R outputs $m = \mathsf{dec}_{\mathsf{sk}_R}(c_{j_0})$, where j_0 is the smallest index such that $c_{j_0} \neq \bot$.

Theorem 7. *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq 1$ for every round $r > 0$. Then, protocol $\mathit{\Pi}^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$ instantiates the functionality $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ in the $(\mathcal{F}^{P_S,P_R}_{\mathsf{PKI}}, \mathcal{F}_{\mathsf{UnstableNet}})$-hybrid model with computational security against an adversary passively corrupting $t < n$ parties.*

Proof. We provide a proof with all the "bells and whistles" in order to illustrate how our formal framework for unstable networks is used. We will construct a simulator $\mathcal{S}$ that interacts with the environment $\mathcal{Z}$ and with the ideal functionality $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ in such a way that $\mathcal{Z}$ cannot distinguish in polynomial time between the execution of $\mathit{\Pi}^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$ and the execution of $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ with $\mathcal{S}$.

The simulator $\mathcal{S}$ is defined as follows. First, it emulates the functionality $\mathcal{F}^{P_S \to P_R}_{\mathsf{PKI}}$ so when the parties request the PKI in the zeroth round it samples and distributes the necessary secret/public key pairs. $\mathcal{S}$ also emulates the functionality $\mathcal{F}_{\mathsf{UnstableNet}}$.

In what follows, $\mathcal{S}$ must emulate the execution of $\mathit{\Pi}^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$. To this end $\mathcal{S}$ emulates all the parties internally, and executes a local copy of the protocol among these parties as instructed by $\mathcal{Z}$ (e.g. activating virtual parties as $\mathcal{Z}$ indicates). Furthermore, for the first round r such that $P_S \in \mathcal{O}_r$, $\mathcal{S}$ sends $(\mathsf{clockin}, P_S)$ to $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$, which allows the functionality $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ to receive input from P_S. When emulating P_S in this round, $\mathcal{S}$ sets $c = \mathsf{enc}_{\mathsf{pk}_R}(0)$ if P_R is honest, or it sets $c = \mathsf{enc}_{\mathsf{pk}_R}(m)$ if $P_R \notin \mathcal{H}$, where m is the value $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{StableNet}}$ after clocking-in P_S (recall that $\mathcal{F}_{\mathsf{StableNet}}$ immediately leaks to $\mathcal{Z}$ the values sent to corrupt parties). Finally, for the first round $r \in \{B+1, \ldots, 2B\}$ in which $P_R \in \mathcal{O}_r$, $\mathcal{S}$ sends $(\mathsf{clockout}, P_R)$ to $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$, which allows P_R to get the message from the functionality.

Now we have to argue that $\mathcal{Z}$ cannot distinguish between the ideal and the real execution. We first begin with the following claim.

Claim. There exists a round $r_R \in \{B+1, \ldots, 2B\}$ such that P_R outputs m in round r_R, where m is the input from $\mathcal{Z}$ to P_S.

To prove this claim, we first observe that, due to the B-assumption, there must be a round $1 \leq r_S \leq B$ in which $P_S \in \mathcal{O}_{r_S}$, so P_S gets to send $c = \mathsf{enc}_{\mathsf{pk}_R}(m)$ to all parties in $\mathcal{O}_{r_S}$. Then, for each round r with $r_S \leq r \leq 2B$, the following invariant holds: all parties in $\mathcal{O}_r$ know c (at the end of round r). Indeed, we argue inductively. The invariant clearly holds for round r_S. Since $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq 1$, assuming that the invariant holds for a round r, we see that it also holds for round $r+1$ since there is at least one party in $\mathcal{O}_r \cap \mathcal{O}_{r+1}$, and this party knows c since it is in $\mathcal{O}_r$, and it also disseminates c to all parties in $\mathcal{O}_r$, being part of that set as well. This shows that the invariant is preserved. This, together with the fact that from the B-assumption there is a round r_R such that $B+1 \leq r_R \leq 2B$ in which $P_R \in \mathcal{O}_{r_R}$, shows that P_R gets c.

With this claim at hand, we can show the indistinguishability of the ideal and real worlds via a reduction to the CPA security of the underlying encryption scheme. We construct an adversary $\mathcal{A}$ that uses $\mathcal{Z}$ internally in order to break the CPA-game. $\mathcal{A}$ works as follows. First, it runs $\mathcal{Z}$ and sees what message m is provided as input for P_S. $\mathcal{A}$ sets the two messages for the CPA-game to be 0 and m. Upon receiving a challenge ciphertext $c = \mathsf{enc}_{\mathsf{pk}}(b)$, where $b \in_R \{0, m\}$, $\mathcal{A}$ plays the role of the simulator $\mathcal{S}$ defined above, interacting with $\mathcal{Z}$, except it uses pk as P_R's public key and the challenge c as the message P_S sends. If $\mathcal{Z}$ believes it is in the ideal execution, then $\mathcal{A}$ guesses the plaintext is 0, else it guesses the plaintext is m.

To analyze the advantage of $\mathcal{A}$, first observe the following:

- If $b = 0$, then this looks to $\mathcal{Z}$ exactly as the execution with the simulator in the ideal world.
- If $b = m$, then this looks to $\mathcal{Z}$ exactly as the real execution. This is because $\mathcal{S}$ runs exactly the real protocol, but with a "dummy" c. If $b = m$, however, then $\mathcal{S}$ is given the real c, so the execution corresponds to the real protocol. Furthermore, in the simulated execution P_R gets the message sent by P_S through the functionality $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$, but this also happens in the real execution thanks to the claim above.

Given this, we see that the advantage of $\mathcal{A}$ is equal to the advantage of $\mathcal{Z}$:

$$\begin{aligned}
\mathsf{Adv}(\mathcal{A}) =& |\Pr[\mathcal{A} = m \mid b = m] - \Pr[\mathcal{A} = m \mid b = 0]| \\
=& |\Pr[\mathcal{Z} = \mathsf{real} \mid b = m] - \Pr[\mathcal{Z} = \mathsf{real} \mid b = 0]| \\
=& |\Pr[\mathcal{Z} = \mathsf{real} \mid \mathsf{real}] - \Pr[\mathcal{Z} = \mathsf{real} \mid \mathsf{ideal}]| = \mathsf{Adv}(\mathcal{Z}).
\end{aligned}$$

We conclude then that $\mathcal{Z}$'s advantage is negligible, given that $\mathcal{A}$'s advantage is negligible since the encryption scheme is CPA-secure. □

E.2 Active Security

The protocol above does not work against active adversaries directly since a corrupt party may lie when sending c. This can be fixed using signatures, since this would allow a receiver to discard a message that was not originally signed by the sender. This protocol also requires the PKI functionality $\mathcal{F}_{\mathsf{PKI}}^{P_R,P_S}$.

In this setting we assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for every round r.[18] The intuition why this is like necessary is similar to the one from the passive setting in Section E.1: since the *actively* corrupt parties could simply refrain from sending any message at all, allowing $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| = 0$ would allow the parties to be partitioned into two disjoint sets that do not communicate among each other.

In the description of the protocol below, and for the rest of the protocols in this work, we relax the notation with respect to the usage of the functionality $\mathcal{F}_{\mathsf{UnstableNet}}$. For example, instead of saying that a party P_i inputs $(m_{i1}, \ldots, m_{in})$ to this functionality, we will say that P_i sends m_{ij} to P_j. Several other intuitive relaxations are made.

Protocol $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{comp,active}}(P_R, P_S, m)$

Setup: The parties call $\mathcal{F}_{\mathsf{PKI}}^{P_S,P_R}$, so each P_R gets $(\mathsf{sk}_R, \mathsf{pk}_R, \mathsf{pk}_S)$ and P_S gets $(\mathsf{sk}_S, \mathsf{pk}_S, \mathsf{pk}_R)$.

- On input (m), P_S does the following: In rounds $1, \ldots, B$, P_S sends (c, σ) to all parties, where $c = \mathsf{enc}_{\mathsf{pk}_R}(m)$ and $\sigma = \mathsf{sign}_{\mathsf{sk}_S}(c)$.
- Every party $P_i \neq P_R$ initializes an variable $\mathtt{msg}_i = \perp$. In rounds $1, \ldots, 2B$, P_i does the following:
  - If P_i receives a message (c_j, σ_j) from P_j, and if $\mathsf{verify}_{\mathsf{pk}_S}(c_j, \sigma_j) = 1$, then P_i sets $\mathtt{msg}_i$ to be equal to c_j.
  - If $\mathtt{msg}_i \neq \perp$, then P_i sends $\mathtt{msg}_i$ to all parties.
- In rounds $B+1, \ldots, 2B$, P_R does the following: If P_R receives a message (c_j, σ_j) from a party P_j, and if $\mathsf{verify}_{\mathsf{pk}_S}(c_j, \sigma_j) = 1$, then P_R outputs $m = \mathsf{dec}_{\mathsf{sk}_R}(c_j)$.

Theorem 8. *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for every $r > 0$. Then, protocol $\mathit{\Pi}_{\mathsf{StableNet}}^{\mathsf{comp,active}}(P_R, P_S)$ instantiates the functionality $\mathcal{F}_{\mathsf{StableNet}}^{P_R \to P_S}$ in the $(\mathcal{F}_{\mathsf{PKI}}^{P_R,P_S}, \mathcal{F}_{\mathsf{UnstableNet}})$-hybrid model with computational security against an adversary actively corrupting $t < n$ parties.*

Proof. At a high level, the simulator $\mathcal{S}$ in this case is defined in a similar manner as the one from the proof of Theorem 7: $\mathcal{S}$ emulates internal honest parties, and executes the protocol exactly as in the real execution, except that it uses an encryption of 0 for the case in which P_R is honest, and the real m received from

[18] This is in particular implied by the alternative assumption $|\mathcal{O}_k \cap \mathcal{H}| > n/2$ for every $k > 0$, which is used in [16].

$\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ otherwise. However, since this time the environment is corrupting some parties maliciously, certain modifications must be made to the simulation.

We assume for now that P_S is honest, and we discuss the other case towards the end. In this case, $\mathcal{S}$ simply emulates the honest parties as indicated above, interacting with the actively corrupt parties that are controlled by $\mathcal{Z}$. As in the simulation from the proof of Theorem 7, $\mathcal{S}$ instructs $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ to read input from P_S in the round in which P_S comes online for the first time, and it instructs $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ to send output to P_R in the first round in $\{B+1, \dots, 2B\}$ in which P_R comes online, if P_R is honest.

To show indistinguishability between the ideal and real worlds, we rely on the following claim:

Claim. There exists a round $r_R \in \{B+1, \dots, 2B\}$ such that P_R outputs m in round r_R, where m is the input from $\mathcal{Z}$ to P_S.

To see this, observe that, from the B-assumption, there must be a round $1 \leq r_S \leq B$ in which $P_S \in \mathcal{O}_{r_S}$, so P_S gets to send (c, σ) to all parties in $\mathcal{O}_{r_S}$. The invariant we claim here is that, for all rounds $r_S \leq r \leq 2B$, all the parties $P_i \in \mathcal{O}_r \cap \mathcal{H}$ set their internal variable $\mathtt{msg}_i$ to the correct message-signature pair (c, σ). To see that this invariant holds, we argue inductively: First, the invariant clearly holds for round r_S. This is because each party $P_i \in \mathcal{O}_{r_S} \cap \mathcal{H}$ receives the message (c, σ) from P_S, and even if they receive other pairs (c', σ') with $c \neq c'$ and $\sigma \neq \sigma'$ from other parties, these messages are discarded as they will satisfy $\mathsf{verify}_{\mathsf{pk}_S}(c', \sigma') = 0$, since these are not produced by P_S.[19]

Now, recall that $|\mathcal{O}_k \cap \mathcal{O}_{k+1} \cap \mathcal{H}| \geq 1$ for every k. Given this, assuming the invariant holds for a round r, we see that it also holds for round $r+1$ since there is at least one *honest* party P_i in $\mathcal{O}_r \cap \mathcal{O}_{r+1}$. This is because this party P_i knows (c, σ) since by induction hypothesis all parties in $\mathcal{O}_r \cap \mathcal{H}$ know (c, σ), and also, since $P_i \in \mathcal{O}_{r+1}$, P_i is able to send this to all parties in $\mathcal{O}_{r+1}$, which preserves the invariant for round $r+1$. This agains uses the fact that the parties can filter out incorrectly-signed messages.

Finally, let $r_R \in \{B+1, \dots, 2B\}$ be a round in which $P_R \in \mathcal{O}_{r_R}$, which exists due to the B-assumption. Due to the invariant, all the honest parties in $\mathcal{O}_{r_R}$ know (c, σ). Hence, P_R gets this pair in this round and is therefore able to learn m.

With this claim at hand, the rest of the analysis is essentially the same as the one from the proof of Theorem 7. We define an adversary $\mathcal{A}$ for the CPA-game for the encryption scheme that interacts with $\mathcal{Z}$ while playing the role of $\mathcal{S}$, and outputs a guess based on the guess of $\mathcal{Z}$. The key is that we can show that, when using the "right" message in the simulation (the one given by $\mathcal{Z}$ to P_S), the execution looks exactly as the one from the real world, which makes use of the claim above to argue that in the real world P_R receives the message sent by P_S, as in the simulated execution.

[19] Here we are making use of the unforgeability of the signature scheme. This could be made more formal by defining an adversary that breaks the EUF-CMA security of the signature scheme, interacting with the environment and playing the role of the simulator. However, we leave such formal approach out for the sake of simplicity.

Finally, if P_S is corrupt, the simulation proceeds with the following changes. $\mathcal{S}$ emulates the honest parties as before, except that this time it can decrypt the potentially multiple signed ciphertexts that P_S sends. As a result, $\mathcal{S}$, following the protocol, is able to determine what is the message that at the end of the execution P_R is supposed to receive, and uses the change command on the $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ to modify the input from P_S to this new value. □

F Security of the Protocol from Section 5

In this section, we provide a *sketch* of the security properties of protocol Π_{MPC} from Section 5.3. Recall that the function to be computed is assumed to be given by a layered circuit $(x_1^{(L)}, \dots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \dots, x_{\ell_0}^{(0)})$, as defined in Section C. Furthermore, it is assumed that the parties have bivariate shares of the inputs $\langle x_1^{(0)} \rangle, \dots, \langle x_{\ell_0}^{(0)} \rangle$, 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 $\mathbb{F}$.[20] 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 $\langle x_1^{(r)} \rangle^{\mathcal{O}_r}, \dots, \langle x_{\ell_r}^{(r)} \rangle^{\mathcal{O}_r}$. To see this we argue inductively. For $r = 0$ this follows trivially as we assumed that the parties start with shares $\langle x_1^{(0)} \rangle, \dots, \langle x_{\ell_0}^{(0)} \rangle$, which in particular means they have shares $\langle x_1^{(0)} \rangle^{\mathcal{O}_0}, \dots, \langle x_{\ell_0}^{(0)} \rangle^{\mathcal{O}_0}$.

Assume the invariant holds for r, and let us show it also holds for $r + 1$. Let $k \in \{1, \dots, \ell_{r+1}\}$. 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 $d \cdot \langle b \rangle^{\mathcal{O}_{r+1}} + e \cdot \langle a \rangle^{\mathcal{O}_{r+1}} + \langle c \rangle^{\mathcal{O}_{r+1}} + d \cdot e$,

[20] 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.

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 $\langle x_1^{(L)} \rangle^{\mathcal{O}_L}, \dots, \langle x_{\ell_L}^{(L)} \rangle^{\mathcal{O}_L}$. As mentioned in Remark 4 in Section 5.3, these shared outputs can be handled in different ways, depending on the application under consideration.

G Results with Pre-Shared Keys

In this section we sketch how our results change if the parties are allowed to interact with a setup functionality before the beginning of the protocol. For the case of computational (malicious) security, nothing changes as the intersection condition is clearly already minimal: if no honest player survives from one round to the next, nothing can be transmitted.

We then consider perfect security. Assume that P_S and P_R have a random shared key $k \in \mathbb{F}$ only known by the two of them. Then we only need to build a protocol where P_S sends $c = m + k$ (instead of m), which does not require any privacy. For this, it is easy to see that the condition $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t + 1$ for every $r > 0$ is sufficient and necessary: P_S can simply send c in the clear to all parties, and all parties relay this message in every round; however, an honest party only relays a message if it hears the given message either directly from the sender, or from at least $t + 1$ parties. The latter condition ensures that you only relay something you heard from at least one honest party. On the other hand, in each round every honest player will hear at least from the $\geq t+1$ honest survivors from that previous round.

For the case of statistical security, we can also let the receiver one-time pad encrypt the message to be sent, so we only need a protocol that transmits a public message m reliably. Recall that with a shared key $K = (a, b)$, a value x can be authenticated by sending along an "unconditional MAC" $m_K(x) = ax + b$ (computed in a finite field). The receiver recomputes the MAC and compares to what she received. An adversary can make the receiver accept a different message only by guessing a, which happens with negligible probability if a is chosen from a sufficiently large field.

Now, let $M(x)$ stand for the following operation: for each party P_i, P_S takes a fresh MAC-key K she shares with P_i and appends $m_K(x)$ to x. Thus, $M(x)$ consists of x followed by n MACs. Now, to send m to P_R, P_S will compute and send $M(M(\cdots M(m)\cdots)) = M^{2B}(m)$. Suppose that, in some round, P_i receives a message that can be parsed as $M^j(m)$. Note that this means $M^j(m)$ consists of $M^{j-1}(m)$ followed by n MACs, one of which is intended for P_i. If this MAC verifies, she will send $M^{j-1}(m)$ to all parties in the next round.

This protocol works if $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 1$ for all r: in the round where P_S is online all honest players will get a message that they can verify, so at least one of them will relay a correct message in the next round, where one layer of MACs has been "peeled off". This continues until P_R comes online, which happens no later than $2B$ rounds after P_S started the exchange, so the parties will not "run out" of MACs. The only way in which P_R can receive an incorrect message is if a MAC was forged, which happens with negligible probability.

As for the communication complexity, note that P_S will need to attempt to start the protocol in each of the first B rounds (she does not know in which of them she is online). For each instance, $O(Bn^2)$ messages may be sent, so we have $O(B^2n^2)$ messages. Each message has size equal to the original message size plus $O(Bn)$ macs. Note here that even if the simple example mac we mentioned has mac size that depends on the message size, it is well known that we can have macs whose size depend only on the security parameter. Total communication is therefore $O(B^2n^2(\ell + Bn\kappa))$ where ℓ is the message length and κ is the security parameter.