MPC for Tech Giants (GMPC): Enabling Gulliver and the Lilliputians to Cooperate Amicably

Bar Alon* Moni Naor† Eran Omri‡ Uri Stemmer§

May 6, 2026

Abstract

In the current digital world, large organizations (sometimes referred to as tech giants) provide service to extremely large numbers of users. The service provider is often interested in computing various data analyses over the private data of its users, which in turn have their incentives to cooperate, but do not necessarily trust the service provider.

In this work, we introduce the *Gulliver multi-party computation model* (GMPC) to realistically capture the above scenario. The GMPC model considers a single highly powerful party, called the *server* or *Gulliver*, that is connected to n users over a star topology network (alternatively formulated as a full network, where the server can block any message). The users are significantly less powerful than the server, and, in particular, should have both computation and communication complexities that are polylogarithmic in n. Protocols in the GMPC model should be secure against malicious adversaries that may corrupt a subset of the users and/or the server.

Designing protocols in the GMPC model is a delicate task, since users can only hold information about polylog(n) other users (and, in particular, can only communicate with polylog(n) other users). In addition, the server can block any message between any pair of honest parties. Thus, reaching an agreement becomes a challenging task. Nevertheless, we design generic protocols in the GMPC model, assuming that at most $\alpha < 1/8$ fraction of the users may be corrupted (in addition to the server). Our main contribution is a variant of Feige's committee election protocol [FOCS 1999] that is secure in the GMPC model. Given this tool we show:

1. Assuming fully homomorphic encryption (FHE), any computationally efficient function with $O\left(n \cdot \text{polylog}(n)\right)$-size output can be securely computed in the GMPC model.
2. Any function that can be computed by a circuit of $O(\text{polylog}(n))$ depth, $O\left(n \cdot \text{polylog}(n)\right)$ size, and bounded fan-in and fan-out can be securely computed in the GMPC model assuming vector commitment schemes (without assuming FHE).
3. In particular, *sorting* can be securely computed in the GMPC model assuming vector commitment schemes. This has important applications for the *shuffle model of differential privacy*, and resolves an open question of Bell et al. [CCS 2020].

*Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel. Email: alonbar08@gmail.com.

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Incumbent of the Judith Kleeman Professorial Chair. Email: moni.naor@weizmann.ac.il.

‡Department of Computer Science Ariel University, Ariel, Israel. Email: omrier@ariel.ac.il.

§Tel Aviv University and Google Research. Email: u@uri.co.il.

Contents

1 Introduction **1**
- 1.1 The Gulliver MPC (GMPC) Model . . . 1
- 1.2 Our Results . . . 2
- 1.3 Challenges of the Gulliver MPC Model . . . 3
- 1.4 Our Techniques . . . 5
- 1.5 Motivation from Differential Privacy: the Shuffle Model . . . 8
- 1.6 On the (Non-)Necessity of a Common Reference String . . . 9
- 1.7 Related Work . . . 10
- 1.8 Organization . . . 11

2 Preliminaries **11**
- 2.1 Notation . . . 11
- 2.2 Secure Computation and the Model of Computation . . . 12
- 2.3 Cryptographic Tools . . . 16
- 2.4 Probabilistically Checkable Proofs of Proximity . . . 20
- 2.5 Committee Election . . . 22

3 Preventing Blocking of Honest Users **22**
- 3.1 Personal Committee Protocol . . . 23
- 3.2 A Locally Sampled Communication Graph With a Small Diameter . . . 25
- 3.3 Preparing For Computation in the GMPC Model: The Setup Protocol . . . 28

4 Towards Reducing User Complexities: Committee Election **33**
- 4.1 Committee Election Protocol . . . 35
- 4.2 From a Single Committee to Many Committees . . . 38

5 Server's Proof of Correctness of Computation **44**
- 5.1 Verifying the Computation . . . 45
- 5.2 Computing Short Output Functionalities . . . 48

6 Putting It All Together **51**
- 6.1 Secure Implementation of The Next-Message Functions . . . 53
- 6.2 Computing Shallow and Small Circuits . . . 55

7 Applications: Secure Protocols in the Star Topology **57**

Bibliography **59**

A Ideal World For Honest Server **63**

1 Introduction

Consider a large organization (such as Apple or Google; henceforth the *server* or *Gulliver*) that is interested in executing a secure computation over its users' data (we call the users *Lilliputians*). As the server is way more powerful than the average user, we would like it to do most of the heavy-lifting in the computation, thereby allowing the users to do only lightweight computations. In addition, it is reasonable to assume that users can communicate only via the server (i.e., a *star* communication model). This is beneficial both for the server (who can better monitor and enforce the progress of the computation) and for the users (who only need to communicate with a supposedly reliable server). For example, in a protocol between Apple and its users, such a restricted communication model makes more sense than having 100M iPhone users communicate arbitrarily.

This suggests a computation model with a powerful server that on the one hand is supposed to do most of the computation work, and on the other hand, if corrupted, might tamper with the communication of the other users (because of the star topology). This combination poses several challenges (to be surveyed next), and indeed all prior works in this vein only presented constructions for *specific functionalities* such as addition; see the related works section. In this work, we set out to explore secure multiparty computation under these (and similar) conditions and present the first *general MPC construction* in this setting.

1.1 The Gulliver MPC (GMPC) Model

We consider a protocol involving a single *server* and n *users*. We think of n as being too big to allow users more than $\text{polylog}(n)$ computational time and communication complexities. On the other hand, we assume that $\text{poly}(n)$ time and communication bandwidth are within the server's reach.[1] We assume that a subset of at most $\alpha n < n/8$ of the users (fixed in advance), as well as the server, might be controlled by a malicious adversary. In particular, controlled users might halt prematurely and become unresponsive throughout the execution (e.g., iPhone users going off-line).

Utility wise, we want to ensure that if the server is honest, then it gets a *correct* outcome for the computation, even if αn users are corrupted. In other words, if the server is honest (and if at most αn users are corrupted), then output delivery is guaranteed. The privacy requirement is that an adversary controlling the server, as well as αn users, cannot learn anything more than the prescribed outcome. That is, even if the server is corrupted (as well as at most αn users), the only information that should be learned about users' inputs is what can be derived from the outcome itself.[2]

The mindset is that all of the communication goes through the server. However, one issue we must address is that the server *should not have the ability to "invent" new users, or to replace too many honest users with malicious ones*. This restriction is challenging to enforce in a star topology, and hence, as in all previous works in this vein, we assume a trusted PKI setup for this model [14, 60, 9]. As we mentioned, these previous works only presented constructions for specific functionalities such as addition. Our main result is to present the first general MPC construction under these conditions.

[1] For example, the server might be running on a computer cluster, and the users might be running on iPhones.

[2] Technically, as we consider a model that allows a malicious server to block a small fraction of honest users, the privacy requirement states that it does not obtain more information than what can be derived from the output over the remaining users.

Towards achieving this, we introduce a slightly different computation model, which we call the GMPC model. Instead of assuming a star topology with a trusted PKI setup, in the GMPC model we assume a complete communication graph (secure point-to-point channels) where the server (or an adversary controlling it) sees all of the communication patterns and can block messages at will. That is, the server knows who is sending a message to whom, and can block any message before it reaches its destination. The server does not get to see the content of the messages, and cannot modify messages. This still allows the server to tamper with the communication. In particular, if an honest user i wants to broadcast (via the server) a message to all of the users, then the server might block this message, or it can forward the message only to a *subset* of the users.

Throughout most of the paper, we focus on the above-mentioned communication model (point-to-point channels with server blocking, without assuming PKI). In Section 7 we show that protocols in the GMPC model can be compiled to operate in a star network with a PKI setup, and hence, working in the GMPC is without loss of generality.[3]

The GMPC model allows for smoother use of composition theorems. We believe that operating in the GMPC model is a better design choice, as it is both a clean abstraction and can help in simplifying the analysis. That is, we believe that the GMPC model is interesting both on its own and as a stepping stone toward constructing protocols in a star network with a PKI setup. In more detail, in our protocol we need to execute many ($\mathrm{poly}(n)$) computations in parallel, and to argue about the security of the resulting overall protocol using composition. Our GMPC setting allows us to isolate key components of our protocol which are information-theoretic, and as a result, argue about their concurrent composition in a smooth way. In contrast, with a general PKI setup, this becomes more challenging.[4] Once we know that our protocol is secure in the GMPC model (after composition has been done under the GMPC umbrella), then we can easily compile it *as a single protocol* to operate in a star network with a trusted PKI setup. See Section 7.

1.2 Our Results

Our main contribution is showing a generic construction in the GMPC model for any functionality where the users' inputs are short, of size $\mathrm{polylog}(n)$, and where only the server gets an outcome, of size $\tilde{O}(n)$, assuming less than $1/8$ of the users are corrupted (and possibly also the server).

Theorem 1.1 (Informalversion of Theorem 6.1)**.** *Let $f : (\{0,1\}^{\mathrm{polylog}(n)})^{n+1} \to \{0,1\}^{n\cdot\mathrm{polylog}(n)}$ be a computationally efficient functionality, and let $\alpha < 1/8$ be a constant. Then, f can be securely computed in the GMPC model, assuming at most αn users are corrupted, and where the server is possibly corrupted as well. Furthermore, the number of rounds in the protocol is at most $\mathrm{polylog}(n)$, the communication and computational complexity of each user is $\mathrm{polylog}(n)$, and the communication and computational complexity of the server is $\mathrm{poly}(n)$. Our construction assumes the existence of signature schemes, vector commitment schemes, pseudorandom generators, and fully homomorphic encryption schemes.*

Remark 1.2. *Several remarks are in order:*

[3] We use a specific form of PKI suitable for the GMPC model, which we call *Gulliver PKI* (GPKI).

[4] Arguing about composability when there is a general PKI setup is subtle. Basically, the difficulty is that the distinguisher is non-uniform. Thus, if the PKI is common to all sessions, then the distinguisher may know the secret key (as its auxiliary input). Now it can distinguish a true encryption of an honest party's input from a simulated encryption (as the simulator does not know the honest party's input).

1. *Our construction can also be used to compute functionalities with a long (polynomial) output, say of length T, at the cost of having users' complexities grow with $\tilde{O}(T/n)$. We chose to focus on functionalities with output length $\tilde{O}(n)$ in order to keep users' complexities polylogarithmic in n.*

2. *We are not restricted to solitary-output functionalities. We can securely compute multi-output asymmetric functionalities (where every party gets an output), at the price of having security with abort: If the server is honest then output delivery is guaranteed, and otherwise some parties may fail to receive their output.*

3. *Since the server is polynomial in n and the users are polylogarithmic in n, we require somewhat non-standard security for the primitives we use. Specifically, security should hold with respect to an adversary that is polynomial in both the security parameter κ and in n. Thus, the adversary may be superpolynomial.*

Following [18, 20, 16], the main tool we use towards proving Theorem 1.1 is a *committee election protocol*. We construct a protocol in the GMPC model that allows all parties to agree on a $\text{polylog}(n)$-sized committee, which will contain an honest majority except with negligible probability.

Theorem 1.3 (Committee election, informal version of Theorem 4.1)**.** *Let $\alpha < 1/8$ be a constant. There exists a protocol in the GMPC model such that the following holds, even if αn of the users and possibly the server are corrupted. Either all honest users abort, or at least $(1-2\alpha)n$ honest users output the same committee $\mathcal{C}$ satisfying*

1. $|\mathcal{C}| = \text{polylog}(n)$.

2. *$\mathcal{C}$ contains a vast honest majority (i.e., strictly more than 3/4 of the users in $\mathcal{C}$ are honest), except with negligible probability.*

Furthermore, the communication and computational complexity of each user is $\text{polylog}(n)$*, while the communication and computational complexity of the server is* $\text{poly}(n)$*. Our construction assumes the existence of vector commitment schemes.*

1.3 Challenges of the Gulliver MPC Model

Before presenting our constructions, we highlight a few of the challenges that arise when constructing protocols in the GMPC model. We explain how we address some of these challenges in Section 1.4.

Challenge 1: Blocks vs. aborts. The first challenge we need to address is that a corrupted server can block user messages at will. In particular, if an honest user catches the server cheating, and tries to advertise this fact to the other users, then the server can simply block this user from the rest of the computation. Furthermore, the other users would not be able to distinguish between whether this user was blocked (meaning that the server is corrupted) and whether this user simply aborted the computation.

To better illustrate this issue, let us first explain why a natural variant of Feige's committee election protocol [35] does not work. Recall that in Feige's protocol, in order to agree on a committee

of size $\approx k$, every party $i \in [n]$ samples a bin $b_i \in [n/k]$ (and broadcasts its choice), and then all parties agree on the lightest bin as the committee. We cannot implement this protocol directly in the GMPC model since (i) there is no broadcast channel, and (ii) users' complexities are bounded by polylog(n) so a user cannot send/receive messages to/from all other users directly. As an attempt to overcome these issues, consider implementing Feige's protocol in the GMPC model via the server. That is, every user sends to the server its choice of the bin, and the server then (supposedly) sends to all users the committee, defined as the users in the lightest bin, alongside the index of the bin. Each user then verifies that it is in the committee if it chose this bin, and not in the committee otherwise. The problem here is that a corrupted server can simply block all of the users who chose the lightest bin, and place corrupted users at that bin instead of the blocked users. The blocked users will have no way of "screaming for help", and the remaining users will not know that the server is cheating. We will overcome this challenge by designing a more complex committee election protocol, based on Feige's protocol, tailored to the GMPC model.

Challenge 2: DDoS attacks. Although all parties are connected to each other, the protocols we present have the property that the communication and computational complexity of each user is polylogarithmic in n. In particular, the number of messages each user sends and receives is polylogarithmic. However, note that an adversary can always blow up the communication complexity of a protocol by flooding honest parties with many garbage messages (even if the server is honest). Boyle et al. [18, 20] handled this by counting only messages that are actually processed by honest parties.[5]

Since we also require the computational complexity of the users to be polylogarithmic in n, the solution of [18, 20] is insufficient for our case. We propose two solutions for this issue. The first solution is purely combinatorial, however, it will allow the adversary to cause a few honest users to abort even if the server is honest. The second solution will prevent that, however, it is somewhat more involved. We refer the reader to Section 2.2 for more details.

Challenge 3: Verifying long statements. We do not assume any specific bound on the runtime needed to evaluate the functionality we compute, only that it can be evaluated in polynomial time. For example, it can require time n^5. As users' runtime is restricted to polylog(n) this means that we must delegate essentially all of the work to the server. This must be done carefully, as the server might be corrupted. In particular, we will need the server to prove to the users that it did the computation correctly. However, as each user is only polylogarithmic in n, any one of them cannot even read the entire statement (this is going to be a statement about a computation with n inputs). To tackle this, we let the users work together as one verifying unit when verifying long statements.

Remark 1.4. *Except for Challenge 2 (DDoS), these challenges are also present when working directly in a star network with PKI. That is, they are not a byproduct of the GMPC model. Furthermore, as we explained in Section 1.1, operating in the GMPC model helps simplify the analysis as it allows for a more smooth use of composition theorems compared to a star network with PKI,*

[5] This is formalized by splitting the "receive phase" into two phases. The first being the *filtering phase*, where each party inspects its incoming messages according to some filtering rules specified by the protocol, and discards some of the messages. This phase is followed by a *processing phase*, where the parties compute their next-message functions based on the non-filtered messages.

1.4 Our Techniques

We now provide an overview of our techniques. We start by addressing the following two challenges that arise from the server's ability to block users:

1. A malicious server can fail-stop honest users and lie about it. That is, it can block an honest user i and tell another user j that i aborted, even though it did not.
2. If an honest user catches the server cheating, then the server can block this user and prevent it from "screaming for help".

In order to deal with these two issues, we construct two tools.

Tool 1: Personal committees. To solve the first issue, each user and the server will replace the user with a *personal committee* (PC) of size polylog(n) chosen uniformly at random. Now, instead of interacting directly with the users, the server will henceforth interact with each PC (as if it is the single party it represents). If the server is honest, then this ensures that *all* personal committees, including those of malicious users, will have an honest majority, and thus they effectively become honest parties and will not abort. Therefore, if the server is malicious and is lying to some PC $\mathcal{P}$ about another PC aborting, then $\mathcal{P}$ knows that the server is cheating and therefore it will abort.

That is, once we know that a user i is represented by a PC $\mathcal{P}$, which was agreed upon by both the server and the user, then this PC should never abort, and the server cannot claim that it aborted. The question is how do we advertise the set of PCs (one for each user), in a way that remains secure even if the server is malicious. In other words, we need to figure out how can the users *agree* on the set of PCs.

Agreeing on the set of PCs. The idea is as follows. The server will commit (for every user separately) on an array of length n whose i^{th} entry is the i^{th} PC (set to $\perp$ if user i was caught cheating during the sampling of the PC). Each user i then verifies that its PC appears in this array. Then, the user chooses a random set of polylog(n) other users to compare the information with them. Specifically, it sends them the commitment it got from the server for the array (as well as the needed public parameters) and its PC. These users respond with the commitments that they got from the server and their PCs. User i then verifies with the server that the information it got from these other users is valid. Note that some users might not respond, or provide information on which the server will not agree (either malicious or blocked), but not too many of them, as otherwise user i learns that the server is cheating, and aborts.

If user i did not abort until now, then in particular, its PC appears correctly in all of the (validated) commitments it collected from the other users. Since these users were chosen at random, by the Chernoff bound(Fact 2.2), this means that user i's PC is stated correctly in the *vast majority* of the commitments given to the users in the protocol (even ones which user i did not sample). Also, by this point, every user j possesses polylog(n) randomly sampled commitments. Hence, again by the Chernoff bound, with overwhelming probability, for every user j (even one who did not communicate with user i), it must hold that the PC of user i is stated correctly in the vast majority of the commitments possessed by user j. That is, by this point the (remaining) users have reached an agreement on the array of PCs in the following sense: Every user holds a collection of commitments (validated by the server) such that for every user who did not abort the computation till now, its PC is stated correctly in the majority of these commitments.

Finally, to ensure that most honest users remain active, each of them samples polylog(n) random users, notifies them it is alive, and requests a feedback. If more than 1/8 of them did not respond, then the user aborts. If a user i did not abort till now, it can trust that at least 7/8 of the users are active, and furthermore, the PC of every active (honest) user is correctly stated and is consistent in the commitments given by the server to active honest users. From this point on, instead of interacting directly with the users, the server and the remaining users interact with the PCs (as if they are the single party they represent). By the above discussion, if the server is honest then *all* of the PCs are honest, and if the server is corrupted then at least 3/4 fraction of the PCs are honest.[6] As we mentioned, the benefit here is that now the server cannot lie about a PC aborting, because no PC should ever abort.

Tool 2: Sampling a communication graph. For the second issue, we let the parties sample an undirected communication graph, where each vertex represents a PC and the diameter of the graph is at most $O(\log n)$. Once this is done, after every message sent from the server, the PCs can simply check if their neighbors are still active for $\log n$ steps by sending a dummy message, and abort if they do not send any message. Recall that (assuming the server is honest) each PC should contain an honest majority and should never abort the execution unless it catches the server cheating. Therefore, such a communication graph allows users (or PCs) to "scream for help" in the sense that if they abort the computation, then in the following $\log(n)$ rounds we will have that *all* of the users abort the computation.

We sample this graph by having every PC sample polylog(n) other PCs as their neighbors in the graph. To see why the graph has a small diameter, consider any two vertices u and v, and suppose that we start from the vertex u and sample polylog(n) neighbors. Then, each of its neighbors samples polylog(n) more vertices, and so on. Observe that as long as less than half of the vertices corresponding to honest PCs were sampled, on expectation, at least half of the vertices sampled at each iteration are new. Therefore, by Hoeffding's inequality it follows that with overwhelming probability at least a quarter of them are new. Since this corresponds to an exponential growth, with overwhelming probability the number of iterations before covering half of the honest vertices is logarithmic. Finally, once the process covers half of the honest vertices, then one of these honest vertices samples the vertex v with overwhelming probability.[7]

1.4.1 Committee Election Protocol

We now turn to describe the ideas behind the proof of Theorem 1.3. Armed with the above two tools, sampling a committee of size $n' = \text{polylog}(n)$ can be done using the following variant of Feige's committee election protocol [35]. Let k denote the number of PCs that remain after the sampling of the communication graph. Each PC $\mathcal{P}_i$ sends to the server a randomly chosen bin $b_i \in [k/n']$, and the server replies with the set of users $\mathcal{C}$ who chose the lightest bin alongside the index i of this bin. Each PC then verifies the following:

1. It is indeed in $\mathcal{C}$ if it chose the bin i, and not in $\mathcal{C}$ otherwise.
2. $\mathcal{C}$ is not larger than the largest size the minimal bin can have, *i.e.,* the PC verifies that $|\mathcal{C}| \leq n'$.

[6] If the server is corrupted, then it can get away with blocking 1/8 fraction of the users and replacing their PCs with malicious PCs. Overall, there could be at most 2/8=1/4 malicious PCs.

[7] This is similar to arguments on the distances in small world graphs [48, 55].

3. Its neighbors in the graph received the same messages from the server.

4. Finally, each PC asks if its neighbors are still active for $\log n$ steps.

We next claim that if no PC aborts, then they all agree on a small committee $\mathcal{C}$ that contains an honest majority, except with negligible probability. First, observe that if the server is honest then no PC aborts (even those that were sampled for malicious users). Next, consider the case where the server is malicious. By the properties of the graph, if any PC aborts then all PCs abort. Thus, we may assume that no PC aborted the execution. This implies that the server sent to all PC's of honest parties the same "small" committee $\mathcal{C}$ and the same bin-index $b \in [k/n']$. Moreover, it must be the case that for all PC's of honest users, they are in $\mathcal{C}$ if and only if they sampled b.

Finally, to show that $\mathcal{C}$ contains an honest majority with overwhelming probability, we use the same analysis as Feige's original protocol. First, as we explained above, the fraction of PC's of *malicious* users is less than $1/4$. By the Chernoff bound, for any bin, the probability that the fraction of honest users is less than $3/4$ is negligible. Thus, by the union bound, the probability that there exists such a bin is negligible. Thus, except with negligible probability, the fraction of honest users in the lightest bin (which is of size at most k) must be at least $3/4$. Note that even if the server blocks another $1/8$-fraction of the users in the elected committee, then the fraction of *honest active* users among the users of the elected committee is still more than $1/2$. This allows the elected committee to securely compute any efficient functionality.[8] We refer the reader to Section 6.1 for further details. We stress that as the size of the committee is only polylogarithmic in n, then it can run computations of polynomial complexities in the number of participants, without compromising the efficiency of each individual participant of the committee.

1.4.2 Secure Computation in the GMPC Model

We proceed with outlining the proof of Theorem 1.1. Since we want low communication and computation load on the users while keeping their inputs hidden, we employ a fully homomorphic encryption (FHE) scheme. The pattern we would like to follow is that the users choose *collectively* a key to such an FHE scheme, encrypt their private inputs with the key, and send the result to the server. The server in turn computes the desired function on the encrypted input. The users then collectively decrypt the result. With a trusted (or semi-honest) server, this would have been a reasonable protocol, where the server does not learn about the users' inputs. But with a malicious server, how can the users know that the value they are decrypting is indeed the correct one and not some other function of the inputs that the server computed?

To resolve this, the users first agree on a committee $\mathcal{C}$ using our committee election protocol. Once the users agree on a committee, this committee samples keys for a fully homomorphic encryption scheme (where the secret key is shared among the members of the committee). Then, similarly to [18, 20, 16], there is a "tree process" in which using $\log n$ rounds every committee spawns two committees (both with the state of holding the secret key). After this process, we have $n+1$ elected committees that everybody "knows" (actually, the users cannot really know all the $n+1$ committees since we aim for polylog(n) complexity for the users). With overwhelming probability, all committees will contain a vast honest majority. Thus we can effectively view them as honest parties.

[8] Specifically, if the server is honest then output delivery is guaranteed, and if the server is corrupted then we get security with abort.

Next, we let the committees hold the users inputs. To do so efficiently, the i^{th} committee will request the i^{th} user for its input. To prove that the committee is honest and was sampled by the original elected committee (and not by a malicious server), we let the original committee generate signature keys and provide the committees it spawns with the secret key. Thus it can sign a message for the user to verify. (For this we need to advertise the public signature key that the original committee sampled. Advertising this public key can be done efficiently via our communication graph.) Next, every committee encrypts its input using a fully homomorphic encryption, and sends the encrypted input to the server. The server then homomorphically computes the function over the encrypted inputs.

Now we want each of the $n+1$ committees to decrypt for the server one chunk of the encrypted result. However, before they decrypt, they need to be convinced that the server did the computation correctly. Thus, the next step in our protocol is to let the server prove to the committees the following statement: "Here is a list of all the encrypted inputs I got, and here is the encrypted result of applying the desired function to these encrypted inputs".

The issue now is that the users (and each committee) are too weak to read even the statement itself (even without the proof). To overcome this challenge, we want the server to encode this statement using a (linear) error-correcting code such that afterwards the users would only need to verify that the encoding was done (roughly) correctly and they would not need to read all of the statement. So the server commits to the statement, its encoding, and the proof. Then, the $n+1$ committees together verify polylog(n) random positions of the encoding (they can do it efficiently since verifying a single position is like computing a summation, which the $n+1$ committees can do using a "tree process"). After the proof verification is over, the committees decrypt the result for the server.

1.5 Motivation from Differential Privacy: the Shuffle Model

Differential privacy [33] is a mathematical definition for privacy that aims to enable statistical analyses of datasets while ensuring that individual-level information does not leak. Traditionally, differentially private algorithms work in two main modalities: curator (centralized) and local. The curator model assumes a trusted centralized curator that collects all the personal information and then analyzes it and publishes the results. The local model, on the other hand, does not involve a central repository. Instead, each piece of personal information is randomized by its provider to protect privacy *before* submitting it to an (untrusted) server, who aggregates all the noisy messages it receives.

While the local model provides a preferable *trust model*, it involves a significant amount of noise (as noise is added to every piece of the data) which results in degraded performances compared to the curator model. In theory, existing MPC constructions can be used to implement any curator model computation without the need for a central repository (thereby essentially matching the trust assumptions of the local model). However, these constructions are still not practical enough to be implemented at the scale at which differentially private algorithms are executed, say when Apple is interacting with 100M iPhone users.

As a result, new models for differential privacy have emerged to alleviate the (inevitable) low accuracy of the local model, while hopefully maintaining its trust guarantees. This includes significant amount of work on the *shuffle model* [44, 13, 34, 29, 4, 5, 7, 28, 39, 62] as well as the *hybrid model* [2, 3, 8, 49]. We view our work as providing another piece of the puzzle, taking an important step towards bridging between MPC and the conditions at which differential privacy is currently

being deployed.

Application for the shuffle model. The shuffle model augments the local model of differential privacy with a *shuffle functionality*, that takes inputs from all the users and delivers them to the server in a random order. The line of work on the shuffle model shows that this assumption allows for significant utility improvements over the (plain) local model. However, it is not clear how one should implement such a shuffle functionality securely and efficiently. We provide a possible answer to this question in the GMPC model. Although our main result (Theorem 1.1) requires the use of heavy cryptographic machinery (e.g., fully homomorphic encryption), we stress that various simple yet important functionalities can be computed more efficiently and without relying on the more exotic types of cryptographic primitives once the parties elect a committee. In particular, aggregation and shuffling can be securely computed *without assuming FHE*. We elaborate on this in Sections 6.2 and 6.2.1.

1.6 On the (Non-)Necessity of a Common Reference String

As mentioned above, our protocol requires each user to run a secure two-party computation with the server for coin tossing. These protocols must be executed in parallel, as otherwise, the total number of rounds in our construction would be too large. There are two standard approaches for arguing about the composition of these parallel executions: Either using the *universal composition (UC)* paradigm of Canetti [23] or using *bounded concurrent composition* [54]. In our setting, bounded concurrent composition is not directly applicable because it requires the complexity of each user to depend polynomially on the number of concurrent executions [53]. As the number of concurrent executions is n, i.e., the number of users, we cannot use bounded concurrent composition (because we aim for polylog(n) user complexities). This leaves us with the UC paradigm of Canetti [23], which is definitely applicable in our setting *but at the cost of assuming a common reference string (CRS)*. Indeed, UC-secure protocols for commitments and coin tossing do not exist in the plain model; they provably require additional assumptions such as a CRS [24, 57]. Thus, to prove Theorems 1.1 and 1.3 using the real vs. ideal paradigm we need to assume a CRS.

An alternative security proof. We put forward a new proof technique that allows us to efficiently compose our n concurrent executions of coin tossing *without assuming a CRS*. To this end, we deviate from the standard real vs. ideal paradigm, and provide an alternative security proof for our coin-tossing protocols (we do not simulate them). Instead, we identify "good events" and claim that they hold with overwhelming probability. (See, for example, Theorem 1.3 above, where we claim that a "good" committee is elected with overwhelming probability.) Then, the security of our full construction is proven using the real vs. ideal paradigm assuming that our predefined "good event" has occurred. In other words, our alternative security proof boils down to: "Protocol A succeeds w.o.p., and conditioned on this then Protocol B is secure". We believe that this opens the door to the question of what to do when simulation is impossible. Do we have to deviate from the plain model, or could we deviate from the paradigm?

We stress that this discussion should be considered a separate contribution of our work, independent of the challenges of the GMPC model mentioned above. We believe this approach could find additional applications in future work. In the context of our GMPC protocol, you could either rely on our "alternative security proof" without assuming a CRS or assume a CRS and achieve "full security" by replacing our coin-tossing protocols with UC-secure protocols. The choice is yours.

1.7 Related Work

We are not the first to study a model where a strong server interacts with many "weak" users using a star topology, and similar models were considered by [14, 60, 9]. However, to the best of our knowledge, we are the first to present a general MPC construction in this setting. For example, Bell et al. [9] presented a constant-round secure-protocol for *summation in finite groups*, in which each user has only polylog(n) time and communication complexities, assuming the existence of vector commitment schemes. However, extending their construction to other functionalities is not clear. In particular, it is not clear if their protocol can be used for bit summation.[9] In contrast, our construction for bounded circuits is much more general (capturing summation as a special case) while still relying only on vector commitments.

Reyzin et al. [60] presented a protocol for a somewhat more general family of functionalities (functionalities requiring limited homomorphism), but their protocol only works in the semi-honest setting, it requires users' runtime to be polynomial in n, and is secure assuming the existence of iO. Our generic construction relies on a significantly lighter assumption (FHE), and works in the malicious setting.

Also very related to our work is the line of work on *communication locality*, initiated by Boyle, Goldwasser, and Tessaro [16]. Similarly to our work, in this model, the parties are connected via a complete graph, but each party communicates only with polylog(n) other parties. Assuming PKI, Boyle et al. [16] constructed a protocol computing any efficient functionality, tolerating at most αn corruptions, for some constant $\alpha < 1/3$. Chandran, Chongchitmate, Garay, Goldwasser, Ostrovsky, and Zikas [27] later showed how to handle *adaptive adversaries* that can corrupt at most αn parties, for some constant $\alpha < 1/2$, assuming the parties are given a *symmetric-key infrastructure*. Boyle, Cohen, Data, and Hubáček [18] considered the communication graph induced by the interaction in communication-locality protocols. Assuming the parties are given a PKI, they presented a protocol with low locality tolerating αn corruptions, where $\alpha < 1/4$ is a constant, such that with overwhelming probability the resulting communication graph is *not an expander*. Boyle, Cohen, and Goel [20] showed communication-locality protocols with the additional property of low *average* communication complexity. That is, not only does each party communicate with only polylog(n) other parties, but also the total communication in the protocol is at most $n \cdot \text{polylog}(n)$ bits. We stress that these works on communication locality are fundamentally different than ours: These works do not involve a server that does all the heavy lifting and can block messages at will, which is the main challenge we address in our work. As these works on communication locality do not involve a server, the *computational complexity* of the parties in all these works is polynomial in n (unlike polylogarithmic in n in our work).

The idea of using small committees to gain efficiency dates back at least to Bracha [21] in 1985. Since then it has been employed by several papers in several settings. In particular, this was used by Boyle et al. [16] in their communication-locality MPC protocol, and was used by Cohen et al. [30] to obtain efficient security uplifting reductions.

Lower bounds for the *bottleneck* communication complexity were investigated by Boyle et al. [19]. They showed that there exists a function such that for any protocol computing it correctly, *i.e.,* without requiring security, there exists a party whose communication complexity is $\Omega(n)$. However, this does not say much about the GMPC model, as the communication complexity of the server is

[9] For bit summation one needs to work in a finite group of size at least n, in order to avoid summation overflows, but then a single malicious user can modify the result almost arbitrarily by choosing its input to be a random number in this group. That is, a single malicious user can affect the sum by a factor of n.

large "by design".

The issue of a server who talks with weaker processors who only communicate locally and only have local information has also been investigated in the context of interactive proofs. Naor et al. [56] showed how to adapt various techniques in proof complexity in order to achieve low communication protocols for proving the correctness of many functions (essentially all NC or even P if one is satisfied with arguments) where the data is distributed among many verifiers. Boneh et al. [15] showed how to use linear PCPs in order to get zero-knowledge proofs when the input statement is not fully available to any single verifier, but can still be efficiently accessed via linear queries.

Another direction to achieve security with many different users is to employ the paradigm of serverless computing where the participants are stateless. In particular, Gentry et al. [38] have suggested the "You Only Speak Once" secure MPC. One important difference with this work is that there are no inputs to the parties that are being aggregated and computed.

1.8 Organization

The preliminaries and definition of the model of computation appear in Section 2. In Section 3 we present our "setup protocol" where the parties construct the two tools of the personal committees and the communication graph. Then, in Section 4 we show our committee election protocol for the GMPC model. In Section 5 we show a protocol that allows the users to delegate the computation to the server. The construction assumes that all users are honest and are connected via a tree network. In Section 6 we combine the committee election protocol and the protocol for delegating the computation to the server, and construct a secure protocol in the GMPC model for computing an arbitrary functionality. Finally, in Section 7 we show an application of the GMPC model, and compile a secure protocol in the GMPC model into a secure protocol in the star network assuming the parties are given a special type of PKI.

2 Preliminaries

2.1 Notation

We use calligraphic letters to denote sets, uppercase for random variables and distributions, lowercase for values, and we use bold characters to denote vectors. For $n \in \mathbb{N}$ we let $[n] = \{1, 2 \dots n\}$. For a set $\mathcal{S}$ we write $s \leftarrow \mathcal{S}$ to indicate that s is selected uniformly at random from $\mathcal{S}$. Given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. For a set $\mathcal{S} \subseteq \mathbb{N}$ and a natural number $n \in \mathbb{N}$, we denote $\mathcal{S} + n = \{s + n : s \in \mathcal{S}\}$. For a vector $\mathbf{v}$ of dimension n and for $i \in [n]$, we write v_i or $v(i)$ for its i^{th} entry. For a set $\mathcal{I} \subseteq [n]$ we denote by either $\mathbf{v}_{\mathcal{I}}$ or $\mathbf{v}(\mathcal{I})$ the vector $(v_i)_{i \in \mathcal{I}}$. A PPT is probabilistic polynomial time, and a PPTM is a PPT (interactive) Turing machine.

A function $\mu(\cdot)$ is *negligible* if for every positive polynomial $q(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < 1/q(\kappa)$. We write $\text{negl}(\cdot)$, $\text{poly}(\cdot)$, and $\text{polylog}(\cdot)$ for an unspecified negligible, polynomial, and polylogarithmic function, respectively.

Given a graph $G = (V, E)$ and a vertex $v \in V$, we let $\mathcal{N}_G(v) = \{u \in V : \{u, v\} \in E\}$ be the set of neighbors of v in the graph G. For two vertices $u, v \in V$, we denote by $\text{dist}_G(u, v)$ the distance between u and v, defined as the length of the shortest path between them (set to ∞ if there is no such path). Finally, we let $\text{diam}(G) = \max_{u,v \in V} \text{dist}_G(u, v)$ denote the diameter of G.

A *distribution ensemble* $X = \{X_{a,n}\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a domain that might depend on n. Computational indistinguishability is defined as follows.

Definition 2.1. *Let $X = \{X_{a,n}\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ be two ensembles. We say that X and Y are computationally indistinguishable, denoted $X \overset{C}{\equiv} Y$, if for every non-uniform* PPT *distinguisher D, there exists a negligible function $\mu(\cdot)$, such that for all n and $a \in \mathcal{D}_n$, it holds that*

$$|\Pr[\mathsf{D}(X_{a,n}) = 1] - \Pr[\mathsf{D}(Y_{a,n}) = 1]| \leq \mu(n).$$

Fact 2.2 (Chernoff's bound)**.** *Let $X_1, \ldots, X_n$ be i.i.d random variables over $\{0, 1\}$, and let $\mu = n \cdot \mathbb{E}[X_1]$. Then for every $t > 0$ it holds that*

$$\Pr\left[\sum_{i=1}^{n} X_i > (t+1)\mu\right] < e^{-\min\{t^2/4, t/2\}\cdot\mu}.$$

Fact 2.3 (One-sided Hoeffding's inequality for random subsets)**.** *Let $m, n \in \mathbb{N}$, where $n < m$, and let $\mathcal{S} \subseteq [m]$ be some set. Suppose a set $T \subseteq [m]$ of size n is sampled uniformly at random. Then for all $t > 0$ it holds that*

$$\Pr[|T \cap \mathcal{S}| - n \cdot |\mathcal{S}|/m \geq t] \leq e^{-2t^2/n},$$

and that

$$\Pr[|T \cap \mathcal{S}| - n \cdot |\mathcal{S}|/m \leq -t] \leq e^{-2t^2/n},$$

2.2 Secure Computation and the Model of Computation

The security of multiparty computation protocols is defined via the real vs. ideal paradigm [40, 22]. According to this paradigm, a protocol in the *real-world* model, i.e., where actual protocols are executed, is deemed secure if it "emulates" the formulated *ideal-model* for executing the task. This ideal-model involves a trusted party whose functionality captures the security requirements of the task, specifically, an adversary in this model is very limited in what it can do. To show that the real-world protocol emulates the ideal-world protocol, it is required to show that for any real-life adversary $\mathcal{A}$, there exists an ideal-model adversary Sim (called simulator) such that the global output of an execution of the protocol with $\mathcal{A}$ in the real-world model is distributed similarly to the global output of running Sim in the ideal model.

The Real Model

A multiparty protocol with n parties is defined by n interactive probabilistic polynomial-time Turing machines $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Each Turing machine (party) holds at the beginning of the execution the common security parameter 1^κ, a private input, and random coins. The *adversary* $\mathcal{A}$ is a *non-uniform* interactive Turing machine describing the behavior of the corrupted parties. It starts the execution with input that contains the identity of the corrupted parties, their inputs, and an additional auxiliary input $\mathsf{aux} \in \{0, 1\}^*$.

The parties execute the protocol over a synchronous network. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their messages for this round) followed by a *receive phase* (where they receive messages from other parties).

Throughout the execution of the protocol, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. The adversary is considered to be *malicious*, meaning that it can instruct the corrupted parties to deviate from the protocol in any arbitrary way. Additionally, the adversary has full-access to the view of the corrupted parties, which consists of their inputs, their random coins, and the messages they see throughout this execution. At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties output nothing, and the adversary outputs a function of its view (containing the views of the corrupted party).

The GMPC Model and GMPC Functionalities

In the GMPC model, we consider $n+1$ parties, $\mathsf{P}_0, \ldots, \mathsf{P}_n$. We refer to P_0 as the server and the other parties as users. The mindset is that there is a huge number of users, all of which are much less computationally powerful than the server. Formally, although all parties are polynomial in the security parameter κ, we assume the users $\mathsf{P}_1, \ldots, \mathsf{P}_n$ to be *polylogarithmic* in n, and the server to be polynomial in n. Furthermore, since we require the server to be computationally bounded, we assume that n is subexponential in κ, *i.e.,* $n = 2^{o(\kappa)}$, or equivalently, $\kappa = \omega(\log n)$. Finally, we let n be held in binary by the users, and in unary by the server. Additionally, when we say the adversary is PPT, it means that it is polynomial in both κ and n. Similarly, when we say a function is negligible, it is shorthand to saying it is negligible in both κ and n.

We consider a complete point-to-point network. Here, every pair of parties is connected via a secure and authenticated channel, and thus the adversary cannot read or modify messages sent between two honest parties. We assume the parties *do not* have access to a broadcast channel. We assume that the server, once corrupted, can *disconnect* any pair of parties. In more details, before the "send phase" of any round, the server receives the list of pairs of users $(\mathsf{P}_i, \mathsf{P}_j)$, where P_i is going to send a message to P_j in the next round, together with the length of the message. Based on this information, a corrupted server can decide which of the messages it blocks (without obtaining the contents of the messages at any point). Additionally, users may abort, however, whenever this occurs we assume that this information is given only to server, and to users that try to interact with the aborting users.

Remark 2.4. *In Section 7 we consider a star network, where the users are connected only to the server. Additionally, in Section 7 (and only there)we assume PKI of a specific form, suitable for the GMPC model, which we call* GPKI*. Intuitively, GPKI is a PKI where Gulliver cannot invent names/public-keys or hide existing ones. We stress that, with the exception of Section 7, we* do not *assume PKI in any of the other parts of this work.*

Distributed denial of service attacks. As mentioned in the introduction, an adversary can flood the network with garbage messages, even if the server is honest. We propose two solutions in order to prevent this. In both solutions, we describe what an *honest* server does to prevent a DDoS attack. If the server is malicious it does not have to prevent the attack. In this case, in all of our protocols it will suffice to instruct an honest user to abort if it receives "too many" messages,[10] as it knows that the server is malicious. Let us start with the first solution. We define a global value $\Delta = \mathrm{poly}(\log n, \kappa)$ that bounds the number of users that any other user can send a message to in

[10]The expected number of messages a user should receive in any given round is known in advance, thus the users know when to abort.

any given round. Moreover, we require the protocols to be such that in an honest execution, the probability that there exists a user receiving more than, say 100Δ, messages from different users is negligible (in both κ and n). Now, whenever some user i receives messages from more than, say, Δ^3 different users, then the *honest* server (who knows which parties interacted at any given round) blocks *all* users that interacted with user i. Note that this includes blocking *honest* users, but not too many of them since for every honest user that is being blocked there are *many* malicious users that are being blocked. For our purposes, this solution suffices since this implies that the fraction of malicious users is getting smaller by performing this attack. Note that this will affect the definition of the ideal world.

Let us now present the second solution. The protocol we construct has the property that for every round, either all users interact with a random subset of the users, or they interact with users that the server can infer from its view. In the latter set of rounds, if a malicious user tries to send a message to a user outside of the set held by the server, then the server simply blocks the user and labels it malicious. In the former case, we let each user and the server interact via a coin-tossing into a well (two-party) protocol where at the end of its execution, the user holds a random subset of the users, and the server holds a commitment to this set. When the users are required to sample a set of users to interact with, they simply open the commitment to the server, and use the subset to which they are committed to. If a malicious user tries to send a message to a user outside of its committed set, the server will catch it with overwhelming probability, and, similarly to the previous case, the server will block the user before it sends the messages.

To simplify the presentation, in all of our protocols and proofs, we assume that the adversary does not perform a DDoS attack in case the server is honest.

GMPC protocol and functionalities. We call a protocol in the above model an n-user GMPC protocol. We next define GMPC functionality. Roughly, these are solitary-output functionalities, where the server alone obtains the output. Furthermore, the length of the inputs and the output is $\tilde{O}(n)$.

Definition 2.5 (GMPC functionality)**.** *An n-ary functionality $f = \{f_\kappa\}_{\kappa\in\mathbb{N}}$ is a sequence of polynomial-time computable, randomized mappings $f_\kappa : (\mathcal{X}_\kappa)^n \to (\mathcal{Y}_\kappa)^n$.*[11] *To alleviate notations, we sometimes omit κ from functions of κ, e.g., we write $\mathcal{X}$ instead of $\mathcal{X}_\kappa$. We call an $(n+1)$-ary functionality f a GMPC functionality if $\log|\mathcal{Y}| = n \cdot \mathrm{poly}(\kappa, \log n)$ and only the first party (i.e., the server) obtains an output.*

For a protocol Π and an adversary $\mathcal{A}$, we denote by $\mathrm{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(\mathbf{x}, \kappa, n)$ the joint output of the adversary $\mathcal{A}$ and the server (assuming it is honest), in a random execution of Π on security parameter $\kappa \in \mathbb{N}$, inputs $\mathbf{x} = (x_0, \ldots, x_n) \in \{0,1\}^*$, the number of users $n \in \mathbb{N}$, and an auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

The Ideal Model

We consider an ideal computation with *guaranteed output delivery* (also referred to as *full security*), where a trusted party performs the computation on behalf of the parties, and the ideal-world adversary *cannot* abort the computation. There is one subtlety in the definition. Observe that

[11] The typical convention is to have $\mathcal{X}_\kappa = \mathcal{Y}_\kappa = \{0,1\}^*$. However, in this work, we deal with functionalities over a domain that is polylogarithmic in n, which is why we introduce this notation.

in the real world, a corrupted server can block some of the users, claiming them to be malicious. Thus, we have to allow a corrupted server to do the same in the ideal world.

We next present an ideal execution for computing a GMPC functionality f assuming a corrupted server. The case where the server is honest is presented in Appendix A (where we define security assuming the adversary cannot perform the DDoS attack described in the previous section).

Ideal world for full security with blocking assuming a corrupted server. We next describe the interaction in the ideal world assuming the server is corrupted. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, which also corrupts the server. In the following, we let α denote an upper bound on the fraction of users that can be corrupted by an adversary (which is known to all parties).

Inputs: Each party holds the security parameter 1^κ and the number of users n (held in binary by the users and in unary by the server). Additionally, the server holds input x_0, user P_i holds x_i, and the adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Parties send inputs to trusted party: The honest parties send their inputs to the trusted party. For each corrupted party, the adversary $\mathcal{A}$ sends to the trusted party some value from their domain as input. Additionally, the adversary sends a set $\mathcal{B} \subseteq [n]$ of users of size at most αn. For every user $i \in \mathcal{B}$ the trusted party replaces x_i with a default input from the same domain. Denote by $(x'_1, \ldots, x'_n)$ the tuple of inputs received (possibly modified) by the trusted party.

Trusted party sends output to the server: The trusted party computes $y \leftarrow f(x'_1, \ldots, x'_n)$ with uniformly random coins and sends the output y to the server.

Outputs: An honest server outputs the value sent by the trusted party, and a corrupted server outputs nothing. Additionally, all users output nothing and $\mathcal{A}$ outputs a function of its view (its inputs, the output, and the auxiliary input aux).

Let $\text{IDEAL}_{f,\mathcal{A}(\mathsf{aux})}(\mathbf{x}, \kappa, n)$ be the random variable consisting of the output of the adversary $\mathcal{A}$ in this ideal world execution and the output of the honest parties in the execution.

We next define secure computation. For an elaborate discussion on this notion, see [40].

Definition 2.6 (malicious security)**.** *Let $n = n(\kappa)$ denote the number of users, let $m = m(\kappa, n) = n \cdot \text{poly}(\kappa, \log n)$ denote the length of each input and the output, let $t = t(\kappa, n)$ be such that $t < n$ denote a bound on the number of corrupted users, and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^m$ be a GMPC functionality. An n-party GMPC protocol Π computing f is said to be t-secure, if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, controlling at most t user and which possibly also corrupts the server, there exists a non-uniform probabilistic polynomial-time adversary Sim in the ideal model, controlling the same parties as $\mathcal{A}$, such that the following holds*

$$\left\{\text{IDEAL}_{f,\mathsf{Sim}(\mathsf{aux})}(\mathbf{x}, \kappa, n)\right\}_{\mathsf{aux} \in \{0,1\}^*, \mathbf{x} \in \mathcal{X}^n, \kappa \in \mathbb{N}} \overset{C}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(\mathbf{x}, \kappa, n)\right\}_{\mathsf{aux} \in \{0,1\}^*, \mathbf{x} \in \mathcal{X}^n, \kappa \in \mathbb{N}}.$$

The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in exactly the same way as in the ideal models described above.

Let f be a functionality. Then an execution of a protocol Π computing a functionality g in the f-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. We consider the setting where the parties may invoke several functionalities concurrently, and where the parties can invoke one functionality during the call to another.

Security Under Composition of Protocols

The security notion defined above is known as stand-alone security, as it deals with an execution of a single protocol, executed in isolation. Generalized security notions take into consideration possible executions of other protocols that run concurrently over the same communication network. Most notable are security under general concurrent composition and universal composition [23]. These definitions deal with a more realistic setting where protocols are executed in an unknown environment, and the protocol may be liable to attacks that are not possible in the stand alone setting. Furthermore, and more relevant to this work, it is often useful to construct protocols as a composition of several sub-protocols that are executed concurrently.

The composition theorem of Canetti [23] states the following. Let ρ be a protocol that securely computes f. Then, if a protocol π computes g in the f-hybrid model, then the protocol π^ρ, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model.

Theorem 2.7 ([23]). *Let $n = n(\kappa)$ denote the number of users, let $m = m(\kappa, n) = n \cdot \mathrm{poly}(\kappa, \log n)$ denote the length of each input and the output, let $t = t(\kappa, n)$ be such that $t < n$ denote a bound on the number of corrupted users, and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^m$ be a GMPC functionality. Suppose we are given a protocol ρ computing f with t security. Further suppose that there exists a protocol π computing g with t-security in the f-hybrid model. Then protocol π^ρ computes g with t security in the real model.*

2.3 Cryptographic Tools

we use fairly standard cryptographic tools, such as secret sharing, signatures and fully homomorphic encryption (the latter being the most "exotic" one).

2.3.1 Secret Sharing

A (threshold) secret-sharing scheme [61] is a method in which a dealer distributes shares of some secret to n parties such that t colluding parties do not learn anything about the secret, and any subset of $t+1$ parties can fully reconstruct the secret.

Definition 2.8 (secret sharing). *A $(t+1)$-out-of-n secret-sharing scheme over a message space $\mathcal{M}$ consists of a pair of algorithms $(\mathsf{Share}, \mathsf{Recon})$ satisfying the following properties:*

1. *t-**privacy:** For every secret $m \in \mathcal{M}$, and every subset $\mathcal{I} \subseteq [n]$ of size $|\mathcal{I}| \leq t$, the distribution of the shares $\{s_i\}_{i \in \mathcal{I}}$ is independent of m, where $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m)$.*
2. *$(t+1)$-**reconstructability:** For every secret $m \in \mathcal{M}$, every subset $\mathcal{I} \subseteq [n]$ of size $t+1$, every set of shares $\mathbf{s} = (s_1, \ldots, s_n)$ s.t. $\Pr_{\mathbf{S} \leftarrow \mathsf{Share}(m)}[\mathbf{S} = \mathbf{s}] > 0$ and every vector $\mathbf{s}' = (s'_1, \ldots, s'_n)$ s.t. $\mathbf{s}_{\mathcal{I}} = \mathbf{s}'_{\mathcal{I}}$ and $\mathbf{s}'_{\bar{\mathcal{I}}} = \perp^{|\bar{\mathcal{I}}|}$ it holds that $m = \mathsf{Recon}(\mathbf{s}')$.*

An error-correcting secret-sharing (ECSS) scheme is a secret-sharing schemes, in which the reconstruction is guaranteed to succeed even if up to t shares are faulty. This primitive has also been referred to as *robust secret sharing* or as *honest-dealer VSS* [59, 26, 31].

Definition 2.9 (error-correcting secret sharing)**.** *A $(t+1)$-out-of-n error-correcting secret-sharing scheme (ECSS) over a message space $\mathcal{M}$ consists of a pair of algorithms* $(\mathsf{Share}, \mathsf{Recon})$ *satisfying the following properties:*

1. t**-privacy:** *As in 2.8.*
2. **Reconstruction from up to t erroneous shares:** *For every secret $m \in \mathcal{M}$, every shares $\mathbf{s} = (s_1, \ldots, s_n)$, and every $\mathbf{s}' = (s'_1, \ldots, s'_n)$ such that $\Pr_{\mathbf{S} \leftarrow \mathsf{Share}(m)}[\mathbf{S} = \mathbf{s}] > 0$ and $|\{i \mid s_i = s'_i\}| \geq n - t$, it holds that $m = \mathsf{Recon}(\mathbf{s}')$ (except for a negligible probability).*

ECSS can be constructed with perfect correctness when $t < n/3$ using Reed-Solomon decoding [10] and with a negligible error probability when $t < n/2$ by authenticating the shares using one-time MAC [59]. In case $t \geq n/2$ it is impossible to construct a $(t+1)$-out-of-n ECSS scheme, or even a secret-sharing scheme that identifies cheaters [45].

2.3.2 Vector Commitments

Our discussion follows the treatment of Fisch [36]. A vector commitment (VC) [25, 52] is a cryptographic commitment to an ordered sequence of m values $(x_1, \ldots, x_m)$ that admits succinct openings at specific positions (e.g., prove that x_i is the i^{th} committed message). For security, VCs are required to satisfy position binding, which states that an adversary should not be able to open a commitment to two different values at the same position. Moreover, VCs are required to be concise, i.e. the size of the commitment string and of its openings is independent of the vector length. Usually VCs are also required to be hiding, meaning that opening at several positions does not leak any information about the committed values at other positions. A Merkle tree is an example of a simple vector commitment that is binding and concise but not hiding.

Vector commitment syntax. We provide a redacted syntax for vector commitments, taken from [36]. A vector commitment scheme $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC.Verify})$ is a 4-tuple of PPT algorithms described as follows.

1. **Setup:** The setup algorithm $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\kappa, m, \mathcal{M})$ is given the security parameter κ, length m of the vector, and message space of vector components $\mathcal{M}$. It outputs the public parameters pp, which are implicit inputs to all the following algorithms.
2. **Commit:** The commitment algorithm $(\tau, c) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}}(\mathbf{v})$ takes an input vector $\mathbf{v} = (v_1, \ldots, v_m)$ and outputs a commitment c and an advice τ.
3. **Open:** The opening algorithm $\Lambda_{\mathcal{S}} \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(\mathbf{v}, c, \mathcal{S}, \tau)$ opens the commitment c of the message $\mathbf{v}$ at locations $\mathcal{S} \subseteq [m]$. It is given the advice τ generated by the commit algorithm. The output $\Lambda_{\mathcal{S}}$ proves that v_i is the i^{th} committed element of c for all $\mathcal{S}(i)$ (where $\mathcal{S}(i)$ denoted the i^{th} element of $\mathcal{S}$).

4. **Verify:** The verification algorithm $b \leftarrow \mathsf{VC.Verify}_{\mathsf{pp}}(c, v'_1, \ldots, v'_q, \mathcal{S}, \Lambda_{\mathcal{S}})$ takes as input the commitment c, a vector $\mathcal{S} \subseteq [m]$ of indices, and an opening proof $\Lambda_{\mathcal{S}}$. It outputs a bit $b \in \{0,1\}$ such that $b = 1$ (accept) if and only if $\Lambda_{\mathcal{S}}$ is a valid proof that c is a commitment to a vector $\mathbf{v}$, *i.e.,* $v_{\mathcal{S}(i)} = v'_i$ for all $i \in [q]$. If $\mathcal{S} = \emptyset$ then $\Lambda_\emptyset$ should be a normal opening, *i.e.,* a proof that c is a commitment to $\mathbf{v}$, namely $v_i = v'_i$ for all $i \in [m]$.

Binding commitments. The main security property of vector commitments (of interest in the present work) is position binding. The security game augments the standard binding commitment game. Roughly, the security guarantees that no PPT adversary can generate a commitment that can be decommitted to two different vectors.

Definition 2.10. *A vector commitment scheme* VC *is said to be* position binding *if for all* $\mathrm{poly}(\kappa, m)$*-time adversaries* $\mathcal{A}$ *and for all* $\mathcal{S}', \mathcal{S} \subseteq [m]$ *with* $|\mathcal{S}| = q$ *and* $|\mathcal{S}'| = q'$*, and for all* $i \in [q]$ *and* $j \in [q']$*, the following probability is at most negligible in* κ*:*

$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\kappa, m, \mathcal{M}) \\ (c, \mathbf{v}, \mathbf{v}', \Lambda, \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})}} \left[\begin{array}{c} \mathsf{VC.Verify}(c, \mathbf{v}, \mathcal{S}, \Lambda) = 1 \wedge \mathsf{VC.Verify}(c, \mathbf{v}', \mathcal{S}', \Lambda') = 1 \\ \wedge\, \mathcal{S}(i) = \mathcal{S}'(j) \wedge v_i \neq v'_j \end{array} \right]$$

2.3.3 Digital Signatures

A digital signature is a scheme for presenting the authenticity of digital messages or documents. We follow the presentation in [47].

Definition 2.11. *A* signature scheme $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ *is a 3-tuple of* PPT *algorithms described as follows.*

1. **Key generation:** *The* key generation *algorithm* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$ *takes as input a security parameter* 1^κ *and outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*, called the* public key *and* private key*, respectively. We assume for convenience that* pk *and* sk *each have length at least* κ*, and that* κ *can be determined from* pk *and* sk*.*

2. **Sign:** *The* signing algorithm $\sigma \leftarrow \mathsf{Sig.Sign}_{\mathsf{sk}}(m)$ *takes as input the private-key* sk *and a message* $m \in \{0,1\}^*$*, and outputs a signature* σ*.*

3. **Verify:** *The deterministic* verification *algorithm* $b \leftarrow \mathsf{Sig.Verify}_{\mathsf{pk}}(m, \sigma)$ *takes as input the public-key* pk*, a message* m*, and a signature* σ*. It outputs a bit* $b \in \{0,1\}$ *such that* $b = 1$ *(accept) if and only if* σ *is a valid signature of* m*.*

It is required that for every $\kappa \in \mathbb{N}$*, every* $(\mathsf{pk}, \mathsf{sk})$ *output by* $\mathsf{Sig.Gen}(1^\kappa)$*, and every message* $m \in \{0,1\}^*$*, it holds that* $\mathsf{Sig.Verify}_{\mathsf{pk}}(m, \mathsf{Sig.Sign}_{\mathsf{sk}}(m)) = 1$*.*

Security of signature schemes. We next define security of signature schemes. We call σ a *valid signature* on a message m (with respect to some public key pk) if $\mathsf{Sig.Verify}_{\mathsf{pk}}(m, \sigma) = 1$. We say that an adversary *forges* a signature if it outputs a message m along with a valid signature σ on m, and furthermore m was not previously signed using the secret key.

Let $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and security parameter κ, denoted as $\mathsf{Sig\text{-}forge}_{\mathcal{A},\mathsf{Sig}}(\kappa)$:

1. $\mathsf{Sig.Gen}(1^\kappa)$ is run to obtain keys $(\mathsf{pk}, \mathsf{sk})$

2. Adversary $\mathcal{A}$ is given pk and oracle access to $\mathsf{Sig.Sign}_{\mathsf{sk}}(\cdot)$ (this oracle returns a signature $\mathsf{Sig.Sign}_{\mathsf{sk}}(m)$ for any message m of the adversary's choice). The adversary then outputs a message-signature pair (m, σ). Let $\mathcal{Q}$ denote the set of messages whose signatures were requested by $\mathcal{A}$ during its execution.

3. The output of the experiment is 1 if $\mathsf{Sig.Verify}_{\mathsf{pk}}(m, \sigma) = 1$ and $m \notin \mathcal{Q}$, and is defined to be 0 otherwise.

The following defines the unforgeability property required from signature schemes. We also consider security against adversaries that can run in polynomial time in a possibly larger parameter $n(\kappa)$.

Definition 2.12. *A signature scheme* Sig *is* existentially unforgeable under an adaptively chosen message attack *if for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\Pr\left[\mathsf{Sig\text{-}forge}_{\mathcal{A},\mathsf{Sig}}(\kappa) = 1\right] \leq \mathrm{negl}(\kappa).$$

For any function $n = n(\kappa)$*, the signature scheme is said to be* n-secure *if the above holds with respect to any adversary that runs in* $\mathrm{poly}(\kappa, n)$ *time.*

2.3.4 Fully Homomorphic Encryption

We next define fully homomorphic encryption (FHE) schemes [37].

Definition 2.13. *A* fully homomorphic encryption scheme *over a circuit family* $\mathcal{C}$*, is a 4-tuple of* PPT *algorithms* $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$*:*

1. **Key generation:** *The* key generation *algorithm* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^\kappa)$ *takes as input a security parameter* 1^κ *and outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*, called the public key and private key, respectively.*

2. **Encrypt:** *The* encryption algorithm $c \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(m)$ *takes as input the public-key* pk *and a plaintext* $m \in \{0,1\}^*$*, and outputs a ciphertext* c*.*

3. **Decrypt:** *The deterministic* decryption *algorithm* $m^* = \mathsf{FHE.Dec}_{\mathsf{sk}}(c)$ *takes as input the private-key* pk *and a ciphertext* c*. It outputs* $m^* \in \{0,1\}^* \cup \{\perp\}$ *which is either a valid plaintext or a special* $\perp$ *symbol denoting failure.*

4. **Evaluate:** *The* evaluation algorithm $c' \leftarrow \mathsf{FHE.Eval}_{\mathsf{pk}}(C, c)$ *takes as input the public key* pk*, a circuit* $C \in \mathcal{C}$*, and a ciphertext* c*. It outputs a new ciphertext* c'*,*

The following are required:

1. **Correctness:** *For every* $\kappa \in \mathbb{N}$*, every* $(\mathsf{pk}, \mathsf{sk})$ *output by* $\mathsf{FHE.Gen}(1^\kappa)$*, every circuit* $C \in \mathcal{C}$*, and every message* $m \in \{0,1\}^*$*, it holds that for a corresponding ciphertext* $c = \mathsf{FHE.Enc}_{\mathsf{pk}}(m)$*,*

$$\mathsf{FHE.Dec}_{\mathsf{sk}}\left(\mathsf{FHE.Eval}_{\mathsf{evk}}(C, c)\right) = C(m).$$

2. **Compactness:** *There exists a polynomial* p *such that for every* $\kappa \in \mathbb{N}$*, the decryption algorithm* $\mathsf{FHE.Dec}$ *can be expressed as a circuit of size at most* $p(\kappa)$*.*

Security against chosen-plaintext attacks. To define chosen-plaintext attack (CPA) security of FHE schemes, let $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be an FHE scheme. Consider the following experiment for an adversary $\mathcal{A}$ and security parameter κ, denoted $\mathsf{FHE\text{-}CPA}_{\mathcal{A},\mathsf{FHE}}(\kappa)$:

1. $\mathsf{FHE.Gen}(1^\kappa)$ is run to obtain keys $(\mathsf{pk}, \mathsf{sk})$.
2. Adversary $\mathcal{A}$ is given pk and oracle access to $\mathsf{FHE.Enc}_{\mathsf{pk}}(\cdot)$. The adversary outputs a pair of messages m_0 and m_1 of equal length.
3. A bit $b \leftarrow \{0,1\}$ is sampled uniformly at random, and then the ciphertext $c \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(m_b)$.
4. The adversary (still having oracle access to $\mathsf{FHE.Enc}_{\mathsf{pk}}(\cdot)$) outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$, and is defined to be 0 otherwise.

The following defines CPA security as we require from FHE schemes. Similarly to signature schemes, we also consider security against adversaries that can run in polynomial times in a possibly larger parameter $n(\kappa)$.

Definition 2.14. *An FHE scheme is said to be* CPA-secure *if for all* PPT *adversaries $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{FHE\text{-}CPA}_{\mathcal{A},\mathsf{FHE}}(\kappa) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(\kappa).$$

For any function $n = n(\kappa)$, the scheme is said to be n-secure *if the above holds with respect to any adversary that runs in* $\mathrm{poly}(\kappa, n)$ *time.*

2.4 Probabilistically Checkable Proofs of Proximity

A *probabilistically checkable proof of proximity* (PCPP) [11, 32] is a proof system, which allows a verifier to be convinced that the input is close to being in a language L. In more detail, the verifier has an explicit input x and an implicit input y given as an oracle. The verifier accepts with high probability if y is close to some y' such that $(x, y') \in L$. We next formalize the notion of PCPP. The definitions below are taken almost verbatim from [12]. We start with defining the Hamming distance.

Definition 2.15 (Hamming distance)**.** *Let $n \in \mathbb{N}$. The* Hamming distance *between two strings $x, x' \in \{0,1\}^n$ is defined as $\Delta(x, x') := |\{i \in [n] : x_i \neq x'_i\}|$. For a string $x \in \{0,1\}^n$ and a set $\mathcal{S} \subseteq \{0,1\}^n$ we let $\Delta(x, \mathcal{S}) = \min_{x' \in \mathcal{S}} \delta(x, x')$. Finally, a string $x \in \{0,1\}^n$ is said to be δ-far from a set $\mathcal{S} \subseteq \{0,1\}^n$ if $\Delta(x, \mathcal{S}) > \delta$.*

Definition 2.16 (Restricted verifier)**.** *Let $r, q : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. An (r, q, t)-*restricted verifier *is a probabilistic oracle Turing machine V, that is given a string x number $k \in \mathbb{N}$ (in binary), an oracle access to an input $y \in \{0,1\}^k$ and a proof $\pi \in \{0,1\}^*$, tosses $r(|x| + k)$ coins, queries the oracle (y, π) at most $q(|x| + k)$ times, runs in time $t(|x|, k)$, and outputs either 0 or 1.*

For a pair language $L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a string $x \in \{0,1\}^*$, we let $L_x = \{y \in \{0,1\}^* : (x, y) \in L\}$.

Definition 2.17 (PCPP for pair languages)**.** *For functions $r, q : \mathbb{N} \to \mathbb{N}$, $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $s, \delta : \mathbb{N} \to [0,1]$, a pair language $L \subseteq \{0,1\}^* \times \{0,1\}^*$ is said to be in $\mathrm{PCPP}_{s,\delta}[r,q,t]$ if there exists an (r,q,t)-restricted verifier V such that the following holds.*

Completeness: *If $(x,y) \in L$ then there exists a proof $\pi \in \{0,1\}^*$ such that*

$$\Pr\left[\mathsf{V}^{y,\pi}(x,|y|) = 1\right] = 1,$$

where the probability is taken over the random coin tosses of V.

Soundness: *If (x,y) is such that y is $\delta(|x|+|y|)$-far from $L_x \cap \{0,1\}^{|y|}$, then for every proof $\pi \in \{0,1\}^*$ it holds that*

$$\Pr\left[\mathsf{V}^{y,\pi}(x,|y|) = 1\right] \leq s\left(|x|+|y|\right),$$

where the probability is taken over the random coin tosses of V.

Theorem 2.18 (Efficient PCPPs for pair languages [12])**.** *For every pair language $L \in \mathrm{NP}$ and every constant $s > 0$, it holds that $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$ where*

- $\delta(m) = m/\operatorname{polylog}(m)$,
- $r(m) = O(\log(m))$,
- $q(m) = \operatorname{polylog}(m)$,
- $t(n,k) = \operatorname{poly}(n, \log(n+k))$.

Moreover, it is implicitly stated that for every $(x,y) \in L$ a correct proof can be generated in polynomial time in $|x|+|y|$ (given a witness).

Repeating the proof a polylogarithmic number of times results in a negligible soundness error.

Corollary 2.19. *For every pair language $L \in \mathrm{NP}$ it holds that $L \in \mathrm{PCPP}_{s,\delta}[r,q,t]$ where*

- $s(m) = e^{-\log^2 m}$.
- $\delta(m) = m/\operatorname{polylog}(m)$,
- $r(m) = \operatorname{polylog}(m)$,
- $q(m) = \operatorname{polylog}(m)$,
- $t(n,k) = \operatorname{poly}(n, \log(n+k))$.

Moreover, it is implicitly stated that for every $(x,y) \in L$ a correct proof can be generated in polynomial time in $|x|+|y|$.

2.5 Committee Election

Feige's lightest-bin protocol [35] is an elegant n-party, public-coin protocol, consisting of a single broadcast round, for electing a committee of size $n' < n$, in the information-theoretic setting. Each party uniformly selects one of $\lceil n/n' \rceil$ bins and broadcasts it choice. The parties that selected the lightest bin are elected to participate in the committee. The protocol ensures that the ratio of corrupted parties in the elected committee is similar to their ratio in the population. The original protocol in [35] considered committees of size $\log(n)$, however, this results in a *non-negligible* failure probability. [17, Lem. 2.6] analyzed Feige's protocol for arbitrary committee sizes and proved the following lemma.

Lemma 2.20 ([17])**.** *For integers $n' < n$ and constants $0 < \alpha < \alpha' < 1$ define*

$$\mathsf{err}\left(n, n', \alpha, \alpha'\right) = \frac{n}{n'} \cdot e^{-\frac{(\alpha'-\alpha)^2 n'}{2(1-\alpha)}}.$$

Feige's lightest-bin protocol is a 1-round, n-party protocol for electing a committee $\mathcal{C}$, such that for any set of corrupted parties $\mathcal{I} \subseteq [n]$ of size $t \leq \alpha n$ the following holds.

1. $|\mathcal{C}| \leq n'$.
2. $\Pr\left[|\mathcal{C} \setminus \mathcal{I}| \leq (1-\alpha') \cdot n'\right] < \mathsf{err}(n, n', \alpha, \alpha')$.
3. $\Pr\left[|\mathcal{C} \cap \mathcal{I}| \geq \alpha' \cdot |\mathcal{C}|\right] < \mathsf{err}(n, n', \alpha, \alpha')$.

Towards proving Lemma 2.20, using Chernoff's inequality and the union bound [17] showed the following.

Lemma 2.21 (Implicit in [17])**.** *Fix integers $n' < n$, constants $0 < \alpha < \alpha' < 1$, and a set $\mathcal{I} \subseteq [n]$ of corrupted parties. For every bin $b \in [\lceil n/n' \rceil]$ we let X_b denote the number of honest parties that sampled b in Feige's protocol. Then*

$$\Pr\left[\exists b \in [\lceil n/n' \rceil] : X_b < (1-\alpha') \cdot n'\right] < \mathsf{err}\left(n, n', \alpha, \alpha'\right).$$

3 Preventing Blocking of Honest Users

Toward constructing our secure protocols, we first develop two tools. The goal of these two tools is to prevent a malicious server from blocking honest users. The first tool is the sampling of *personal committees* (PC) that are chosen uniformly at random and will replace each user. The idea is that with high probability every PC of an honest party (hereinafter, honest PC) will contain an honest majority. Furthermore, if the server is honest, then the same holds for malicious users. Thus, if a PC aborts after all of them were sampled, then this constitutes a proof of the server being malicious. The second tool is locally sampling a communication graph (*i.e.,* each PC samples a small number of neighbors) with a small diameter that the parties will use to essentially allow any single PC to raise a flag indicating to all other PCs that the server is malicious. If the server is honest, then since all PCs contain an honest majority, no PC would ever raise such a flag.

We stress that the security of the construction is not defined via the real vs. ideal paradigm. Instead, we define a "good" event and show that it occurs except with negligible probability (in both the security parameter κ and the number of users n). In Section 3.1 we present a protocol

for sampling personal committees. Then, in Section 3.2 we present the formal definition of the graph distribution we use, and prove that with high probability it has a small diameter. Then, in Section 3.3 we combine the two results, and present the final setup protocol for sampling the PCs and graph with the desired security properties.

3.1 Personal Committee Protocol

We present a protocol for sampling the personal committees. Recall that the goal of the PCs is to replace each user, thus any malicious behaviour of the committee as a whole implies that the server is malicious. Before formally describing the protocol, we first give a short overview of the construction.

First, each user samples together with the server a PC using a coin-tossing protocol. From here, the goal of the honest users and the (honest) server, is to provide a proof for other users that the PC was sampled by them. However, as the users are assumed to be polylogarithmic in n, they cannot actually hold the entire proof. Instead, we let each user verify consistency of information with $\kappa = \omega(\log n)$ randomly sampled other users. If not too many users aborted, then except with negligible probability, the server must have played honestly.

The idea is as follows. The server will commit to an array of length n whose i^{th} entry is the PC of user i (set to $\perp$ if the user aborted early). Each user then verifies that it appears in this array, and in addition, samples κ other users uniformly at random. These κ users will be used for consistency checks.

The neighboring users compare their information, and further verify it with the server (by requesting to open the commitment at the correct positions). Observe that it could be the case that some of the neighboring users are blocked by a malicious server, and the user cannot distinguish this from the case of a malicious user. However, if the server is honest then, with overwhelming probability, at least $\approx (1-\alpha)\kappa$ users will respond with information that is consistent with the information provided by the server. Thus, we let each user verify that this is indeed the case.

We then let each user notify the users in its PC that they belong to its personal committee. These users then verify this with the server, and ignores the message (without aborting) from the user in case the server's answer is inconsistent. We then let the users in each PC *broadcast* to all other users in the same PC that they participate in the PC,[12] and verify that at least $1-\alpha$ fraction of them sent a message. To implement the broadcast channel, the users will use a variant of the broadcast protocol due to [58, 51] that is secure against at most $\alpha < 1/8$ corruptions. We refer the reader to Section 6.1 for a detailed description of the broadcast protocol. Note that if the server is honest, then each PC will contain more than roughly $(1-\alpha)\kappa$ honest users. Therefore, the users can safely abort if less than $(1-\alpha)\kappa$ users are active in the PC. Finally, to ensure that most honest users remain, each of them samples κ random users, notifies them they are alive, and requests a feedback. If more than roughly $\alpha\kappa$ of them did not respond, then the user aborts.

We now present the personal committee protocol. In the following, we let $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC}$ be a vector commitment scheme, let C be a perfectly hiding commitment scheme[13], and let $\varepsilon = 1/8 - \alpha$ be a positive constant.

[12] Formally speaking, the parties compute the multicast functionality, where only a subset of the users obtain the output.

[13] Using a perfectly hiding commitment scheme is done only for convenience.

Protocol 1: `PersonalCommittee`

Common inputs: All parties hold the security parameter 1^κ and the number of users n (held in binary by the users, and in unary by the server).

1. The server and each user i interact as follows.
   (a) Let $k = \kappa \cdot \log n$ be the length of the representation of each PC. User i samples a random string $r_i \leftarrow \{0,1\}^k$, computes a commitment $\hat{c}_i \leftarrow \mathsf{C.Com}(r_i)$, and sends $\hat{c}_i$ to the server.
   (b) The server responds with a random string $s_i \leftarrow \{0,1\}^k$.
   (c) Upon receiving s_i from the server, user i computes $\mathcal{P}_i := r_i \oplus s_i$ and sends to the server a decommitment to $\hat{c}_i$. In case the server does not send s_i to user i, the user outputs $\perp$.
   (d) If the decommitment of user i is invalid, the server labels i as inactive.
2. Each remaining user i computes the public parameters $\mathsf{pp}_i \leftarrow \mathsf{VC.Setup}(1^\kappa, n, [n] \times \binom{[n]}{\kappa})$ for the commitment, and sends pp_i to the server.
3. The server does the following:
   (a) Set $\mathbf{v}$ to be the vector of length n whose i^{th} entry is $\mathcal{P}_i$ in case i is active, and is set to $\perp$ otherwise.
   (b) For each remaining user i do the following:
      i. Compute the commitment $(\tau_i, c_i) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}_i}(\mathbf{v})$ of the vector $\mathbf{v}$.
      ii. Open the commitment at position i to obtain $\Lambda_i \leftarrow \mathsf{VC.Open}_{\mathsf{pp}_i}(\mathbf{v}, c_i, i, \tau_i)$.
      iii. Send c_i and Λ_i to user i.
4. User i aborts if the server did not decommit properly, *i.e.,* $\mathsf{VC.Verify}_{\mathsf{pp}_i}(c_i, \mathcal{P}_i, i, \Lambda_i) = 0$.
5. Each user i does the following:
   (a) Sample a set of users $\mathcal{S}_i^{\mathtt{out}} \subseteq [n]$ of size κ uniformly at random and notify each of them and the server[14] that they were sampled by it.
   (b) Set $\mathcal{S}_i^{\mathtt{in}}$ to be the set of users that sampled it. It aborts if it was sampled by too many users, *i.e.,* $|\mathcal{S}_i^{\mathtt{in}}| > 3\kappa$.
   (c) Otherwise, send (pp_i, c_i) to all of users in $\mathcal{S}_i^{\mathtt{in}}$.
   (d) Let $\mathcal{S}_i \subseteq \mathcal{S}_i^{\mathtt{out}}$ be the set of users j that responded with a message (pp_j, c_j).
   (e) For each $j \in \mathcal{S}_i$, compare the information with the server: The server decommits to c_j at location i by sending $\Lambda_{i,j} \leftarrow \mathsf{VC.Open}_{\mathsf{pp}_j}(\mathbf{v}, c_j, i, \tau_j)$ to user i.
   (f) Check that at least $(1 - 2\alpha - \varepsilon/2)\kappa$ of the users in $\mathcal{S}_i$ sent a message that is consistent with the server, namely, $\mathsf{VC.Verify}_{\mathsf{pp}_j}(c_i, \mathcal{P}_i, i, \Lambda_{i,j}) = 1$ for all $j \in \mathcal{S}_i$. If this is not the case, then abort.

[14] Note that by the assumption on the network, the server can obtain $\mathcal{S}_i^{\mathtt{out}}$ by simply observing which parties interact. We decided to explicitly have the user notify the server for the sake of presentation.

Protocol 1: `PersonalCommittee` (Continued)

6. The server and each user i interact as follows:
   (a) The user samples a *query set* $\mathcal{Q}_i \subseteq [n]$ and sends it to the server.
   (b) The server opens each of the commitments c_j, where $j \in \mathcal{S}_i \cup \{i\}$ and locations $\mathcal{Q}_i$. That is, it sends $\mathbf{v}(\mathcal{Q}_i)$ and $\Lambda^{\mathcal{Q}}_{i,j} \leftarrow \mathsf{VC.Open}_{\mathsf{pp}_j}(\mathbf{v}, c_j, \mathcal{Q}_i, \tau_j)$ for every $j \in \mathcal{S}_i \cup \{i\}$.
   (c) Let $\mathbf{u}_i$ be the purported value of $\mathbf{v}(\mathcal{Q}_i)$ sent by the server. The user verifies all decommitments, *i.e.,* it checks that $\mathsf{VC.Verify}_{\mathsf{pp}_j}(c_j, \mathbf{u}_i, \mathcal{Q}_i, \Lambda^{\mathcal{Q}}_{i,j}) = 1$ for all $j \in \mathcal{S}_i \cup \{i\}$. If this is not the case then the user aborts.
7. Each remaining user i sends the set $\mathcal{P}_i$ to all users in $\mathcal{P}_i$.
8. If some user j received more than 3κ messages, then it aborts.
9. Otherwise, for every $\mathcal{P}_i \ni j$ received, user j compares this information with the server: the server decommits to c_j at location i by sending $\Lambda'_{i,j} \leftarrow \mathsf{VC.Open}_{\mathsf{pp}_j}(\mathbf{v}, c_j, i, \tau_j)$ to user j.
10. If $\mathsf{VC.Verify}_{\mathsf{pp}_j}(c_j, \mathcal{P}_i, i, \Lambda'_{i,j}) = 0$, then the user *ignores* the message without aborting.
11. For each personal committee $\mathcal{P}_i$, all users in it broadcast alive to all other users in $\mathcal{P}_i$ (see Section 6.1 for a secure implementation).
12. The personal committee $\mathcal{P}_i$ is labeled inactive and does not participate in subsequent interaction, if more than $\alpha + \varepsilon/2$ of its users did not send alive, in which case user i aborts.
13. Each remaining user i samples κ users and sends them the message alive. It then replies with alive for every alive received.
14. User i aborts if it received less than $(1 - 2\alpha - \varepsilon/2)\kappa$ replies.

Output phase: Each user i outputs $\{(j, \mathcal{P}_j)\}_{j:i\in\mathcal{P}_j}$, pp_i, and c_i. An honest server outputs the list $\mathcal{L} = \{(i, \mathcal{P}_i)\}_i$ of alive users and their corresponding PC, and $(\mathsf{pp}_i, c_i, \tau_i)_{i\in\mathcal{L}}$.

The security properties of the protocol (stated as an event occurring with overwhelming probability) appear in Section 3.3, as part of the security properties of the setup protocol presented in the same section.

We next claim that all users are polylogarithmic in n. By construction, this boils down to showing that no user will be sampled by too many users (in any of the sets sampled throughout the protocol). Observe that this directly follows from Chernoff's inequality (Fact 2.2) since every user is sampled by at most κ user on expectation.

3.2 A Locally Sampled Communication Graph With a Small Diameter

In this section, we discuss the distribution on graphs that we use for sampling the communication graph. Recall that the goal of the graph is to allow the users to quickly notify each other if any of them caught the server cheating. The distribution over the graphs that we use is the union of stars, each of a fixed size d, sampled uniformly at random. We next formalize the definition of this distribution.

Definition 3.1. *Let $n, d \in \mathbb{N}$ where $1 \le d \le n$ and let $\mathsf{Bad} \subseteq [n]$. We define the distribution $\mathcal{G}(n, d, \mathsf{Bad})$ over* directed *graphs $G = ([n], E)$ with n vertices as follows. For each vertex $i \in [n]\backslash\mathsf{Bad}$*

sample a subset $\mathcal{N}_i^{\mathtt{out}} \subseteq [n]$ *of size* d *and set* $(i,j) \in E$ *(i.e., there is an edge from* i *to* j*) if and only if* $j \in \mathcal{N}_i^{\mathtt{out}}$.

The following lemma asserts that with overwhelming probability, for a graph sampled according to the distribution $\mathcal{G}(n,d,\mathcal{S})$ any induced subgraph over a sufficiently large constant fraction of the vertices of $[n] \setminus \mathsf{Bad}$, will have a small diameter with high probability.

Looking ahead, each vertex in the graph will represent a personal committee of some user. Therefore, the lemma can be used to show that the induced subgraph over the set of honest PCs will have a small diameter with overwhelming probability. However, recall that a malicious server might abort some honest users. Thus, the set of "bad" users, which the induced subgraph ignores, is of size $2\alpha n$, and not αn. In the following, we let $\beta = 2\alpha < 1/4$ denote an upper bound on the fraction of malicious and blocked users.

Lemma 3.2. *Let* $n, d \in \mathbb{N}$ *such that* $4 \leq d \leq n$*, let* $\beta < 1/4$*, let* $\ell = \log_{d/4}(n/3)$*, and let* $\mathsf{Bad} \subseteq [n]$ *be of size* $|\mathsf{Bad}| \leq \beta n$*. Consider the following distribution over graphs with* n *vertices: sample* $G \leftarrow \mathcal{G}(n,d,\mathsf{Bad})$ *and let* G' *denote the induced subgraph of* G *over the vertices* $[n] \setminus \mathsf{Bad}$*. Then*

$$\Pr\left[\operatorname{diam}\left(G'\right) > \ell + 1\right] < n^2 \cdot (\ell+1) \cdot e^{-4d/9},$$

where the probability is taken over the sampling of G'*.*

Proof. Let $u, v \in [n] \setminus \mathsf{Bad}$ be two distinct vertices in G'. We show that

$$\Pr\left[\operatorname{dist}_{G'}(u,v) > \ell + 1\right] < (\ell+1) \cdot e^{-4d/9}. \tag{1}$$

The proof then follows from the union bound.

We define the following sequence of random variables. For $i \in \{0,\ldots,\ell\}$ let

$$S_i = \{w \in [n] \setminus \mathsf{Bad} : \operatorname{dist}(u,w) = i\}$$

be the set of vertices of distance *exactly* i from u in the graph G', and let $S_i^* := \cup_{j=0}^{i} S_j$ denote the set of distance at most i from u in the graph G'. It suffices to show that except with probability at most $\ell \cdot e^{-4d/9}$, it holds that $|S_\ell^*| \geq n/3$. Indeed, let $\mathcal{N}_v^{\mathtt{in}} = \{i \in [n] : v \in \mathcal{N}_i^{\mathtt{out}}\}$ denote the set of vertices that sampled v. Then

$$\Pr\left[\mathcal{N}_v^{\mathtt{in}} \cap S_\ell^* = \emptyset \mid |S_\ell^*| \geq n/3\right] \leq \left(1 - \frac{d}{(1-\beta)n}\right)^{|S_\ell|} < e^{-\frac{d}{3(1-\beta)}} < e^{-4d/9}.$$

Toward proving that $|S_\ell^*| \geq n/3$ holds with high probability, we prove the following claim. Roughly, the claim asserts that for every $i \leq \ell$, if $|S_\ell^*| < n/3$ and S_{i-1} is somewhat large, then S_i is somewhat large as well with high probability.

Claim 3.3. *For every* $i \in \{0,\ldots,\ell\}$ *it holds that*

$$\Pr\left[|S_i| < (d/4)^i \;\middle|\; |S_\ell^*| < n/3 \wedge \forall j \in [i-1] : |S_j| \geq (d/4)^j\right] \leq e^{-d/8},$$

if $i > 0$*, and that*

$$\Pr\left[|S_i| < (d/4)^i \;\middle|\; |S_\ell^*| < n/3\right] = 0,$$

if $i = 0$*.*

The claim is proven below. We first show that it indeed implies that $|S^*_\ell| \geq n/3$ with high probability. Since the S_i's are pairwise disjoint and ℓ satisfies

$$\sum_{i=0}^{\ell}(d/4)^i = \tfrac{(d/4)^{\ell+1}-1}{d/4-1} \geq (d/4)^\ell = n/3,$$

it follows that if $|S^*_\ell| < n/3$ then there exists $i \in \{0,\ldots,\ell\}$ such that $|S_i| < (d/4)^i$. Thus

$$\begin{aligned}
\Pr\left[|S^*_\ell| < n/3\right] &= \Pr\left[|S^*_\ell| < n/3 \wedge \exists i \in [\ell] : |S_i| < (d/4)^i\right] \\
&\leq \Pr\left[\exists i \in [\ell] : |S_i| < (d/4)^i \;\middle|\; |S^*_\ell| < n/3\right] \\
&\leq \sum_{i=1}^{\ell} \Pr\left[|S_i| < (d/4)^i \;\middle|\; |S^*_\ell| < n/3 \wedge \forall j \in [i-1] : |S_j| \geq (d/4)^j\right] \\
&\leq \ell \cdot e^{-d/8}
\end{aligned}$$

where the second inequality follows from the union bound. It remains to prove Claim 3.3.

Proof of Claim 3.3. The proof is by induction on i. Clearly, the claim holds for $i = 0$. Assume that the claim holds for $i \leq \ell - 1$. We next prove it for $i+1$. First, observe that if $|S^*_\ell| < n/3 < \frac{(1-\beta)}{2} \cdot n$, then half of the vertices in G' are not in S_{i+1}. Thus, for any fixation $\mathcal{S}_i$ of S_i, the (conditional) expected size of S_{i+1} is at least

$$\mathbb{E}\left[|S_{i+1}| \;\middle|\; |S^*_\ell| < n/3 \wedge S_i = \mathcal{S}_i\right] \geq d \cdot |\mathcal{S}_i|/2.$$

Therefore,

$$\mathbb{E}\left[|S_{i+1}| \;\middle|\; |S^*_\ell| < n/3 \wedge |S_i| \geq (d/4)^i\right] \geq (d/2) \cdot (d/4)^i.$$

In particular, it holds that

$$\mu := \mathbb{E}\left[|S_{i+1}| \;\middle|\; |S^*_\ell| < n/3 \wedge \forall j \in [i] : |S_j| \geq (d/4)^j\right] \geq (d/2) \cdot (d/4)^i.$$

Thus, by Fact 2.3, it follows that

$$\begin{aligned}
&\Pr\left[|S_{i+1}| < (d/4)^{i+1} \;\middle|\; |S^*_\ell| < n/3 \wedge \forall j \in [i] : |S_j| \geq (d/4)^j\right] \\
&\leq \Pr\left[|S_{i+1}| - \mu < -(d/4)^{i+1} \;\middle|\; |S^*_\ell| < n/3 \wedge \forall j \in [i] : |S_j| \geq (d/4)^j\right] \\
&\leq e^{-\frac{2(d/4)^2}{d}} \\
&= e^{-d/8}.
\end{aligned}$$

□

□

3.3 Preparing For Computation in the GMPC Model: The Setup Protocol

In this section, we combine Protocol 1 for sampling PCs and the result from Section 3.2 for sampling a graph, and construct our setup protocol. Roughly, the protocol proceeds by first executing Protocol 1, which samples the PCs and forms some sort of agreement (see Theorem 3.5 below the protocol's description). Then, each PC samples its neighbors uniformly at random. To ensure that the graph is undirected, the user requests a receipt from each of the users it sampled, which in turn add the requesting users to their set of neighbors.

We abuse notions and describe the protocol as if each PC is a single party. Formally, whenever we say that the server sends a message to $\mathcal{P}_i$, it means that it (supposedly) shares it among the users of $\mathcal{P}_i$. Similarly, when we say that $\mathcal{P}_i$ sends a message to the server, it means that every user in $\mathcal{P}_i$ sends its share of the message to the server. Finally, whenever a PC $\mathcal{P}_i$ sends a message msg to another PC $\mathcal{P}_j$, it means that each user in $\mathcal{P}_i$ shares its share of msg among the users in $\mathcal{P}_j$.

Formally, we describe the protocol in a hybrid world, where the hybrid functionalities compute the "next-message" function of each PC.[15] Let NxtMsg_i denote the next-message function of $\mathcal{P}_i$. The inputs of the users to each call of NxtMsg_i are a $(1-\alpha-\varepsilon/2)\kappa$-out-of-$\kappa$ Shamir's secret sharing scheme of the view of $\mathcal{P}_i$. If the server or i are honest and at least $(1-\alpha-\varepsilon/2)\kappa$ inputs are provided to the functionality, then it proceeds to compute the next-message as specified by the protocol, and sharing the output among the users in $\mathcal{P}_i$. If less than $(1-\alpha-\varepsilon/2)\kappa$ inputs are provided, the functionality sends $\perp$ to all of its users. If both the server and i are corrupted, then the adversary chooses the output the users receive from the functionality. In all cases, if a user obtain $\perp$ as the output from some NxtMsg_i, then it aborts. In Section 6.1 we present a protocol for implementing each call.

We further describe the output with respect to the PC rather than the users. The output of each user is then defined as the set of PCs containing it, and a share of the output of each such PC, in a $(1-\alpha-\varepsilon/2)\kappa$-out-of-$\kappa$ Shamir's secret sharing scheme.

[15] Although the PCs are sampled during the execution of the protocol, the next-message they compute are functionalities that are known in advance. Therefore, the hybrid functionality is well-defined.

Protocol 2: `CommitteeSetup`

Common inputs: All parties hold the security parameter 1^κ and the number of users n (held in binary by the users, and in unary by the server).

1. The parties execute protocol `PersonalCommittee`. Recall that each user i obtains a set $\{(j, \mathcal{P}_j)\}_{j:i\in\mathcal{P}_j}$ of size at most 3κ, and the server holds the set $\mathcal{L} = \{(i, \mathcal{P}_i)\}_i$ where each $\mathcal{P}_i$ is active. Additionally, the server is committed (via a vector commitment scheme) to a vector $\mathbf{v}$ whose i^{th} entry is $(i, \mathcal{P}_i)$.
2. Each PC $\mathcal{P}_i$ samples a set of neighbors $\mathcal{N}_i^{\mathtt{out}} \subseteq [n]$ of size κ uniformly at random, and sends it to server.
3. For each remaining $\mathcal{P}_i$, the server opens the commitment to every user in $j \in \mathcal{P}_i$ at position j. The user aborts from $\mathcal{P}_i$ if the verification of the decommitment fails.
4. The PC $\mathcal{P}_i$ then notifies each PC $\mathcal{P}_j$, where $j \in \mathcal{N}_i^{\mathtt{out}}$.
5. $\mathcal{P}_i$ sets $\mathcal{N}_i^{\mathtt{in}}$ to be the set of PCs that sampled it. It aborts if it was sampled by too many users, *i.e.*, $|\mathcal{N}_i^{\mathtt{in}}| > 3\kappa$.
6. Let $\mathcal{N}_i = \mathcal{N}_i^{\mathtt{out}} \cup \mathcal{N}_i^{\mathtt{in}}$ be the total set of neighbors of PC $\mathcal{P}_i$ (known to both the PC and the server).
7. Send $\mathcal{P}_i$ to each PC $\mathcal{P}_j$, where $j \in \mathcal{N}_i$.
8. For $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$ iterations, each personal committee $\mathcal{P}_i$ sends alive to every $\mathcal{P}_j$ where $j \in \mathcal{N}_i$. At any iteration, if $\mathcal{P}_i$ did not receive the message alive from any of its neighbors, it aborts.
9. Each honest $\mathcal{P}_i$ outputs the set of neighbors $\mathcal{N}_i$, and an honest server outputs the list $\mathcal{L} = \{(i, \mathcal{P}_i)\}_i$ of alive users and their corresponding PC., and the (undirected) graph $G = (\mathcal{L}, E)$, where $\{(i, \mathcal{P}_i), (j, \mathcal{P}_j)\} \in E$ if and only if $i \in \mathcal{N}_j$.

We next state the security properties of Protocol 2. Instead of describing it using the standard real vs. ideal paradigm, we formalize the security notion by describing an event, which captures the desired properties. We then show that the event occurs with overwhelming probability.

Definition 3.4 (Personal committees and small graph event)**.** *We define the* personal committees and small graph event*, denoted* PCSG*, as* $\mathsf{PCSG}_{\mathsf{hon}}$ *if the server is honest, and define it as* $\mathsf{PCSG}_{\mathsf{mal}}$ *if the server is malicious, where* $\mathsf{PCSG}_{\mathsf{hon}}$ *and* $\mathsf{PCSG}_{\mathsf{mal}}$ *are defined as follows.* $\mathsf{PCSG}_{\mathsf{hon}}$ *is the conjunction of the following events.*

- *All honest users are alive, and all their PCs are labeled active.*
- *Each alive user i (whether it is honest or malicious) holds a personal committee $\mathcal{P}_i$ of size κ and a set of neighbors $\mathcal{N}_i$ of size at most 4κ, where both are known to the server.*
- *For every alive user i (whether it is honest or malicious), its PC $\mathcal{P}_i$ contains at least $(1-\alpha-\varepsilon/2)\kappa > 7\kappa/8$ honest users.*
- *For every alive i, all of its neighbors in $\mathcal{N}_i$ hold $(i, \mathcal{P}_i)$.*
- *For every alive i and every alive $j \in \mathcal{N}_i$, all users in $\mathcal{P}_i$ hold $(j, \mathcal{P}_j)$.*

- *The graph over the PCs of alive users induced by the set of neighbors $\{\mathcal{N}_i\}_i$, has diameter at most $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$.*

$\mathsf{PCSG}_{\mathsf{mal}}$ *is the conjunction of the following events.*

- *At least $(1 - 2\alpha - \varepsilon)n = (7/8 - \alpha)n$ honest users are alive, and their PCs are labeled active.*
- *Each alive user i (whether it is honest or malicious), holds a personal committee $\mathcal{P}_i$ of size κ and a set of neighbors $\mathcal{N}_i$ of size at most 4κ, where both are known to the server.*
- *For every alive and honest i, its PC $\mathcal{P}_i$ contains at least $(1 - \alpha - \varepsilon/2)\kappa > 7\kappa/8$ honest users.*
- *For every alive i, all of its neighbors in $\mathcal{N}_i$ hold $(i, \mathcal{P}_i)$.*
- *For every alive i and every alive $j \in \mathcal{N}_i$, all users in $\mathcal{P}_i$ hold $(j, \mathcal{P}_j)$.*
- *The graph over the PCs of alive honest users induced by the set of neighbors $\{\mathcal{N}_i\}_i$, has diameter at most $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$.*

The next theorem asserts that at the end of Protocol 2, the event PCSG holds except with negligible probability.

Theorem 3.5. *Let $\alpha < 1/8$ be a constant (i.e., it does not depend on κ or the number of users n). Assume the existence of position binding vector commitment schemes. Let $\mathcal{A}$ be a* PPT *for Protocol 2 corrupting at most αn users, and which possibly corrupts the server as well. Then the event PCSG occurs except with negligible probability.*

Observe that by construction, if a PC contains at most $(\alpha + \varepsilon/2)\kappa$ corrupted users and it is labeled active at the end of the execution, then all of its users hold the neighbors of the user corresponding to the PC and PCs of the neighbors. Thus, Theorem 3.5 directly follows from the next two lemmata, which split (and simplify) the statement depending on whether the server is honest or corrupted.

Lemma 3.6. *Let $\alpha < 1/8$ be a constant and assume the existence of position binding vector commitment schemes. Let $\mathcal{A}$ be a* PPT *adversary for Protocol 2, corrupting a set $\mathcal{I} \subseteq [n]$ of at most αn users and which does* not *corrupt the server. Then the following hold.*

1. *Every personal committee (including those of malicious users that did not abort) will contain less than $\kappa/8$ corrupted users with overwhelming probability. That is, let $\mathcal{I}' \subseteq \mathcal{I}$ denote the set of malicious users that are labeled active by the server by the end of the protocol's execution. Then*
$$\Pr\left[\exists i \in \mathcal{H} \cup \mathcal{I}' : |\mathcal{P}_i \cap \mathcal{I}| \geq \left(\alpha + \frac{\varepsilon}{2}\right) \cdot \kappa\right] \leq n \cdot e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$
2. *All honest users remain alive at the end of the execution, except with negligible probability.*

Lemma 3.7. *Let $\alpha < 1/8$ be a constant and assume the existence of position binding vector commitment schemes. Let $\mathcal{A}$ be a* PPT *adversary for Protocol 2, corrupting a set $\mathcal{I} \subseteq [n]$ of at most αn users and which also corrupts the server. Let $\mathcal{L} \subseteq \mathcal{H}$ denote the set of honest users that did not abort the execution of the protocol. Assume that $\mathcal{L} \neq \emptyset$. Then the following hold.*

1. *$|\mathcal{L}| \geq (1 - 2\alpha - \varepsilon)n = (7/8 - \alpha)n$ except with negligible probability.*

2. *Every personal committee of an alive honest user in will contain less than $\kappa/8$ malicious users, except with negligible probability. Formally,*

$$\Pr\left[\exists i \in \mathcal{L} : |\mathcal{P}_i \cap \mathcal{I}| \geq \left(\alpha + \frac{\varepsilon}{2}\right) \cdot \kappa\right] \leq n \cdot e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$

In particular, the PCs of alive honest users are active at the end of the execution.

3. *The graph over the active honest PCs induced by the sets $\{\mathcal{N}_i\}_i$ (i.e., $\{\mathcal{P}_i, \mathcal{P}_j\}$ is an edge if and only if $i \in \mathcal{N}_j$ or $j \in \mathcal{N}_i$), has diameter at most $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$, except with negligible probability. Moreover, each user has at most 4κ neighbors.*

4. *For every alive honest user $i \in \mathcal{L}$, all its neighbors $j \in \mathcal{N}_i$ and the users in $\mathcal{P}_j$, hold $\mathcal{P}_i$ (as it was sent to them at Step 5c).*

We first prove Lemma 3.6, where the server is assumed to be honest.

Proof of Lemma 3.6. We first prove Item 1. First, by the binding property of the commitment scheme, any malicious user will decommit properly, except with negligible probability. Therefore, $\mathcal{P}_i$ will be uniformly random for all $i \in \mathcal{I}'$ except with negligible probability. Now, observe that for every $i \in \mathcal{H} \cup \mathcal{I}'$ it holds that

$$\mathbb{E}\left[\left|\mathcal{P}_i \cap \mathcal{I}'\right|\right] \leq |\mathcal{I}| \cdot \frac{\kappa}{n} \leq \alpha\kappa.$$

By Hoeffding's inequality (Fact 2.3), for every user $i \in \mathcal{H} \cup \mathcal{I}$, it holds that

$$\Pr\left[|\mathcal{P}_i \cap \mathcal{I}| \geq \left(\alpha + \frac{\varepsilon}{2}\right) \cdot \kappa\right] \leq \Pr\left[|\mathcal{P}_i \cap \mathcal{I}| - \mathbb{E}\left[|\mathcal{P}_i \cap \mathcal{I}|\right] \geq \frac{\varepsilon}{2} \cdot \kappa\right] \leq e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$

The claim now follows from the union bound.

We now show Item 2. Since the server is honest, all users will obtain a commitment to the same vector. Thus, all users remain if and only if no user will be sampled by too many other users (in Steps 5b and 8) and if no personal committee will be labeled inactive. The former holds with negligible probability and it directly follows from Chernoff's inequality (Fact 2.2) and the union bound. Then latter follows from Item 1 and the security of the broadcast protocol. □

We now prove Lemma 3.7, where the server is assumed to be malicious.

Proof. For simplicity, throughout the entire proof we assume the malicious server always opened the commitments properly. Furthermore, for an alive honest user $i \in \mathcal{L}$ we let $\mathbf{v}_i$ denote the vector to which the (malicious) server send a commitment to user i.[16]

Proof of Item 1. Assume that less than $(1 - 2\alpha - \varepsilon)n$ honest users are alive, and let $\mathcal{T}_i$ denote the set of users sampled by user i at Step 13. Then $\mathbb{E}\left[|\mathcal{T}_i \cap \mathcal{L}|\right] < (1 - 2\alpha - \varepsilon)\kappa$. Therefore, by Hoeffding's inequality (Fact 2.3)user i aborts at Step 13 except with probability at most

$$\Pr\left[|\mathcal{T}_i \cap \mathcal{L}| \geq \left(1 - 2\alpha - \frac{\varepsilon}{2}\right) \cdot \kappa\right] \leq \Pr\left[|\mathcal{T}_i \cap \mathcal{L}| - \mathbb{E}\left[|\mathcal{T}_i \cap \mathcal{L}|\right] \geq \frac{\varepsilon}{2} \cdot \kappa\right] \leq e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$

Item 1 now follows from the union bound.

[16] Note that the existence of $\mathbf{v}_i$ is only guaranteed from a computational perspective, namely, the computationally bounded malicious server cannot decommit to any other value. To formalize this argument, observe that if there exists another vector $\mathbf{v}'_i$ to which the server is able to decommit without user i aborting, then this directly translate to an attacker breaking the binding assumption of the vector commitment scheme.

Proof of Item 2. Assume towards contradiction that the inequality is false for some adversary $\mathcal{A}$ corrupting the server. Then there exists an honest user $i \in \mathcal{L}$ whose PC does not contain at least $(\alpha + \varepsilon/2)\kappa$ honest users, with probability at least $e^{-\varepsilon^2 \cdot \kappa/2}$. Consider the following thought experiment, where two parties, a user u and a server, interact. At the beginning of the interaction, the (malicious) server chooses two subsets $\mathcal{I}$ and $\mathcal{L}$, where $|\mathcal{I}| \leq \alpha n$. The server and user u agree on a random subset $\mathcal{P} \subseteq [n]$ of size κ, using the same interaction as in Step 1 of the protocol. We say that the server wins if it succeeds in forcing the output $\mathcal{P}$ to admit $|\mathcal{P} \cap \mathcal{I}| \geq (\alpha + \varepsilon/2)\kappa$. Similarly to the case where the server is honest (see Item 1 of Lemma 3.6), by Fact 2.3, for every malicious server (that decommits properly) it holds that the probability that the adversary wins is

$$\Pr\left[|\mathcal{P} \cap \mathcal{I}| \geq \left(\alpha + \frac{\varepsilon}{2}\right) \cdot \kappa\right] \leq e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$

We now show that using the above adversary $\mathcal{A}$ for the multiparty protocol, the server in the two-party experiment can achieve its goal with probability higher than the above quantity. The server (in the two-party setting) sets $\mathcal{I}$ to be the set of users corrupted by $\mathcal{A}$, and interacts with the user u the same as $\mathcal{A}$ interacts with user i, emulating in its head the entire multiparty protocol (except the interaction with i). Clearly, the server in the two-party protocol succeeds with too high probability, resulting in a contradiction.

Proof of Item 3. Consider the directed graph G over the users (and not the PCs), where the edges between honest alive users are defined via the sets $\{\mathcal{N}_i^{\mathtt{out}}\}_i$ (*i.e.,* there is an edge from i to j if and only if $j \in \mathcal{N}_i^{\mathtt{out}}$). Then at least $(1 - 2\alpha - \varepsilon/2)n > 3n/4$ of its vertices are users in $\mathcal{L}$. Now, by Item 2 there exists at least one alive honest PC. Therefore, by Lemma 3.2 applied for the graph G with $\mathsf{Bad} := [n] \setminus \mathcal{L}$, it follows that if one PC of an alive honest user aborted, then all such PCs would have aborted at Step 8, except with negligible probability. Thus, the PCs are all active, and the corresponding graph has diameter $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$, except with negligible probability.

Proof of Item 4. Fix $i \in \mathcal{L}$. We prove the stronger statement asserting that for every alive honest user $j \in \mathcal{L}$, the proper decommitment to c_j at position i returns $\mathcal{P}_i$. We show this in two steps. First, we show that for all $i \in \mathcal{L}$, it holds that the majority (rather than all) of alive honest users j are such that the proper decommitment of c_j at position i results in $\mathcal{P}_i$. In the second step, we conclude that the above statement holds for *all* users $j \in \mathcal{L}$.

Before presenting the formal proof, let us first provide an intuitive explanation. For the first step, assume towards contradiction that the majority of alive honest users j disagree with user i at position i in their committed vector. Then the set of users $\mathcal{S}_i$ to which user i received the commitments at Step 5d, on expectation will contain a majority of users j that disagree at position i, thus user i will sample many disagreeing users with overwhelming probability, causing it to abort. For the second step of the proof, if most users agree at position i, then the query set $\mathcal{Q}$ of any user that disagrees with the majority, will contain at least one of those in the majority with overwhelming probability, thus causing the user in the minority to abort.

We now formalize the above intuition. Let $\mathcal{L}_i := \{j \in \mathcal{L} : \mathbf{v}_j(i) = \mathcal{P}_i\}$ denote the set of all alive honest users j that agree with i at position i. We first show that $|\mathcal{L}_i| \geq (1/2 + \varepsilon/2)n = (9/16 - \alpha/2)n > n/2$. Assuming otherwise, it holds that $\mathbb{E}[\mathcal{S}_i \cap \mathcal{L}_i] < (1/2 + \varepsilon/2)\kappa$, hence by Fact 2.3 it follows that

$$\Pr\left[|\mathcal{S}_i \cap \mathcal{L}_i| \geq \left(\frac{1}{2} + \varepsilon\right) \cdot \kappa\right] \leq \Pr\left[|\mathcal{S}_i \cap \mathcal{L}_i| - \mathbb{E}\left[|\mathcal{S}_i \cap \mathcal{L}_i|\right] \geq \frac{\varepsilon}{2} \cdot \kappa\right] \leq e^{-\frac{\varepsilon^2}{2} \cdot \kappa}.$$

Thus, except with negligible probability, user i sampled less than

$$\left(\frac{1}{2}+\varepsilon\right)\kappa = \left(\frac{5}{8}-\alpha\right)\kappa < \left(\frac{15}{16}-\frac{3\alpha}{2}\right) = \left(1-2\alpha-\frac{\varepsilon}{2}\right)\kappa$$

users with consistent information. However, since we assume i did not abort, it follows that *all* honest users it sampled must be from $\mathcal{L}_i$, resulting in a contradiction. Therefore, the event $|\mathcal{L}_i| < (1/2+\varepsilon/2)n$ occurs with negligible probability.

To conclude the proof of Item 4, we next show that any user $j \notin \mathcal{L}_i$ aborts Step 6c, except with negligible probability. Indeed, the probability that j does not abort is at most

$$\Pr\left[\mathcal{Q}_j \cap \mathcal{L}_i = \emptyset\right] \leq \Pr\left[|\mathcal{L}_i| < (1/2+\varepsilon/2)n\right] + \Pr\left[\mathcal{Q}_j \cap \mathcal{L}_i = \emptyset \mid |\mathcal{L}_i| \geq (1/2+\varepsilon/2)n\right].$$

The above quantity is negligible since

$$\Pr\left[\mathcal{Q}_j \cap \mathcal{L}_i = \emptyset \mid |\mathcal{L}_i| \geq (1/2+\varepsilon/2)n\right] \leq \left(1-\frac{(1/2+\varepsilon/2)n}{n}\right)^{\kappa} < e^{-\frac{1+\varepsilon}{2}\cdot\kappa}.$$

□

4 Towards Reducing User Complexities: Committee Election

Similarly to [18, 20, 16], a key component of our construction is a committee election protocol. Our protocol follows ideas similar to Feige's lightest-bin protocol [35], but is adjusted to the special GMPC setting, where the users are polylogarithmic (in the number of users n) and communication be blocked by the server. These requirements make the process of jointly agreeing on a single committee quite challenging.

We show how the parties can perform this task assuming the PCSG event occurred, namely, the parties are replaced with personal committees and they hold together a graph with small diameter. Specifically, even if the server is malicious, then with overwhelming probability more than 7/8 of users in the elected committee are honest (note that this guarantee is not the same as in Feige's protocol). In addition, the committee will be of size $\Theta(\kappa) = \omega(\log n)$.

We next formally define the committee election functionality f_{elect}. To simplify the presentation, we split the description into two cases, depending on whether the server is honest or malicious. Note that unlike in Feige's protocols, here a malicious server can abort some honest users, hence the users do not know the number of remaining users.

Ideal world for electing a committee assuming an honest server. We next describe the interaction in the ideal world of f_{elect} assuming the server is honest. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, and which does *not* corrupt the server.

Inputs: All parties hold the security parameter 1^κ and the number of users (held in unary by the server and in binary by the users). The adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Adversary aborts some malicious users: The adversary sends to T a set $\mathcal{L} \subseteq [n]$ of alive users, where $\mathcal{H} \subseteq \mathcal{L}$.

Trusted party chooses a committee: The trusted party does the following.

1. Set $b = \lceil |\mathcal{L}|/\kappa \rceil$ to be the number of bins.
2. For every alive (possibly malicious) user $i \in \mathcal{L}$, sample a bin $x_i \in [b]$ independently and uniformly at random.
3. Set $\mathcal{C}$ to be the lightest bin and send it to all parties (note that $|\mathcal{C}| \leq \kappa$ by the pigeonhole principle).

Output: Each honest user and the server output whatever it received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

Ideal world for electing a committee assuming a malicious server. We next describe the interaction in the ideal world of f_{elect} assuming the server is corrupted. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, which also corrupts the server.

Inputs: All parties hold the security parameter 1^κ and the number of users (held in unary by the server and in binary by the users). The adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Adversary aborts some users and choose number of bins: The adversary either sends abort to T or sends it a set $\mathcal{L} \subseteq [n]$ of alive (possibly malicious) users of size at least $|\mathcal{L}| \geq 7n/8$, and the number of bins b, where $b \geq \lceil 7n/8\kappa \rceil$.

Trusted party splits users into bins: If T received abort, then it sends $\perp$ to all parties and halts. Otherwise, for every alive honest user $i \in \mathcal{L} \cap \mathcal{H}$, the trusted party samples a bin $x_i \in [b]$ independently and uniformly at random. It then sends $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$ to the adversary.

The adversary chooses a bin and adds corrupted users: For every $j \in [b]$, let $\mathcal{B}_j = \{i \in \mathcal{L} \cap \mathcal{H} : x_i = j\}$ denote the set of honest users that sampled the j^{th} bin. The adversary sends back either abort or a set $\mathcal{C}$ of size at most κ, such that there exists $j \in [b]$ for which it holds that $\mathcal{B}_j \subseteq \mathcal{C}$ and $\mathcal{C} \setminus \mathcal{B}_j \subseteq \mathcal{I}$ (note that such a j always exists if the server is honest, hence, when such a j does not exist then it implies the server must behaved maliciously).

The trusted party sends outputs: If T received abort it sends $\perp$ to all parties and halts. Otherwise, it sends $\mathcal{C}$ to all users in $\mathcal{L}$ and sends $\perp$ to all other users.

Output: Each honest user outputs whatever it received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

We prove that conditioned on the event PCSG, the parties can compute f_{elect} securely. Formally, we prove the following.

Theorem 4.1. *Let $\alpha < 1/8$ be a constant (i.e., it does not depend on κ or the number of users n). Then there exists an n-user GMPC protocol computing f_{elect}, such that conditioned on the event PCSG, the protocol is αn-secure.*

The proof of Theorem 4.1 is presented in Section 4.1. Then, in Section 4.2 we present a protocol for electing many committees, such that each committee will be assigned to one user. This will later be used in Sections 5 and 6 for the construction of secure protocols for general GMPC functionalities. Before proving the theorem, we first show that the elected committee contains a vast majority of honest users. In fact, we take into account the number of users stated to be alive by the adversary. This will later allow us to claim that users in the committee can securely compute any functionality (see Section 6.1). Recall that we let $\varepsilon = 1/8 - \alpha > 0$.

Lemma 4.2. *Let $\alpha < 1/8$, let $\mathcal{A}$ be an adversary corrupting αn of the users (which possibly corrupts the server), and let $\beta = |([n] \setminus \mathcal{L}) \cup \mathcal{I}|/n$ denote the fraction of users that are either malicious or not alive, among the set of all users. Then in an execution of f_{elect} holds that if the parties output a committee $\mathcal{C}$, then*

$$\Pr\left[|\mathcal{C} \setminus \mathcal{I}| < (1 - \beta - \varepsilon/2) \cdot \kappa\right] < n \cdot e^{-\frac{\varepsilon^2}{8} \cdot \kappa}.$$

In particular, $\mathcal{C}$ contains strictly more than 3/4 fraction of honest users, except with negligible probability. Moreover, if the server is honest then all honest users always output $\mathcal{C}$.

The proof follows from an immediate application of Lemma 2.21. To see why the fraction of honest users in $\mathcal{C}$ is more than 3/4, observe that

$$\beta = \frac{|([n] \setminus \mathcal{L}) \cup \mathcal{I}|}{n} \leq \frac{1}{8} + \alpha < \frac{1}{4}.$$

4.1 Committee Election Protocol

We are now ready to introduce our protocol for electing a committee of size κ. Given the event PCSG, the protocol for electing a committee proceeds as follows. First, the server will send to all personal committees the number of remaining users. Each PC then checks with its neighbors if they received the same number. For $O(\log n/\log \kappa)$, each PC notify its neighbors that it did not abort.

The PCs then execute the following variant of Feige's protocol: each PC randomly choose a bin and send it to the server. The server will send back the set of users $\mathcal{C}$ who chose the lightest bin alongside the index of this bin. Each PC $\mathcal{P}_i$ then verifies that $i \in \mathcal{C}$ if $\mathcal{P}_i$ chose the bin, and $i \notin \mathcal{C}$ otherwise. Additionally, the PCs will check that $\mathcal{C}$ is not larger than κ. Finally, each PC will ask if its neighbors on the graph are still active for $O(\log n/\log \kappa)$ steps.

Similarly to Protocol 2, we describe the protocol in a hybrid world, where the hybrid functionalities compute the "next-message" function of each PC.We also abuse notions and describe the protocol as if each PC is a single party. Formally, whenever we say that the server sends a message to $\mathcal{P}_i$, it means that it (supposedly) shares it among the users of $\mathcal{P}_i$. Similarly, when we say that $\mathcal{P}_i$ sends a message to the server, it means that every user in $\mathcal{P}_i$ sends its share of the message to the server. Finally, whenever a PC $\mathcal{P}_i$ sends a message msg to another PC $\mathcal{P}_j$, it means that each user in $\mathcal{P}_i$ shares its share of msg among the users in $\mathcal{P}_j$. We let $\mathsf{NxtMsg} = \{\mathsf{NxtMsg}_i\}_{i=1}^n$ and describe the protocol in the NxtMsg-hybrid model. To obtain a protocol in the real world (*i.e.,* without

ideal access to NxtMsg), the parties will implement each call via the protocol from Section 6.1.

Protocol 3: `Committee`

Common inputs: All parties hold the security parameter 1^κ and the number of users n (held in binary by the users, and in unary by the server).
Event assumption: We assume that the event PCSG occurred. Recall that an honest server holds the list $\mathcal{L} = \{(i, \mathcal{P}_i)\}_i$ of all alive users and their PCs. Additionally, it holds the graph $G = (\mathcal{L}, E)$ induced by the neighbors of each alive user. The neighbors $\mathcal{N}_i$ of users i is held by user i and all users in its personal committee $\mathcal{P}_i$.

The users agree on the number of alive users:

1. The server sends to all PCs the number of alive users. Let n' denote this value.
2. Each PC $\mathcal{P}_i$ sends n' to all of its neighbors, *i.e.,* to all $\mathcal{P}_j$ where $j \in \mathcal{N}_i$.
3. A PC aborts if it received a value from one of its neighbors that differs from the value it received from the server.
4. For $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$ iterations, each personal committee $\mathcal{P}_i$ sends alive to every $\mathcal{P}_j$ where $j \in \mathcal{N}_i$. At any iteration, if $\mathcal{P}_i$ did not receive the message alive from any of its neighbors, it aborts.

The parties run a variant of Feige's protocol:

1. Let $b = \lceil n'/\kappa \rceil$ denote the number of bins. Each PC $\mathcal{P}_i$ samples a bin $x_i \in [b]$ uniformly at random and sends it to the server.
2. The server:
   (a) Let $x = \mathsf{minority}(x_1, \ldots, x_{|\mathcal{V}|})$.
   (b) Let $\mathcal{C} = \{i \in \mathcal{V} : x_i = x\}$.
   (c) Send $\mathcal{C}$ and x to all personal committees.
3. Each personal committee $\mathcal{P}_i$ sends $\mathcal{C}$ and x to each of its neighbors, *i.e.,* to every $\mathcal{P}_j$ where $j \in \mathcal{N}_i$.
4. A personal committee $\mathcal{P}_i$ sends $\perp$ to every neighbor $\mathcal{P}_j$ where $j \in \mathcal{N}_i$ and aborts if one of the following holds.
   - It received different $\mathcal{C}$ or x from one of its neighbors, or
   - $|\mathcal{C}| > \kappa$, or
   - $x_i = x$ and $i \notin \mathcal{C}$, or
   - $x_i \neq x$ and $i \in \mathcal{C}$.
5. For $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$ iterations, each personal committee $\mathcal{P}_i$ sends alive to every $\mathcal{P}_j$ where $j \in \mathcal{N}_i$. At any iteration, if $\mathcal{P}_i$ did not receive the message alive from any of its neighbors, it aborts.
6. Each user outputs $\mathcal{C}$.

Clearly, since we assume PCSG to occur, no user is in more than 3κ PCs, thus the running time of all users is polylogarithmic in n. The next lemma states that the protocol securely computes

f_{elect}. In Section 6.1, we show how to implement the NxtMsg hybrid functionalities, with security under parallel composition. Together with the composition theorem, this proves Theorem 4.1.

Lemma 4.3. *Let $\alpha < 1/8$ be a constant. Then protocol* `Committee` *computes f_{elect} with αn-security in the* NxtMsg*-hybrid model, conditioned on the event* PCSG*.*

Proof. Fix an adversary $\mathcal{A}$ corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, of size at most $|\mathcal{I}| \leq \alpha n$. We separate the proof into two cases, depending on whether the server is honest or corrupted.

Honest server. Since PCSG is assumed to have occurred, it follows that all PCs honest users are active, and all PCs (whether of malicious or honest users) contain at least $(1 - \alpha - \varepsilon/2)\kappa > 7\kappa/8$ honest users. Therefore, the adversary cannot prevent any of the calls to each NxtMsg_i functionality to be delayed. Furthermore, at any round of the protocol, the adversary sees at most $(\alpha + \varepsilon/2)\kappa < \kappa/8$ shares from each PC. Thus, its view is comprised of only random messages sampled uniformly at random and independently.

Malicious server. We next define the simulator $\mathsf{Sim}_{\mathcal{A}}$. To simplify the presentation, we do not concern ourselves with the view of malicious users in honest PCs, similarly to the case where the server was honest, they correspond to shares that reveal no information to $\mathcal{A}$. Furthermore, we assume the server does not block any message sent from one honest PC to another, as this will immediately cause all honest users to abort. The simulator does as follows.

1. **Simulate "the users agree on the number of alive users":**
   (a) Query the adversary $\mathcal{A}$ for the number of alive users that it sends to the honest PCs.[17] If two of them differ or the number is less than $7n/8$, then send abort to the trusted party T, output whatever $\mathcal{A}$ outputs, and halt.
   (b) Otherwise, let n' denote the number of alive users as was sent by the server.
   (c) Query $\mathcal{A}$ for the messages that each malicious PC sends to its neighbors (recall that messages sent to other PCs are considered DDoS attacks, and where handled in Section 2.2). If any of the message differ from n' during the simulation of Step 3, or differ from alive during the simulation of Step 4, then send abort to T, output whatever $\mathcal{A}$ outputs, and halt.
   (d) Otherwise, send to T the set $\mathcal{L}$ of alive users as given by the event PCSG (note that it could be the case that $n' \neq |\mathcal{L}|$), and the number of bins $b = \lceil n'/\kappa \rceil$.
   (e) The simulator obtains from the trusted party the bins that each alive honest user sampled. That is, it receives $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$, where $x_i \in [b]$.
2. **Simulate "the parties run a variant of Feige's protocol":**
   (a) Send the collection of choices $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$ to $\mathcal{A}$.
   (b) The adversary replies with a set C and a bin-index x for every alive honest PC. If one of the following occur, then send abort to T, output whatever $\mathcal{A}$ outputs, and halt.
      - Two PCs received different messages.

[17] Formally it is given as shares, on for each party. However, it is clear that the simulator can reconstruct the message.

- $|\mathcal{C}| > \kappa$.
- There exists $i \in \mathcal{L} \cap \mathcal{H}$ such that either $x_i = x$ and $i \notin \mathcal{C}$, or $x_i \neq x$ and $i \in \mathcal{C}$.

(c) Query the adversary for the message that the malicious PCs sends during Step 5. If any of them is not alive, then send abort to T, output whatever $\mathcal{A}$ outputs, and halt.

(d) Otherwise, send $\mathcal{C}$ to T, output whatever $\mathcal{A}$ outputs, and halt.

We claim that real (hybrid) world is identically distributed as the ideal world. Clearly, the messages the adversary sees in the real world are identically distributed as the messages it receives from the simulator in the ideal world. In particular, it responds with the same messages. Moreover, by the assumption that the event PCSG occurred, it follows that if any PC does not send the message alive in the protocol at Steps 4 and 5, then all users abort. By the construction of the simulator, this occurs in the real world with exactly the same probability as in the ideal world. Finally, observe the committee $\mathcal{C}$ send by the simulator in the ideal world satisfy the constrains it must abide by, namely, it is of size at most κ, and there exists $j \in [b]$ for which it holds that $\mathcal{B}_j \subseteq \mathcal{C}$ and $\mathcal{C} \setminus \mathcal{B}_j \subseteq \mathcal{I}$, where $\mathcal{B}_j = \{i \in \mathcal{L} \cap \mathcal{H} : x_i = j\}$ is the j^{th} bin. □

4.2 From a Single Committee to Many Committees

Recall that our final goal is to construct a secure protocol for computing an arbitrary function over the users' inputs. The first step towards constructing such a protocol, is to elect *many committees*, where each will effectively act as an honest party, replacing one user. Once the committees are elected, we let each of them hold an input of one party, and then run a secure protocol with the server (see Sections 5 and 6). We show how to securely elect $n + 1$ committees so that each committee contains sufficiently many honest users, and can thus be treated as an honest party.

We next formally describe the functionality, denoted f_{many}. Roughly, the first committee sampled by the functionality, is sampled similarly to f_{elect}, while the other are sampled honestly regardless of whether the server is corrupted or not. The functionality then sends a different committee to each user. Additionally, the users holds the committees in a "tree-like" structure, that is, all users in one committee will hold all users in the corresponding children and parent committees.

Ideal world for electing many committees assuming an honest server. We next describe the interaction in the ideal world of f_{many} assuming the server is honest. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, and which does *not* corrupt the server.

Inputs: All parties hold the security parameter 1^κ and the number of users (held in unary by the server and in binary by the users). The adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Adversary aborts some malicious users: The adversary sends to T a set $\mathcal{L} \subseteq [n]$ of alive users, where $\mathcal{H} \subseteq \mathcal{L}$.

Trusted party samples committees: The trusted party does the following.

1. Set $b = \lceil |\mathcal{L}|/\kappa \rceil$ to be the number of bins.
2. For every alive (possibly malicious) user $i \in \mathcal{L}$, sample a bin $x_i \in [b]$ independently and uniformly at random.
3. Set $\mathcal{C}_0$ to be the lightest bin (note that $|\mathcal{C}_0| \leq \kappa$ by the pigeonhole principle).

4. For every $i \in [n]$, sample a committee $\mathcal{C}_i \subseteq \mathcal{L}$ of size κ uniformly at random.
5. Send $\{\mathcal{C}_i\}_{i=0}^{n}$ to the server, and for every user $i \in [n]$, send $\mathcal{C}_i$ to user i, and to all users in $\mathcal{C}_{2i+1}$ and $\mathcal{C}_{2i+2}$ (assuming $2i+1, 2i+2 \in [n]$) and vice versa.

Output: Each honest user and the server output whatever they received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

Ideal world for electing many committees assuming a malicious server. We next describe the interaction in the ideal world of f_{many} assuming the server is corrupted. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, which also corrupts the server.

Inputs: All parties hold the security parameter 1^κ and the number of users (held in unary by the server and in binary by the users). The adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Adversary aborts some users and choose number of bins: The adversary either sends abort to T or sends it a set $\mathcal{L} \subseteq [n]$ of alive (possibly malicious) users of size at least $|\mathcal{L}| \geq 7n/8$, and the number of bins b, where $b \geq \lceil 7n/8\kappa \rceil$.

Trusted party splits users into bins: If T received abort, then it sends $\perp$ to all parties and halts. Otherwise, for every alive honest user $i \in \mathcal{L} \cap \mathcal{H}$, the trusted party samples a bin $x_i \in [b]$ independently and uniformly at random. It then sends $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$ to the adversary.

The adversary chooses a bin and adds corrupted users: For every $j \in [b]$, let $\mathcal{B}_j = \{i \in \mathcal{L} \cap \mathcal{H} : x_i = j\}$ denote the set of honest users that sampled the j^{th} bin. The adversary sends back either abort or a set $\mathcal{C}_0$ of size at most κ, such that there exists $j \in [b]$ for which it holds that $\mathcal{B}_j \subseteq \mathcal{C}_0$ and $\mathcal{C}_0 \setminus \mathcal{B}_j \subseteq \mathcal{I}$ (note that such a j always exists if the server is honest, hence, when such a j does not exist then it implies the server must behaved maliciously).

Trusted party samples other committees and sends output: If T received abort it sends $\perp$ to all parties and halts. Otherwise, it does the following.

1. For every $i \in [n]$, sample a committee $\mathcal{C}_i \subseteq [n]$ of size κ uniformly at random.
2. Send $\{\mathcal{C}_i\}_{i=1}^{n}$ to the server.

The adversary chooses whether to end to computation or not: The adversary replies with either abort or a new set of alive $\mathcal{L}' \subseteq [n]$ of users of size at least $|\mathcal{L}'| \geq 7n/8$. In the former case, T send $\perp$ to all parties and halt. In the latter case, for every alive user $i \in \mathcal{L}'$, send $\mathcal{C}_i$ to user i, and to all (alive) users in $\mathcal{C}_{2i+1}$ and $\mathcal{C}_{2i+2}$ (assuming $2i+1, 2i+2 \in [n]$) and vice versa.

Output: Each honest user outputs whatever it received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

We prove the following result, asserting there exists a secure protocol for computing f_{many} assuming the event PCSG occurred.

Lemma 4.4. *Let $\alpha < 1/8$ be a constant. Then there exists an n-user GMPC protocol computing f_{many}, such that conditioned on the event PCSG, the protocol is αn-secure.*

We now briefly explain the idea for sampling many committees. First, the parties will generate a global common committee $\mathcal{C}_0$ known to all parties, using Protocol 3. Then, $\mathcal{C}_0$ constructs a tree of $n+1$ committees, with itself as the root. This is done by having each node (committee) sample two children (*i.e.,* two new committees). Doing this for $\log n$ iterations results in the foramtion of $n+1$ committees. By Lemma 4.2 and the union bound, with overwhelming probability, all of these committees will have a vast majority of honest users. Hence, they can be effectively viewed as honest parties. Moreover, it is highly unlikely that any user will appear in more then polylog(n) committees, hence efficiency is maintained.

After the tree of (honest) committees is completed, each committee is assigned to one user, that is, committee $\mathcal{C}_i$ will be assigned to user i. There are two main difficulties in doing so. The first one is that the committee must prove to the user it was sampled honestly during the creation of the tree. The second issue is that the committee cannot distinguish an aborting user from one that is blocked by the server, hence the input could be lost.

To overcome to the first issue, we let $\mathcal{C}_0$ sample signature keys, and pass them to all committees in the tree via its children. Thus, each committee can prove honesty by signing a random message. For a user to be able to verify the signed message, $\mathcal{C}_0$ must deliver it the public key. To do so efficiently, we use the assumption that the event PCSG occurred. Recall that the event asserts that each user has an associated personal committees replacing it, and furthermore, the PCs agree on a communication graph of diameter $O(\log n/\log\kappa)$, in the sense that each PC knows its set of neighbors.

With this in mind, we let all users in $\mathcal{C}_0$ send the public key to their respective PCs, as well as to the server. Each PC then compares the public key to the one sent to the server. If they are the same, then it passes it to all other PCs along the communication graph. Otherwise, it sends $\perp$ to all its neighbors (without aborting). If there is no value that appears more than (roughly) $3/4$ of the time, then the users know the server is malicious. Finally, all PCs send the public key to their associated user (assuming the user is still active). By the small diameter property of the graph (assumed to hold by the PCSG event), it follows that in $O(\log n/\log\kappa)$ rounds all users, will obtain the public key.

As for the second issue, we let the committees request a response from its assigned user, and simply aggregate along the tree responds. If too many users did not respond, then they necessarily know the server is corrupted and they all abort (note that this holds true regardless of how many users aborted previously).

We now formalize the above process. Similarly to protocols `CommitteeSetup` and `Committee`, the protocol is described in a hybrid world, denoted NxtMsg, where the hybrid functionalities compute the next-message function of each committee and each PC. The NxtMsg functionalities that correspond to the committees, are defined similarly to those that correspond to PCs, with the difference being the sharing threshold of the sharing scheme used for the inputs and outputs of the participating users. Specifically, the shares are generated in a $(1-\beta-\varepsilon/2)\kappa$-out-of-$\kappa$ Shamir's secret sharing scheme, where $\beta = |\mathcal{L}\cap\mathcal{I}|/|\mathcal{L}|$ is the fraction of malicious users among the alive users.[18] In Section 6.1, we show how to implemented these functionalities.

If at any point during the computation, the server blocks the communication between two committees (*i.e.,* the users in the receiving committee do not hold sufficiently many shares), then the

[18]Formally, the functionality cannot obtain $\mathcal{L}$, and hence it cannot compute β. However, the fraction of alive users can be estimated by sampling κ random users and requesting a response. To simplify the presentation, we assume the functionality receives β. We stress that the implementation provided in Section 6.1, does not assume this.

receiving committee aborts. If the parent (global) committee aborts, it sends this information to κ random PCs who passes this information through the communication graph. To simplify the presentation of the protocol, we will not concern ourselves with these issues. We are now ready to describe the protocol. It is presented in the $\{\mathsf{NxtMsg}, f_{\mathsf{elect}}\}$-hybrid model.

Protocol 4: `ManyCommittees`

Common inputs: All parties hold the security parameter 1^κ and the number of users n (held in binary by the users, and in unary by the server).

Event assumption: We assume that the event PCSG occurred. Recall that an honest server holds the list $\mathcal{L} = \{(i, \mathcal{P}_i)\}_i$ of all alive users and their PCs. Additionally, it holds the graph $G = (\mathcal{L}, E)$ induced by the neighbors of each alive user. The neighbors $\mathcal{N}_i$ of users i is held by user i and all users in its personal committee $\mathcal{P}_i$.

Sample first committee and send public verification key to all users:

1. The parties call f_{elect}. If they receive $\perp$, the abort. Otherwise, let $\mathcal{C}_0$ denote the sampled committee.
2. $\mathcal{C}_0$ prepares signature keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$. It then sends pk to the server and to all users $i \in \mathcal{C}_0$.
3. For every $i \in \mathcal{C}_0$, the server sends pk to PC $\mathcal{P}_i$.
4. Each PC $\mathcal{P}_i$ sets pk_i to be an empty array of length $|\mathcal{C}_0|$, indexed with the users in $\mathcal{C}_0$. If $\mathcal{P}_i$ received the same value pk from user i and from the server, it sets $\mathsf{pk}_i(i) = \mathsf{pk}$. Otherwise, it sets $\mathsf{pk}_i(i) = \perp$. It then sends pk_i to all its neighbors $\mathcal{P}_j$, *i.e.,* where $j \in \mathcal{N}_i$.
5. For $\lceil \frac{\log(n/4)}{\log(\kappa/4)} \rceil + 1$ iterations, each PC updates its array according to values it received from its neighbors, and sends the updated array to its neighbors.
6. If there exists a value in pk_i that appears at least $(3/4 - \varepsilon/2)\kappa$ times (where $\varepsilon = 1/8 - \alpha$), then $\mathcal{P}_i$ sets it to be the public key and sends this value to users i. Note that by Lemma 4.2, with overwhelming probability, this value will be the public key pk that was sampled by $\mathcal{C}_0$. To alleviate notations, we will use pk to denote this value.

Sample the rest of the committees:

1. For $i = 0$ to $\lceil \log n \rceil - 1$:
   (a) Committee $\mathcal{C}_i$ samples two more committees $\mathcal{C}_{2i+1}, \mathcal{C}_{2i+2} \subseteq [n]$, each of size κ, independently and uniformly at random, and sends them the secret signature key sk (held shared among the users in the committee) and its identity $\mathcal{C}_i$.
   (b) For $b \in \{1, 2\}$ such that $2i + b \in [n]$:
      i. Committee $\mathcal{C}_{2i+b}$ sends to user $2i + b$ the pair $(r_{2i+b}, \sigma_{2i+b})$, where $r_{2i+b} \leftarrow \{0,1\}^\kappa$ is a random string and $\sigma_{2i+b} \leftarrow \mathsf{Sig.Sign}_{\mathsf{sk}}(r_{2i+b})$ is its signature.
      ii. If $\mathsf{Sig.Verify}_{\mathsf{pk}}(r_{2i+b}, \sigma_{2i+b}) = 1$, then user $2i+b$ responds to $\mathcal{C}_{2i+b}$ with the message alive. Otherwise, if it never received a signed message it aborts.
2. The committees aggregate the total number of alive message they received. Let ℓ denote the final result, held by $\mathcal{C}_0$.
3. If $\ell < (1 - \alpha - \varepsilon/2)n$ then each committee notifies its user to abort.
4. Otherwise, an honest user i outputs $\mathcal{C}_i$, and for every $j \in \{0, \ldots, n\}$, where $i \in \mathcal{C}_j$, the user outputs $\mathcal{C}_{2j+1}$, $\mathcal{C}_{2j+2}$, and $\mathcal{C}_{\lfloor (j-1)/2 \rfloor}$ as well (assuming the indexes are in $\{0, \ldots, n\}$). An honest server outputs $\{\mathcal{C}_i\}_{i=0}^n$.

By Fact 2.2, no user will be in more than 3κ committees except with negligible probability, hence the protocol maintains efficiency. As for security, the intuition is that by Lemma 4.2 and the union bound, all committees sampled during the execution of the protocol will contain a vast honest majority with overwhelming probability. In particular, the number of honest users is sufficient so that the calls to NxtMsg will be be executed honestly. Finally, since the adversary cannot forge a signed random message, no user will appear in a maliciously fabricated committee. We next formalize this intuition.

Lemma 4.5. *Let $\alpha < 1/8$ be a constant. Then protocol* `ManyCommittees` *computes f_{many} with αn-security in the $\{\mathsf{NxtMsg}, f_{\mathsf{elect}}\}$-hybrid model, conditioned on the event PCSG.*

Proof. Fix an adversary $\mathcal{A}$ corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, of size at most $|\mathcal{I}| \leq \alpha n$. The case where the server is honest follows similar reasoning to the proof of Lemma 4.3 and is therefore omitted.

We next assume the adversary corrupts the server. We assume without loss of generality that the adversary did not send abort to the functionality f_{elect}, as this causes all users to abort. Let $\mathcal{L} \subseteq [n]$ denote the set of alive users the adversary sent to the functionality f_{elect} at the start of the protocol (where $|\mathcal{L}| \geq 7n/8$). Observe that by Lemma 4.2 and the union bound, it follows that

$$|\mathcal{C}_i \setminus \mathcal{I}| \geq (\beta + \varepsilon/2)\kappa$$

for all $i \in \{0, \ldots, n\}$, except with negligible probability, where $\beta = |([n] \setminus \mathcal{L}) \cup \mathcal{I}|/n$ is the fraction of users that are either not alive or malicious, among the total number of users. In the rest of the proof we condition on this event occurring.

We next define the simulator $\mathsf{Sim}_{\mathcal{A}}$. Similarly to the proof of Lemma 4.3, to simplify the presentation, we do not concern ourselves with the view of malicious users in honest PCs or in committees, as they correspond to random share. Furthermore, we assume the server does not block any message sent from one honest PC to another, as this will immediately cause all honest users to abort. The simulator does the following.

1. **Simulate "Sample first committee and send public verification key to all users":**
   (a) Simulate the interaction between $\mathcal{A}$ and the functionality f_{elect}:
      i. Query $\mathcal{A}$ for the set $\mathcal{L}$ of alive users, and the number of bins b. Send these to the trusted party, and obtain $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$.
      ii. Send $\{(i, x_i)\}_{i \in \mathcal{L} \cap \mathcal{H}}$ to the adversary $\mathcal{A}$, who responds with a set $\mathcal{C}_0$ of size at most κ, such that $\mathcal{B}_j \subseteq \mathcal{C}_0$ and $\mathcal{C}_0 \setminus \mathcal{B}_j \subseteq \mathcal{I}$ for some $j \in [b]$, where $\mathcal{B}_j = \{i \in \mathcal{L} \cap \mathcal{H} : x_i = j\}$, is the j^{th} bin.
   (b) Generate signature keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$ and send pk to the adversary.
   (c) For every honest $i \in \mathcal{C}_0$, the adversary sends a value pk'_i to the PC $\mathcal{P}_i$.
   (d) If less than $(3/4 - \varepsilon/2)\kappa$ of them received pk from $\mathcal{A}$, then send abort to the trusted party T, output whatever $\mathcal{A}$ outputs, and halt (note that simulating the view of $\mathcal{A}$ during Step 5 is redundant since the adversary can compute it by itself).
2. **Simulate "Sample the rest of the committees":**
   (a) Send $\mathcal{C}_0$ to the trusted party, who replies with committees $\{\mathcal{C}_i\}_{i=1}^n$, where $\mathcal{C}_i \subseteq [n]$ is of size κ for all $i \in [n]$.

(b) For $i = 0$ to $\lceil \log n \rceil - 1$:

i. Send $\mathcal{C}_{2i+1}$ and $\mathcal{C}_{2i+2}$ to $\mathcal{A}$ (and shares of sk).

ii. For $b \in \{1, 2\}$ such that $2i + b \in \mathcal{I}$ is corrupted, send to $\mathcal{A}$ the pair $(r_{2i+b}, \sigma_{2i+b})$, where $r_{2i+b} \leftarrow \{0,1\}^\kappa$ is a random string and $\sigma_{2i+b} \leftarrow \mathsf{Sig.Sign}_{\mathsf{sk}}(r_{2i+b})$ is its signature.

iii. For $b \in \{1, 2\}$ such that $2i + b \in \mathcal{H}$ is honest, the adversary can either block the message alive that user $2i + b$ is suppose to send, or not.[19]

(c) If the server blocked at least $7n/8$ users during Step 2(b)iii, then send abort to T, output whatever $\mathcal{A}$ outputs, and halt.

(d) Otherwise send the set $\mathcal{L}' \subseteq [n]$ of all users that are either malicious or honest that were not blocked, to T, output whatever $\mathcal{A}$ outputs, and halt.

Clearly, the view of $\mathcal{A}$ in both worlds are identically distributed. In particular, the adversary's replies are identical. We next show that, conditioned on the view of $\mathcal{A}$ being the same in both worlds, the output of the honest users in both worlds is statistically close.

Indeed, in the ideal world, the committees are sampled honestly by the trusted party. Conversely, in the real world, they are sampled honestly if and only if all of them contain sufficiently many honest users, which holds due to Lemma 4.2 except with negligible probability. Additionally, in the ideal world no honest user will output a committee fabricated by the adversary. In the real world, such an event can occur if and only if the adversary forged a signature of a κ-length random message. By the security of the signature scheme, the adversary will be able to do this with only a negligible probability. □

Remark 4.6. *Recall that when we formally defined the GMPC model in Section 2.2, we noted the adversary can perform a DDoS attack, even without corrupting the server. We proposed two solutions to this issue, where one of them was a combinatorial solution, that in general could allow the adversary to cause a small fraction of honest users to abort even when the server is honest.*

Observe that given these committees, they parties can overcome this issue, and have the committees represent honest users that aborted during the election of the first committee. To do so, the users that previously aborted, can obtain the public signature keys by requesting it from κ randomly sampled PCs and comparing the information. Since the server is honest, the information will always be consistent (since all PC have more than 7/8 fraction honest users in them), hence the users will not abort. Furthermore, the attacker cannot perform a DDoS attack at this stage, since the server knows which users suppose to interact with each other, hence it can block the malicious users in case they attack.

5 Server's Proof of Correctness of Computation

We present a protocol allowing a server to compute an arbitrary function over the users' inputs, while proving to the users that the computation was done correctly. For the entire section, we assume that *all users are honest*, while only the server might be corrupted. We further assume the users are connected via a binary tree network, with the root of the tree known to all parties. These

[19] Observe that the simulator can obtain the set of blocked users from the malicious server, since at the beginning of every round, the server expects to receive the set of pairs (i, j), where user i sends a message to user j.

assumption are then removed in Section 6 by electing many committees using the protocol from Section 4, and letting each committee replace a user.

Before presenting the protocol in Section 5.2, we present in Section 5.1 a protocol that allows the users to verify a certain computation by the server was done properly.

5.1 Verifying the Computation

We present a protocol that allows a prover to prove a P statement[20] to $n+1$ (honest) verifiers, so that a cheating prover is caught with overwhelming probability. The prover holds an input x of length $\tilde{O}(n)$, and each verifier holds a single polylog(n)-sized substring of x. We assume the verifiers run in time that is polylogarithmic in the number of users and polynomial in the security parameters. Furthermore, the verifiers are connected to each other via a binary tree network.

In the construction of the protocol we use *probabilistically checkable proofs of proximity* (PCPP) [11, 32]. Roughly speaking, a PCPP allows a verifier to be convinced that the input is close to being in a language L. In more details, the verifier has an explicit input x and an implicit input y given as an oracle. The verifier accepts with high probability if y is close to some y' such that $(x, y') \in L$.

We now present the protocol. We first describe its setting. There is a single verifier V, n helping parties $\mathsf{H}_1, \ldots, \mathsf{H}_n$, and a single prover P. The prover is connected to every other party via a secure point-to-point channel, and the verifier and the helpers are connected via a binary tree network. Specifically, the root of the tree is V whose identity is known to all parties, and is connected to both H_1 and H_2. Additionally, for every $i \in [n]$, helper H_i is connected to H_{2i+1} and H_{2i+2} (assuming $2i+1, 2i+2 \in [n]$).

The goal of the prover is to prove some P statement (encoded as $x \in L$ for some language $L \in \mathrm{P}$) to the verifier. Our setup assumption is that the string x is divided among the helpers and the verifier (in addition to the prover knowing x in its entirety). The main difficulty is that the verifier and the helpers are polylogarithmic in n, and hence, any one of them cannot even read the entire statement. To overcome this, we let them work together as one verifying unit, where the helpers assist the (designated) verifier in completing this process.

As a first step, the prover encodes x using a linear error-correcting code, partitions the codeword, and sends to the verifier and each helper a single polylog(n)-sized substring of the codeword. The verifier and the helpers then check that the encoding was done properly. Since the ECC is linear, this can be done as follows. The verifier samples a random subset of size polylog(n) of the rows in the generating matrix of the ECC. Each helper computes its part of the encoding of the input with respect to this subset of rows.

To complete the verification of the encoding, the helpers sum up their computed values, and send the sum to the verifier. This summation can be performed efficiently, by propagating the summed values along the topology of the tree, where the helper at each node sums the values it receives from its children and pass it on to its parent. In addition, the verifier gathers the encoding values attributed with the selected subset of rows, directly from the helpers that were assigned this values (by the server). The verifier then compares the two values for each selected row, and accepts the encoding if they all match.

If the ECC verification is accepted, the prover generates a PCPP proof for the encoding of the

[20] Though we state and prove assuming the language belong to P, the same protocol works for any language in NP, assuming the prover is given the witness to the input.

language, which is then verified by the verifier. To implement the oracle access of the PCPP's verifier, the protocol's verifier V will use the aid of the helpers. Specifically, since they hold the encoding of the input, V can simply query them via the correct path on the tree topology.

Intuitively, if the ECC can tolerate more errors than the distance parameter δ of the PCPP proof, then accepting the proof indicates that the codeword is close to a codeword in the encoding of the language, thus the original input x is in the language.

In order to simplify the presentation, whenever we say the verifier sends a message to some helper or vice versa, this is shorthand for having the helpers pass the message along the tree to its correct destination. Additionally, we will assume that $n+2$ is a power of 2, *i.e.,* the binary tree is full. Finally, we do not assume the prover can block message sent between two other (honest) parties. This will be handled in Section 6.

In the following, we let $L \subseteq \{0,1\}^* \times \{0,1\}^*$ be a pair language in P, and let LECC be an efficient $[3(n+1)\log^c n, (n+1)\log^c n, 2(n+1)\log^c n + 1]$ linear error-correcting code, where $c \in \mathbb{N}$ is a constant (possibly 0). Additionally, we denote by G be the generator matrix of LECC and we assume that any entry in G can be computed in time $\mathrm{polylog}(n)$.[21] Define $L' = \{(x, \mathsf{LECC.Enc}(y)) : (x,y) \in L\}$. We let $\delta(m) = m/\log^d m$, where $d \in \mathbb{N}$ is a constant, be the distance parameter of the PCPP proof for the pair language L', as given by Corollary 2.19. Finally, we let $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC.Verify})$ be a vector commitment scheme.

[21] Reed-Solomon encoding is an example of such encoding.

Protocol 5: `EfficientMultiverifierProof`

Inputs: Each helper H_i holds an input $x_i \in \{0,1\}^{\log^c n}$, where $c \in \mathbb{N}$ is some constant, the verifier holds $x_0 \in \{0,1\}^{\log^c n}$, and the prover P holds $(x_0, \ldots, x_n)$.

Common inputs: All parties hold the security parameter 1^κ, the number of helpers n (held in binary by the verifier and the helpers, and in unary by the prover), and a public input $\mathsf{pub} \in \{0,1\}^{\mathrm{poly}(\kappa)}$.

Goal: P wishes to prove the statement $(\mathsf{pub}, (x_0, \ldots, x_n)) \in L$.

1. The prover P computes $\tilde{x} = \mathsf{LECC.Enc}(x_0, \ldots, x_n)$, where we view each x_i as a field element.
2. P partitions $\tilde{x}$ into $n+1$ strings $(\tilde{x}_0, \ldots, \tilde{x}_n) = \tilde{x}$, where $\tilde{x}_i \in \{0,1\}^{3\log^c n}$ for all $i \in \{0, \ldots, n\}$. It then sends $\tilde{x}_0$ to V and for all $i \in [n]$ it sends $\tilde{x}_i$ to H_i.
3. The verifier and the helpers check that the encoding was done correctly:
   (a) V samples a set $\mathcal{S} \subseteq \{0, \ldots, n\}$ of size $\kappa \cdot \log^d 4n$ uniformly at random and sends $\mathcal{S}$ to all helpers (recall that d is the exponent of the logarithm for the distance parameter δ of the PCPP proof for L').
   (b) For every $i \in [n]$ and every $j \in \mathcal{S}$, helper H_i computes $\tilde{x}_j[i] := G(j,i) \cdot x_i$.
   (c) For every $i \in \mathcal{S} \setminus \{0\}$ helper H_i sends $\tilde{x}_i$ to V.
   (d) For $\ell = 1$ to $\log(n+2) - 1$:
      i. For every node $i \in \{(n+2)/2^\ell - 1, \ldots, (n+2)/2^{\ell-1} - 2\}$ on the ℓ^{th} level and every $j \in \mathcal{S}$, helper H_i computes
      $$\tilde{s}_j[i] := \tilde{s}_j[2i+1] + \tilde{s}_j[2i+2] + \tilde{x}_j[i],$$
      where $\tilde{s}_j[2i+1] = \tilde{s}_j[2i+2] = 0$ if $2i+1, 2i+2 > n$.
      ii. Helper H_i sends $\tilde{s}$ to its parent $\mathsf{H}_{\lfloor (i-1)/2 \rfloor}$ if $i > 2$, and to V otherwise.
   (e) For every $j \in \mathcal{S}$, V computes $\tilde{s}_j := \tilde{s}_j[1] + \tilde{s}_j[2] + G(j,0) \cdot x_0$.
   (f) If there exists $j \in \mathcal{S}$ such that $\tilde{x}_j \neq \tilde{s}_j$, then V sends $\perp$ to all parties and outputs `reject`.
4. The prover P generates a PCPP proof π for the assertion $(\mathsf{pub}, \tilde{x}) \in L'$.[22]
5. V computes $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(1^\kappa, T(\kappa, n), \{0,1\})$, where $T(\kappa, n)$ is the time it takes for P to generate π, and sends pp to P.
6. P sends to V a commitment to π.
7. The verifier V executes the PCPP's verifier $\mathsf{V}_{\mathrm{PCPP}}$ for π, with the oracle access being implemented as follows:
   - Whenever V queries a bit from π, it sends the query to the prover who responds with a decommitment of the bit.[23] If the verification of the decommitment rejects, then the verifier sends $\perp$ to all parties and outputs `reject`.
   - Whenever V queries a bit from the implicit input $\tilde{x}_i$ for some $i \in [n]$, it asks from the corresponding helper H_i to send the bit to it.
8. V outputs whatever $\mathsf{V}_{\mathrm{PCPP}}$ outputs.

[22] Observe that this step can be done in parallel to the verification of the LECC.

Clearly, the verifier V and helpers $\mathsf{H}_1, \ldots, \mathsf{H}_n$ are all polynomial in $\log n$ and κ. The next lemma states the correctness and soundness of the protocol.

Lemma 5.1. *Assume the existence of position binding vector commitment scheme. If* $(\mathsf{pub}, (x_0, \ldots, x_n)) \in L$ *and the prover is honest, then* V *always outputs* accept. *If* $(\mathsf{pub}, (x_0, \ldots, x_n)) \notin L$*, then for any malicious* PPT *(in* κ *and* n*) prover* P^**, it holds that* V *outputs* accept *with negligible probability.*

Proof. Clearly, if $(\mathsf{pub}, (x_0, \ldots, x_n)) \in L$ and the prover is honest, then by the completeness of the PCPP proof, and the fact that for every $j \in \mathcal{S}$ it holds that

$$\tilde{s}_j = \sum_{i=0}^{n} G(j,i) \cdot x_i,$$

it follows that V accepts the proof with probability 1.

Next, assume that $(\mathsf{pub}, (x_0, \ldots, x_n)) \notin L$ and fix a malicious PPT prover P^*. For simplicity we assume that P^* always decommits properly whenever it is required to do so. Let $\tilde{x}^*$ be the purported encoding sent to the helpers and the verifier, and let $\mathcal{D}$ be the set of indexes $i \in \{0, \ldots, n\}$ such that $\tilde{x}^*_i \neq \tilde{x}_i$. Recall that the encoding of any given input x_i can be computed efficiently by V and the helpers. Moreover, this is done for all inputs x_i, where $i \in \mathcal{S}$. Therefore, as

$$\delta\left(|\tilde{x}|\right) = \frac{|\tilde{x}|}{\log^d |\tilde{x}|} = \frac{3(n+1)\log^c n}{\log^d \left(3(n+1)\log^c n\right)} \geq \frac{3(n+1)\log^c n}{\log^d 4n},$$

for all sufficiently large n's, it follows that if $|\mathcal{D}| \geq \delta$ then the probability that V accepts the encoding is at most

$$\Pr\left[\mathsf{V} \text{ accepts encoding}\right] \leq \Pr\left[\mathcal{S} \cap \mathcal{D} = \emptyset\right] \leq \left(1 - \frac{|\mathcal{D}|}{n+1}\right)^{|\mathcal{S}|} \leq e^{-\frac{\frac{3(n+1)\log^c n}{\log^d 4n} \cdot \kappa \cdot \log^d 4n}{n+1}} = e^{-3\kappa \cdot \log^c n}.$$

We may now assume that $|\mathcal{D}| < \delta(|\tilde{x}|)$, that is, $\Delta(\tilde{x}^*, \tilde{x}) < \delta(|\tilde{x}|)$. Since the LECC can tolerate $(n+1)\log^c n \geq \delta(|\tilde{x}|)$ errors and by the assumption that $(\mathsf{pub}, \tilde{x}) \notin L'$, it follows that for any possible codeword $\tilde{y} \in \{0,1\}^{|\tilde{x}^*|}$ where $(\mathsf{pub}, \tilde{y}) \in L'$, it holds that $\Delta(\tilde{x}^*, \tilde{y}) \geq \delta(|\tilde{x}^*|)$. Thus, $(\mathsf{pub}, \tilde{x}^*)$ is such that $\tilde{x}^*$ is δ-far from any codeword $\tilde{y}$ satisfying $(\mathsf{pub}, \tilde{y}) \in L'$, hence the soundness of the PCPP implies that the verifier accepts with only negligible probability. □

5.2 Computing Short Output Functionalities

We are now ready to present a protocol for computing any function with a short output, assuming that all users are honest. Let us first describe the setting. Similarly to the previous section, there are $n+1$ users $\hat{\mathsf{P}}_0, \ldots, \hat{\mathsf{P}}_n$ connected via a binary tree network, and a server that is connected to every party. The root of the tree is $\hat{\mathsf{P}}_0$ whose identity is known to all parties. The parties wishe to compute a function $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^{m \cdot (n+1)}$, where $m = m(n, \kappa) = \mathrm{poly}(\log n, \kappa)$, over the inputs of the users such that only the server obtains the output.

Given `EfficientMultiverifierProof` the idea is rather simple: The users first encrypt their inputs using a fully homomorphic encryption scheme. Then, the server homomorphically evaluates the function over the encrypted inputs, and proves to the users it did so honestly using

[23] Observe that since the verifier is honest, it will never query π on an index larger than $|\pi| \leq T(\kappa, n)$, hence the prover can always decommit.

`EfficientMultiverifierProof`. If the proof is accepted, the users decrypt the output. Finally, since we aim to handle the case where there are n users and the server has an input, we will assume that $\hat{\mathsf{P}}_0$ has no input and the server sends to it an input at the start of the protocol. Similarly to Section 5.1, in order to simplify the presentation in this section, we do not assume the server can block message sent between two other (honest) users. This will be handled in Section 6.

In the following, write the function f as $f(x) = (f_0(x), \ldots, f_n(x))$, where $f_i(x) \in \{0,1\}^m$ is the i^{th} block of the output. Additionally, we let $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be a fully homomorphic encryption scheme and let $s = s(\kappa)$ be the number of random bits used in $\mathsf{FHE.Eval}$. Finally, let $\mathsf{G} : \{0,1\}^{n+1} \to \{0,1\}^{s\cdot(n+1)}$ be a pseudorandom generator, and write it as $\mathsf{G}(r) = (\mathsf{G}_0(r), \ldots, \mathsf{G}_n(r))$ where $\mathsf{G}_i : \{0,1\}^{n+1} \to \{0,1\}^s$ for every $i \in \{0, \ldots, n\}$.

Protocol 6: `GeneralProtocol`

Inputs: Each user $\hat{\mathsf{P}}_i$, where $i \in [n]$, holds an input x_i. The server holds x_0.
Common inputs: All parties hold the security parameter 1^κ and the number of users $n+1$ (held in binary by the users and in unary by the server).

1. The server sends x_0 to $\hat{\mathsf{P}}_0$.
2. $\hat{\mathsf{P}}_0$ generates FHE keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^\kappa)$, sends $(\mathsf{pk}, \mathsf{sk})$ to all other users, and sends the public key pk to the server.
3. Each user $\hat{\mathsf{P}}_i$, where $i \in \{0, \ldots, n\}$, encrypts its input x_i to obtain $\hat{x}_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(x_i)$, sample a uniform random bit $r_i \leftarrow \{0,1\}$, and sends $\hat{x}_i$ and r_i to the server.
4. The server homomorphically evaluates all functions f_i, where $i \in \{0, \ldots, n\}$, over the encrypted inputs using the randomness provided by the users, and obtains the encrypted outputs
$$\hat{y}_i = \mathsf{FHE.Eval}_{\mathsf{pk}}(f_i, \hat{x}_0, \ldots, \hat{x}_n; \mathsf{G}_i(r_0, \ldots, r_n)).$$
5. The server sends $\hat{y}_i$ to $\hat{\mathsf{P}}_i$, for every $i \in \{0, \ldots, n\}$.
6. The parties execute protocol `EfficientMultiverifierProof`, where the server takes the role of the prover P, user $\hat{\mathsf{P}}_0$ takes the role of the verifier V, and for every $i \in [n]$ user $\hat{\mathsf{P}}_i$ takes the role of the helper H_i, for the pair language L defined as follow:
$$L = \left\{ (\mathsf{pk}, (z_0, \ldots, z_n)) : \forall i \in \{0, \ldots, n\} \quad \begin{array}{l} z_i = (r_i, \hat{x}_i, \hat{y}_i) \text{ , where} \\ \hat{y}_i = \mathsf{FHE.Eval}_{\mathsf{pk}}\left(f_i, \hat{x}_0, \ldots, \hat{x}_n; \mathsf{G}_i(r_0, \ldots, r_n)\right) \end{array} \right\}$$
Observe that $L \in \mathrm{P}$, hence the protocol can be executed efficiently.
7. $\hat{\mathsf{P}}_0$ sends all parties the output it obtained from the computation.
8. If the output is reject then all users abort.
9. Otherwise, each user $\hat{\mathsf{P}}_i$ decrypts $\hat{y}_i$ to obtain $y_i \leftarrow \mathsf{FHE.Dec}_{\mathsf{sk}}(\hat{y}_i)$, and sends the decrypted values back to the server.
10. The server outputs $(y_0, \ldots, y_n)$.

Lemma 5.2. *Let $m = m(\kappa, n) = \mathrm{poly}(\log n, \kappa)$ and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^{m\cdot(n+1)}$ be a GMPC functionality. Assume the existence of pseudorandom generators, position binding vector commitment schemes, and n-secure fully homomorphic encryption schemes. Then* `GeneralProtocol` *computes f with security against any* PPT *adversary corrupting the server.*

Proof. Correctness clearly holds. Assume that the server is corrupted by an adversary $\mathcal{A}$. Its simulator $\mathsf{Sim}_{\mathcal{A}}$ works as follows.

1. Query $\mathcal{A}$ to obtain the input it sends to $\hat{\mathsf{P}}_0$, and send it to the trusted party.
2. Receive an output $(y_0, \ldots, y_n)$ from the trusted party (recall that the server has no input and all users are honest).
3. Generate FHE keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(1^\kappa)$.
4. Compute $n+1$ dummy encryptions, sample $n+1$ random bits $r_0, \ldots, r_n$, and send them and the public key pk to $\mathcal{A}$.
5. The adversary $\mathcal{A}$ replies with purported encrypted values $(\hat{y}_0, \ldots, \hat{y}_n)$.
6. Simulate `EfficientMultiverifierProof`:
   (a) The adversary sends a purported codeword $\tilde{x}_i$ for each $i \in \{0, \ldots, n\}$.
   (b) Perform the same error correction verification as done in the real world. If the verification fails send $\perp$ to $\mathcal{A}$, output whatever $\mathcal{A}$ outputs, and halt.
   (c) Otherwise, execute the PCPP verifier $\mathsf{V}_{\mathrm{PCPP}}$, while querying $\mathcal{A}$ whenever $\mathsf{V}_{\mathrm{PCPP}}$ queries the proof.
   (d) If $\mathsf{V}_{\mathrm{PCPP}}$ rejects the proof, send $\perp$ to $\mathcal{A}$, output whatever $\mathcal{A}$ outputs, and halt.
7. If there was no abort during the simulation of `EfficientMultiverifierProof`, send to $\mathcal{A}$ the output $(y_0, \ldots, y_n)$ as given by the trusted party, output whatever $\mathcal{A}$ outputs, and halt.

By the semantic security of the FHE scheme, it follows that the adversary's view in the real world is indistinguishable from the view generated in the ideal world. In particular, $\mathcal{A}$'s replies are indistinguishable. Next, by Lemma 5.1, except with negligible probability, either all users abort and $\mathcal{A}$ does not obtain the output in both worlds, or no user aborts and $\mathcal{A}$ receives an output in both worlds. By the definition of the language L, the output in the real world are the decryptions of

$$\hat{y}_i = \mathsf{FHE.Eval}_{\mathsf{pk}}\left(f_i, \hat{x}_0, \ldots, \hat{x}_n; \mathsf{G}_i(r_0, \ldots, r_n)\right).$$

Observe that if the distribution of the above $(\hat{y}_0, \ldots, \hat{y}_n)$ can be distinguished from the encryptions

$$\left(\mathsf{FHE.Enc}_{\mathsf{pk}}\left(y_i\right)\right)_{i=0}^{n}$$

in the ideal world, then $\mathsf{FHE.Eval}$ can be used to break the security of the PRG G. Thus, the outputs of $\mathcal{A}$ in both worlds are indistinguishable. □

6 Putting It All Together

In this section, we combine the committee-tree election protocol from Section 4 and the protocol for proving correct computation from Section 5, and present a protocol for securely computing any arbitrary short-output functionality assuming at most α-fraction of the users are corrupted, where $\alpha < 1/8$ is a constant. Formally, we assume the parties are given ideal access to the functionality f_{many} for sampling the tree of committees. This assumption can be removed by implementing the functionality using Protocol 4, however, this comes at the cost of conditioning on the event PCSG defined in Section 3.3.

Theorem 6.1. *Let $\alpha < 1/8$, let $m = m(\kappa, n) = \mathrm{poly}(\log n, \kappa)$, and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^{m\cdot(n+1)}$ be a GMPC functionality. Assume the existence of pseudorandom generators and n-secure fully homomorphic encryption schemes. Then there exists an αn-secure n-user GMPC protocol computing f in the f_{many}-hybrid model.*

The idea is to have the parties generate the committees in a tree-like structure by calling f_{many}, and let each committee $\mathcal{C}_i$ sampled by the functionality emulate a single user $\hat{\mathsf{P}}_i$ from protocol Protocol 6. This result in a secure protocol computing any GMPC functionality in the hybrid model. Similarly to the definitions used in Section 4.2, we formalize this by letting the parties have access to the "next-message" function of each committee. In Section 6.1 we show to implement these.

We first define the "next-message" hybrid functionality of the i^{th} committee $\mathcal{C}_i$, denoted NxtMsg_i. The inputs of the users to each call of NxtMsg_i are a $(1-\beta-\varepsilon/2)\kappa$-out-of-$\kappa$ Shamir's secret sharing scheme of the view of $\mathcal{C}_i$, where β is the fraction of malicious users among the alive users. If at least $(1-\beta-\varepsilon/2)\kappa$ inputs are provided to the functionality, then it proceeds to compute the next-message as specified by the protocol, and sharing the output among the users in $\mathcal{C}_i$. If less then $(1-\beta-\varepsilon/2)\kappa$ inputs are provided (*e.g.,* the user is corrupted or a corrupted server blocked an honest user), the functionality sends $\perp$ to all of its users. If a user obtain $\perp$ as the output from some NxtMsg_i, then it aborts.

We abuse notions and describe the protocol as if each committee is a single party. Formally, whenever we say that the server sends a message to $\mathcal{C}_i$, it means that it (supposedly) shares it among the users of $\mathcal{C}_i$. Similarly, when we say that $\mathcal{C}_i$ sends a message to the server, it means that every user in $\mathcal{C}_i$ sends its share of the message to the server. Finally, whenever a committee $\mathcal{C}_i$ sends a message msg to another committee $\mathcal{C}_j$, it means that each user in $\mathcal{C}_i$ shares its share of msg

among the users in $\mathcal{C}_j$. In the following, we let $\mathsf{NxtMsg} = \{\mathsf{NxtMsg}_i\}_{i=0}^n$.

Protocol 7: `MainProtocol`

Inputs: Each user $i \in [n]$ holds an input x_i. The server holds x_0.
Common inputs: All parties hold the security parameter 1^κ and the number of users n (held in binary by the users, and in unary by the server).

1. The parties call f_{many} to elect $n+1$ committees $\{\mathcal{C}_i\}_{i=0}^n$. Recall that each user i holds the committee $\mathcal{C}_i$, and all committee are held in a tree structure, *i.e.,* the users in $\mathcal{C}_i$ holds $\mathcal{C}_i$, $\mathcal{C}_{2i+1}$, $\mathcal{C}_{2i+2}$, and $\mathcal{C}_{\lfloor (i-1)/2 \rfloor}$ (assuming the indexes are in $\{0,\ldots,n\}$).
2. Each user i sends its input x_i to $\mathcal{C}_i$ (held shared). If user i was blocked, then x_i is replaced with a default value.[24]
3. The committees aggregate the total number of inputs they received. If it is less than $(1-\alpha)n$, then they abort (note that with overwhelming probability every user belong to at least one committee, hence all users abort).
4. The parties emulate an execution of `GeneralProtocol`, with each committee $\mathcal{C}_i$ taking the role of user $\hat{\mathsf{P}}_i$. Specifically, whenever $\hat{\mathsf{P}}_i$ supposes to compute a message, the users in $\mathcal{C}_i$ call NxtMsg_i to obtain the shares of this message.
   - If at any point during the emulation, the functionality NxtMsg_i outputs $\perp$ to the users in $\mathcal{C}_i$, then they all abort.
5. The server outputs whatever it receives from the emulation of `GeneralProtocol`.

The following lemma asserts the security of `MainProtocol`. Combined with the secure implementation of NxtMsg we will present later in Section 6.1 and the composition theorem, this proves Theorem 6.1.

Lemma 6.2. *Let $\alpha < 1/8$ be a constant, let $m = m(\kappa, n) = \mathrm{poly}(\log n, \kappa)$, and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^{m\cdot(n+1)}$ be a GMPC functionality. Assume the existence of pseudorandom generators and n-secure fully homomorphic encryption schemes. Then* `MainProtocol` *is an αn-secure n-user GMPC protocol computing f in the $\{f_{\mathsf{many}}, \mathsf{NxtMsg}\}$-hybrid world.*

Proof. Fix an adversary $\mathcal{A}$ corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, of size at most $|\mathcal{I}| \leq \alpha n$. Observe that if the server is honest, then except with negligible probability, all committees will contain sufficiently many honest users, so that the calls to NxtMsg will not output $\perp$. Thus, no honest user will ever abort, and the server obtains the output.

Assume that the server is corrupted. Intuitively, security holds, since the view of $\mathcal{A}$ when emulating `GeneralProtocol` is, up to shares that reveal no information, exactly the same as in a real execution of `GeneralProtocol` (with honest parties and no blocking of messages). We next define the simulator $\mathsf{Sim}_{\mathcal{A}}$. To simplify the presentation, we will not concern ourselves with the shares the adversary receives throughout the execution of the protocol.

1. Query $\mathcal{A}$ for its inputs to the functionality f_{many}. If it sent abort at any point, then all users abort and the protocol halts. In this case, the adversary does no receive any more messages. Output whatever $\mathcal{A}$ outputs, and halt.

[24] Formally, x_i is held shared by the users in $\mathcal{C}_i$, and is supposed to be reconstructed by NxtMsg_i. If the functionality cannot reconstruct the input, then instead of aborting, it replaces it with a default value.

2. Otherwise, the server blocks a set of users $\mathcal{B} \subseteq [n]$ from sending their inputs to $\mathcal{C}_i$.[25]

3. If the server blocks more than αn users from providing their inputs to their committees, then the protocol halts and the adversary does no receive any more messages. The simulator outputs whatever $\mathcal{A}$ outputs, and halt.

4. Otherwise, query $\mathcal{A}$ for the inputs of the malicious users that it sends to the committees (formally, $\mathcal{A}$ sends it in the form of shares).

5. Execute the simulator $\mathsf{Sim}'_{\mathcal{A}}$ for Protocol 6 with the following modification.
   - Suppose the adversary blocks too many users in some committee $\mathcal{C}_i$ from sending a message to another committee $\mathcal{C}_j$ (hence NxtMsg_j outputs $\perp$), *before* the parties decrypt the output for the server (see Step 9 of Protocol 6). In this case, `MainProtocol` aborts. Then the adversary does no receive any more messages. and we let the simulator output whatever $\mathcal{A}$ outputs, and halt.

6. Otherwise, send to the trusted party T whatever $\mathsf{Sim}'_{\mathcal{A}}$ sent alongside the set $\mathcal{B}$ of blocked users. Then, send the output to $\mathcal{A}$, output whatever it outputs, and halt (recall that this is exactly what $\mathsf{Sim}'_{\mathcal{A}}$ does in case the adversary did not cause an abort).

Since the view of $\mathcal{A}$ when emulating Protocol 6 is the same as a real execution (up to shares), it follows that the view generated by $\mathsf{Sim}'_{\mathcal{A}}$ is indistinguishable. Therefore, the probability the adversary causes all honest users to abort in the real world, is indistinguishable from the ideal world. Finally, conditioned on the protocol not aborting, the output received by the adversary in the ideal world, is the same output it receives when simulating an execution of Protocol 6. Therefore, as shown in Lemma 5.2, the output of simulator is indistinguishable from the output of $\mathcal{A}$ in the real world. □

6.1 Secure Implementation of The Next-Message Functions

We explain how to implement the next-message functionality NxtMsg used in our protocols to compute the messages generated for each personal committee or each (global) committee. Specifically, the implementation will be secure under *parallel composition*. The implementation for both cases is done using the same protocol. Essentially, our protocol is a slight variation of the protocol given in Goldwasser and Lindell [41, Corollary 2].

Remark 6.3. *We view every committee as a different "computation unit". In particular, if an honest user aborts from one committee, it does* not *automatically abort from other committees it participates in. This is important as otherwise a malicious user i might affect committees it does not participate in, by causing an honest user j to abort from* another *committee which both i and j participate in.*

We first show how to implement a broadcast channel. Then, we will show how the parties can securely compute any functionality, given the broadcast channel. We stress that these computations are always performed by subsets of users that are of size that is poly-logarithmic in the number of users. Hence, the protocols implementing these computations may be of polynomial complexities in the number of participants. In the following we fix positive constants $\alpha < 1/8$ and $\varepsilon = 1/8 - \alpha$.

[25] Observe that the simulator can obtain this set from the malicious server, since at the beginning of every round, the server expects to receive the set of pairs (i, j), where user i sends a message to user j.

Computing broadcast. We present a deterministic protocol for computing the broadcast functionality. First, the sender sends its message to the server, which in turn passes it on to everyone else. Second, all users exchange with each other what they received from the server. A user aborts if less than $1-\alpha-\varepsilon/2$ of the messages it received are the same as what it received from the server. In particular, the sender verifies that at least $1-\alpha-\varepsilon/2$ of the users sent it its input. Finally, the users check that at most $\alpha+\varepsilon/2$ of them have aborted, and abort if this is not the case.

Observe that, although when the server is malicious there are roughly 2α corrupted users in the (global) committees, when it is honest there are roughly α. Thus, any user seeing inconsistencies with more than α users can abort. Now, clearly, if the server is honest then all honest users agree on the output (even if the sender is corrupted). On the other hand, if the server is malicious, then any user that disagrees with at least $\alpha+\varepsilon/2$ of the users aborts. Moreover, if more than $\alpha+\varepsilon/2$ of the users abort at the second step, then everyone will abort at the last step. Finally, as the protocol admits perfect security, by the work of Kushilevitz, Lindell, and Rabin [50] it follows that it is UC-secure and thus can be securely run in parallel.

Observe that in the above protocol, a malicious server can decide which users to block, depending on the messages it received. Furthermore, it can change the message at the cost of having the sender abort. As we show below, these two issues do not affect the security of the overall protocol.

Computing any functionality. We next explain how the user in each PC and committee can securely compute any functionality. The security that the protocol admits is such that if more than $\alpha+\varepsilon/2$ honest users are blocked, then the protocol is secure-with-abort (*i.e.,* the adversary can abort the computation after obtaining the output). Otherwise, the protocol admits guaranteed output delivery. Specifically, the computation will be secure under parallel composition. This will allow the committees and PCs of honest users, to effectively be viewed as honest parties (recall that unlike honest users, if the server blocks a PC or a committee then all users abort).

We let the users emulate an execution of the protocol of Rabin and Ben-Or [59]. The protocol admits information theoretic security, assuming secure point-to-point channels between every pair of users, an honest majority, and the availability of a broadcast channel. We show that the implementation of broadcast presented above is sufficient for the users to emulate the protocol of [59]. Consider the case a user broadcasted a message using the above protocol at some round. Then, even though the server can block the users based on the value of the message to be sent, it could have done so at the beginning of the next round. Moreover, the server can only block at most $(\alpha+\varepsilon/2)$-fraction of the honest users (viewed as aborting, malicious parties, with security guaranteed to hold by the adaptive security of [59]), in addition to having at most $(2\alpha+\varepsilon/2)$-fraction corrupted users (with overwhelming probability). Furthermore, changing the sender's message each computation of broadcast causes it to abort, hence it can be viewed as one of the aborting malicious parties. Thus, the security guarantees from the broadcast protocol suffices.

Next, observe that the users *do not* have secure channels, as a malicious server can block messages between any pair of users. To overcome this, whenever a party does not receive a message from another party, it uses the broadcast protocol to send an accusation as well as its identity. Then, all other users label *both* users as malicious (even if one of them is honest). Note that a malicious server can abort $2\alpha+\varepsilon/2$ honest users by sending false accusations, however, aborting more will cause all users in the committee to abort with overwhelming probability.

Finally, in order to maintain an honest majority, recall that in every (global) committee, a malicious server can have $2\alpha+\varepsilon/2$ malicious users. Since it can abort an additional $2\alpha+\varepsilon/2$, we

require that $4\alpha + \varepsilon < 1/2$, which indeed holds since $\alpha < 1/8$.

We stress that even if the server blocks the communication with different users for each user, the protocol is still secure due to the use of broadcast (recall that the users check that sufficiently many users are active at the end of the broadcast protocol). Furthermore, even though the server can block users adaptively, the protocol remains secure as the protocol of Rabin and Ben-Or [59] is secure against adaptive adversaries.[26] Finally, as shown by Canetti [23] and Kushilevitz el al. [50], the protocol of Rabin and Ben-Or [59] is secure under concurrent general composition,[27] thus the parties can run it in parallel while maintaining overall security.

Remark 6.4. *We observe that the constant 1/8 can be improved assuming an honest server is allowed to block messages. This is because the server knows whether the sender of the message actually sent it or not, and block it from every other user in case it did not send any message. Therefore, unlike in the proposed solution, a corrupted server can only block α more honest users (in addition to the 2α corrupted users there are in each committee). Thus, for honest majority it is required to have $\alpha < 1/6$.*

6.2 Computing Shallow and Small Circuits

In this section we sketch a simple protocol for computing any circuit of size $\tilde{O}(n)$, and polylogarithmic depth, fan-in, and fan-out. Unlike the general protocol, here we do not need to assume the existence of a fully homomorphic encryption scheme. Two important example of functionalities computable by such circuits are summation and sorting.

The idea is the following: the parties will use protocol Protocol 4 to generate $\tilde{O}$ committees in a "tree-like manner", where each committee corresponds to either a single gate, an input wire, or an output wire in the circuit. Then, using the help of the server, each committee receives the list of committees to which it should send the message according to the circuit. Since with overwhelming probability, each input and output wire, and each gate, will have a committee associated with it, where all committees contain sufficiently many honest users, the committees can perform this computation securely.

To ensure the server doesn't cheat, and connect committees to incorrect ones, or to fabricated ones, we let each committee sign its identity as well as its associated wire/gate in the circuit (recall that in Protocol 4, all committees obtain a secret signature key, while all parties obtain the corresponding public key). Then, when the server sends a list of committees, each committee verifies that the signature is valid, and that it is consistent with the circuit.

Finally, as the size of the circuit is $\tilde{O}(n)$, it follows that each user appears in a polylogarithmic number of committees, except with negligible probability. In addition, the complexity of each committee is polylogarithmic since the fan-in, fan-out, and the depth of the circuit are polylogarithmic.

For a fixed circuit C, we denote the functionality for sampling the committees by $f_{\mathsf{many},C}$. Formally, we have the following result.

Lemma 6.5. *Let $\alpha < 1/8$ and let C be a circuit of size $n \cdot \log^c n$, depth at most $\log^c n$, and fan-in and fan-out at most $\log^c n$. Assume the existence of n-secure signature schemes and position binding*

[26] Rabin and Ben-Or [59] referred to them as dynamic adversaries.

[27] Formally, for security with concurrent general composition to hold, when a user obtains its input to the protocol it first broadcasts a message indicating it is ready. The actual protocol of Rabin and Ben-Or [59] starts after all users broadcast the message.

vector commitment schemes. Then there exists an n-user GMPC protocol computing $f_{\mathsf{many},C}$, such that conditioned on the event PCSG*, the protocol is αn-secure.*

Theorem 6.6. *Let $\alpha < 1/8$, let $m = m(\kappa, n) = \mathrm{poly}(\log n, \kappa)$ let $c > 0$, and let $f : (\{0,1\}^m)^{n+1} \to \{0,1\}^m$ be a GMPC functionality. Assume that f can be computed by a circuit C of size $n \cdot \log^c n$, depth at most $\log^c n$, and fan-in and fan-out at most $\log^c n$. Then there exists an αn-secure n-user GMPC protocol computing f in the $f_{\mathsf{many},C}$-hybrid world.*

6.2.1 Application to the Shuffle Model of Differential Privacy

Differential privacy [33] is a rigorous definition for privacy. It typically involves a server or a data curator that aggregates the data of n individuals (or users) in a way that guarantees that the outcome of the computation does not leak too much information on the data of any single individual. Most of the work on differential privacy focuses on the case where the server is fully trusted with the raw data, and privacy is only provided w.r.t. an observer that sees the outcome of the computation. In recent years, however, there has been a growing interest in studying models where the server is untrusted. The first model in this vein, called the *local model* [33, 46], requires each user to randomize its input before sending it to the untrusted server, who aggregates all the noisy reports. The amount of randomization needs to be sufficiently high such that the server learns almost no information about the data of any single user, and hence, this model generally suffers from significant loss of accuracy.[28]

The *shuffle model of differential privacy* [33, 44, 13, 34, 29, 4] aims to alleviate the loss of accuracy of the local model while maintaining its trust guarantees. This is achieved by augmenting the local model with a *shuffle functionality* that takes messages from the users and delivers them to the server after randomly permuting them. The fruitful line of work on the shuffle model shows that having access to such a shuffle functionality can lead to dramatic accuracy improvements over the plain local model of differential privacy. This naturally raises the question of efficiently implementing such a shuffle functionality. Recently, Bell at al. [9] presented a secure implementation of the shuffle functionality with constant rounds of communication. However, the runtime and the communication complexity of the users in their protocol is linear in n (the number of users), and reducing users' complexities to polylogarithmic in n was stated as an open question. As we next explain, Theorem 6.6 resolves this question as an important special case.

Observe that in order to implement the shuffle functionality, it suffices to implement a sorting functionality, in which the server obtains all the messages in an ascending order. (The reason is that the server can then permute the messages on its own.) Recall that sorting can be done using a sorting networks of depth $O(\log n)$ (and hence size $O(n \log n)$) using the famed *AKS network* [1]. Instantiating Theorem 6.6 with the AKS network, denoted AKS, we obtain the following theorem.

Theorem 6.7. *Let* Shuffle $: (\{0,1\}^m)^n \to \{0,1\}^{m \cdot n}$ *be the shuffle GMPC functionality, whose output is the input of the users in an order chosen uniformly at random. Then the following holds. Let $\alpha < 1/8$, let $m = m(\kappa, n) = \mathrm{poly}(\log n, \kappa)$ let $c > 0$. Then there exists an αn-secure n-user GMPC protocol computing* Shuffle *in the f_{AKS}-hybrid world.*

Theorem 6.7 shows that the shuffle functionality can be securely computed in the GMPC model with polylogarithmic runtime and communication complexity for the users, which resolves an open question of [9]. Note that this construction does not assume FHE.

[28] We stress that the server can still learn global statistics about the population.

Alternatively, Theorem 6.7 can be proved using *bitonic sorting* [6] instead of the AKS sorting network. While the AKS network has better asymptotics ($O(\log n)$ depth and $O(n \log n)$ size vs $O(\log^2 n)$ depth and $O(n \log^2 n)$ size), it is far from being practical (see Goodrich [43] for an estimation of the constants in various variants and alternatives and Goodrich [42] for a randomized oblivious sort), and bitonic sorting is much more efficient for relevant sizes.

Remark 6.8. *For simplicity, in Theorem 6.7 we considered a shuffle functionality that takes a single input from each user. Our techniques also apply to a setting where every user submits $\ell = \text{polylog}(n)$ messages to the shuffle functionality, which then deliverers all $n\ell$ messages to the server in a random order.*

7 Applications: Secure Protocols in the Star Topology

In this section, we explain how to compile a protocol in the GMPC model to the star network assuming a special form of PKI. We first formally define the model in the star network and define the PKI. We then show how to compile any protocol in the GMPC protocol to the star model. The parties are connected via a star network with the server in the middle, *i.e.,* each user is connected only to the server. Thus, the server has full control over communication traffic and can block any message it wants. All parties are given the security parameter 1^κ, and the number of users n, held in binary by the users and in unary by the server.

The Board Functionality

To emulate users wishing to securely send a message to another user, we assume the parties are given a special form of PKI. Observe that since the users are assumed to be polylogarithmic in the number of users n, we cannot simply give each of them the public keys of all other users. Instead, the parties are given a special *board functionality*, denoted Board. This functionality holds the public keys (for both encryption and signature schemes) of all users, and it allows any user to request the public keys of any other user. This can be motivated by allowing the parties to access a third-party forum, whereupon registration the keys are sampled by the website.

Intuitively, having access to the Board functionality allows any user to send a message to any user of its choice via the server, such that the server cannot learn the contents of the message nor modify it. In the following, we let $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme, and we let $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a non-malleable public key encryption scheme.[29]

Roughly, the Board functionality allows the following. Any user can register *exactly once*, and obtain the public and secret keys for both the encryption scheme and the signature scheme. The functionality then stores the public keys in its database. Additionally, every user can request the functionality to send it the public keys of any other user that appears in the database.

We stress that the Board functionality *never* interacts with the server. In particular, this means that it cannot invent new users or public keys, nor can it prevent any user from obtaining the public keys of another user. We now formally describe the functionality.

Input: The functionality holds the security parameter 1^κ and the number of users 1^n.

[29] Roughly, in a non-malleable encryption scheme no computationally bounded adversary that is given an encryption of some message m can generate an encryption of a different correlated message m'.

Initialize: Set up a new database $\mathcal{D}$ as an array of length n.

Register: Whenever user i wishes to register, it samples keys $(\mathsf{pk}^i_{\mathsf{Sig}}, \mathsf{sk}^i_{\mathsf{Sig}}) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$ and $(\mathsf{pk}^i_{\mathsf{PKE}}, \mathsf{sk}^i_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$. It then sends the message $(\mathsf{register}, \mathsf{pk}^i_{\mathsf{PKE}}, \mathsf{pk}^i_{\mathsf{Sig}}, i)$ to the functionality. Upon receiving this message, verify that i did not register previously. If this is not the case, then ignore the message. Otherwise, add $(\mathsf{pk}^i_{\mathsf{Sig}}, \mathsf{pk}^i_{\mathsf{PKE}})$ to $\mathcal{D}$ at position i.

Get Keys: Upon receiving a message $(\mathsf{get}\colon i, \mathsf{from}\colon j)$, verify that i is in $\mathcal{D}$. If not, then ignore the message. Otherwise, send to user j the public keys $(\mathsf{pk}^i_{\mathsf{Sig}}, \mathsf{pk}^i_{\mathsf{PKE}})$ of user i.

Compiling Secure GMPC Protocols to the Star Model

Given access to the Board functionality, the parties can emulate the full network with blocking setting we considered throughout the paper. Whenever user i wishes to send some message to user j, it will sign and encrypt its message using its secret signature key and j's public encryption key. It will send to the server a request to transfer the encrypted and signed message to user j.

Since the message is encrypted, the server cannot learn its contents. Moreover, as the message was also signed, it follows that the server cannot forge a new message without j noticing.[30] Thus, the only attack a malicious server can do is to block the message from reaching j. Note that it could still be the case that user i is malicious and purposely signed the message incorrectly, hence j will not abort in this case and view it as if the message was blocked.

Observe that since the encryption scheme is assumed to be non-malleable, it follows that the protocol maintains security even if the parties call the Board functionality concurrently. We conclude that the above model is equivalent to the GMPC model considered throughout the paper, in the sense that any attacker for a protocol in one model can be mapped to an attacker in the other model. Thus, all of our results can be translated to this setting as well.

Acknowledgements

The authors are very grateful to Muthuramakrishnan (Muthu) Venkitasubramaniam for many helpful discussions.

Funding

The work of B.A. was supported in part by grants from the Israel Science Foundation (no.152/17 and no.391/21), and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office. The work of M.N. was supported in part by grants from the Israel Science Foundation (no.2686/20), by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness and by a Data Science grant of the PCB. The work of E.O. was supported in part by grants from the Israel Science Foundation (no.152/17), by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office, and by the Robert L. McDevitt, K.S.G., K.C.H.S. and Catherine H. McDevitt L.C.H.S. endowment at Georgetown University. Part of this work was done when E.O. was hosted by Georgetown University. The work of U.S. was partially supported by the Israel Science Foundation (grant 1871/19) and by Len Blavatnik and the Blavatnik Family foundation.

[30] Note we need to have both the encryption and signature schemes to be secure against adversaries that are polynomial in the number of users n, which is possibly superpolynomial in the security parameter κ.

Bibliography

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An 0(n log n) sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 19, New York, NY, USA, 1983. Association for Computing Machinery.

[2] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits. BLENDER: Enabling local search with a hybrid differential privacy model. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 747–764. USENIX Association, 2017.

[3] B. Avent, Y. Dubey, and A. Korolova. The power of the hybrid model for mean estimation. *Proc. Priv. Enhancing Technol.*, 2020(4):48–68, 2020.

[4] B. Balle, J. Bell, A. Gascón, and K. Nissim. The privacy blanket of the shuffle model. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 638–667. Springer, 2019.

[5] B. Balle, J. Bell, A. Gascón, and K. Nissim. Private summation in the multi-message shuffle model. In *CCS*, pages 657–676. ACM, 2020.

[6] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314, 1968.

[7] A. Beimel, I. Haitner, K. Nissim, and U. Stemmer. On the round complexity of the shuffle model. In *TCC (2)*, volume 12551 of *Lecture Notes in Computer Science*, pages 683–712. Springer, 2020.

[8] A. Beimel, A. Korolova, K. Nissim, O. Sheffet, and U. Stemmer. The power of synergy in differential privacy: Combining a small curator with local randomizers. In *ITC*, volume 163 of *LIPIcs*, pages 14:1–14:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[9] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1253–1269. ACM, 2020.

[10] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

[11] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. In *Proceedings of the 36th annual ACM Symposium on Theory of Computing*, pages 1–10. Association for Computing Machinery, 2004.

[12] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short pcps verifiable in polylogarithmic time. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 120–134. IEEE, 2005.

[13] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In

Proceedings of the 26th Symposium on Operating Systems Principles, pages 441–459. ACM, 2017.

[14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 11751191, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468.

[15] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97. Springer, 2019.

[16] E. Boyle, S. Goldwasser, and S. Tessaro. Communication locality in secure multi-party computation. In *Theory of Cryptography Conference*, pages 356–376. Springer, 2013.

[17] E. Boyle, S. Goldwasser, and Y. T. Kalai. Leakage-resilient coin tossing. *Distributed computing*, 27(3):147–164, 2014.

[18] E. Boyle, R. Cohen, D. Data, and P. Hubáček. Must the communication graph of mpc protocols be an expander? In *IACR Crypto 2018*, pages 243–272. Springer, 2018.

[19] E. Boyle, A. Jain, M. Prabhakaran, and C.-H. Yu. The bottleneck complexity of secure multiparty computation. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[20] E. Boyle, R. Cohen, and A. Goel. Breaking the $O(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 319–330, 2021.

[21] G. Bracha. An O(log n) expected rounds randomized byzantine generals protocol. In *STOC*, 1985.

[22] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[23] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[24] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[25] D. Catalano and D. Fiore. Vector commitments and their applications. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, pages 55–72, 2013.

[26] A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 195–208. Springer, 2012.

[27] N. Chandran, W. Chongchitmate, J. A. Garay, S. Goldwasser, R. Ostrovsky, and V. Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 153–162, 2015.

[28] A. Cheu and J. R. Ullman. The limits of pan privacy and shuffle privacy for learning and estimation. In *STOC*, pages 1081–1094. ACM, 2021.

[29] A. Cheu, A. D. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via shuffling. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 375–403. Springer, 2019.

[30] R. Cohen, I. Haitner, E. Omri, and L. Rotem. From fairness to full security in multiparty computation. In *SCN*, pages 216–234, 2018.

[31] R. Cramer, I. Damgård, and S. Fehr. On the cost of reconstructing a secret, or vss with optimal reconstruction phase. In *Annual International Cryptology Conference*, pages 503–523. Springer, 2001.

[32] I. Dinur and O. Reingold. Assignment testers: towards a combinatorial proof of the pcp-theorem. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004. doi: 10.1109/FOCS.2004.16.

[33] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[34] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2468–2479. SIAM, 2019.

[35] U. Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 142–152. IEEE, 1999.

[36] B. Fisch. Poreps: Proofs of space on useful data. *IACR Cryptology ePrint Archive*, 2018:678, 2018. URL https://eprint.iacr.org/2018/678.

[37] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[38] 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 *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2021.

[39] B. Ghazi, N. Golowich, R. Kumar, R. Pagh, and A. Velingker. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy. In *EUROCRYPT (3)*, volume 12698 of *Lecture Notes in Computer Science*, pages 463–488. Springer, 2021.

[40] O. Goldreich. *Foundations of Cryptography, Volume II – Basic Applications*. Cambridge University Press, 2004.

[41] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[42] M. T. Goodrich. Randomized shellsort: A simple data-oblivious sorting algorithm. *J. ACM*, 58(6):27:1–27:26, 2011.

[43] M. T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in o(n log n) time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 684–693. ACM, 2014.

[44] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 239–248. IEEE Computer Society, 2006.

[45] Y. Ishai, R. Ostrovsky, and H. Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography Conference*, pages 21–38. Springer, 2012.

[46] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM J. Comput.*, 40(3):793–826, 2011.

[47] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. ISBN 978-1-58488-551-1.

[48] J. M. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000*, pages 163–170. ACM, 2000.

[49] R. Kohen and O. Sheffet. Transfer learning in differential privacy's hybrid-model. *CoRR*, abs/2201.12018, 2022.

[50] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM Journal on Computing*, 39(5):2090–2112, 2010.

[51] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[52] B. Libert and M. Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 499–517, 2010.

[53] Y. Lindell. Lower bounds for concurrent self composition. In *Theory of Cryptography Conference*, pages 203–222. Springer, 2004.

[54] Y. Lindell. Lower bounds and impossibility results for concurrent self composition. *J. Cryptol.*, 21(2):200–249, 2008.

[55] G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 54–63. ACM, 2004.

[56] M. Naor, M. Parter, and E. Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1096–115. SIAM, 2020.

[57] R. Pass, H. Lin, and M. Venkitasubramaniam. A unified framework for UC from only OT. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 699–717. Springer, 2012.

[58] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[59] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

[60] L. Reyzin, A. D. Smith, and S. Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In S. Dolev, O. Margalit, B. Pinkas, and A. A. Schwarzmann, editors, *CSCML 2021*, volume 12716 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2021.

[61] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[62] J. Tenenbaum, H. Kaplan, Y. Mansour, and U. Stemmer. Differentially private multi-armed bandits in the shuffle model. In *Advances in Neural Information Processing Systems*, volume 34, pages 24956–24967. Curran Associates, Inc., 2021.

A Ideal World For Honest Server

Ideal world for full security with blocking assuming an honest server. We next describe the interaction in the ideal world assuming the server is honest. Let $\mathcal{A}$ be an adversary corrupting a subset $\mathcal{I} \subseteq [n]$ of the users, and which does *not* corrupt the server.

Inputs: Each party holds the security parameter 1^κ and the number of users n (held in binary by the users and in unary by the server). Additionally, the server holds input x_0, user P_i holds x_i, and the adversary is given auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Parties send inputs to trusted party: The honest parties send their inputs to the trusted party. For each corrupted user, the adversary $\mathcal{A}$ sends to the trusted party some value from their domain as input. Denote by $(x'_1, \ldots, x'_n)$ the tuple of inputs received by the trusted party.

Trusted party sends output to the server: The trusted party computes $y \leftarrow f(x'_1, \ldots, x'_n)$ with uniformly random coins and sends the output y to the server.

Outputs: An honest server outputs the value sent by the trusted party, and a corrupted server outputs nothing. Additionally, all users output nothing and $\mathcal{A}$ outputs a function of its view (its inputs and the auxiliary input aux).