The Bottleneck Complexity of Secure Multiparty Computation

Elette Boyle
IDC Herzliya
elette.boyle@idc.ac.il

Abhishek Jain
Johns Hopkins University
abhishek@cs.jhu.edu

Manoj Prabhakaran
Indian Institute of Technology Bombay
mp@cse.iitb.ac.in

Ching-Hua Yu
University of Illinois at Urbana-Champaign
cyu17@illinois.edu

Abstract

In this work, we initiate the study of *bottleneck* complexity as a new communication efficiency measure for secure multiparty computation (MPC). Roughly, the bottleneck complexity of an MPC protocol is defined as the maximum communication complexity required by any party within the protocol execution.

We observe that even without security, bottleneck communication complexity is an interesting measure of communication complexity for (distributed) functions and propose it as a fundamental area to explore. While achieving $O(n)$ bottleneck complexity (where n is the number of parties) is straightforward, we show that: (1) achieving *sublinear* bottleneck complexity is *not* always possible, even when no security is required. (2) On the other hand, several useful classes of functions do have $o(n)$ bottleneck complexity, when no security is required.

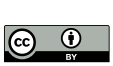
Our main positive result is a compiler that transforms any (possibly insecure) efficient protocol with fixed communication-pattern for computing any functionality into a secure MPC protocol while preserving the bottleneck complexity of the underlying protocol (up to security parameter overhead). Given our compiler, an efficient protocol for any function f with sublinear bottleneck complexity can be transformed into an MPC protocol for f with the same bottleneck complexity.

Along the way, we build cryptographic primitives – incremental fully-homomorphic encryption, succinct non-interactive arguments of knowledge with ID-based simulation-extractability property and verifiable protocol execution – that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols, Theory of computation → Communication complexity

Keywords and phrases distributed protocols, secure computation, communication complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.24



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 24; pp. 24:1–24:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Secure multi-party computation (MPC) [32, 20] is a fundamental notion in cryptography, enabling a collection of mutually distrusting parties to jointly evaluate a function on their private inputs while revealing nothing beyond the function output. In the past decades, a great deal of research has been dedicated to the design and optimization of efficient MPC protocols.

In this work, we study one fundamental metric of MPC efficiency: the required *communication* between parties. In particular, we focus on the communication complexity of MPC in large-scale settings, where the number of participants is significant.

In nearly all existing works in MPC literature, the communication complexity goal has been to minimize the *total* communication of the protocol across all n parties. However, for many important applications, such as peer-to-peer computations between lightweight devices,[1] total costs (such as total communication) are only secondarily indicative of the feasibility of the computation, as opposed to the primary issue of *per-party* cost. Indeed, while a total communication bound L implies average per-party communication of the protocol is L/n, the computation may demand a subset of the parties to *each* communicate as much as $\Theta(L)$. When all parties contribute input to the computation, then $L \geq n$, meaning these parties must bear communication proportional to the *total number of parties*. In large-scale distributed settings, or when the protocol participants are lightweight devices, such a requirement could be prohibitive.

New efficiency measure: (MPC) Bottleneck Complexity

To address these concerns, we initiate the study of *bottleneck complexity* of MPC. The bottleneck complexity of a protocol Π is defined as the *maximum communication required by any party* within the protocol execution. One may further specialize this to incoming versus outgoing communication. The MPC bottleneck complexity of a (distributed) function is the minimum possible bottleneck complexity of a secure MPC protocol for the function. In this work, our goal is to explore this notion as a complexity measure for distributed computations, and to develop secure protocols with low bottleneck complexity.

Bottleneck complexity addresses certain (practically important) aspects ignored by standard communication complexity. For instance, if two messages are transmitted in two different parts of a network, say $A \to B$ and $C \to D$, they would be delivered faster than two messages sent to/from the same party, say $A \to B$ and $C \to B$. While both have same total communication, the latter has higher bottleneck communication.

Bottleneck Complexity without Security

Before studying bottleneck communication complexity for secure protocols, we first consider this measure for arbitrary protocols without any security considerations. Indeed, this already forms an interesting measure of complexity for (distributed) functions, and we propose it as a fundamental area to explore. As in the case of total communication complexity (which coincides with bottleneck complexity for the case of 2 parties), there is a trivial upper bound of $O(n)$ bottleneck complexity for any n-party functionality (with boolean inputs), where all parties simply send their input to a central party who computes the functionality. On

[1] For example, optimizing navigation routes based on traffic information contributed by the cell phones of drivers on the road, without revealing the locations of individual users.

the other hand, in many functions, bottleneck complexity brings out structures that total communication complexity overlooks. For instance, in computing say the XOR or AND of n bits, total communication complexity is $\Theta(n)$, but the bottleneck complexity is $O(1)$. These functions naturally allow for *incremental computation* along a chain, in which each party receives and sends a single bit. Indeed, there is a large class of useful functions which have protocols with low bottleneck complexity, as discussed below.

However, a priori it is not clear whether *all* functions can be computed in a similar manner. This brings us to the first question considered in this work:

Can all functions be computed (without security)
with *sublinear* bottleneck complexity?

For concreteness, we may consider n-party functions, with n inputs (one for each party) each k bits long, and a single-bit output. Because of the trivial $O(nk)$ upper bound on total communication complexity for any such function (as discussed above), each party *on average* needs only to communicate $O(k)$ bits. But in this protocol, the communication complexity of the central party—and thus bottleneck complexity of the protocol—is $(n-1)k$ bits. Surprisingly, we show that this is the best one can ask for, for general functions. That is, there exist n-party functionalities with k-bit inputs for which the bottleneck complexity is $\Omega(nk)$.

▶ **Theorem 1.** *(Informal.) There exist n-party functions with k-bit input for each party that have bottleneck complexity close to that in the trivial upperbound, namely $(n-1)k$.*

Our proof is based on a counting argument, and quantifies over possibly inefficient functions too. Interestingly, giving an explicit efficient function f with such a lower bound will require a breakthrough in complexity theory, as it would imply an $\Omega(n^2)$ lower bound on the circuit size of computing f. (We discuss this connection below.)

Functions with Low Bottleneck Complexity

Despite the above lower bound, there is a large class of interesting functions which do have sublinear bottleneck complexity. One simple but widely applicable example is addition in a finite group: the sum of n group elements distributed among n parties can be aggregated bottom-up (and then disseminated top-down) using a constant-degree tree, with every party communicating $O(d)$ group elements, where d is its degree in the tree. [2]

A wider class of functions are obtained from the literature on streaming algorithms [18, 26]. Indeed, any streaming algorithm with a small memory and a small number of passes corresponds to a low bottleneck complexity function. (Here, we refer to the actual function that the streaming algorithm computes, which may in fact be an approximation to some other desired function.) This is because we can design a protocol which passes around the state of the streaming algorithm from one party to the next, in the order in which their inputs are to be presented to the algorithm.

On the other hand, low bottleneck complexity protocols appear to be much more general than streaming algorithms. Indeed, observe that the low bottleneck complexity protocol described above has a very special communication structure of a chain (or multi-pass cycle). We leave it as an open problem to separate these two notions – i.e., find functions which have low bottleneck complexity protocols, but do not have low-memory streaming algorithms.

[2] If the group is not abelian, the tree used should be such that its in-order traversal should result in the parties to be ordered in the same way their inputs are ordered in the sum being computed.

Finally, we note that, any n-input function with a constant fan-in circuit of subquadratic size (i.e., $o(n^2)$ gates) has a sublinear bottleneck complexity protocol. To see this, first we note that such a circuit can be made to have constant fan-out as well, by increasing the circuit size by a constant factor. [3] Then, a sublinear bottleneck complexity protocol can be obtained from the circuit by partitioning all the $o(n^2)$ gates roughly equally among the n parties, and letting the parties evaluate the gates assigned to them, communicating with each other when wires cross party boundaries. The communication incurred by each party is bounded by the number of wires incident on all the gates it is assigned, which is $o(n)$.

MPC Bottleneck complexity

We next turn our attention to achieving low bottleneck complexity for *secure* computation of functionalities. We focus on the general setting where up to $n-1$ out of n parties can be corrupted. As a baseline, we observe that the MPC protocol of Dodis *et al.* [15] based on "additive-spooky encryption" can be easily adapted to obtain generic secure computation with $O(n)$ bottleneck complexity (where $O(n)$ hides factors of the security parameter). Therefore, as in the insecure setting, we focus on constructing MPC protocols with sublinear $o(n)$ bottleneck complexity.

Specifically, we ask the question:

> If a function f can be computed with bottleneck complexity L,
> can it be computed *securely* with the same bottleneck complexity,
> up to a multiplicative overhead of the security parameter?

We note that the goal of sublinear bottleneck complexity is strictly stronger than the recently studied problem of MPC with sublinear communication locality [5]. The locality of a protocol is the maximum number of other parties that any party must communicate with during the course of the protocol. It is easy to see that sublinear bottleneck complexity directly implies sublinear locality (since sending/receiving $o(n)$ bits means that a party can only communicate with $o(n)$ neighbors); however, as locality does not place any requirements on the number of bits communicated by a party, the converse is not true. Indeed, without security requirements, every function has an $O(1)$-local protocol, which is not the case for bottleneck complexity.

We show a general compiler which transforms any (possibly insecure) efficient multi-party protocol Π for computing a function f into a protocol Π' for *securely* computing f, preserving the per-party communication and computation requirements up to $O(\lambda^c)$ factors in the security parameter λ for small constant c. The original protocol Π can have an arbitrary communication pattern; however, we require that this pattern must be fixed a priori (independent of inputs) and known to all parties. Our compiler additionally preserves the topology of communication graph of Π (and in particular, preserves locality).

▶ **Theorem 2** (Informal). *There is a transformation which maps any (possibly insecure) efficient protocol with fixed-communication-pattern for an n-party distributed function f into a secure MPC protocol for f with asymptotically (as a function of n) the same communication and computational requirements per party, and using the same communication graph as the original protocol.*

[3] Given a gate with fan-out $d > 2$, consider the depth-1 tree T rooted that gate with d leaves being the gates to which its outputs are connected. T can be replaced by an equivalent *binary* tree T' with the same root and leaves, and $d-2$ new internal nodes. The new internal nodes of T' can be "charged" to the leaves of T. On doing this for all gates in the circuit, each gate gets charged at most as many times as its fan-in. Since each gate in the original circuit has constant fan-in, this transformation increases the circuit size by at most a constant factor.

The transformation to achieve security against passive corruption is based on a new tool that we develop, namely *Incremental Fully Homomorphic Encryption*, which we show can be instantiated from the Learning With Errors (LWE) assumption, à la [19]. For security against active corruption (possibly for restricted auxiliary information), we build zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK [2]) with an"ID-based" simulation-extractability property (see Section 1.1 for details). We rely on a setup that includes a common random string and a (bare) public-key infrastructure, where all the n parties have deposited keys for themselves, and which all the parties can access for free. The setup can be reused for any number of executions.

Our Contributions

To summarize, our main contributions in this work are as follows:

- We introduce a new measure of per-party communication complexity for (distibuted) functions, called bottleneck complexity.
- We demonstrate the existence of n-party functions with k bits of input for each party, that have bottleneck complexity $\Theta(nk)$. Showing an explicit function with $\Omega(n)$ bottleneck complexity will require showing an explicit function with $\Omega(n^2)$ circuit size complexity. On the other hand, we observe that many useful classes of functions do have $o(n)$ bottleneck complexity.
- We show a general transformation from arbitrary efficient protocols to secure MPC protocols (in a model with public setup) that asymptotically (as a function of n) preserves the communication and computational requirements per party, and preserves the same communication graph.
- As part of our transformation, we introduce cryptographic primitives—Incremental FHE, Verifiable Protocol Execution—and give a construction of ZK-SNARKs with an ID-based simulation-extractability property. These may be of independent interest.

We expand on the transformation in the following section.

1.1 Our Techniques

We describe the main ideas underlying our positive result: the bottleneck-complexity-preserving transformation from arbitrary protocols to secure ones.

At a high-level, we follow an intuitive outline for our compiler: (1) We first compile an insecure protocol into a protocol that is secure against semi-honest (or honest-but-curious) adversaries using fully homomorphic encryption (FHE). (2) We then use zero-knowledge succinct arguments of knowledge (ZK-SNARKs) to compile it into a protocol that is (standalone) secure against malicious adversaries. However, we run into several technical challenges along the way, requiring us to develop stronger guarantees for FHE and SNARKs, as well as some other new ideas. We elaborate on these challenges and our solutions below.

Semi-honest Security

A natural starting idea to obtain semi-honest security is to execute an "encrypted" version of the underlying (insecure) protocol by using FHE. Once the parties have the encrypted output, they execute the FHE decryption process to learn the output. The immediate problem with implementing this idea in the multiparty setting is which key must we use for encryption and decryption. If a single party knows the (entire) decryption key, then we cannot guarantee security.

To address this problem, two approaches have been developed in the literature: threshold FHE [1], where the parties jointly generate a public key for an FHE scheme such that each party only knows a share of the decryption key, and multi-key FHE [25], where each party has its own public and secret key pair and FHE evaluation can be performed over ciphertexts computed w.r.t. different public keys.

While these approaches have been shown to suffice for constructing round-efficient MPC protocols, they are not directly applicable to our setting. This is for two reasons:

- Threshold FHE and multi-key FHE systems are defined in the broadcast model of communication where each party gets to see the messages sent by all the other parties. In contrast, our setting is inherently point-to-point, where a party only communicates with its neighbors in the communication graph of the underlying insecure protocol. Indeed, in order to maintain sublinear bottleneck complexity, we cannot afford each party to communicate with all the other parties.
- Further, in all known solutions for threshold FHE [1] and multi-key FHE [25, 9, 27, 6, 29], the size of one or more protocol messages of each party grows at least linearly with the number of parties. This directly violates our sublinear bottleneck complexity requirement.

To address these issues, we define and implement a new notion of *incremental* FHE (IFHE). Roughly, an IFHE scheme is defined similarly to threshold FHE, with the following key strengthened requirements: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

We construct an IFHE scheme with appropriate security guarantees based on the Gentry-Sahai-Waters FHE scheme [19]. Using IFHE, we are able to directly compile an insecure protocol into a semi-honest secure protocol. In fact, this protocol can withstand a slightly stronger adversary – called a *semi-malicious* adversary [1] – which is allowed to maliciously choose its random tape. This will be crucially exploited in the next step, because without it, one will need to enforce honest random-tapes for all the parties (using n-way coin-tossing-in-the-well) which would incur $\Omega(n)$ communication already.

From Semi-Malicious to Malicious Security

A natural approach to achieve security against malicious adversaries is to use the GMW paradigm [20]. Roughly, in the GMW compiler, each party first commits to its input and random tape. Later, whenever a party sends a message of a semi-malicious protocol,[4] it also proves in ZK to *all* the other parties that it is behaving correctly w.r.t. the committed input and random tape.

The GMW commit-and-prove methodology is problematic in our setting since we cannot allow a party to talk to all other parties (directly or indirectly through the other nodes). Yet, in order to achieve security, each honest party must verify not just that its neighbors behave correctly, but that *all* corrupt parties (many of whom may not directly interact with any honest party) behaved honestly. A priori, these may seem to be contradictory goals.

We address all of these challenges by presenting a new generic compiler for *Verifiable Protocol Execution* (VPE), modeled as a functionality $\mathcal{F}_{\mathsf{vpe}}$. Our protocol Π_{vpe} for implementing

[4] The standard GMW compiler is defined for semi-honest protocols and also involves a coin-tossing step. Here, we consider a natural variant that works for semi-malicious protocols.

$\mathcal{F}_{\mathsf{vpe}}$ asymptotically preserves the per-party communication and computational complexity (up to a multiplicative factor polynomial in the security parameter) of the underlying semi-malicious protocol. We construct Π_{vpe} from two main ingredients: (1) a new commitment protocol that allows the parties to compute a succinct "aggregate" commitment over the inputs and randomness of all of the parties. (2) ZK-SNARKs with a strong extraction property as well as simulation-soundness to ensure that adversary cannot prove false statements even upon receiving simulated proofs. We refer the reader to the full version for details on our commitment protocol. Here, we discuss our use of ZK-SNARKs.

ID-Based Simulation-Extractable ZK-SNARKs

We rely on ZK-SNARKs to let parties provide not just proofs of correctly computing their own messages, but also of having verified previous proofs *recursively*. This use of SNARKs for recursive verification resembles prior work on proof-carrying data [8, 3]. The key difference is that proof-carrying data only addresses correctness of computation, whereas in our setting, we are also concerned with privacy. In particular, in order to argue security, we also require these proofs to be simulation-sound with extractability (or simply *simulation-extractable*) [30, 31], which presents a significant additional challenge.

The core challenge in constructing simulation-extractable ZK-SNARKs (SE-ZK-SNARKs) arises from the inherent limitation that extraction from the adversary must be non-black-box (since the size of the extracted witness is larger than the proof itself), but the adversary receives simulated proofs which he cannot directly produce on his own. Indeed, for this reason SE-ZK-SNARKs are impossible to achieve with strong universal composability (UC) security [23]. To reduce the security of an SE-ZK-SNARK construction to an underlying knowledge assumption (such as standard SNARKs), one must thus either (a) start with an assumption that guarantees non-black-box extraction even in the presence of an oracle (which can be problematic [16]), or (b) somehow in the reduction be able to provide *the code* to answer the adversary's simulated proof queries, without voiding the reduction by including the simulation trapdoor itself.

Two recent works have presented constructions of SE-ZK-SNARKs, each adopting a different approach. Groth and Maller [21] embody approach (a), constructing full SE-ZK-SNARKs from a new specific pairing-based knowledge assumption which assumes extraction in the presence of black-box access to an oracle with the trapdoor. Alternatively, Garman *et al.* [17] take approach (b), basing their construction on standard SNARKs; however, their construction is only applicable to a restricted security model where the statements on which the adversarial prover requests simulated proofs are fixed in advance (in which case these proofs can be hardcoded in the reduction). The case where the adversary's queries are chosen *adaptively* as a function of previously simulated proofs (which we need for our transformation) is not currently addressed in this setting.

We provide a new solution for handling adaptive queries, without relying upon oracle-based assumptions as in [21]. We consider an ID-based notion of SE-ZK-SNARKs, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). In our definition, the adversary must fix a set ID^* of "honest" identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement w.r.t. an identity $\mathsf{id} \notin \mathsf{ID}^*$.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARKs by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to "puncture" the trapdoor trap for the CRS w.r.t. an identity set

ID^*. A punctured trapdoor $\mathsf{trap}_{\mathsf{ID}^*}$ can only be used to simulate the proofs w.r.t. identities $\mathsf{id} \in \mathsf{ID}^*$, but cannot be used to simulate proofs w.r.t. identities $\mathsf{id} \notin \mathsf{ID}^*$. Using such a punctured trapdoor, we are able to successfully implement approach (b) in the adaptive setting. We implement this idea by using identity-based signatures, which can be readily constructed using certificate chains from a standard signature scheme.

Ultimately, we obtain recursively verifiable ID-based SE-ZK-SNARK generically from signatures and (standard) SNARKs with an "additive extraction overhead." While the latter is a relatively strong requirement [22], such primitives have been considered in prior work [10, 3] and appears to be as justified as the standard SNARK assumption.

Auxiliary Inputs

In order to achieve the standard MPC security, we require SE-ZK-SNARKs that allow extraction even w.r.t. adversaries with auxiliary inputs from arbitrary distribution, which in turn requires SNARKs with the same extraction property. However, even assuming SNARKs which are secure only with respect to specific auxiliary information distribution, our construction obtains MPC that is secure with respect to restricted auxiliary information (see the full version for details).

1.2 Related Work

Communication complexity models

The vast majority of study in communication complexity (c.f. [24]) focuses on the setting of only two parties, in which case the total and bottleneck complexities of protocols align (asymptotically). In the multi-party setting, several models are considered regarding how the input to f begins initially distributed among the players. The most common such models are the "number-on-forehead" model, in which parties begin holding all inputs except their own, and the model considered in this work (as is standard in MPC), frequently known as the "number in hand" model, where each party begins with his own input. In all cases, the "communication complexity" within the given model refers to the total communication of all parties.

Communication complexity of MPC

Communication complexity of secure multiparty computation (MPC) has been extensively studied over the years. Communication complexity preserving compilers from insecure to secure protocols were introduced in the 2-party setting by [28]. The setting of MPC with many parties was first predominantly considered in the line of work on scalable MPC [11, 12]. Here the focus was on optimizing the complexity as a function of the circuit size $|C|$, and the resulting n-party protocols have per-party communication $\tilde{O}(|C|/n) + \mathrm{poly}(n)$. Some of these works explicitly achieve load-balancing (e.g., [14, 4]), a goal similar in spirit to bottleneck complexity, where the complexity of the protocol is evenly distributed across the parties. To the best of our knowledge, however, the $\mathrm{poly}(n)$ term in the per-party communication complexity is $\Omega(n)$ in all works aside from [33], which achieves $\tilde{O}(|C|/n)$ *amortized* per-party communication but $\tilde{O}(|C|/n + n)$ *bottleneck* complexity (due to its dependence on [7]).

Communication Locality

A related notion to bottleneck complexity is communication locality [5]. The locality of a party is the number of total other parties it must communicate with throughout the protocol, and the locality of the protocol is the worst case locality of any party. In [5], Boyle et al.

studied locality in secure MPC and showed (based on various computational assumptions) that any efficiently computable function has a polylog(n)-locality secure MPC protocol.

Lower bounds on MPC communication complexity

As discussed, lower bounds on standard multi-party communication complexity cannot directly imply meaningful lower bounds on bottleneck complexity, as no such bound can exceed $\Omega(n)$ (attainable by all parties sending their input to a single party), but this implies only a bound of $\Omega(1)$ bottleneck complexity. For *secure* computation, in [13], Damgård et al. showed that securely evaluating a circuit of m multiplication gates requires $\Omega(n^2 m)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity in their setting. We note, however, that their lower bound does not apply to us, as we consider computational security, and further, their lower bound does not apply to the setting where the number of parties is larger than the security parameter.

2 Preliminaries and Definitions

By $x \in_R \mathbb{Z}_q^n$ we denote that x is uniformly sampled from $\mathbb{Z}_q^n$, and by $x \leftarrow D$ we denote that x is sampled from a distribution D. By $\overset{c}{\approx}$ we denote computational indistinguishability. We denote an N-party additive secret sharing of $x \in \mathbb{Z}_q$ by $[x]_q^N$. That is, each P_i owns $x_i \in \mathbb{Z}_q$ such that $x = \sum_{i \in [N]} x_i$. When it is clear, we write $[x]$ for brevity.

Communication Model

We consider a synchronous network among n parties, $P_1, \cdots, P_n$, that allows secure communication between (some) pairs of parties; the channels are authenticated and leak nothing except the number of bits in each message.

An n-party protocol π in this model is a "next-message function," that takes as input the round number t, two identities $i, j \in [n]$ and the view of P_i, and outputs the message from P_i to P_j in round t. The view of a party consists of its input, random-tape, and all the messages received in prior rounds of the protocol. If the next-message function is invoked with the keyword out instead of a receiver, π generates the output for the "sender" (for simplicity, we shall restrict ourselves to protocols that produce output only on termination).

We shall require that in a protocol, the message on any edge (i, j) at any round t is encoded using a *prefix-free code* that is agreed up on between the sender and the receiver. Adopting this model precludes communication by not sending any bits. [5]

Security for MPC

We use a standard simulation-based security definition for MPC protocols in the *standalone security* model. We consider two notions of corruptions: (1) active adversaries, who may arbitrarily deviate from the protocol strategy, and (2) semi-malicious adversaries, who follow the protocol instructions but may choose their random tapes arbitrarily. Both of these

[5] While one may argue that it is reasonable to allow zero-cost communication in this manner, it can be abused to communicate large amounts of information at the cost of a single bit, by using a large number of rounds. Further, such signalling cannot be used if multiple such protocols are executed concurrently. Also, practically, the prefix-free communication is more amenable to implementing a synchronous model over an asynchronous network, as delays will not be mistaken for shorter (or empty) messages.

adversaries can abort the execution whenever they choose. We shall also consider a notion of MPC security in which the auxiliary information given to the adversary at the beginning of the protocol is restricted to be from a class of distributions. We refer the reader to the full version for further details.

2.1 Bottleneck Complexity

We introduce a new *per-party* communication metric for distributed computations.

▶ **Definition 3** (Bottleneck Complexity of Protocol)**.** The *individual communication complexity* of a party P_i in an n-party protocol π, denoted as $\mathcal{CC}_i(\pi)$, is the expected number of bits sent or received by P_i in an execution of π, with worst-case inputs.

The *bottleneck complexity* ($\mathcal{BC}$) of an n-party protocol π is the worst-case communication complexity of any party. That is, $\mathcal{BC}(\pi) = \max_{i\in[n]} \mathcal{CC}_i(\pi)$.

▶ **Definition 4** (Bottleneck Complexity of Function)**.** The *bottleneck complexity of an n-input function f* is the minimum value of $\mathcal{BC}(\pi)$ when quantified over all n-party distributed protocols π which correctly evaluate f.

Analogously, we define the *MPC bottleneck complexity* of f as the minimum $\mathcal{BC}(\pi)$ quantified over all n-party protocols π which *securely* evaluate f.

Admissible Protocols

We will show techniques that transform general (insecure) protocols to secure ones. Here we define the required minimal assumption of the original protocols, which we refer to as admissibility. Roughly, a protocol is admissible if its next-message function is polynomial-time computable and it has a fixed communication pattern. Below $\mathbb{Z}_+$ denotes non-negative integers.

▶ **Definition 5** (Admissible Protocol)**.** Let f be a polynomial function, k be a security parameter, and let $\pi = \{\pi_1, ..., \pi_n\}$ be a possibly randomized n-party protocol, where π_i is a next message function of P_i. Let $x = \{x_1, ..., x_n\}$ and $r = \{r_1, ..., r_n\}$ be the input set and the random string set respectively. Denote $\left\{m^t_{i,j}(x,r)\right\}^{t\in[T]}_{i,j\in[n]}$ as the set of the messages generated by $\pi(x,r)$, and let $|m^t_{i,j}(x,r)| \in [0, f(k)]$ be the length of message from P_i to P_j at time t.[6] We say π is admissible if it satisfies the following two conditions:

- **Polynomial-Time Computable:** For each i, next-message function π_i is expressed by a circuit of fixed polynomial-size in $|x_i| + |r_i|$, with a universally bounded depth.
- **Fixed Communication Pattern:** A protocol π is said to have a *fixed communication pattern* if, irrespective of the input and random-tapes of the parties, the total number of rounds $t_{\max}$ is fixed and there is a function len : $[t_{\max}] \times [n] \times [n] \to \mathbb{Z}_+$ that maps (t, i, j) to the length of message (possibly 0) from P_i to P_j at round t as determined by π for any view of P_i.

Note that above we allow randomized protocols, as some interesting low bottleneck complexity protocols (e.g., those derived from streaming algorithms) tend to be randomized.

---

[6] Precisely, for each $i \in [n], t \in [T]$, $\{m_{i,j}(x,r)\}^t_{j\in[n]} \leftarrow \pi_i\left(x_i, r_i, \{m_{j,i}(x,r)\}^{t\in[t-1]}_{j\in[n]}\right)$

3 Lowerbound on Bottleneck Complexity of Distributed Functions

In this section we show that for most functions f on n inputs (each input could be as short as 1 bit, and the output a single bit delivered to a single party), for any distributed computation protocol π that implements f, the (incoming) bottleneck complexity $\mathcal{BC}(\pi)$ is at least $n - O(\log n)$ bits. In fact, this holds true even without any security requirement. This is tight in the sense that, even with a security requirement, there is a protocol in which only one party has individual communication complexity $\Omega(n)$, and all others have communication proportional to their inputs and outputs (with a multiplicative overhead independent of the number of parties).

This is somewhat surprising since many interesting functions do have protocols with constant communication complexity. As mentioned before, any (possibly randomized) function which has a streaming algorithm or a sub-quadratic sized circuit (with small fan-in gates) gives rise to low botteleneck complexity protocols.

To show our lower-bound, we need to therefore rely on functions with roughly a quadratic lower-bound on circuit size. Given the current lack of explicit examples of such functions, we present an *existential result*, and leave it as a conjecture that there are n-bit input boolean functions with polynomial sized circuits with bottleneck communication complexity of $\tilde{\Omega}(n)$. For simplicity, we discuss the case of perfectly correct protocols, but as we shall point out, a small constant probability of error does not change the result significantly. This result says that there is a function (in fact, most functions) such that the best bottleneck complexity is almost achieved by the trivial protocol, in which one party receives the inputs of all the other $n - 1$ parties and carries out the computation locally. We prove the following in the full version.

▶ **Theorem 6.** *$\exists f : \{0,1\}^{k \times n} \to \{0,1\}$ such that any n-party, each with k bits input, distributed computation protocol that computes f correctly will have at least one party receiving at least $(n-1)k - O(\log nk)$ bits in the worst-case.*

4 Incremental FHE

We define and implement a new notion of incremental FHE (IFHE), which is used within our main positive result. We start by providing syntax and security definitions for IFHE in Section 4.1. We describe our construction of IFHE in Section 4.2. Our construction builds upon the FHE scheme of Gentry, Sahai and Waters (GSW) [19]; we refer the reader to the full version for discussion on the specific properties of their scheme that we rely upon.

4.1 Definitions

(Leveled) Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme consists of algorithms (KEYGEN, ENCRYPT, EVAL, DECRYPT), where (KEYGEN, ENCRYPT, DECRYPT) constitute a semantically secure public-key encryption scheme, and EVAL refers to the homomorphic evaluation algorithm on ciphertexts. An ℓ-leveled FHE scheme supports homomorphic evaluation of circuits of depth at most ℓ.

Incremental FHE

An IFHE scheme is defined similarly to threshold FHE [1], with the following key modifications: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be

decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

For technical reasons, it is convenient to describe the joint decryption procedure via three sub-algorithms: A procedure PreDec which pre-processes a homomorphically evaluated ciphertext to be safe for joint decryption; PartDec run by each individual party on a ciphertext (with his share of the secret key) to generate his contribution toward the decryption; and CombineDec which combines the outputs of PartDec from each party for a given ciphertext to reconstruct the final decrypted output. In addition to standard semantic security, we also require the output of PartDec to hide information about the secret key share that was used; this is captured by the Simulatability of Partial Decryption property below.

▶ **Definition 7** (Incremental FHE)**.** An *incremental fully homomorphic encryption (IFHE)* scheme is an FHE scheme with an additional algorithm IFHE.CombineKeys and with Decrypt replaced by three algorithms IFHE.PreDec, IFHE.PartDec and IFHE.CombineDec. By PK_S we denote a combined public key of a subset $S \subseteq [n]$ of parties. Particularly, $\mathrm{PK}_{\{i\}} = \mathrm{PK}_i$ is generated by P_i using the algorithm Keygen, and $\mathrm{PK} = \mathrm{PK}_{[n]}$ is the final public key. Similarly, by v_S we denote a combined decryption, and by v_i when $S = \{i\}$. For the completeness of notations, let PK_S and v_S be empty strings when $S = \emptyset$. We describe the syntax of the four algorithms as follows:

- IFHE.CombineKeys$(\mathrm{PK}_S, \mathrm{PK}_T)$: On input 2 combined public keys $\mathrm{PK}_S, \mathrm{PK}_T$, where $S \cap T = \emptyset$, output a combined public key $\mathrm{PK}_{S\cup T}$.
- IFHE.PreDec$(\mathrm{PK}, \mathbf{C})$: On input a final public key PK and a ciphertext $\mathbf{C}$, sample a public random $\mathbf{R}$, and output a re-randomized ciphertext $\mathbf{C}'$ of the same plaintext.
- IFHE.PartDec$(\mathrm{PK}, \mathrm{SK}_i, \mathbf{C})$: On input a final public key PK, ith secret key SK_i, ciphertext $\mathbf{C}$, output a partial decryption v_i.
- IFHE.CombineDec(v_S, v_T): On input 2 partial decryptions v_S, v_T, where $S \cap T = \emptyset$, if $|S \cup T| < n$, output a partial decryption $v_{S\cup T}$; otherwise, output a plaintext y as the final decryption.

Also, we require the following additional properties:

Efficiency: There are polynomials $poly_1(\cdot), poly_2(\cdot)$ such that for any security parameter λ and any $S \subseteq [n], S \neq \emptyset$, $|\mathrm{PK}_S| = poly_1(\lambda)$ and $|v_S| = poly_2(\lambda)$.

Correctness: Given a set of plaintexts and a circuit to evaluate, the correctness of IFHE says that the FHE evaluation of the circuit over the ciphertexts can always be decrypted to the correct value, where the ciphertexts are encryption of plaintexts using a single combined public key.

Furthermore, by "Incremental" FHE, we mean that the final combined public key as well as the final combined decryption can be formed in an arbitrary incremental manner. We defer the formal definition of correctness to the full version.

Semantic security under Combined Keys (against Semi-Malicious Adversary): Given the parameters prepared in the initial setup, the (corrupted) parties $\{P_j\}_{j\neq i}$, instead of using random strings to compute $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$, can use an arbitrary string to generate $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$. Then as long as an honest party generates $(\mathrm{PK}_i, \mathrm{SK}_i)$ independently, the encryption using the final combined public key $(\mathrm{PK}_{[n]}, \mathrm{SK}_{[n]})$ is semantically secure. Formally, we say that the IFHE scheme has semantic security under combined keys if the advantage $\Pr[\beta' = \beta] - 1/2$ in the following experiment is negligible for all PPT Adv.

1. $(\mathsf{params}) \leftarrow \mathrm{Setup}(1^\lambda, 1^d)$

2. $\forall j \neq i$, Adv computes $(\text{PK}_j, \text{SK}_j)$ according to $\text{KEYGEN}(\mathsf{params})$ but replaces the randomly sampled string by a chosen one. Then Adv computes a combined key $\text{PK}_{[n]\setminus\{i\}}$ according to COMBINEKEYS, picks $x \in \{0,1\}$ and sends $(\text{PK}_{[n]\setminus\{i\}}, x)$ to the challenger.
3. The challenger computes $(\text{PK}_i, \text{SK}_i) \leftarrow \text{KEYGEN}(\mathsf{params})$, $\text{PK} \leftarrow \text{COMBINEKEYS}(\text{PK}_i, \text{PK}_{[n]\setminus\{i\}})$, and chooses a random bit $\beta \xleftarrow{\$} \{0,1\}$. If $\beta = 0$, it lets $\mathbf{C} := \text{ENCRYPT}(\text{PK}, 0)$, and else $\mathbf{C} := \text{ENCRYPT}(\text{PK}, x)$, and sends $\mathbf{C}$ to Adv.
4. Finally Adv outputs a bit β'.

Simulatability of Partial Decryption: There is a PPT simulator SIM, s.t. for every combined public key PK, ciphertext $\mathbf{C}'$ of a plaintext y, index $i \in [n]$:

$$\left\{\text{PK}, \mathbf{C}', \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, v_i\right\} \overset{c}{\approx} \left\{\text{PK}, \mathbf{C}', \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, v_i'\right\},$$

where $v_i \leftarrow \mathsf{IFHE}.\text{PARTDEC}(\text{PK}, \text{SK}_i, \mathbf{C}')$ and $v_i' \leftarrow \text{SIM}(\text{PK}, \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, y, \mathbf{C}')$.

4.2 Construction of IFHE

We present an IFHE scheme building on the FHE scheme of Gentry et al. [19]. The SETUP, KEYGEN, ENCRYPT, and EVAL parts are the same as those of [19], while COMBINEKEYS, PREDEC, PARTDEC and COMBINEDEC parts are new. The algorithms are described as follows:

An IFHE scheme.

- **Setup:** $(\mathsf{params}) \leftarrow \mathsf{IFHE}.\text{SETUP}(1^\lambda, 1^d)$. Same as GSW, where $\mathsf{params} = (q, n, m, \chi, B_\chi, \mathbf{B})$.
- **Key Generation:** $(\text{PK}_i, \text{SK}_i) \leftarrow \mathsf{IFHE}.\text{KEYGEN}(\mathsf{params})$. Same as GSW, where $\text{PK}_i = (\mathbf{B}, \mathbf{b}_i)$ and $\text{SK}_i = \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$.
- **Combining Keys:** $\text{PK}_{S\cup T} \leftarrow \mathsf{IFHE}.\text{COMBINEKEYS}(\text{PK}_S, \text{PK}_T)$
  On input $\text{PK}_S = (\mathbf{B}, \mathbf{b}_S)$ and $\text{PK}_S = (\mathbf{B}, \mathbf{b}_T)$, output $\text{PK}_{S\cup T} = (\mathbf{B}, \mathbf{b}_S + \mathbf{b}_T)$. Particularly $\text{PK}_{[n]}$ is abbreviated as PK or a matrix $\mathbf{A}$.
- **Encryption:** $\mathbf{C}_i \leftarrow \mathsf{IFHE}.\text{ENCRYPT}(\text{PK}, x_i)$. Same as GSW.
- **Evaluation:** $\mathbf{C} \leftarrow \mathsf{IFHE}.\text{EVAL}(\mathbf{C}_1, ..., \mathbf{C}_\tau; f)$. Same as GSW.
- **Preparing Decryption:** $\mathbf{C}' \leftarrow \mathsf{IFHE}.\text{PREDEC}(\text{PK}_i, \mathbf{C})$
  On input $\text{PK} = \mathbf{A}$ and $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$, sample a public random matrix $\mathbf{R}$ in $\{0,1\}^{m\times m}$ and output $\mathbf{C}' = \mathbf{C} + \mathbf{AR}$.
- **Partial Decryption:** $v_i \leftarrow \mathsf{IFHE}.\text{PARTDEC}(\text{PK}_i, \text{SK}_i, \mathbf{C}')$
  On input $\text{PK} = \mathbf{A}$, $\text{SK}_i \equiv \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$, and $\mathbf{C}' \in \mathbb{Z}_q^{n\times m}$, sample $\mathbf{e}_i' \leftarrow \chi^m$, set $\mathbf{t}_i' = t_i$ if $i = 1$ and $\mathbf{t}_i' = (-\mathbf{s}_i, 0)$ if $i > 1$, and output $v_i = (\mathbf{t}_i'\mathbf{C}' - \mathbf{e}_i')\mathbf{G}^{-1}(\mathbf{w}^T)$, where $\mathbf{w} = (0, ..., 0, \lceil q/2 \rceil) \in \mathbb{Z}_q^n$.
- **Combining Decryption:** $\mathsf{IFHE}.\text{COMBINEDEC}(v_S, v_T)$
  On input two partial decryptions v_S, v_T with $S \cap T = \emptyset$, compute a partial decryption $v_{S\cup T} = v_S + v_T$. For $|S \cup T| < n$, output $v_{S\cup T}$; for $|S \cup T| = n$, output a plaintext $y = \left\lfloor \frac{v}{q/2} \right\rceil$.

---

5 Summary of Further Results

Due to space limitations we defer the remaining technical sections to the full version. We first demonstrate how to convert an arbitrary admissible multi-party distributed protocol Π (as per Definition 5) for computing a function f to a protocol Π_{sm} for computing f secure against

semi-malicious adversaries, while preserving *per-party* computation and communication. Note that as per Definition 5 the communication pattern of the starting protocol Π can be arbitrary, but we require that it be fixed (i.e., not data dependent).

▶ **Theorem 8.** *Let* IFHE *be an incremental FHE scheme, and* Π *be an* n*-party protocol for evaluating a function* f *with fixed communication pattern. Then there exists a protocol* Π_{sm} *that securely evaluates* f *against up to* $(n-1)$ semi-malicious *corruptions, preserving the per-party computation and communication requirements of* Π *up to* $\mathrm{poly}(\lambda)$ *multiplicative factors (independent of the number of parties* n*). Moreover, the communication pattern of* Π_{sm} *is identical to that of* Π *plus two additional traversals of a communication spanning tree of* Π*.*

At a very high level, the parties will run a one-pass protocol to (incrementally) construct a joint key for the incremental FHE scheme, then execute the original protocol Π underneath FHE encryption, and finally run one more pass to (incrementally) decrypt.

Next, we give a general compiler to transform the above protocol into a maliciously-secure MPC protocol, while preserving the bottleneck complexity. Our compiler relies upon multiple cryptographic ingredients, most notably, ID-based simulation-extractable succinct non-interactive arguments of knowledge (ZK-SNARKs) – a notion that we define and construct in this work (see the full version).

▶ **Theorem 9.** *Let* Π_{sm} *be an MPC protocol that securely evaluates a functionality* f *against* semi-malicious *corruptions, as in Theorem 8. Then, assuming the existence of an ID-based simulation-extractable succinct non-interactive arguments of knowledge, non-interactive commitment schemes and a family of collision-resistant hash functions, there exists a compiler that transforms* Π_{sm} *into another MPC protocol* Π *in the (bare) public-key and common reference string model such that* Π *computes the same functionality* f *and preserves the per-party computation and communication of* Π_{sm} *up to* $\mathrm{poly}(\lambda)$ *multiplicative factors (independent of the number of parties* n*).*

We remark that in the above theorem, the ID-based SE-ZK-SNARK is w.r.t. arbitrary auxiliary inputs. We stress, however, that in order to achieve a weaker MPC definition where the adversary only receives restricted auxiliary inputs, the requirement on the SE-ZK-SNARK can be similarly relaxed as well.

References

1. 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.
2. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349, 2012.
3. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120, 2013.
4. Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 742–762, 2015.

5 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation: how to run sublinear algorithms in a distributed setting. In *Proceeding TCC'13 Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, pages 356–376, 2013.

6 Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 190–213, 2016.

7 Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 57–64, 2013.

8 Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331, 2010.

9 Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 630–656, 2015.

10 Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 54–74, 2012.

11 Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 501–520, 2006.

12 Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 241–261, 2008.

13 Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In *Crypto'16*, pages 459–488, 2016.

14 Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, pages 242–256, 2014.

15 Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 93–122, 2016.

16 Dario Fiore and Anca Nitulescu. On the (in)security of snarks in the presence of oracles. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 108–138, 2016.

17 Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 81–98, 2016.

18 Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 2016. doi:10.1007/978-3-540-28608-0.

19 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto'13*, pages 75–92, 2013.

20 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.

21 Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

22 Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 71–88, 2014.

23 Ahmed E. Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T.-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. How to use snarks in universally composable protocols. *IACR Cryptology ePrint Archive*, 2015:1093, 2015. URL: `http://eprint.iacr.org/2015/1093`.

24 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

25 Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234, 2012.

26 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43:9–20, 2014.

27 Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 735–763, 2016.

28 Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 590–599, 2001.

29 Chris Peikert and Sina Shiehian. Multi-key FHE from lwe, revisited. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 217–238, 2016.

30 Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 543–553, 1999.

31 Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 566–598, 2001.

32 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

33 Mahdi Zamani, Mahnush Movahedi, and Jared Saia. Millions of millionaires: Multiparty computation in large networks. *IACR Cryptology ePrint Archive*, 2014:149, 2014.