# 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

**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. Unlike previous works in the literature, we allow parties to return to the computation according to an adversarially chosen schedule and, moreover, we do not assume that these parties receive the messages that were sent to them while being offline. However, we do 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.

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 that avoids this overhead by following a more "direct" approach rather than building a stable network on top. We introduce our network model in the UC-framework and prove the security of our protocols within this setting.

## 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.

⋆ Work partially done while Daniel Escudero was at Aarhus University

For example, the protocol construction may require the parties to send certain messages before the first 10 seconds mark, such that all parties can wait for this time period to receive all the messages before proceeding to the next "round" of the protocol. More generally, in a synchronous network the computation 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 they have the following limitations. First, the protocol always takes time equal to the number of rounds times the stipulated duration of each round. Furthermore, 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.

Fixing the above synchronous network problems by having each round last the longest network time possible is not a realistic option, as protocols would become too slow for practical applications. A different solution is to consider an *asynchronous network*. In this model, the parties are not assumed to have a clock anymore, so the protocol cannot make use of "time" in its design. Instead, protocol rules are written in terms of conditions on messages received. For example, it may instruct the parties to send a certain message, after a certain number of previous messages has been received. 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. Unfortunately, one important drawback of an asynchronous network is that, 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. This typically implies that asynchronous protocols 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 to consider 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 optimal corruption thresholds, allow all parties to contribute input, and guarantee termination at a certain time. A series of works has studied MPC in different variant of this model, see Section 1.3 for a detailed comparison of prior works. However, it is still an open ques-

tion 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, and parties are not assumed to receive messages sent while they were offline.

### 1.1 Our Contribution

In this work we given a positive answer to the above question. Before getting to the answer, we describe the model in a bit more detail: A *stable network* is a standard synchronous network. On the other hand, an *unstable network* is a synchronous network where we allow the adversary to choose in each round a subset of parties that will be offline in that specific round, and will therefore not be able to send or receive messages[3]. This models honest parties dropping out in that specific round. Note that the adversary has the power to decide which parties drop out at which point in time, which serves to model certain failures like weak mobile connections. Note also that the set the adversary chooses may be different for every round, which models the fact that parties are allowed to come back and rejoin the computation.

Finally, the model has, as a parameter, a bound $B$, where we require that an honest party is never offline for more than $B$ rounds, i.e. an honest party is 'immortal' like the mythological phoenix. If no such bound is assumed, and we are only promised that a party eventually rejoins, protocols would not be able to guarantee termination at a certain time, or that all parties get to contribute input. These are natural properties that virtually all synchronous protocols satisfy, so we set as a goal to also achieve them on an unstable network.

A crucial aspect of our model is that, when a party rejoins the computation after being offline for some rounds, it does not receive the messages the adversary blocked during this period, and moreover, an honest party does not know whether she was set to be offline in a particular round. This implies that an honest party does not know if she failed to receive a certain message because the sender was offline or corrupt, or because she herself was offline. This is a natural way to model failures due to a cable being cut and later repaired, for instance, but it also imposes a challenge when designing protocols since it is not possible for the parties to selectively send messages depending on whether they have been offline or not, for example.

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 $\pi$, $\pi'$ are *equivalent* if they tolerate the same number of corruptions, achieve the same type of security (computational/statistical/perfect)

[3] In fact we allow the adversary to let some messages make it, which only makes the model more general.

and the same security guarantee (security with abort/fairness/guaranteed output delivery).

Let $n$ be the number of parties and let $t$ be the number of (actively) corrupt parties. Let $\mathcal{O}_r$ denote the set of online parties in round $r$, and let $\mathcal{H}$ denote the set of honest parties. Our first set of results is as follows:

**Perfect security.** (Section 4) 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 5) 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 B 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). The intersection condition is required for any computationally secure MPC protocol to exist over an unstable network.

Note that even if the (minimal) assumptions in our results say that at least some parties 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 harder. Had we assumed that an honest party always stays online for more than one round, we believe that much simpler and efficient protocols could be designed. In this work, however, our goal is to get feasibility results for the most adversarial version of the model, so we leave the study of relaxed models for future work.

It is also important to note that our results imply a necessary tradeoff between instability and corruptions: taking perfect security as an example, it is well known that we must have $n \geq 3t + 1$ to have perfect security at all. So for a maximal value of $t$, we have only $2t + 1$ honest parties, and the result above then says that all honest parties must stay online all the time. On the other hand, as we increase $n$ above $3t + 1$, an increasing number of honest players can be sent offline.

The results above are obtained via a generic and modular approach in which we emulate a stable synchronous network using the unstable network. Since we do this in the UC framework, we can compose with any synchronous MPC protocol, and get a protocol for the unstable network, which inherits all the properties of the underlying MPC protocol. For instance, malicious security can be achieved "automatically" by starting from a maliciously secure MPC protocol designed for a stable network.

However, this modularity and generality 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) We therefore also consider the construction of more efficient MPC protocols directly on top of an unstable network, and we present a construction in the perfect security setting. The idea is to start from the standard idea of preparing multiplication triples in a preprocessing phase. Since this can be done in constant-round, we can do this in $O(B)$ rounds on the unstable network using our general approach. This leads to a protocol taking the same number of rounds as for a stable network, except for an additive $O(B)$ term. Note that such a term is unavoidable: protocols on an unstable network where even one party get to contribute input must take $\Omega(B)$ rounds. Namely, we must wait at least until that party has had a chance to send at least one message. This also shows that the case of computational security is less problematic: in this setting, we can get a $O(B)$-round protocol already from the generic compilation approach by starting from the constant-round protocols in the computational setting.

Finally, we consider a different type of improvement of our general results: if we assume that each pair of players share a sufficient amount of secret key material, the lower bounds on the number of parties that must survive from one round to the next no longer hold for the case of statistical and perfect security. 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.

In the technical overview, we give more details on our constructions and their complexities. As an additional contribution, necessary to achieve the results above, we formalize in Section A the concept of a unstable network with dropouts and comebacks, which enables the adversary to set parties offline and online arbitrarily, as an extension of the universal composability (UC) framework [7]. Our results are proven secure in this framework, and therefore they enjoy the same composability features that protocols set in the traditional UC framework have.

### 1.2 Technical Overview

**Modular Constructions.** We start by briefly discussing the main results of our generic approach in which we emulate a stable synchronous network using the unstable network. The main complication when designing protocols in an unstable network is that parties may not stay online for enough rounds as to "contribute" to the computation. For example, consider the standard approach to secure multiplication based on Beaver triples $([a], [b], [c])$, where $a, b$ are random elements in $\mathbb{F}$ and $c = a \cdot b$. To multiply two sharings $[x], [y]$ the parties open

two secret-shared values $d \leftarrow [x] - [a]$ and $e \leftarrow [y] - [b]$, and later take the linear combination $[x \cdot y] = e[a] + d[b] + [ab] + de$. Only the parties that are online in the current round can learn the reconstructed values, so only these parties can define their shares of $x \cdot y$. However, even if there are enough of these parties so that the sharings of $x \cdot y$ are "well-defined", it could happen that in the next round many of these parties are offline. As a result, not enough parties know their shares of $x \cdot y$ to be able to continue with the computation.

To solve this issue, our approach consists of designing multi-round protocols that ensure that all the parties can learn the necessary data for the protocols to make progress. In order to achieve this, we abstract away the problem of designing MPC protocols, and focus only on the task of guaranteeing message delivery between honest parties, which is the main obstacle when working on an unstable network.

To this end, 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).

To achieve the general "feasibility" results presented in Section 1.1, 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.

The instantiation of $\mathcal{F}_{\mathsf{StableNet}}$ in the computational security setting is presented in Section B in the Supplementary Material, and it makes use of signature and encryption schemes to transfer messages between honest parties. Since we can use encryption, it is relatively easy to make sure that a secret message eventually makes it to the receiver, by asking parties to echo the encrypted messages that are in transit. Of course, this does not work for perfect security, and hence the instantiation for this case (discussed in Section 4) is more complicated. It makes a novel use of (a variant of) the method for secret-sharing using bivariate polynomials that has been used for verifiable secret sharing in the literature before [4]. At a high level, a sender secret-shares the message she wants to send using a symmetric bivariate polynomial $f(x, y)$, giving each party the "share" $f(x, i)$. Since the receiver may not be online in the exact same round, the parties transfer these sharings to the parties who are online in the next round by letting each $P_i$ send to each $P_j$ the value $f(j, i)$. Once each $P_j$ has received enough values $\{f(j, i)\}_i$, he can perform error correction to recover the polynomial $f(j, x)$, which by symmetry equals $f(x, j)$. Once the receiver comes online, she will receive the secret $f(0, 0)$.

The computational and perfect implementations of $\mathcal{F}_{\mathsf{StableNet}}$ both make a number of calls to the unstable network that is polynomial in the number of

players and the bound $B$. Also, they reliably transfer a message in at most $2B$ rounds which is optimal. See Section 4.2 and 4.1 for the passive and active constructions, respectively.

The instantiation with statistical security, which is much more involved, is presented in Section 5. At a high level, one of the reasons 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 techniques such as error-correction, that guarantee continuation, but instead only ensure error-detection. To overcome this issue, we first design a method, rooted in robust secret-sharing, in which a pair of parties can communicate, assuming that the rounds in which they are online are not too far from each other. Then, we devise a novel and non-trivial recursive construction that leverages the first method to communicate between parties that become online at potentially very different times. 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 more efficient protocol (in the statistical setting) by generated pre-shared secret key material, see Section E.

Last but not least, we show that the intersection requirements of online-and-honest parties from one round to the next to be at least $2t + 1$, $t + 1$ and 1 for perfect statisitcal and computational security, respectively, to be optimal.

**Direct Constructions.** While our approach of emulating a stable network using an unstable one is a clean and modular approach to MPC, there are also drawbacks. We pay an overhead in practice since each message sent in the emulated stable network takes several "real" rounds in the underlying unstable network. To overcome this issue, we present an MPC protocol in the perfectly secure setting that builds directly on top of an unstable network. We start from the standard idea of preparing multiplication triples in a preprocessing phase. Since this can be done in constant-round, we can do this in $O(B)$ rounds on the unstable network using our general construction. 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 is essentially the same number of rounds as 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 6.

**Constructions with Pre-shared Keys.** The reason why results are different with preshared keys is that a sender can one-time pad encrypt his message and then the only remaining problem is the simpler one of getting the encrypted message unchanged to the receiver by relaying the message in every round.

Note that these results allow for an alternative MPC construction that can be used even in the case where no shared keys are given: the parties first use the (statistical or perfect) protocol mentioned before to send keys secretly between each pair of players. Then, to run the actual MPC protocol, they use the stable network emulation using shared keys, that we just mentioned. 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 have to be satisfied during the first part of the protocol where the keys are set up, see Section E.

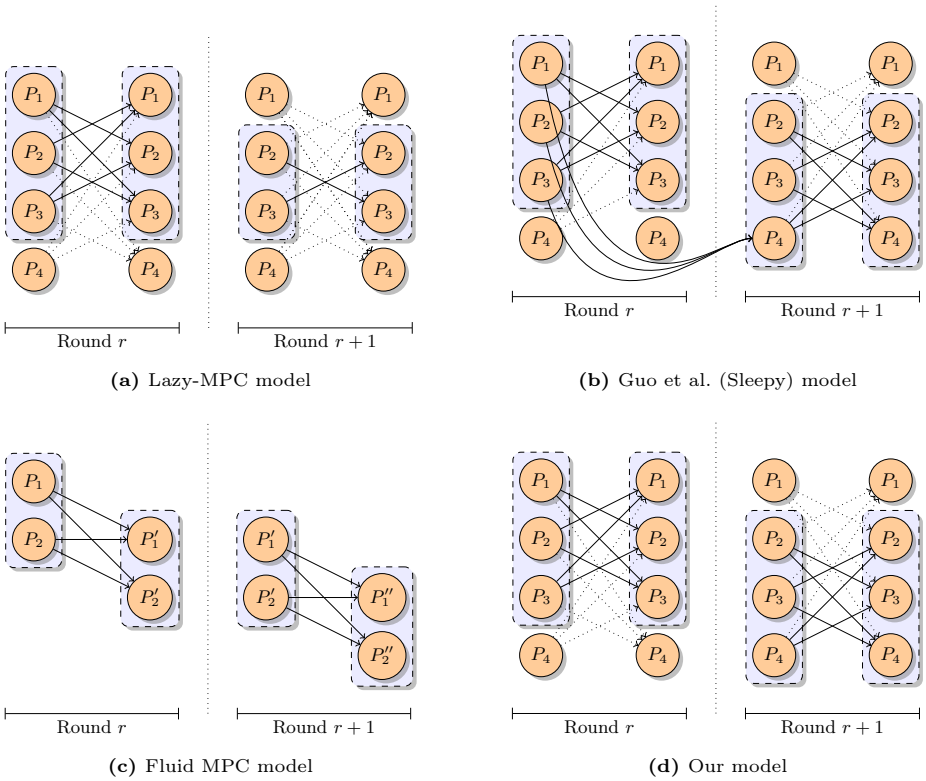
### 1.3 Related Work

In what follows we discuss some of the works that study a similar problem to the one we address in this work. Also, we present in Figure 1 a more graphical comparison of our model with respect to the works of [1]

*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.




**(a)** Lazy-MPC model

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

**(c)** Fluid MPC model

**(d)** Our model

**Fig. 1** – 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. 1a, the parties cannot return. In the model of Guo et al., Fig. 1b, 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. 1c, 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. 1d, the parties can return to the computation and it is not assumed that they receive the messages sent to them while they were offline.

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 of 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 5, 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 *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. 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. In [14] a statistically secure but very inefficient protocol is given, and no perfectly secure protocol is known in the YOSO model. An additional caveat with the yoso model is that one can only have information theoretically 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.

## 2 Preliminaries

In this section we introduce the necessary preliminaries that we require to present our results. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of all parties, and $\mathcal{H}$ be the set of honest parties. We assume that the adversary corrupts $t$ out of the $n$ parties. Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| > n$.

### 2.1 Shamir Secret-Sharing

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

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

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

- If at least $t+1$ are guaranteed to be correct, *error-detection* can be performed by checking if these shares all lies in a polynomial of degree at most $t$, and if this is the case, the reconstructed polynomial is guaranteed to be correct since it is determined by the $t+1$ correct shares.
- If at least $2t+1$ are guaranteed to be correct, *error-correction* is possible by looping through all possible subsets of these shares of size $2t+1$ and checking if all shares in the given subset are consistent with a polynomial of degree at

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

### 2.2 Crytographic Tools

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]).

### 2.3 Layered Circuits

As described before, we present in Section 6 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 $0, \ldots, L$. Wires in layer 0 are input wires, and these in layer $L$ are output wires. We denote by $\ell_i$ the number of wires in layer $i$, and we denote by $x_1^{(i)}, \ldots, x_{\ell_i}^{(i)}$ 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 $2w$.

## 3 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]. A much more detailed description of our networking model, within the setting of the UC framework, is presented in Section A in the Supplementary Material.

Our starting point is a synchronous network, where an upper bound $\Delta$ 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 $\Delta$ time and consisting of all parties sending messages to each other at the beginning of each round. Since each round $r$ takes $\Delta$ 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 A.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 $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^{n}$ 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 $\{\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}\}_{i,j=1}^{n}$-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.

### 3.1 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:

- 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.

More details are given in Section A.2 in the Supplementary Material. Observe that if a party $P_i$ is set to be offline in a given round $r$, so $P_i \notin \mathcal{O}_r$, this does not mean that$P_i$ isnot allowed to send or receive any message in that round. In reality, $P_i$ may go offline after receiving and sending part of the messages, which is modeled by the fact that there are not requirements in how the environment

should handle offline parties,[4] and this is precisely one of the main challenges in the protocol design over a unstable network. Also, notice that an honest party does not know if it was set to be offline in a given round. For example, if a party $P_i \in \mathcal{H}$ does not receive a given message from a party $P_j$ in a given round $r$, it might be because (1) $P_i$ is offline in that round (so $P_i \notin \mathcal{O}_r$), (2) $P_j$ is honest but offline, or (3) $P_j$ is actively corrupt. Finally, a crucial factor of our model is that, once a party is set to be online after being offline for some rounds, the messages sent to it during these rounds are lost. This is captured implicitly by the functionality above.

### 3.2 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])[5], and perfectly secure protocols with $t < n/3$ (e.g. [3]). As a result, due to the composability of the UC framework, 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 $i, j = 1, \ldots, n$.

*Intersections of online parties from round to round.* It turns out that the feasibility of instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_i \to P_j}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model depends, essentially, in 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), which measures the amount of honest parties that are online *from one round to the next one*. An overview of the intersection sizes required in each of the settings considered in our work is presented in Fig 2 in Section A.3 in the Supplementary Material.

*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.

[4] In fact, a trivial environment could allow all traffic between offline parties. In general, we can see here that the environment has full control about the order in which messages are delivered and which parties receive which messages, which models a very strong and practical adversarial setting in which an attacker has full control of the network.

[5] 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 A.4 in the Supplementary Material.

**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 B, 4 and 5 guarantee that each message is delivered within $2B$ rounds, which is optimal according to the reasoning above.

We recall that, in Section 6, 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.

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

In this section we take care of instantiating the functionality for a stable network with perfect security. First, in Section 4.1 we discuss the simplest setting of passive security. Then, in Section 4.2 we extend this to active security, while retaining perfect simulation.

### 4.1 Passive Security

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

The reason why the condition $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$ is required when we aim for information-theoretic security is more involved compared to the analogous restriction in the computational setting from Section B in the Supplementary Material. We discuss this in more detail later in this section.

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

- On input $(m)$, $P_S$ samples random elements $c_{ij} \in \mathbb{F}$ for $i, j = 0, \ldots, t$, subject to $c_{0,0} = m$ and $c_{ij} = c_{ji}$, and lets $f(x, y) = \sum_{i,j=0}^{t} c_{ij} x^i y^j$. Then, in rounds $1, \ldots, B$, $P_S$ sends $f(x, i)$ to each party $P_i$.

- Every party $P_i$ initializes a variable $\mathtt{f}_i = \perp$. In rounds $1, \ldots, 2B$, $P_i$ does the following:
  - If $\mathtt{f}_i$ is not set already:
    - If $P_i$ receives a polynomial $f_i(x) = f(x, i)$ from $P_S$, then $P_i$ sets $\mathtt{f}_i = f_i$.
    - Else, if $P_i$ receives messages $m_j \in \mathbb{F}$ from at least $t+1$ parties $P_j$, then $P_i$ sets $\mathtt{f}_i$ to be the polynomial $f_i(x)$ such that $f_i(j) = m_j$ for the first $t+1$ messages $m_j$.
  - If $\mathtt{f}_i \neq \perp$, then $P_i$ sends $\mathtt{f}_i(j)$ to each party $P_j$ and $\mathtt{f}_i(0)$ to $P_R$.

- In rounds $B+1, \ldots, 2B$, $P_R$ does the following: If $P_R$ receives messages $m_j \in \mathbb{F}$ from at least $t+1$ parties $P_j$, then $P_R$ computes the polynomial $f_0(x)$ such that $f_0(j) = m_j$ for the first $t+1$ messages $m_j$, and outputs $m = f_0(0)$.

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

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

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

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

The fact that the adversary does not learn anything if both $P_S$ and $P_R$ are honest follows from the fact that its view is limited to $t$ polynomials of the form $f(x,i)$, which look uniformly random as described in Section 2.1.

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 $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security against a passive adversary, the assumption that the adversary's schedule satisfies $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq t+1$ in every round $r$ is necessary. However, we have to be careful about what this should actually mean: consider an adversary who respects the $B$-assumption and breaks the intersection condition in one, or some finite number of rounds. Now, if the sender happens to start our protocol for sending a message after the last bad round, it will clearly succeed. So we cannot hope to show that communication between sender and receiver is impossible, unless we consider an adversary who keeps breaking the intersection condition "for ever". So we construct below an adversary that breaks this condition once every $B$ rounds, and by doing so it is able to learn the message sent by an honest sender using *any* instantiation of $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$.

Assume the existence of an implementation of $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security that tolerates an adversary that schedules the parties as follows:

- 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 $k > 0$.
- 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 $\ell$ 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.

### 4.2 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.

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

- On input $(m)$, $P_S$ samples random elements $c_{ij} \in \mathbb{F}$ for $i, j = 0, \ldots, t$, subject to $c_{0,0} = m$ and $c_{ij} = c_{ji}$, and lets $f(x,y) = \sum_{i,j=0}^{t} c_{ij} x^i y^j$. Then, in rounds $1, \ldots, B$, $P_S$ sends $f(x,i)$ to each party $P_i$.

- Every party $P_i$ initializes a variable $\mathtt{f}_i = \perp$. In rounds $1, \ldots, 2B$, $P_i$ does the following:
  - If $\mathtt{f}_i$ is not set already:
    - If $P_i$ receives a polynomial $f_i(x) = f(x,i)$ from $P_S$, then $P_i$ sets $\mathtt{f}_i = f_i$.
    - Else, $P_i$ collects the messages $m_j \in \mathbb{F}$ for $P_j$ received in that round and, 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 $m_j$. If this succeeds, then $P_i$ sets $\mathtt{f}_i = f_i$.
  - If $\mathtt{f}_i \neq \perp$, then $P_i$ sends $\mathtt{f}_i(j)$ to each party $P_j$ and $\mathtt{f}_i(0)$ to $P_R$.

- In rounds $B+1, \ldots, 2B$, $P_R$ does the following: If $P_R$ receives 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 $m_j$. If this succeeds then $P_R$ outputs $m = f_0(0)$.

**Theorem 2.** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every $r > 0$. Then, protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_R, P_S)$ 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 actively corrupting $t < n/3$ parties.*[6]

The proof is similar to the one in Theorem 1, and we present it in Section C in the Supplementary Material.

**Optimality of $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$.** As in Section 4.1, 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

[6] 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 $\ell$ 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.

## 5 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 4.2 with $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$. As shown in Section 4.2, perfect security cannot be achieved in this setting, so we settle with statistical security.

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

Now, we would like to "bootstrap" the protocol $\Pi_1$ into a protocol $\Pi_2(P_S, P_R, m)$ that guarantees that a receiver $P_R$ gets the message $m$ sent by a sender $P_S$, as long as $P_R$ comes online either in the same round where $P_S$ is, in the next one, *or in the one after that*. To this end, the parties run $\Pi_1(P_S, P_R, m)$, which guarantees that $P_R$ gets $m$ if it comes online in the same round as $P_S$, or at most in the round after. However, to deal with the case in which $P_R$ comes online two rounds after $P_S$, the parties also execute the following *in parallel*: $P_S$ computes $(m_1, \ldots, m_n) = f(m)$ and executes $\Pi_1(P_S, P_R, m_i)$ for $i = 1, \ldots, n$. This ensures that every $P_i \in \mathcal{O}_2$ will get $m_i$, and at this point, the parties in $\mathcal{O}_3 \cap \mathcal{O}_2$ can send these to $P_R$ in the third round. Upon receiving $m'_i$, $P_R$ outputs $m = g(m'_1, \ldots, m'_n)$.

To analyze the protocol $\Pi_2$, assume for simplicity that $P_S \in \mathcal{O}_1$. We first observe that if $P_R \in \mathcal{O}_1 \cup \mathcal{O}_2$, then $P_R$ gets $m$ as $\Pi_1(P_S, P_R, m)$ is being executed. If, on the other hand, $P_R \in \mathcal{O}_3$, $P_R$ gets $m$ as $g(m_1, \ldots, m_n)$ since the parties $P_i \in \mathcal{O}_2$ get $m_i$ from $\Pi_1(P_S, P_R, m_i)$. This idea can be iterated to obtain protocols that deliver messages as long as $P_R$ comes online at most $k$ rounds after $P_S$ comes online.

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

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

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

Several robust secret-sharing constructions can be found in the literature. However, since we consider a non-standard version of robust secret-sharing, we present below a concrete construction that fits Definition 2, which is motivated on the so-called information-checking signatures from [20].

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

$\mathsf{share}(m)$: Compute Shamir shares $m_1,\ldots,m_n$ of $m$. Sample $(\alpha_1,\beta_1),\ldots,(\alpha_n,\beta_n) \in_R \mathbb{F}^2$ and let, for every $i,j \in \{1,\ldots,n\}$, $\tau_{ij} = \alpha_j m_i + \beta_j$. Return $(s_1,\ldots,s_n)$, with $s_i = (m_i, (\alpha_i,\beta_i), \{\tau_{ij}\}_{j=1}^n)$.

---

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

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

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

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 $\mathsf{share}(m) = (s_1, \ldots, s_n)$ with $s_i = (m_i, (\alpha_i, \beta_i), \{\tau_{ij} = \alpha_j m_i + \beta_j\}_{j=1}^n)$. First we argue privacy. It is clear that the $n$ Shamir shares $m_1, \ldots, m_n$ do not leak anything about the secret $m$ towards the adversary. Additionally, the keys $(\alpha_i, \beta_i)$ are simply random values, which do not leak anything either. Finally, each $P_i$ receives $\{\tau_{ij} = \alpha_j m_i + \beta_j\}_{j=1}^n$, but these only involve $m_i$, which is already known by $P_i$.

Now, to see the guaranteed reconstruction property, let $(s'_1, \ldots, s'_n)$ be as in Definition 2. Assume that $|A^c \cap B| > t$, we want to show that $\mathsf{rec}(s'_1, \ldots, s'_n)$ outputs $m$ in this case. Let us write each $s'_i$ for $i \in A \cap B$ as $s'_i = (m'_i, (\alpha'_i, \beta'_i), \{\tau'_{ij}\}_{j=1}^n)$. We claim that if $m'_i = m_i + \delta_i$ with $\delta_i \neq 0$, then $\tau'_{ij} = \alpha_j m'_i + \beta_j$ for at least $j \in A^c \cap B$ can only happen with negligible probability. To see why this holds, let us write $\tau'_{ij} = \tau_{ij} + \epsilon_{ij}$, so $\tau'_{ij} = (\alpha_j m_i + \beta_j) + \epsilon_{ij} = (\alpha_j m'_i + \beta_j) - \alpha_j \delta_i + \epsilon_{ij}$. For this to be equal to $\alpha_j m'_i + \beta_j$, it has to hold that $\alpha_j = \delta_i^{-1} \epsilon_{ij}$. However, $\delta_i$ and $\epsilon_{ij}$ are functions of $\{s_\ell\}_{\ell \in A}$, so they are computed independently of the uniformly random value $\alpha_j$ 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, $\tau'_{ij} \neq \alpha_j m'_i + \beta_j$ for every $j \in A^c \cap B$, so in particular $\tau'_{ij} = \alpha_j m'_i + \beta_j$ can only be satisfied for $j \in A \cap B$, but since $|A \cap B| \leq t$, we see that $m'_i$ would be set to $\perp$ from the definition of $\mathsf{rec}(\cdot)$. As a result, only values with $m'_i = m_i$ remain, and since there are at least $|A^c \cap B| > t$ of these, we see that $\mathsf{rec}(\cdot)$ outputs $m$ correctly in this case.

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

**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 $\Pi$ 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** $\Pi_1(P_S, P_R, m)$

$P_S$ does the following:

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

Every party $P_i$ does the following:

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

$P_R$ does the following in every round:

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

**Proposition 2.** *$\Pi_1(P_R, P_S, m)$ 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 $\Pi_{\mathsf{k}-1}(P_R, P_S, \cdot)$ that achieves $(k-1)$-delivery, one can obtain a protocol that achieves $k$-delivery.

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

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

The parties execute $\Pi_{\mathsf{k}-1}(P_S, P_R, m)$. In parallel, they execute the following.

- Let $(s_1, \ldots, s_n) = \mathsf{share}(m)$. The parties run $n$ protocol instances $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$ for $i = 1, \ldots, n$.
- Each $P_i$, upon outputting $s_i$ from $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$, send $(s_i)$ to $P_R$ in all subsequent rounds.

– $P_R$ initializes variables $\mathtt{s}_1, \ldots, \mathtt{s}_n = \perp$. Then $P_R$ does the following in every round:
  - Upon outputting $s_i$ from some execution $\Pi_{\mathsf{k}-1}(P_S, P_i, s_i)$, $P_R$ sets $\mathtt{s}_i = s_i$.
  - Upon receiving $s'_i$ from some party, sets $\mathtt{s}_i = s'_i$.
  - $P_R$ outputs $\mathsf{rec}(\mathtt{s}_1, \ldots, \mathtt{s}_n)$ if this value is not $\perp$.

**Proposition 3.** *Protocol $\Pi_{\mathsf{k}}(P_S, P_R, m)$ achieves k-delivery.*

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

Given the above, it remains to analyze the case in which $P_R \in \mathcal{O}_{r_S+k}$. From the properties of $\Pi_{\mathsf{k}-1}$, every party $P_i \in \mathcal{O}_{r_S+(k-1)}$ receives $s_i$ from $P_S$ in round $r_S + (k-1)$. In particular, each party $P_i \in \mathcal{O}_{r_S+(k-1)} \cap \mathcal{O}_{r_S+k}$ sends $s_i$ to $P_R$ in round $r_S + k$. An analysis similar to the one in the proof of Proposition 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 $\Pi_{\mathsf{k}}$ 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 3.** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq t+1$ for every $r > 0$. Then, protocol $\Pi_{\mathsf{2B}}(P_R, P_S, \cdot)$ instantiates the functionality $\mathcal{F}_{\mathsf{StableNet}}^{P_R \to P_S}$ in the $\mathcal{F}_{\mathsf{UnstableNet}}$-hybrid model with statistical security against an adversary actively corrupting $t < n/2$ parties.*[7]

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

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

## 6 A More Efficient Protocol with Perfect Security

Recall that in Section 4.2 we presented a protocol to instantiate the functionality $\mathcal{F}_{\mathsf{StableNet}}$, which is intended to represent a traditional stable and secure network among the $n$ parties. This is the typical communication model used in several MPC protocols, and, assuming $t < n/3$, we can find perfectly secure protocols in this model which can be used together with our protocol $\Pi^{\mathsf{perf,active}}_{\mathsf{StableNet}}(P_S, P_R)$ from Section 4.2 to obtain a perfectly secure protocol over an unstable network.

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

Round-count is a very sensitive metric in distributed protocols, especially in high-latency scenarios where every communication trip incurs in a noticeable waiting time. Furthermore, the $\theta(B)$ overhead may not be so noticeable if the higher level protocol has a low round count, but unfortunately, it is a well-known open problem to achieve constant round protocols with *perfect security* for functionalities outside $\mathsf{NC}^1$ 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.[8] 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 $\theta(B)$ 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 [19]. For instance, we could use the 3-round protocol from [1] together with $\Pi^{\mathsf{comp,active}}_{\mathsf{StableNet}}(P_R, P_S)$ from

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

Section B.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.

### 6.1 Bivariate Sharings and Transition of Shares

We describe the input and preprocessing phases of our protocol in Section 6.2, and in Section 6.3 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 2.1, 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 4.2 to instantiate $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ with perfect security against an active adversary. This type of sharings proved useful in Protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf,active}}(P_S, P_R)$ to "transfer" a state between a set of parties to another one, and this is the purpose of this primitive in this section as well. In a bit more detail, during the execution of our protocol it will not hold that all parties have shares of certain given values, but rather only specific subsets corresponding to online parties will do. Since the set of online parties potentially changes from round to round, a crucial primitive our protocol relies on is what we call *transition of shares*, which takes care of transmitting the shared state from one set of parties to another.

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

A protocol for transition of shares is a one-round protocol in which the parties start with $\langle s \rangle^{\mathcal{O}_r}$ in round $r$, and they obtain $\langle s \rangle^{\mathcal{O}_{r+1}}$ in the next round $r + 1$. In

what follows we present a protocol for transition of shares, which is motivated in the perfectly secure protocol for instantiating $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ from Section 4.

**Protocol $\Pi_{\mathsf{transfer}}$**

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

Parties do the following:

1. For each $i = 1, \ldots, n$, if $P_i$ has a share $f(x, i)$ of $\langle s \rangle^{\mathcal{O}_{r+1}}$ (different to $\perp$), then $P_i$ sends $f(j, i)$ to $P_j$ for $j = 1, \ldots, n$.
2. For each $j = 1, \ldots, n$, if $P_j$ receives at least $2t+1$ messages $\{f(j, i)\}_i$, then $P_j$ performs *enhanced* error correction (see Section 2.1) to either recover $f(j, x)$ or output an error $\perp$.

**Theorem 4.** *If executed in round $r$, protocol $\Pi_{\mathsf{transfer}}$ 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 4 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 6.3.

### 6.2 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 2.3. 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 B, these shares can be computed by using *any* generic MPC protocol for these tasks, together with our compiler from Section 4.2. 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 $c = a \cdot b$.

### 6.3 Computation Phase

With the primitives described above, the protocol for computing the given functionality $F$ is relatively straightforward: by making use of the $\Pi_{\mathsf{transfer}}$ and $\Pi_{\mathsf{rec}}$ 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 6.2, *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 6.1, and a sketch of this proof can be found in Section D 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 $\Pi_{\mathsf{MPC}}$**

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

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

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

[a] Here is where Remark 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 $\langle x_1^{(L)}\rangle^{\mathcal{O}_L}, \ldots, \langle x_{\ell_L}^{(L)}\rangle^{\mathcal{O}_L}$ in round $L$, where $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$ is the result of the computation. This output can be dealt with in multiple different ways:

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

## 7 Acknowledgments

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2021 JP Morgan Chase & Co. All rights reserved.

## 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 T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st*

*Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 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. 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.
20. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). pages 73–85, 1989.

# Supplementary Material

## A Unstable Network with Dropouts and Comebacks

Our starting point is a synchronous network, where an upper bound $\Delta$ 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 $\Delta$ time and consisting of all parties sending messages to each other at the beginning of each round. Since each round $r$ takes $\Delta$ 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 $r$.[9] 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.

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

[9] 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 $\mathcal{O}_r \cap \mathcal{H}$.

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.[10]

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

### A.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 A.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.

**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.

**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, orches-

[10] If we require the adversary to let parties who return to the computation stay online for at least two rounds (that is, if $P_i \in \mathcal{O}^c_{r-1} \cap \mathcal{O}_r$ 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.

trating it by *activating*[11] 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.

**Security.** At a high level, a protocol $\Pi$ for a given functionality $\mathcal{F}$ is secure if, for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ that makes the real and ideal executions indistinguishable. Computational security relates to the fact that such indistinguishable is conditioned on a given computational problem to be hard. Statistical security means that the distributions arising from the real and ideal world are negligibly close, and perfect security means that these two distributions are exactly the same.

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 A.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

[11] 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.

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 A.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 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 A.3.

The functionality $\mathcal{F}_{\mathsf{StableNet}}$ is described below. It is inspired by the functionality $\mathsf{F}_{\mathsf{sc}}$ 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.

### A.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 $(m_{i1}, m_{i2}, \ldots, m_{in})$. 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.
- On input $(\mathsf{erase}, P_i, P_j)$, look for a record of the form $(P_i, m_{i1}, \ldots, m_{in})$, 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 $(P_j, m_{j1}, \ldots, m_{jn})$.

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 $\mathsf{erase}$ 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.

### A.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 5.**
- *(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.*[12]
- *(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.[13]

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

[12] These protocols require *broadcast*. We elaborate on this in Section A.4.

[13] Recall, however, that [16] assumes that the parties who return to the computation get the messages sent to them while being offline. See Section 1.3 for details.

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.

| | **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. 2** – 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.

An overview of the intersection sizes required in each of the settings considered in our work is presented in Fig 2.

***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 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 $r = 1, \ldots, B$, 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 B, 4 and 5 guarantee that each message is delivered within $2B$ rounds, which is optimal according to the reasoning above.

### A.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 a 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 a 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 1.3 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 practical perspective, over an unstable network.

## B 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.

### B.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 $S$.[14] This functionality is executed before the protocol starts. Observe that, since the environment $\mathcal{Z}$

[14] 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 $\mathcal{F}_{\mathsf{StableNet}}$.

follows the rules for synchronized computation from Section A.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.

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

**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 $(m)$ 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 6.** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1}| \geq 1$ for every round $r > 0$. Then, protocol $\Pi^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$ instantiates the functionality $\mathcal{F}^{P_S \to P_R}_{\mathsf{StableNet}}$ in the $(\mathcal{F}_{\mathsf{PKI}}^{P_S,P_R}, \mathcal{F}_{\mathsf{UnstableNet}})$-hybrid model with computational security against an adversary passively corrupting $t < n$ parties.*

*Proof.* We provide a proof with all the "bells and whistles", although we will reduce details in upcoming proofs. 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 $\Pi^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$ 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 $\mathcal{F}_{\mathsf{PKI}}^{P_S \to P_R}$ 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 $\Pi^{\mathsf{comp,passive}}_{\mathsf{StableNet}}(P_S, P_R)$. 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 $\mathsf{pk}$ 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. □

### B.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 $r$.[15] The intuition why this is like necessary is similar to the one from the passive setting in Section B.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** $\Pi_{\mathsf{StableNet}}^{\mathsf{comp,active}}(P_R, P_S, m)$

**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 $(c, \sigma)$ 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 $\mathtt{msg}_i = \bot$. In rounds $1, \ldots, 2B$, $P_i$ does the following:
  - If $P_i$ receives a message $(c_j, \sigma_j)$ 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 $\mathtt{msg}_i \neq \bot$, then $P_i$ sends $\mathtt{msg}_i$ to all parties.

[15] 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].

– In rounds $B+1,\ldots,2B$, $P_R$ does the following: If $P_R$ receives a message $(c_j,\sigma_j)$ 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 7.** *Assume that $|\mathcal{O}_r\cap\mathcal{O}_{r+1}\cap\mathcal{H}|\geq 1$ for every $r>0$. Then, protocol $\Pi^{\mathsf{comp,active}}_{\mathsf{StableNet}}(P_R,P_S)$ instantiates the functionality $\mathcal{F}^{P_R\to P_S}_{\mathsf{StableNet}}$ in the $(\mathcal{F}^{P_R,P_S}_{\mathsf{PKI}},\mathcal{F}_{\mathsf{UnstableNet}})$-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 6: $\mathcal{S}$ 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 $\mathcal{F}^{P_S\to P_R}_{\mathsf{StableNet}}$ 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 6, $\mathcal{S}$ instructs $\mathcal{F}^{P_S\to P_R}_{\mathsf{StableNet}}$ to read input from $P_S$ in the round in which $P_S$ comes online for the first time, and it instructs $\mathcal{F}^{P_S\to P_R}_{\mathsf{StableNet}}$ to send output to $P_R$ in the first round in $\{B+1,\ldots,2B\}$ 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 $r_R\in\{B+1,\ldots,2B\}$ 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 $(c,\sigma)$ 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 $(c,\sigma)$. 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 $(c,\sigma)$ from $P_S$, and even if they receive other pairs $(c',\sigma')$ 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 $P_S$.[16]

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 $(c,\sigma)$ since by induction hypothesis all parties in $\mathcal{O}_r\cap\mathcal{H}$ know $(c,\sigma)$, and also,

[16] 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.

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 $r_R \in \{B+1, \ldots, 2B\}$ 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 $(c, \sigma)$. 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 6. 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.

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 $\mathsf{change}$ command on the $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ to modify the input from $P_S$ to this new value. □

## C Proof of Theorem 2

**Theorem 8 (Theorem 2 re-stated).** *Assume that $|\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}| \geq 2t+1$ for every $r > 0$. Then, protocol $\Pi_{\mathsf{StableNet}}^{\mathsf{perf},\mathsf{active}}(P_R, P_S)$ 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.*[17]

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

To see this, let $r_S \in \{1, \ldots, B\}$ be the smallest value such that $P_S \in \mathcal{O}_{r_S}$. We claim the following invariant: at the end of every round $r$ with $r_S \leq r \leq 2B$, each $P_i \in \mathcal{O}_r \cap \mathcal{H}$ has $\mathtt{f}_i \neq \perp$, and these polynomials satisfy that $\mathtt{f}_i(x) = f(x, i)$, where $f(x, y)$ is the polynomial sampled by $P_S$ at the beginning of the protocol. We use induction in order to show that the invariant holds. First, notice that the invariant is true for $r = r_S$ given that parties $P_i \in \mathcal{O}_{r_S} \cap \mathcal{H}$ receive the polynomial directly from $P_S$. For the inductive step assume that the invariant holds for some round $r$, and we show that it holds for round $r+1$. By the hypothesis assumption each party $P_i \in \mathcal{O}_r \cap \mathcal{H}$ has set its variable $\mathtt{f}_i$, and $\mathtt{f}_i(x) = f(x, i)$. In particular, this holds for the parties in $\mathcal{O}_r \cap \mathcal{O}_{r+1} \cap \mathcal{H}$, which means that each party $P_i$ in this set sends $\mathtt{f}_i(j)$ to every other party $P_j$ in round $r+1$, which is received by

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

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 2.1, even if $P_j$ receives more shares, some of them potentially incorrect, $P_j$ can still recover $f(x,j)$ via error correction, as instructed by the protocol. We see then that for $P_j$ $\mathtt{f}_j = f(x,j)$, so the invariant is preserved.

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

As with the case with passive security, the analysis above enables the construction of a simulator $\mathcal{S}$ for the proof in a straightforward manner. As with the proof of Theorem 7, 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 $\mathsf{change}$ command in the $\mathcal{F}_{\mathsf{StableNet}}^{P_S \to P_R}$ 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. □

## D Security of the Protocol from Section 6

In this section, we provide a *sketch* of the security properties of protocol $\Pi_{\mathsf{MPC}}$ from Section 6.3. Recall that the function to be computed is assumed to be given by a layered circuit $(x_1^{(L)}, \ldots, x_{\ell_L}^{(L)}) = F(x_1^{(0)}, \ldots, x_{\ell_0}^{(0)})$, as defined in Section 2.3. Furthermore, it is assumed that the parties have bivariate shares of the inputs $\langle x_1^{(0)} \rangle, \ldots, \langle x_{\ell_0}^{(0)} \rangle$, and also, for every multiplication gate, a triple $(\langle a \rangle, \langle b \rangle, \langle c = a \cdot b \rangle)$ with $a, b$ uniformly random in $\mathbb{F}$.[18] Recall that $\langle s \rangle^{\mathcal{O}_r}$ means that there is a large enough subset $\mathcal{S}_r \subseteq \mathcal{O}_r \cap \mathcal{H}$ such that every party $P_i \in \mathcal{S}_r$ has $f(x,i)$ such that $f(0,0) = s$, and parties in $(\mathcal{O}_r \cap \mathcal{H}) \setminus \mathcal{S}_r$ either have $f(x,i)$ or a special symbol $\perp$.

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

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

[18] 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.

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

Since the invariant holds for every layer, in particular it holds for $r = L$, which shows that, after $L$ rounds, the parties obtain $\langle x_1^{(L)} \rangle^{\mathcal{O}_L}, \ldots, \langle x_{\ell_L}^{(L)} \rangle^{\mathcal{O}_L}$. As mentioned in Remark 4 in Section 6.3, these shared outputs can be handled in different ways, depending on the application under consideration.

## E 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 $\ell$ is the message length and $\kappa$ is the security parameter.