Communication Locality in Secure Multi-Party Computation

How to Run Sublinear Algorithms in a Distributed Setting

Elette Boyle[1], Shafi Goldwasser[2], and Stefano Tessaro[1]

[1] MIT CSAIL, eboyle@mit.edu, tessaro@csail.mit.edu
[2] MIT CSAIL and Weizmann, shafi@theory.csail.mit.edu

Abstract. We devise multi-party computation protocols for general secure function evaluation with the property that *each party is only required to communicate with a small number of dynamically chosen parties*. More explicitly, starting with n parties connected via a complete and synchronous network, our protocol requires each party to send messages to (and process messages from) at most $\mathsf{polylog}(n)$ other parties using $\mathsf{polylog}(n)$ rounds. It achieves secure computation of any polynomial-time computable randomized function f under cryptographic assumptions, and tolerates up to $(\frac{1}{3} - \epsilon) \cdot n$ statically scheduled Byzantine faults.
We then focus on the particularly interesting setting in which the function to be computed is a *sublinear algorithm*: An evaluation of f depends on the inputs of at most $q = o(n)$ of the parties, where the identity of these parties can be chosen randomly and possibly adaptively. Typically, $q = \mathsf{polylog}(n)$. While the sublinear query complexity of f makes it possible in principle to dramatically reduce the *communication complexity* of our general protocol, the challenge is to achieve this while maintaining security: in particular, while keeping the *identities* of the selected inputs completely hidden. We solve this challenge, and we provide a protocol for securely computing such sublinear f that runs in $\mathsf{polylog}(n) + O(q)$ rounds, has each party communicating with at most $q \cdot \mathsf{polylog}(n)$ other parties, and supports *message sizes* $\mathsf{polylog}(n) \cdot (\ell + n)$, where ℓ is the parties' input size.
Our optimized protocols rely on a multi-signature scheme, fully homomorphic encryption (FHE), and simulation-sound adaptive NIZK arguments. However, we remark that multi-signatures and FHE are used to obtain our bounds on message size and round complexity. Assuming only standard digital signatures and public-key encryption, one can still obtain the property that each party only communicates with $\mathsf{polylog}(n)$ other parties. We emphasize that the scheduling of faults can depend on the initial PKI setup of digital signatures and the NIZK parameters.

1 Introduction

Multiparty computation (MPC) protocols for secure function evaluation (SFE) witnessed a significant body of work within the cryptography research community in the last 30 years.

This research was initiated and done in part while the authors were visiting the Isaac Newton Institute for Mathematical Sciences in Cambridge, UK.

These days, an emerging area of potential applications for secure MPC is to address privacy concerns in data aggregation and analysis, to match the explosive current growth of available data. Large data sets, such as medical data, transaction data, the web and web access logs, or network traffic data, are now in abundance. Much of the data is stored or made accessible in a distributed fashion. This necessitated the development of efficient distributed protocols to compute over such data. In order to address the privacy concerns associated with such protocols, cryptographic techniques such as MPC for SFE where *data items are equated with servers* can be utilized to prevent unnecessary leakage of information.

However, before MPC can be effectively used to address today's challenges, we need protocols whose efficiency and communication requirements scale practically to the modern regime of massive data. An important metric that has great effect on feasibility but has attracted surprisingly little attention thus far is the *number of other parties* that each party must communicate with during the course of the protocol. We refer to this as the communication *locality*. Indeed, if we consider a setting where potentially hundreds of thousands, or even millions of parties are participating in a computation over the internet, requiring coordination between each pair of parties will be unrealistic.

In this work, we work to optimize the communication locality for general secure function evaluation on data which is held distributively among n parties. These parties are connected via a complete synchronous communication network, of whom $(\frac{1}{3} - \epsilon)n$ may be statically scheduled, computationally bounded Byzantine faults. We do *not* assume the existence of broadcast channels.

We also focus on a particularly interesting setting in which the randomized function f to be computed is a *sublinear algorithm*: namely, a random execution of $f(x_1, ..., x_n)$ depends on at most $q = o(n)$ of the inputs x_i. We consider both non-adaptive and adaptive sublinear algorithms, in which the identities of the selected inputs may depend on the randomness r of execution, or on both r and the values of x_i queried thus far. Sublinear algorithms play an important role in efficiently testing properties and trends when computing on large data sets. The sublinear query complexity makes it possible in principle to dramatically reduce the *amount* of information that needs to be communicated within the protocol. However, the challenge is to achieve this while maintaining security—in particular, keeping the identities of the selected inputs completely hidden.

Straightforward application of known general MPC techniques results in protocols where each party sends and receives messages from all n parties, and where the overall communication complexity is $O(n^2)$, regardless of the complexity of the function to be computed. We remark that this is obviously the case for the classical general SFE protocols (beginning with [26, 14, 5]) in which every party first secret shares its input among all other parties (and exchanges messages between all n parties at the evaluation of every gate of the circuit of the function computed). Furthermore, although much progress was made in the MPC literature of the last two decades to make MPC protocols more efficient and suitable for practice, this is still the case both in works on scalable MPC [17, 20, 19, 18]

and more recent works utilizing the existence of fully homomorphic encryption schemes [35, 3] for MPC. The latter achieve communication complexity that is independent of the circuit size, but not of the number of parties when broadcast channels are not available.

A recent notable exception to the need of each party to communicate with all other parties is the beautiful work of King, Saia, Sanwalani and Vee [34] on what they call scalable protocols for a relaxation of the Byzantine agreement and leader election problems. Their protocols require each honest party to send and process a $\mathsf{polylog}(n)$ number of bits. On the down side, the protocols of [34] do not guarantee that all honest parties will achieve agreement, but only guarantee that $1 - o(1)$ fraction of the good processors reach agreement—achieving only so-called *almost everywhere agreement*. In another work of King et al [32], it is shown how using $\tilde{O}(\sqrt{n})$ communication, full Byzantine agreement can be achieved. The technique of almost-everywhere leader election of [34] will be the technical starting point of our work.

1.1 Our Results

We provide multiparty computation protocols for general secure function evaluation with communication locality that is *polylogarithmic* in the number of parties. That is, starting with n parties connected via a complete and synchronous network, we prove the following main theorem:

> **Theorem 1.** Let f be any polynomial-time randomized functionality on n inputs. Then, for every constant $\epsilon > 0$, there exists an n-party protocol Π_f that securely computes a random evaluation of f, tolerating $t < (1/3 - \epsilon)n$ statically scheduled active corruptions, with the following complexities:
> (1) Communication locality: $\mathsf{polylog}(n)$.
> (2) Round complexity: $\mathsf{polylog}(n)$.
> (3) Message sizes: $O(n \cdot l \cdot \mathsf{polylog}(n))$, where $l = |x_i|$ is the individual input size.
> (4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{polylog}(n)$, as well as a $\mathsf{polylog}(n)$-long additional common random string (CRS).[3]
>
> The protocol assumes a secure multisignature scheme, a fully homomorphic encryption (FHE) scheme, simulation-sound NIZK arguments, as well as pseudorandom generators.
> Assuming *only* a standard signature scheme and semantically secure public-key encryption, and setup as in (4), there exists a protocol for securely computing f with $\mathsf{polylog}(n)$ communication locality.

Multisignatures [39, 36] are digital signatures which enable the verification that a large number of signers have signed a given message, where the number

[3] Adversarial corruptions may be made as a function of this setup information.

of signers is not fixed in advance. The size of a multisignature is independent of the number of signers, but in order to determine their identities one must attach identifying information to the signature. Standard instantiations of such schemes exist under the bilinear computational Diffie-Hellman assumption [44, 36].

The use of multisignatures rather than standard digital signatures enables us to bound the *size* of the messages sent in the protocol. Further, the use of FHE enables us to bound the *number* of messages sent, rather than depend on the time complexity of the function f to be computed and polynomially on the input size. However, we can obtain the most important feature of our complexity, the need of every party to send messages to (and process messages from) only $\mathsf{polylog}(n)$ parties in the network, solely under the assumption that digital signatures and public-key encryption exist.

In addition, we show how to convert an arbitrary sublinear algorithm with query complexity $q = \mathsf{polylog}(n)$ into a multi-party protocol to evaluate a randomized run of the algorithm with $\mathsf{polylog}(n)$ communication locality and rounds, and where the *total communication complexity* sent by each party is only $O(\mathsf{polylog}(n) \cdot (l + n))$ for $l = |x|$ an individual input size. We prove that participating in the MPC reveals no information beyond the output of the sublinear algorithm execution using a standard Ideal/Real simulation-based security definition.

For underlying query complexity q, our second main theorem is as follows:

> **Theorem 2.** Let SLA be a sublinear algorithm which retrieves $q = q(n) = o(n)$ different inputs. Then, for all constant $\epsilon > 0$, there exists an n-party protocol Π_{SLA} that securely computes an execution of the sublinear algorithm SLA tolerating $t < (1/3 - \epsilon)n$ statically scheduled active corruptions, with the following complexities, where l is the size of the individual inputs held by the parties:
> (1) Communication locality: $q \cdot \mathsf{polylog}(n)$.
> (2) Round complexity: $O(q) + \mathsf{polylog}(n)$.
> (3) Message sizes: $O((l + n) \cdot \mathsf{polylog}(n))$.
> (4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{polylog}(n)$, as well as a $\mathsf{polylog}(n)$-long additional CRS.
> The protocol assumes a secure multisignature scheme, an FHE scheme, simulation-sound NIZK arguments, and pseudorandom generators.

Techniques. We first describe how to achieve our second result, for the case when f is a sublinear algorithm. This setting requires additional techniques in order to attain the communication complexity gains. After this, we describe the appropriate modifications required to maintain $\mathsf{polylog}(n)$ communication locality for general functions f.

There are three main technical components to our protocol for sublinear algorithms. The first is to set up a committee structure constituted of a *supreme committee* C and n *input committees* $C_1, ..C_n$. These committees will all be of size $\mathsf{polylog}(n)$ and with high probability have a 2/3 majority of honest parties. Each committee C_i will (to begin with) hold shares of the input x_i whereas the

role of the supreme committee will essentially be to govern the running of the protocol. A major challenge is to ensure that all parties in the network know the identity of parties in all the committees. The starting point to address this challenge is to utilize the communication-efficient *almost-everywhere* leader election protocol of [34]. We remark that [34] achieves better total communication complexity of $\mathsf{polylog}(n)$ bits and offers unconditional results, but only achieves an almost-everywhere agreement: there may be a $o(1)$ fraction of honest parties who will not reach agreement and, in our context, will not know the makeup of the committees. The main idea to remedy this situation is to add an iterated certification procedure using multi-signatures to the protocol of [34], while keeping the complexity of only $\mathsf{polylog}(n)$ messages sent and processed by any honest party. In the process, however, we move from unconditional to computational security and our message sizes grow, as they will be signed by multi-signatures. Whereas the size of the multi-signatures depends only on the security parameter, the messages should indicate the identities of the signers – this is cause for the increased size of messages.

The second component is to implement a randomly chosen secret reshuffling ρ of parties' inputs within the complexity restrictions we have alloted. At the end of the shuffling, committee $C_{\rho(i)}$ will hold the input of committee C_i. Informally, this will address the major privacy issue in executing a sublinear algorithm in a distributed setting, which is to ensure that the adversary does not learn which of the n inputs are used by the algorithm. We implement the shuffling via distributed evaluation of a switching network with very good mixing properties under random switching, all under central coordination by the supreme committee. We assume that a fixed switching network over n wires is given, with depth $d = \mathsf{polylog}(n)$, and is known to everyone.

The third component, once the inputs will be thus permuted, is to actually run the execution of the sublinear algorithm. For lack of space, let us illustrate how this is done for the sub class of non-adaptive sublinear algorithms. This is a class of algorithms that proceed in two steps:

- First, a random subset I of size q of the indices $1, ..., n$ is selected.
- Second, an arbitrary polynomial-time algorithm is computed on inputs x_j for $j \in I$.

To run an execution of such an algorithm, the supreme committee: first selects a random and secret $q = \mathsf{polylog}(n)$ size subset I of the inputs; and second, runs a secure function evaluation (SFE) protocol on the set of inputs in $\rho(I)$ with the assistance of parties in committees C_j for $j \in \rho(I)$. In the adaptive case, one essentially assumes queries are asked in sequence, and executes in a similar way the sublinear algorithm query after query, contacting committee $\rho(i)$ for each query i, instead of parallelizing the computation for all inputs from I. The price to pay is an additive factor q in the number of rounds of the protocol. However, note that in the common case $q = \mathsf{polylog}(n)$, this does not affect the overall asymptotic complexity.

Now, consider the case when f is a general polynomial-time function, whose evaluation may depend on a large number of its inputs. In this case, we can skip

the aforementioned shuffling procedure, and instead simply have *each party* P_i send his (encrypted) input up to the supreme committee C to run the evaluation of f. That is, each P_i gives an encryption of his input to the members of his input committee C_i, and each party in C_i sends the ciphertext up to C via a communication tree that is constructed during the process of electing committees (in Step 1). Then, the members of the supreme committee C (who collectively have the ability to decrypt ciphertexts) are able to evaluate the functionality f directly via a standard SFE.

Remarks. A few remarks are in order.

- *Flooding by faulty parties.* There is no limit (nor can there be) on how many messages are sent by faulty parties to honest parties, as is the case in the works mentioned above. To address this issue in [34, 32, 33, 21], for example, it is (implicitly) assumed that the authenticated channels between parties can "recognize" messages from unwarranted senders which should not be processed and automatically drop them, whereas we will use a digital signature verification procedure to recognize and drop these messages which should not be processed.
- *Security definition for sublinear algorithms.* The security definition we achieve is the standard definition of secure multiparty computation (MPC). Informally, the parties will receive the output corresponding to a random execution of the sublinear algorithm but nothing else. Formally, we use the ideal/real simulation-based type definition. We note that in works of [29, 23, 31] on MPC for approximation algorithms for functions f, privacy is defined so as to mean that no information is revealed beyond the *exact* value of f, rather than beyond the approximate value of f computed by the protocol. One may ask for a similar privacy definition for sublinear algorithms, which are an approximation algorithm of sorts. However, this is an orthogonal concern to the one we address in this work.

1.2 Further Related Work

Work on MPC in partially connected networks, such as the recent work of Chandran, Garay and Ostrovsky [12, 13], shows MPC protocols for network graphs of degree $\mathsf{polylog}(n)$ (thus each party is connected to no more than $\mathsf{polylog}(n)$ parties). They can only show how to achieve MPC amongst all but $o(n)$ honest parties. Indeed, in this setting it is unavoidable for some of the honest parties to be cut out from every other honest party. In contrast, in the present work, we assume that although the n parties are connected via a complete network and potentially any party can communicate with any other party, our protocols require each honest party to communicate with only at most $\mathsf{polylog}(n)$ parties whose identity is only determined during the course of the protocol execution.

The problem of sublinear communication in MPC has also been considered in the realm of *two-party* protocols, e.g. by [40] who provide communication-preserving protocols for secure function evaluation (but which require super-polynomial computational effort), and in a recent collection of works including

[28] which achieve amortized sublinear time protocols, and the work of [31] which show polylogarithmic communication for specific functions.

An interesting point of comparison to our result is the work of Halevi, Lindell and Pinkas [30]. They design computationally secure MPC protocols for n parties in which one party is singled out as a server and all other parties communicate directly with the server in sequence (in one round of communication each). However, it is easy to see that protocols in this model can only provide a limited privacy guarantee: for example, as pointed out by the authors, if the last i parties collude with the server then they can always evaluate the function on as many input settings as they wish for variable positions $n-i, n-i+1, \ldots, n$. No such limitations exist in our model.

In a recent and independent work to the current paper, King et al [21] extends [32] to show a protocol for unconditionally secure SFE for general f that requires every party to send at most $O(\frac{m}{n}+\sqrt{n})$ messages, where m is the size of a circuit representation of f. A cursory comparison to our work shows that in [21] each party sends messages to $\Omega(\sqrt{n})$ other parties.

Finally, let us point out that our approach to anonymize access patterns to parties is similar in spirit to problems arising in the context of Oblivious RAM [27], and uses similar ideas to the obfuscated secret shuffling protocols of Adida and Wilkström [2].

2 Preliminaries

We recall first the definitions of standard basic tools used throughout the paper, and then move to some important results on shuffling and our notation for sublinear algorithms.

2.1 Basic Tools

Non-Interactive Zero Knowledge. We make use of a standard non-interactive zero knowledge (NIZK) argument system $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{Proof}}))$ with unbounded adaptive simulation soundness, as defined in [22, 6, 7]. That is, soundness of the argument system holds even against PPT adversaries who are given access to an oracle that produces *simulated* proofs of (potentially false) statements. For a formal definition, we refer the reader to, e.g., [22, 6, 7].

Theorem 1. *[42] There exists an unbounded simulation-sound NIZK proof system for any* NP *language L, based on trapdoor one-way permutations, with proof length* $\mathsf{poly}(|x|, |w|)$*, where x is the statement and w is the witness.*

Fully Homomorphic Encryption. We make use of a fully homomorphic public-key encryption (FHE) scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ as defined in, e.g., [25]. For our purposes, we require an FHE scheme with the additional property of *certifiability*. A certifiable FHE scheme is associated with a set R of "good" encryption randomness such that (repeated execution of) the Eval algorithm and the decryption algorithm Dec are correct on ciphertexts derived from those using randomness from R to encrypt. A formal definition follows.

Definition 1. *For a given subset $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ of possible randomness values, we (recursively) define the class of* R-evolved *ciphertexts with respect to a public key* pk *to include all ciphertexts c of the form:*

- $c = \mathsf{Enc}_{\mathsf{pk}}(m; r)$ *for some m in the valid message space and randomness $r \in R$, and*
- $c = \mathsf{Eval}_{\mathsf{pk}}((c_i)_{i \in I}, f)$ *for some $\mathsf{poly}(k)$-size collection of R-evolved ciphertexts $(c_i)_{i \in I}$ and some poly-size circuit f.*

Definition 2. *A FHE scheme is said to be* certifiable *if there exists a subset $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ of possible randomness values for which the following hold.*

1. $\Pr[r \in R] = 1 - \mathsf{negl}(k)$*, where the probability is over uniformly sampled $r \leftarrow \{0,1\}^{\mathsf{poly}(k)}$.*
2. *There exists an efficient algorithm $\mathcal{A}_R$ such that $\mathcal{A}_R(r) = 1$ for $r \in R$ and 0 otherwise.*
3. *With overwhelming probability,* Gen *outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$ *such that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}((c_i)_{1 \le i \le n}, f)) = f((x_i)_{1 \le i \le n})$ for all poly-sized circuits f and for all R-evolved ciphertexts $c_1, \ldots, c_n$, where $x_i = \mathsf{Dec}_{\mathsf{sk}}(c_i)$.*

Certifiable FHE schemes have been shown to exist based on the Learning with Errors assumption, together with a circular security assumption (e.g., Brakerski and Vaikuntanathan [10] and Brakerski, Gentry, and Vaikuntanathan [9]). For the readers who are familiar with these constructions, the set of "good" certifying randomness R corresponds to encrypting with sufficiently "small noise."

Multisignatures. A multisignature scheme is a digital signature scheme with the ability to combine signatures from multiple signers on the same message into a single short object (a *multisignature*).[4] The first formal treatment of multisignatures was given by Micali, Ohta, and Reyzin [39].

Definition 3. *A* multisignature *scheme is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Combine}, \mathsf{MultiVerify})$*, where syntactically* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *are as in a standard signature scheme, and* $\mathsf{Combine}, \mathsf{MultiVerify}$ *are as follows:*

$\mathsf{Combine}(\{\{\mathsf{vk}_j\}_{j \in J_i}, \sigma_i\}_{i=1}^{\ell}, m)$**:** *For disjoint $J_1, \ldots, J_\ell \subseteq [n]$, takes as input a collection of signatures (or multisignatures) σ_i with respect to verification keys vk_j for $j \in J_i$, and outputs a combined multisignature, with respect to the union of verification keys.*

$\mathsf{MultiVerify}(\{\mathsf{vk}_i\}_{i \in I}, m, \sigma)$**:** *Verifies multisignature σ with respect to the collection of verification keys $\{\mathsf{vk}_i\}_{i \in I}$. Outputs 0 or 1.*

All algorithms satisfy the standard natural correctness properties, except with negligible probability. Moreover, the scheme is secure if for any PPT adversary $\mathcal{A}$, the probability that the challenger outputs 1 in the following game is negligible in the security parameter k:

[4] Note that multisignatures are a special case of *aggregate* signatures [8], which in contrast allow combining signatures from n different parties on n *different* messages.

Setup. *The challenger samples n public key-secret key pairs,* $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^k)$ *for each* $i \in [n]$*, and gives* $\mathcal{A}$ *all verification keys* $\{\mathsf{vk}_i\}_{i\in[n]}$*.* $\mathcal{A}$ *selects a proper subset* $M \subset [n]$ *(corresponding to parties to corrupt) and receives the corresponding set of secret signing keys* $\{\mathsf{sk}_i\}_{i\in M}$*.*

Signing queries. $\mathcal{A}$ *may issue multiple adaptive signature queries, of the form* (m, i)*. For each such query, the challenger responds with a signature* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}_i}(m)$ *on message* m *with respect to the signing key* sk_i*.*

Output. $\mathcal{A}$ *outputs a triple* $(\bar{\sigma}^*, m^*, I^*)$*, where* $\bar{\sigma}^*$ *is an alleged forgery multisignature on message* m^* *with respect to a subset of verification keys* $I^* \subset [n]$*. The challenger outputs 1 if there exists* $i \in I^* \setminus M$ *such that the message* m^* *was not queried to the signature oracle with key* sk_i*, and* $1 \leftarrow \mathsf{MultiVerify}(\{\mathsf{vk}_i\}_{i\in I^*}, m^*, \sigma^*)$*.*

The following theorem follows from a combination of the (standard) signature scheme of Waters [44] together with a transformation from this scheme to a multisignature scheme due to Lu et. al. [36].

Theorem 2. *[44, 36] There exists a secure multisignature scheme with signature size* $\mathsf{poly}(k)$ *(independent of message length and number of potential signers), based on the Bilinear Computational Diffie-Hellman assumption.*

Multi-party protocols: Model and Security Definitions. We consider the setting of n parties $\mathcal{P} = \{P_1, ..., P_n\}$ within a synchronous network who wish to jointly compute any PPT function f over their private inputs. We allow up to t statically chosen Byzantine (malicious) faults and a rushing adversary. In our protocols below, we consider $t \leq (\frac{1}{3} - \epsilon)n$ for any constant $\epsilon > 0$. We assume that every pair of parties has the ability to initiate direct communication via a point-to-point private, authenticated channel. (However, we remark that in our protocol, each (honest) party will only ever send or process information along subset of only $\mathsf{polylog}(n)$ such channels.) We assume the existence of a public-key infrastructure, but allow the adversary's choice of corruptions to be made as a function of this public information.

The notion of security we consider is the standard simulation-based definition of secure multiparty computation (MPC), via the real/ideal world paradigm. Very loosely, we require that for any PPT adversary $\mathcal{A}$ in a real-world execution of the protocol, there exists another PPT adversary who can simulate the output of $\mathcal{A}$ given only access to an "ideal" world where he learns only the evaluated function output. We refer the reader to, e.g., [11] for a formal definition of (standalone) MPC security.

General secure function evaluation. The following theorem is well known and will be use throughout this paper. Let $\mathbb{C}$ be a circuit with n inputs, and let $F_{\mathbb{C}}$ the functionality that computes the circuit.

Theorem 3. *[5] For any* $t < n/3$*, there exists a protocol that securely computes the functionality* $F_{\mathbb{C}}$ *functionality, with* perfect *security. The protocol proceeds in* $O(|\mathbb{C}|)$ *rounds, and each party sends* $\mathsf{poly}(n)$ *messages of size* $\mathsf{poly}(k, n)$ *each.*

Verifiable Secret Sharing. A secret sharing scheme is a protocol that allows a dealer who holds a secret input s, to share his secret among n parties such that any t parties do not gain any information about the secret s, but any set of (at least) $t+1$ parties can reconstruct s. A *verifiable secret sharing* (VSS) scheme, introduced by Chor et al. [15], is a secret sharing scheme with the additional guarantee that after the sharing phase, a dishonest dealer is either rejected, or is committed to a single secret s, that the honest parties can later reconstruct, even if dishonest parties do not provide their correct shares.

For concreteness, we consider a class of VSS constructions that takes advantage of reconstruction and secrecy properties of low-degree polynomials [43, 38]. In particular, security of such a VSS protocol Share is formalized as emulating the ideal functionality F_{VSS}^t for parties $P_D, P_1, ..., P_n$ with distinguished dealer P_D such that $F_{\mathsf{VSS}}(q, (\emptyset, ..., \emptyset)) = \big(\emptyset, (q(\alpha_1), ..., q(\alpha_n))\big)$ for fixed evaluation points $\alpha_1, \ldots, \alpha_n$ if $\deg(q) \leq t$, and $F_{\mathsf{VSS}}(q, (\emptyset, ..., \emptyset)) = (\emptyset, (\perp, ..., \perp))$ otherwise. The party can also run a *reconstruction protocol* $\mathsf{Reconst}$ such that if honest parties input the correct shares output by the above functionality to them, then they recover the right value. The following result is well known.

Theorem 4. *[5, 4] For any $t < n/3$, there exists a constant-round protocol* Share *that securely computes the F_{VSS}^t functionality, with* perfect *security. Each party sends* $\mathsf{poly}(n)$ *messages of size $O(l \log l)$, where $l = \max\{|x|, n\}$.*

Also, we will be interested in the case where the dealer D can be any of the n parties, and he sends shares to a subset P' of the n parties of size n' (e.g., $n' = \mathsf{polylog}(n)$), and we may not necessarily have $D \in P'$. The above functionality can be extended to this case naturally, and it is a folklore result that the protocols given by the above theorem also remain secure in this case as long as less than a fraction $1/3$ of the parties in P' are corrupted.

Broadcast. Another important functionality we need to implement is broadcast. To define, a broadcast protocol can be seen as an example of an MPC implementing a functionality F_{BC} for parties $P_D, P_1, ..., P_n$ with distinguished dealer P_D, defined as $F_{\mathsf{BC}}(m, (\emptyset, ..., \emptyset)) = \big(\emptyset, (m, \ldots, m)\big)\big)$, where m is the message to be broadcast.

Theorem 5. *[24] For any $t < n/3$, there exists a constant-round protocol that securely computes the F_{BC} functionality, with* perfect *security. Each party sends* $\mathsf{poly}(n)$ *messages of size $O(|m|)$ each.*

2.2 Random Switching Networks and Random Permutations.

Our protocol will employ what we call an *n-wire switching network*, which consists of a sequence of *layers*, each layer in turn consisting of one or more swapping gates which decide to swap the values of two wires depending on a bit. Formally, given an input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ (which we assume to be integers wlog), a swap gate operation $\mathsf{swap}(i, j, \boldsymbol{x}, b)$ returns $\boldsymbol{x'}$, where if $b = 0$ then $\boldsymbol{x} = \boldsymbol{x'}$, and if $b = 1$ then we have $x'_i = x_j$, $x'_j = x_i$, and $x'_k = x_k$ for all $k \neq i, j$. A switching

layer is a set $L = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ of pairwise-disjoint pairs of distinct indices of $[n]$. A d-depth switching network is a list $SN = (L_1, \ldots, L_d)$ of switching layers. Note that for each assignment of the bits of the gates in SN, the network defines a permutation from $[n]$ to $[n]$ by inputting the vector $\boldsymbol{x} = (1, 2, \ldots, n)$ to the network. The question we are asking is the following: If we set each bit in each swap gate uniformly and independently at random, how close to uniform is the resulting permutation? The following theorem guarantees the existence of a sufficiently shallow switching network giving rise to an almost-uniform random permutation.

Theorem 6. *For all $c > 1$, there exists an efficiently computable n-wire switching network of depth $d = O(\mathsf{polylog}(n) \cdot \log^c(k))$ (and size $O(n \cdot d)$) such that the permutation $\widehat{\pi} : [n] \to [n]$ implemented by the network when setting swaps randomly and independently has negligible statistical distance (in k) from a uniformly distributed random permutation on $[n]$.*

Proof. By Theorem 1.11 in [16], there exists such network SN of depth $d = O(\mathsf{polylog}(n))$ where the statistical distance is of the order $O(1/n)$. Consider now the switching network SN' obtained by cascading r copies of SN. Then, when setting switching gates at random, the resulting permutation $\widehat{\pi}$ equals $\widehat{\pi}_1 \circ \cdots \circ \widehat{\pi}_n$, where $\widehat{\pi}_i$ are independent permutations obtained each by setting the gates in SN uniformly at random. With π being a random permutation, a well-known property of the statistical distance $\Delta(\cdot, \cdot)$, combined with the fact permutation composition gives a group (see e.g. [37] for a proof) yields

$$\Delta(\widehat{\pi}, \pi) \leq 2^{r-1} \cdot \prod_{i=1}^{r} \Delta(\widehat{\pi}_i, \pi) \leq O\left(\left(\frac{2}{n}\right)^r\right) \leq O(2^{r(\log 2 - \log(n))}) \,,$$

which is negligible in k for $r = \log^c(k)$. □

Note that in particular this means that each wire is connected to at most $d = O(\mathsf{polylog}(n) \cdot \log^c(k))$ other wires via a switching gates, as each wire is part of at most one gate per layer.

2.3 Sublinear algorithms

We consider a model where n inputs $x_1, \ldots, x_n$ are accessible to an algorithm SLA via individual queries for indices $i \in [n]$. Formally, a *Q-query algorithm* in the n-input model is a tuple of (randomized) polynomial time algorithms $\mathsf{SLA} = (\mathsf{SLA.Sel}_1, \mathsf{SLA.Sel}_2, \ldots, \mathsf{SLA.Sel}_Q, \mathsf{SLA.Exec})$. During an execution with inputs $(x_1, \ldots, x_n)$, $\mathsf{SLA.Sel}_1$ takes no input and produces as output a state σ_1 and a query index $i_1 \in [n]$, and for $j = 2, \ldots, n$, $\mathsf{SLA.Sel}_j$ takes as input a state σ_{j-1} and input $x_{i_{j-1}}$, and outputs a new state σ_j and a new query index i_j. Finally, $\mathsf{SLA.Exec}$ takes as input σ_Q and x_Q, and produces a final output y. We say that SLA is *sublinear* if $Q = o(n)$. We will also consider the special case of *non-adaptive* algorithms which consist without loss of generality of only two randomized algorithms $\mathsf{SLA} = (\mathsf{SLA.Sel}, \mathsf{SLA.Exec})$, where $\mathsf{SLA.Sel}$ outputs a

subset $I \subseteq [n]$ of indices of inputs to be queried, and the final output is obtained by running $\mathsf{SLA.Exec}$ on input $(x_i)_{i \in I}$.

Examples of sublinear algorithms, many of them non-adaptive, include algorithms for property testing such as testing sortedness of the inputs, linearity, approximate counting, and numerous graph properties, etc. Surveying this large area and the usefulness of these algorithms goes beyond the scope of this paper, and we refer the reader to the many available surveys [1].

3 Multi-Party Computation for Sublinear Algorithms

We present a high-level overview geared at illustrating the techniques used within our sublinear algorithm compiler (Theorem 2), which is the more involved of our two results. For exposition, we focus on the case of non-adaptive algorithms. Given a Q-query non-adaptive sublinear algorithm SLA, we would like to evaluate it in a distributed fashion along the following lines. First, a small committee C consisting of $\mathsf{polylog}(n)$ parties is elected, with the property that at least two thirds of its members are honest. This committee then jointly decides on a random subset of Q parties I, output by $\mathsf{SLA.Sel}$, from which inputs are obtained. The parties in $C \cup I$ jointly execute a multi-party computation among themselves to produce the output of the sublinear algorithm according to the algorithm $\mathsf{SLA.Exec}$, which is then broadcasted to all parties.

But things will not be as simple. Interestingly, one main challenge is very unique to the setting of sublinear algorithms: An execution of the protocol needs to hide the subset I of parties whose inputs contribute to the output! More precisely, an ideal execution of the sublinear algorithm via the functionality $\mathcal{F}_{\mathsf{SLA}}$ only reveals the output of the sublinear algorithm. Therefore, we need to ensure that the adversary does not learn any additional information about the composition of I from a protocol execution beyond what leaked via the final output. Our protocol will indeed hide the set I completely. This will require modifying the above naive approach considerably.

The second challenge is complexity theoretic in nature. Enforcing low complexity of our protocol when implementing the above steps, while realizing our mechanism to hide the subset I, will turn out to be a delicate balance act.

In particular, at a high level our protocol will consist of the following components:

Committee election phase. The n parties jointly elect a supreme committee C, as well as individual committees $C_1, \ldots, C_n$ on which they *all agree*, sending each at most $\mathsf{polylog}(n)$ messages of size each $n \cdot \mathsf{poly}(\log n, \log k)$. All committees have size $\mathsf{polylog}(n)$ and at least a fraction $2/3$ of the parties in them are honest. As part of this process, the parties set up a communication structure that allows the supreme committee to communicate messages to all parties.

Commitment phase. Each party P_i commits to its input so that C_i holds shares of these inputs.

Shuffling phase. To hide the access pattern of the algorithm (i.e., which inputs are included in the computation), the committees will randomly shuffle the inputs they hold with respect to a random permutation ρ. This will happen by using a switching network with good shuffling properties. For each swap gate (i, j) in the switching network, committees C_i and C_j will swap at random the sharings they hold via a multi-party computation under a random decision taken by the supreme committee C. The supreme committee then holds a secret sharing of ρ.

Evaluation phase. The parties in the supreme committee C sample a random query set I according to $\mathsf{SLA.Sel}$ via MPC and learn $\rho(I)$ only. They will then include the parties in committees C_i for $i \in \rho(I)$ in a multi-party computation to evaluate the sublinear algorithm on the inputs they hold. (Recall that C holds ρ in shared form.)

Output phase. The supreme committee broadcasts the output of the computation to all parties, using the communication structure from the first stage.

In addition, we carefully implement sharings and multi-party computations using FHE to improve complexity, making the dependency of both the communication and round complexities linear in the input length $|x|$, rather than polynomial, and independent of the circuit sizes to implement the desired functionalities.

The following paragraphs provide a more detailed account of the techniques used within our protocol. In addition, a high-level description of the protocol procedure is given in Figure 1.

Committee election phase. The backbone behind this first phase is given by the construction of a *communication tree* using a technique of King et al [34]. Such tree is a sparse communication subnetwork which will ensure both the election of the supreme committee, as well as a basic form of communication between parties and the supreme committee where each party communicates only with $\mathsf{polylog}(n)$ other parties and only $\mathsf{polylog}(n)$ rounds of communication are required. Informally, the protocol setting up the tree assigns (possibly overlapping) subsets of parties of polylogarithmic size to the nodes of a tree with polylogarithmic height and logarithmic degree. The set of parties assigned to the root will take the role the supreme committee C. Communication from the root to the parties (or the other way round) occurs by communicating messages over paths from the root to the leaves of the tree, with an overall communication cost of $\mathsf{polylog}(n)$ messages per party. To elect the committees $C_1, \ldots, C_n$, we can have the supreme committee agree on the seed s of a PRF family $\mathcal{F} = \{F_s\}_s$ via a coin tossing protocol, where F_s maps elements of $[n]$ to subsets of $[n]$ of size $\mathsf{polylog}(n)$, and send s to all parties. We then let $C_i = F_s(i)$.

However, a closer look reveals that it is only possible for the protocol building the communication tree to enforce that a vast majority of the nodes of the tree are assigned to a set of parties for which a $2/3$ majority is honest, but some nodes are unavoidably associated with too large a fraction of corrupted parties. Indeed, some parties may be connected to too many bad nodes and their communication ends up being essentially under adversarial control. As a

consequence, the supreme committee is only able to correctly communicate with a $1 - o(1)$ fraction of the (honest) parties. Moreover, individual parties are not capable of determining whether the value they hold is correct or not. We refer to this situation as *almost-everywhere (ae) agreement*.

Our main contribution here is the use of cryptographic techniques to achieve *full agreement* on C and s in this stage, while maintaining $\mathsf{polylog}(n)$ communication locality; this improves on previous work in the information-theoretic setting [32, 33, 21] which requires each party to talk to $O(\sqrt{n} \cdot \mathsf{polylog}(n))$ other parties to reach agreement. We tackle these two issues in two separate ways.

1. *From ae agreement to ae certified agreement.* We first move to a stage where a large $1 - o(1)$ fraction of the parties learn the value sent by the supreme committee, together with a *proof* that the output is the one sent by the committee, whereas the remaining parties who do not know the output are also aware of this fact. We refer to this scenario as *almost-everywhere certified agreement*. Let us start with the basic idea using traditional signatures (we improve on this below using multisignatures). After having the supreme committee send a value m to all parties with almost-everywhere agreement, each party P_i receiving a value m_i will *sign* m_i with his own signing key, producing a signature σ_i. Then, P_i sends (m_i, σ_i) up the tree to the supreme committee, and each member will collect at least $n/2$ signatures on σ_i on some message m. Note that this will always be possible, as a fraction $1 - o(1) > n/2$ of the honest parties will receive the message $m_i = m$ and send a valid signature up the tree. Moreover, the adversary would need to forge signatures for honest parties in order to produce a valid certificate for a message which was not broadcast by the supreme committee.
2. *From ae certified agreement to full agreement.* We finally describe a transformation from ae certified agreement to full agreement. If a committee wants to broadcast m to all parties, the committee additionally generates a seed s for a PRF and broadcasts (m, s) in a certified way using the above transformations. Each party i receiving (m, s) with a valid certificate π forwards (m, s, π) to all parties in "his" committee $F_s(i)$. Whenever a party receives (m, s, π) with a valid certificate, it stops and outputs m. Note that no party sends more than $\mathsf{polylog}(n)$ additional messages in this transformation. Moreover, it is not hard to see that with very high probability every honest party will be in at least one of the $F_s(i)$ for a party i who receives (m, s) correctly with a certificate, by the pseudorandomness of $\mathcal{F}$. Note in particular that the same seed s can be used over multiple executions of this broadcast procedure from the committee to the parties, and can be used directly to generate the committees $C_1, \ldots, C_n$.

While we do guarantee that every party *sends* at most $\mathsf{polylog}(n)$ messages, a problem of the above approach is the potentially high complexity of processing incoming messages if dishonest parties flood an honest party by sending too many messages. Namely, the $t = \Theta(n)$ corrupted parties can always each send (m, s) with an invalid certificate to some honest party P_i, who needs to verify all signatures in the certificate to confirm that these messages are not valid.

We propose a solution based on multisignatures that alleviates this problem by making certificates only consist of an individual *aggregate* signature (instead of of $\Theta(n)$), as well as of a description of the subset of parties whose signatures have been aggregated. The main idea is to have all parties initially sign the value they receive from the supreme committee with their own signing keys. However, when sending their values up the tree, parties assigned to inner nodes of the tree will aggregate valid signatures on the message which was previously sent down the tree, and keep track of which signatures have contributed.

Commitment phase. Our instantiations of multi-party computations among subsets of parties will be based on fully homomorphic encryption (FHE). To this end, we want parties in each input committee C_i to store an FHE encryption $\mathsf{Enc}(\mathsf{pk}, x_i)$ of the input x_i that we want to be committing. The FHE public key pk is generated by the supreme committee (who holds secret shares of the matching secret key sk), and sent to all parties using the methods outlined above. A party i is committed to the value x_i if the honest parties in C_i all hold the *same* ciphertext encrypting x_i. This presents some challenges which we address and solve as follows:

1. First, a malicious party P_i must not be able to broadcast an invalid ciphertext to the members of the committee C_i. This is prevented by appending a simulation-sound NIZK argument π to the ciphertext c that there exists a message x and "good" randomness r such that $\mathsf{Enc}(\mathsf{pk}, x; r) = c$.
2. Second, for a security proof to be possible, it is well known that not only the encryption needs to be hiding and binding, but a simulator needs to be able to have some way to extract the corresponding plaintext from a valid ciphertext-proof pair (c, π). A major issue here is that the simulated setup must be independent of the corrupted set in our model. This prevents the use of NIZK arguments of knowledge. Moreover, we can expect the FHE encryption to be secure against chosen plaintext attacks only. We will solve this by means of *double encryption*, following Sahai's construction [41] of a CCA-secure encryption scheme from a CPA-secure one. Namely, we provide an additional encryption c_2 of x under a different public-key (for which no one needs to hold the secret key), together with an additional NIZK argument that c_1 and c_2 encrypt the same message. The ciphertext c_2 will not be necessary at any later point in time and serves only the purpose of verifying commitment validity (and permitting extraction in the proof).
3. Third, a final problem we have to face is due to rushing adversaries and the possibility of mauling commitments, in view of the use of the same public key pk for all commitments. This can be prevented in a black-box way by letting every party P_i first (in parallel) VSS its commitment to the parties in C_i, and then in a second phase letting every committee C_i reconstruct the corresponding commitment. If the VSS protocol is perfectly secure, this ensures input-independence.

Another challenge is how to ensure that ciphertext sizes and the associated NIZK proof length are all of the order $|x| \cdot \mathsf{poly}(k)$, instead of $\mathsf{poly}(|x|, k)$. We achieve

this by encrypting messages bit-by-bit using a bit-FHE scheme, whose ciphertexts are hence of length $\mathsf{poly}(k)$. The corresponding NIZK proof is obtained by sequentially concatenating individual proofs (each of length $\mathsf{poly}(k)$) for the encryptions of individual bits.

Shuffling phase. The major privacy issue in executing a sublinear algorithm in a distributed setting is that the adversary must not learn which parties have contributed their inputs to the protocol evaluation, beyond any information that the algorithm's output itself reveals. Ideally, we would like parties to shuffle their inputs in a random (yet oblivious) fashion, so that at the end of such a protocol each party P_i holds the input of party $P_{\pi(i)}$ for a random permutation π, but such that the adversary has no information about the choice of π and for which party $\pi(i)$ he holds an input. At the same time, the supreme committee jointly holds information about the permutation π in a shared way. Unfortunately, this seems impossible to achieve: A disrupting adversary may always refuse to hold inputs for other parties. However, we can now exploit the fact that the inputs are held by *committees* $C_1, \ldots, C_n$ containing a majority of honest parties.

The actual shuffling is implemented via distributed evaluation of a switching network SN, under central coordination by the supreme committee. We assume that a switching network over n wires is given, with depth $d = \mathsf{polylog}(n)$, and is known to everyone, and with the property given by Theorem 6: i.e., it implements a nearly uniform permutation on $[n]$ under random switching. For each swap gate (i, j) in the network, the supreme committee members jointly produce an encryption $\hat{b}_{i,j}$ of an (unknown) random bit $b_{i,j}$, indicating whether the inputs x_i and x_j are to be swapped or not when evaluating the corresponding swapping gate. The value $\hat{b}_{i,j}$ is broadcast to all parties in C_i and C_j. At this point, each party in C_i broadcasts his copy of $\widehat{x}_i$ to all parties in C_j, and each party in C_j does the same with $\hat{x}_j$ to all parties in C_i. (Each party then, given ciphertexts from the other committee, will choose the most frequent one as the right one.) Then, each party in C_i (or C_j) will update his encryption $\hat{x}_i$ to be an encryption of $\mathsf{Dec}(\mathsf{sk}, \hat{x}_j)$ or $\mathsf{Dec}(\mathsf{sk}, \hat{x}_i)$, depending on the value of $\hat{b}_{i,j}$, using homomorphic evaluation of the swap-or-not function. We note that this operation can be executed in parallel for all gates on the same layer, hence the swapping requires d rounds.

Evaluation phase. Once the parties' inputs have been (obliviously) shuffled, we are ready to run the sublinear algorithm. The execution is controlled by the supreme committee C. First, the members of C will run an MPC to randomly select the subset of inputs $I \subset [n]$ to be used by the algorithm. The output of the MPC will be the set of *permuted* indices $\sigma(I) := \{\sigma(i) : i \in I\}$. The corresponding committees $\{C_j : j \in \sigma(I)\}$ are invited to join in a second MPC. Each member of C_j enters the MPC with input equal to his currently held encrypted secret share (of some unknown input x_i, for which $j = \sigma(i)$). Each member of C enters the MPC with input equal to his share of the secret decryption key sk. Collectively, the members of $C \cup (\bigcup_{j \in \sigma(I)} C_j)$ run an MPC which (1) recombines

Protocol for Non-adaptive Sublinear Algorithm Evaluation (Overview)

Committee Election Phase

1. Execute *almost-everywhere* committee election protocol of [34] to generate a communication tree together with a committee C at its root (where $(1-o(1))$ fraction of honest parties agree on C).
2. Achieve *certified* almost-everywhere agreement on C and individual committees $\{C_i\}_{i\in[n]}$ as follows. Members of C collectively sample a PRF seed s and communicate it to (almost) all parties. Each C_i is defined by $F_s(i)$. Every party signs his believed value of (C, s) and passes it up the communication tree to C, where agreeing signatures are aggregated into a single multisignature at each inner node. The message and "certificate" multisignature that contains signatures from a majority of all parties is sent back down the tree.
3. Achieve *full* agreement on $C, \{C_i\}_{i\in[n]}$ as follows. Each party P_i possessing a valid certificate π on (C, s) sends (C, s, π) to each party in $C_i := F_s(i)$. Each party P_j who does *not* have a valid certificate listens for incoming messages and adopts the first properly certified tuple. (Note steps 2-3 enable C to broadcast messages).

Commitment Phase

4. Parties in the primary committee C run the (standard) MPC protocol of [5] amongst themselves to generate keys for the FHE scheme and a second standard PKE scheme. Parties in C receive the public keys $\mathsf{pk}, \mathsf{pk}'$ and a secret share of FHE key sk. They broadcast $\mathsf{pk}, \mathsf{pk}'$ to all parties.
5. In parallel, each party P_i acts as dealer to VSS the following values to his input committee C_i: (1) an FHE encryption of his input $\hat{x}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$, (2) a second encryption of x_i under the standard PKE with pk', and (3) NIZK proofs that $\hat{x}_i$ is a valid encryption and the two ciphertexts encrypt the same value.

Shuffling Phase

6. Parties in primary committee C run an MPC to generate a random permutation ρ, expressed as a sequence of random swap bits in the switching network SN. The output is an *FHE encryption* $\hat{\rho}$ of ρ, which they broadcast to all parties.
7. The committees C_i obliviously shuffle their stored input values, as follows.
   For each layer $L_1, ..., L_d$ in the sorting network SN,
   - Let $L_\ell = ((i_1, j_1), ..., (i_{n/2}, j_{n/2}))$ be the swapping pairs in the current layer ℓ.
   - In *parallel*, the corresponding pairs of committees $(C_{i_1}, C_{j_1}), ..., (C_{i_{n/2}}, C_{j_{n/2}})$ exchange their currently held input ciphertexts $\hat{x}_p, \hat{x}_q$ (using broadcast then majority vote) and homomorphically evaluate the swap-or-not function on $\hat{x}_p, \hat{x}_q$, and the appropriate encrypted swap bit $\hat{b}$ contained in $\hat{\rho}$.

   Outcome: each party in committee C_i holds encryption of input $x_{\rho(i)}$.

Evaluation Phase

8. Parties in primary committee C run an MPC to execute the input selection procedure $I \leftarrow \mathsf{SLA.Sel}$. The output of the MPC is the set of *permuted* indices $\rho(I) \subset [n]$.
9. Every party in C sends a message "Please send encrypted input ℓ" to every party P_j in C_ℓ for which $\ell \in \rho(I)$.
10. Each party $P_j \in C_\ell$ who receives consistent messages "Please send encrypted input ℓ" from a *majority* of the parties in C, broadcasts his currently held encrypted input $\hat{x}^j_{p_\ell}$ to all parties in C. (Recall that this allegedly corresponds to an encryption of the input x_p held by the committee $C_\ell = C_{\rho(p)}$ after the ρ-permutation shuffle).
11. The parties of C evaluate the second portion of the sublinear algorithm, $\mathsf{SLA.Exec}$ via an MPC. Each party of C broadcasts the resulting output answer to all parties.

Fig. 1: High-level overview of the protocol Π_{SLA} for secure distributed evaluation of a non-adaptive sublinear algorithm $\mathsf{SLA} = (\mathsf{SLA.Sel}, \mathsf{SLA.Exec})$.

the shares of sk, (2) decrypts the secret shares held by each C_j, (3) reconstructs each of the relevant inputs x_i, $i \in I$, from the corresponding set of secret shares, (4) executes the sublinear algorithm on the reconstructed inputs, and (5) outputs *only* the output value dictated by the sublinear algorithm (e.g., for many algorithms, this will simply be YES/NO).

The main challenge is making the complexity of this stage such that only $\mathsf{poly}(\log n, \log k)$ rounds are executed, and only messages of size $|x| \cdot \mathsf{poly}(\log k, \log n)$ will be exchanged. This will be achieved by performing most of the computations locally via FHE by the parties in the supreme committee, and by generating the randomness to be used in $\mathsf{SLA.Sel}$ and $\mathsf{SLA.Exec}$ by first agreeing on a $\mathsf{poly}(k)$-short seed of a PRG via coin-tossing, and then subsequently using the PRG output as the actual randomness.

Extension: Adaptive algorithms. The above protocol can be modified to accommodate *adaptive* sublinear algorithms $\mathsf{SLA} = (\mathsf{SLA.Sel}_1, \ldots, \mathsf{SLA.Sel}_q, \mathsf{SLA.Exec})$ simply by modifying the evaluation phase such that an MPC is run for each next-query $\mathsf{SLA.Sel}_j$ to obtain the permuted index of the next query $\rho(i_j)$. Note that without loss of generality all queries are distinct. As a result of this modification, the number of rounds unavoidably increases: Namely, we need $O(q)$ additional rounds to obtain inputs from the committees $C_{\rho(i_j)}$ one by one. However, the proof and the protocol are otherwise quite similar, and we postpone a more detailed description to the final version of this paper.

Acknowledgments. This research was initiated and done in part while the authors were visiting the Isaac Newton Institute for Mathematical Sciences in Cambridge, UK. This work was partially supported by NSF contract CCF-1018064, a Simon Investigator award, and a Visiting Fellowship of the Isaac Newton Institute for Mathematical Sciences.

This material is based on research sponsored by DARPA under agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. Collection of surveys on sublinear algorithms. `http://people.csail.mit.edu/ronitt/sublinear.html`.
2. Ben Adida and Douglas Wikström. How to shuffle in public. In *TCC*, pages 555–574, 2007.
3. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

4. Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly-secure multiparty computation. *IACR Cryptology ePrint Archive*, 2011:136, 2011.
5. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.
6. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
7. Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.
8. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.
9. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. ECCC, Report 2011/111, 2011.
10. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, 2011.
11. Ran Canetti. Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465, 2006. http://eprint.iacr.org/.
12. Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. Improved fault tolerance and secure computation on sparse networks. In *ICALP (2)*, pages 249–260, 2010.
13. Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. Edge fault tolerance on sparse networks. In *ICALP (2)*, pages 452–463, 2012.
14. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *CRYPTO*, page 462, 1987.
15. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. pages 383–395, 1985.
16. Artur Czumaj, Przemka Kanarek, Krzysztof Lorys, and Miroslaw Kutylowski. Switching networks for generating random permutations, 2001.
17. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.
18. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.
19. Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.
20. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.
21. Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Breaking the o(nm) bit barrier: Secure multiparty computation with a static adversary. *CoRR*, abs/1203.0289, 2012.
22. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, pages 308–317, 1990.
23. Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.
24. Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *STOC*, pages 148–161, 1988.

25. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
26. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
27. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
28. S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure computation with sublinear amortized work. *IACR Cryptology ePrint Archive*, 2011:482, 2011.
29. Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private approximation of np-hard functions. In *STOC*, pages 550–559, 2001.
30. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.
31. Piotr Indyk and David P. Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC*, pages 245–264, 2006.
32. Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *ICDCN*, pages 203–214, 2011.
33. Valerie King and Jared Saia. Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18, 2011.
34. Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.
35. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.
36. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.
37. Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In *CRYPTO*, pages 130–149, 2007.
38. R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24:583–584, September 1981.
39. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2001.
40. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *In Proc. of 33rd STOC*, pages 590–599, 2001.
41. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.
42. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.
43. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), November 1979.
44. Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.