

Secure Distributed Network Optimization Against Eavesdroppers

Yael Hitron

Weizmann Institute of Science, Rehovot, Israel

Merav Parter

Weizmann Institute of Science, Rehovot, Israel

Eylon Yogev

Bar-Ilan University, Ramat-Gan, Israel

Abstract

We present a new algorithmic framework for distributed network optimization in the presence of eavesdropper adversaries, also known as *passive wiretappers*. In this setting, the adversary is listening to the traffic exchanged over a fixed set of edges in the graph, trying to extract information on the private input and output of the vertices. A distributed algorithm is denoted as f -secure, if it guarantees that the adversary learns nothing on the input and output for the vertices, provided that it controls at most f graph edges.

Recent work has presented general simulation results for f -secure algorithms, with a round overhead of $D^{\Theta(f)}$, where D is the diameter of the graph. In this paper, we present a completely different white-box, and yet quite general, approach for obtaining f -secure algorithms for fundamental network optimization tasks. Specifically, for n -vertex D -diameter graphs with (unweighted) edge-connectivity $\Omega(f)$, there are f -secure congest algorithms for computing MST, partwise aggregation, and $(1 + \epsilon)$ (weighted) minimum cut approximation, within $\tilde{O}(D + f\sqrt{n})$ congest rounds, hence nearly tight for $f = \tilde{O}(1)$.

Our algorithms are based on designing a secure algorithmic-toolkit that leverages the special structure of congest algorithms for global optimization graph problems. One of these tools is a general secure compiler that simulates *light-weight* distributed algorithms in a congestion-sensitive manner. We believe that these tools set the ground for designing additional secure solutions in the congest model and beyond.

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Distributed algorithms

Keywords and phrases congest, secure computation, network optimization

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.71

Funding *Merav Parter*: This project is funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 949083).

Eylon Yogev: Supported in part by a grant from the Israel Science Foundation (no. 2893/22), and by an Alon Young Faculty Fellowship.

1 Introduction

Modern large-scale graph processing and network analysis involve exchanging sensitive information between network participants. This computation is commonly employed by distributed message-passing systems (such as Google’s Pregel, Apache’s Giraph) in which processors send small messages to their neighbors in the network. Such a collaboration-based communication approach usually forces the vertices to sacrifice their privacy for the greater good, e.g., solving a global network optimization task. In this paper, we study *secure* distributed algorithms that, on the one hand, perform almost as well as their non-secure counterparts (in terms of running time and quality of the output solution) and, at the same time, do not compromise on the privacy of the network’s individuals.



© Yael Hitron, Merav Parter, and Eylon Yogev;

licensed under Creative Commons License CC-BY 4.0

14th Innovations in Theoretical Computer Science Conference (ITCS 2023).

Editor: Yael Tauman Kalai; Article No. 71; pp. 71:1–71:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We focus on the adversarial setting of *eavesdropping* (also known as *passive wiretapping*) where the adversary overhears the entire communication exchanged over a fixed set of edges in the graph, $F^* \subseteq E$. The identity of these adversarial edges is *unknown* to the network participants. The adversary is assumed to know the graph topology but has no information about the edge *weights*, which are considered as part of the *private* input of the vertices. A distributed algorithm is said to be *f-secure* if it is guaranteed that an adversary, controlling any subset of f edges, learns nothing new, in the *information-theoretic* sense, about the individual inputs of the vertices from the communication it tapped into.

While there are standard techniques to provide security under cryptographic assumptions (e.g., public-key encryption), in this paper, we strive for *information-theoretic* solutions that make no computational assumptions on the adversary's power. Such security notion is the holy-grail objective in secure multi-party computation in which the underlying network topology is the complete graph [12]. We also note that information-theoretic fits the perspective of the congest model. The latter assumes that the vertices are computationally *unbounded*, and therefore it makes sense to assume that the adversary is *unbounded* as well.

Our goal is to provide information-theoretic secure distributed graph algorithms in the congest model of distributed computing [31]. In the secure setting addressed in this paper, the edge weights of the graph are considered to be *private* information that should be *hidden* from the adversary, while the unweighted version of the graph is considered to be public (see also [34, 3]). These secure computations have various applications. For example, in road networks whose unweighted topology is public, but the edge weights which correspond to the amount of traffic in each road segment are private information as they are based on GPS locations of vehicles or mobile data [3]. The edge weights also play a role in social graphs in which they are associated with sensitive information, e.g., the price of commercial trade [25].

Within this setting, we focus on the family of global optimization graph problems, arguably among the most studied distributed graph problems these days. This family includes the fundamental graph problems of minimum-spanning-tree (MST), minimum cut, subgraph connectivity, shortest paths, etc. Several critical milestones mark the three-decade-old quest for round-efficient congest algorithms for this class of problems: Starting with the influential paper of Garay, Kutten, and Peleg [7] on existential vs. universal optimal bounds, the powerful lower bound technique of Das Sarma et al. [33], and fast-forwarding to the very recent breakthrough results on universal optimal solutions by Haeupler, Wajc, and Zuzic [16, 13]. This extensive body of work provides us with an almost complete characterization of the inherent complexity of this graph family. It is known that most of the classical optimization problems have worst-case (near) optimal round complexity of $\tilde{O}(D + \sqrt{n})$ rounds, where D is the graph's diameter and n is the number of vertices. Our main objective is to provide *f-secure* algorithms for this class of problems that (nearly) match the round complexity of the non-secure counterparts.

To illustrate our ideas, we mainly focus on MST computation which is arguably the most canonical graph problem, which initiates the study of distributed optimization [7]. In the standard distributed setting, it is required for each vertex to learn its incident edges in the MST. In our secure setting, all vertices are required to learn this local output, while the adversary must learn *nothing* about the edge weights of the graph, in the information-theoretic sense. Our secure MST algorithm can be naturally adapted to the wide class of global network optimization problems, for which a unified algorithmic approach has been provided by Ghaffari and Haeupler [9] and Haeupler, Wajc and Zuzic [16].

Quick History on Secure Distributed Algorithms against Eavesdroppers. In an earlier work of Dolev, Dwork, Waarts and Yung [4], a secure unicast algorithm was provided for general graph topologies. In this setting, a given vertex pair s, t is required to exchange a secret message m^* . Their secure solution was based on exchanging messages along multiple s - t edge-disjoint paths. A recent result of Hitron and Parter [17] shows that there are D -diameter graphs for which the maximal length in a collection of f edge-disjoint paths might be $(D/f)^{\Omega(f)}$. This state of affairs is quite unfortunate: handling $f = \Omega(\log n)$ faults requires polynomial time when sending information along f edge-disjoint paths.

Considerably improved solutions for the secure unicast problem are known when using the framework of *secure network coding*, introduced by Cai and Yeung [2]. For example, Jain [20] designed a network coding based algorithm that can be implemented in $O(D)$ congest rounds, as noted recently by [18]. Round-efficient algorithms are currently only known for restricted multicast tasks and under the assumption that the network is a DAG [6].

A very recent work of Hitron, Parter and Yogev [18] overcomes the above mentioned $(D/f)^{\Omega(f)}$ barrier for the secure broadcast task¹. Specifically, they presented an f -secure broadcast algorithm against eavesdroppers that runs in $\tilde{O}(D + \sqrt{fn})$ rounds, for every n -vertex D -diameter graph, with edge-connectivity $\Theta(f)$. While the work of [18] serves as a basic starting point of our work, we note that there is a fundamental difference between broadcast and the class of network optimization tasks, considered in this paper. In the secure broadcast setting, it is required to hide from the adversary only one secret, namely the $O(\log n)$ -bit broadcast message, m^* ; while the communication graph itself is *not* hidden from the adversary. Consequently, the computation of a convenient communication backbone (for the secure transmission of m^*) can be done in a *non-secure* manner. In contrast, we consider graph problems (rather than information dissemination problems), where it is required to hide from the adversary the edge weights of the graph, as the latter determine the output solution (e.g., MST, minimum weighted cut). This brings along challenges of a different flavor than those encountered in [18]. In particular, in comparison to [18] we also need to make sure that the *communication pattern* of the algorithm would not leak information on the input weighted graph. Indeed, in most distributed (non-secure) algorithms for these problems, an adversary gains information merely by viewing the edges on which messages are exchanged (without knowing the content of these messages).

1.1 Our Contribution

Secure Network Optimization. We provide a unified framework for the secure computation of network optimization graph problems in the congest model of distributed computing. This includes f -secure algorithms for MST, minimum cut approximation, and partwise aggregation. Our algorithms are nearly existentially optimal for $f = \tilde{O}(1)$. In contrast to the approach taken in many recent works, our secure algorithms are not obtained by a black-box compilation of the non-secure solutions. Instead, we zoom into a *particular framework* for non-secure network optimization tasks, and compile algorithms in this framework to be secure. An informal statement of our key result is as follows:

¹ In this task, all vertices learn a secret message m^* while the adversary should learn nothing on m^* .

► **Theorem 1 (Informal).** *For every n -vertex weighted D -diameter graph $G = (V, E, W)$ with unweighted edge-connectivity $(2f + 3)(1 + o(1))$, there are f -secure randomized congest algorithms for computing MST, $(1 + \epsilon)$ -Minimum (Weighted) Cuts and Partwise Aggregation that run in $\tilde{O}(D + f\sqrt{n})$ rounds.*

*The output of these algorithms is given in the **distributed secure** format: each vertex v knows its output (e.g., its incident edges in the output MST, a $(1 + \epsilon)$ approximation on the minimum cut, etc). It is guaranteed that w.h.p., an adversary listening on any subset of f edges in the graph learns **nothing** (in a statistical sense) about any of these input and output values.*

Note that for $f = \tilde{O}(1)$, the f -secure solutions nearly match the non-secure complexity of these problems, which is also known to be existentially tight by Das-Sarma et al. [33]. In contrast, all prior algorithms provided by the general compilers of [17] yield f -secure algorithms with $D^{O(f)}$ rounds, hence super-linear for $f = \Omega(\log n)$.

New Tool: Congestion-Sensitive Compilers. Our secure algorithms are based on a collection of secure subroutines. One of the most notable tools, which we believe to be of independent interest, is a general *congestion-sensitive* compiler whose performances are optimized for distributed algorithms with low congestion. The *congestion* of a distributed algorithm is measured by the maximum number of $(\log n)$ -bit messages that the algorithm sends over a given edge in the graph, throughout its entire execution. This measure is important in many distributed contexts, e.g., scheduling, as observed by [24, 8]. An algorithm is then said to be *light-weight* if its congestion is at most poly-logarithmic in the number of vertices. While network optimization problems (provably) admit round-efficient solutions with *large* congestion (as large as \sqrt{n}), they are nevertheless based on critical light-weight subroutines. This is the point where we take advantage of our optimized compiler.

It is noteworthy that the prior distributed compilers in the adversarial setting (e.g., [29, 28, 27, 17]) have an inherent round complexity *overhead*. That is, given an r -round non-secure algorithm \mathcal{A} , these compilers securely simulate \mathcal{A} in a round-by-round manner, resulting in a secure algorithm \mathcal{A}' with round complexity of $r \cdot T(n)$, where $T(n)$ is the time it takes to securely simulate a single round of \mathcal{A} . Our compilers deviate from this recipe, and consequently, for light-weight algorithms, the round complexity depends only additively in r (neglecting logarithmic factors), we get the following:

► **Theorem 2 (Congestion-Sensitive Compilers).** *For every $(2f + 3)(1 + o(1))$ edge-connected D -diameter n -vertex graph G , any r -round **cong**-congestion algorithm \mathcal{A} for G can be compiled into an equivalent f -secure algorithm \mathcal{A}' that runs in $\tilde{O}(r + D + f \cdot \sqrt{\mathbf{cong} \cdot n} + f \cdot \mathbf{cong})$ rounds. The security of \mathcal{A}' holds w.h.p.*

Recent work [19, 1] noted that most of the classical network optimization problems (MST, minimum cut, etc.) can be solved by using $O(\log n)$ applications of the following two building blocks: (i) partwise aggregation² and (ii) one-round neighborhood communication (i.e., a single congest round). Altogether, our work provides the secure implementation of these

² The computation of aggregate function in a collection of vertex-disjoint connected subgraphs

building blocks: Namely, Theorem 1 provides f -secure algorithms for partwise aggregation; and using our secure compiler of Theorem 2, we can securely simulate $O(\log n)$ congest rounds within $\tilde{O}(D + f\sqrt{n})$ rounds. While we present a complete description of f -secure algorithms for MST, partwise and approximate minimum cut³, our approach should be applicable for the wide class of graph problems captured in [9, 19].

1.2 Preliminaries

Throughout, we are given an n -vertex weighted graph $G = (V, E, W)$, with unweighted diameter of D . The neighbors of a vertex $v \in V$ are denoted by $N(v)$. For a subset of items X and a probability $p \in [0, 1]$, let $X[p]$ denote the random sample obtained by sampling each item of X independently with probability of p . For a subgraph $G' \subseteq G$, let $D(G')$ denote the diameter of G' . We use the following useful subroutine from [30].

▷ **Claim 3 ([30]).** Convergecast of an associative and commutative function g over a rooted tree T with diameter $D(T)$ can be performed in $O(D(T))$ many rounds, while sending only one message on each tree edge. In addition, the convergecast of k functions g_1, \dots, g_k can be done in $O(D(T) + k)$ rounds, sending at most k message along each tree edge.

The Adversarial Congest Model. The communication model used throughout is the classical congest model [31]. In this model, the network is abstracted as an n -vertex graph $G = (V, E)$, where each vertex has a unique identifier of $O(\log n)$ bits. Initially, the vertices only know the identifiers (and possibly also the edge weights) of their incident edges. The algorithm works in synchronous rounds, where neighbors can exchange $O(\log n)$ bits of information in each round.

In our adversarial setting, an eavesdropper adversary controls a fixed set of edges F^* in the graph. The edges of F^* are denoted as *adversarial*, and the remaining edges $E \setminus F^*$ are *reliable*. The vertices do not know the identity of F^* , but they do know the bound f on the cardinality of F^* . The adversary is assumed to be computationally *unbounded*, is allowed to know the topology of the graph G (but not the edge weights) and the algorithm description run by the vertices. The adversary, however, is oblivious to the randomness of the vertices. The security guarantees are formally defined as follows.

Security against Eavesdroppers. For a given randomized algorithm \mathcal{A} running on an n -vertex graph $G = (V, E)$, denote the n -length input vector of the algorithm \mathcal{A} by X (an input for each vertex). Let $R_e(X) \in \{0, 1\}^*$ be the random variable specifying all messages sent through the edge e over the execution of \mathcal{A} , for every edge $e \in E$. For a subset of edges $F = \{e_1, \dots, e_f\} \subseteq E$, let $R_F(X) = (R_{e_1}(X), \dots, R_{e_f}(X))$.

For a given parameter $\delta \in (0, 1)$ and integer f , an algorithm \mathcal{A} is said to be (f, δ) -secure against an eavesdropper if for every n -vertex graph G any subset $F \subseteq E$ of size at most f , and any two input vectors X_1, X_2 , the following two distributions are δ -close in statistical distance: $\{R_F(X_1)\}$ and $\{R_F(X_2)\}$.

³ In a very recent work, Ghaffari and Zuzic [11] used the partwise framework to provide exact, rather than $(1 + \epsilon)$ approximate, minimum cut computation. Since their framework is based on partwise computation, it is very plausible that our tools can be used to provide f -secure exact min-cut computation. This claim should be taken with a grain of salt until all the details are verified.

An algorithm running on n -vertex graphs is said to be f -secure w.h.p. if it is (f, δ) -secure, for $\delta = 1/n^c$ for any given constant $c \geq 1$. Note that in contrast to prior work, and more specifically in comparison to the recent work of [18], the f -security guarantees of our network optimization algorithms are *statistical*.

Our algorithms encrypt messages by XORing them with random keys, this is known as one-time pad encryption [21].

► **Definition 4 (One-Time-Pad Encryption).** *Let $x \in \{0, 1\}^b$ be a b -bit message. In the one-time pad encryption x is encrypted using a uniform random key $K \in \{0, 1\}^b$, by setting $\hat{x} = x \oplus K$. To decrypt \hat{x} using K , simply let $x = \hat{x} \oplus K$.*

Useful Secure Procedures from [18]. Our algorithms use the secure algorithms for the broadcast and unicast tasks, presented in the recent work of [18].

► **Theorem 5 (f -Secure Broadcast [18]).** *For every $(2f + 3)(1 + o(1))$ edge-connected graph $G = (V, E)$, there exists a randomized f -secure broadcast algorithm for sending a b -bit message m^* that runs in $\tilde{O}(D + \sqrt{f \cdot b \cdot n} + b)$ rounds. The edge congestion of the algorithm is bounded by $\tilde{O}(\sqrt{f \cdot b \cdot n} + b)$.*

Indeed our secure network optimization algorithms also require (unweighted) edge-connectivity of $(2f + 3)(1 + o(1))$. The source of this requirement is that it applies as a subroutine the secure broadcast algorithm of Theorem 5.

► **Theorem 6 (Secure Unicast [20, 18]).** *Given is a D -diameter graph G , with a collection of k pairs $\{(s_i, t_i)\}$ (known to all vertices), where each s_i holds a b -bit message m_i to be sent to t_i . There exists a randomized $\tilde{O}(D + k \cdot b)$ -round algorithm **SecureSimUnicast** that exchange the m_i 's from s_i to t_i . The security holds provided that every s_i, t_i are connected in $G \setminus F^*$.*

Secure Distributed Guarantees. Throughout our algorithms (and internal subroutines), we maintain the guarantee that the input and output to the problems at hand are given in a secure distributed manner, in the following sense. At the beginning of the algorithm, each vertex knows its relevant input while the adversary knows nothing on the values of these inputs. At the end of the execution, each vertex knows its own part in the solution, and we are guaranteed (sometimes w.h.p.), that the adversary learns nothing about these output values, as well. Note that since we assume that the adversary knows the graph topology (but not the edge-weights), our framework addresses problems whose output either depends on the edge weights (e.g., MST), or on the inputs of the vertices (e.g., partwise aggregation). This is precisely the set of problems addressed by Haeupler, Wajc, and Zuzic in [16] who considered the supported congest model in which vertices know the graph topology. We note, however, that in this work only the adversary knows the topology, while the vertices do not.

Roadmap. In Section 2, we provide a high-level exposition of our techniques. In Section 3, we provide a full description of our congestion-sensitive compiler, and establish Theorem 2. In Section 4, we describe our f -secure MST algorithm. In the full version we show that this algorithm can also be extended to provide f -secure partwise aggregation and approximate minimum (weighted) cuts.

2 Key Techniques

2.1 Congestion-Sensitive Compilers

We give a high-level overview of how our secure compiler works and how it exploits the small congestion of a given algorithm \mathcal{A} with congestion **cong** and round complexity r . Our compiler is randomized which is necessary for getting the security, and the correctness of the resulting algorithm holds only with high probability. The basis for the compiler is to provide the vertices with a sufficient number of random keys K that are *unknown* to the adversary, using the secure broadcast algorithm of [18] (see Theorem 5). A vertex u can then send a secret message m to its neighbor v by simply sending the *cipher* message, $m \oplus K$; v can decrypt the cipher using the key K .

This immediately leads to a costly but straightforward solution: use the secure broadcast algorithm of [18] to share a secret K for every edge $(u, v) \in E$ and for every round of the protocol of \mathcal{A} (notice that we cannot re-use the same key between rounds as it would leak information to the adversary). The number of keys required to share by this approach might be roughly $n^2 \cdot r$, which is clearly too much. Our final compiler uses only $\tilde{O}(f \cdot \mathbf{cong})$ keys. We explain this bound in steps, first reducing the number of keys to $\tilde{O}(f^2 \cdot r)$, then to $\tilde{O}(f \cdot r)$ keys, and finally to $\tilde{O}(f \cdot \mathbf{cong})$ keys. Recall that the adversary listens to only f edges, and notice that it is safe for two neighboring pairs (u_1, v_1) and (u_2, v_2) to use the same key K , as long as the adversary is *not listening to both edges*. Since we do not know which edges are tapped by the adversary, one can let the vertices pick keys at random from a small set of available keys (per round), hoping that no two pairs of edges in F^* choose the same key. From the birthday paradox, we get that the number of keys we need per round is f^2 ⁴, leading to $\tilde{O}(f^2 \cdot r)$ keys.

Beating the Birthday Paradox by Subset Keys. To improve the quadratic dependency in f , we must overcome the curse of birthdays. Instead of picking a *single* key to compile a single round, we let each vertex pick⁵ a *small subset* of ℓ keys (where $\ell \approx \log n$), $K_{i_1}, \dots, K_{i_\ell}$ from a collection of $f \cdot \log n$ shared keys. Then, they use the XORed key $K^* = K_{i_1} \oplus \dots \oplus K_{i_\ell}$ as the key for the one-time pad. The advantage of this approach is that it escapes the birthday paradox, and lets us get a much higher success probability. The probability that two messages are encrypted using the same subset of keys is very small (i.e., $n^{-O(\log f)}$). However note that we still need to rule out other events in addition to a collision, e.g., the event where one chooses a subset that is contained in the union of all the other subsets. In such a case, security might not hold. We show that, with high probability, each encryption will use at least one fresh key (that has not been used in the encryption of other message) which allows us to provide the security guarantees. This gets us to a solution with $\tilde{O}(f \cdot r)$ keys.

Sensitivity to Congestion rather than to Round Complexity. So far, we have not yet enjoyed the fact that the (non-secure) algorithm \mathcal{A} is light-weight. Our starting observation is that the *secure* algorithm \mathcal{A}' must exchange messages on each of its edges, and in each round. This is despite the fact that the (non-secure) algorithm at hand has bounded congestion. The reason is that if the secure algorithm does not exchange a message through a given

⁴ For a constant success probability, which could be amplified.

⁵ This is done w.r.t. each of its neighbors.

edge (u, v) , it already leaks information to the adversary (e.g., on the weight of that edge). This leads to the following classification of messages: *dummy messages* which correspond to empty messages exchanged by the non-secure algorithm, and *meaningful messages* that correspond to non-empty messages exchanged in the original algorithm \mathcal{A} . The congestion bound implies that the secure algorithm \mathcal{A} should send **cong** meaningful messages through each given edge.

A. Encrypting Dummy Messages. As we wish to have only $\tilde{O}(f \cdot \mathbf{cong})$ random keys, we do not want to spend this precious pool of keys to encrypt dummy messages. Instead, we will be encrypting these messages using independent random strings, locally generated by their sending vertices u (which the other endpoints v do not know). To allow the receiver vertex v to distinguish between dummy messages and meaningful messages, we always pad the original (meaningful) messages with a prefix of $\Theta(\log n)$ consecutive 0's and encrypt them with the pool of $\tilde{O}(f \cdot \mathbf{cong})$ random keys (as explained in the next paragraph). The receiver vertex v upon receiving the message attempts to decrypt it using the pool of $\tilde{O}(f \cdot \mathbf{cong})$ keys that it knows. For the dummy messages, it would hold, w.h.p., that none of these attempts would result in a valid message with a 0's prefix, and those messages will be dropped.

B. Encrypting Meaningful Messages. To encrypt the meaningful messages of Alg. \mathcal{A} , a natural (but flawed) approach is to have **cong** different independent pools of keys. When a vertex u sends its i -th message, it will use the i -th pool. The main issue with this approach is that there is no synchrony when different vertices send their i -th message⁶. This lack of synchrony leads to a problematic scenarios where an adversary listening to f edges accumulate many messages (much more than f) encrypted using the i -th pool over many rounds. It then gains information about the keys in that pool and, consequently, on the messages that got encrypted with this set. This was not an issue when we used a pool for each round.

Instead, our compiler works by setting a *single pool* of keys for all vertices for all rounds. As the vertices use this pool only for the meaningful messages of \mathcal{A} , the size of the pool needs to suffice for encrypting $\tilde{O}(f \cdot \mathbf{cong})$ different messages, at unknown times and order. This bound nearly matches the total number of meaningful messages exchanged over the edges of F^* during \mathcal{A} , i.e., as $|F^*| \leq f$ and the congestion bound is **cong**. The pool of $\tilde{O}(f \cdot \mathbf{cong})$ keys are shared securely at an initialization phase, which is then used to securely simulate all the rounds of \mathcal{A} . Note that our encryption approach makes it impossible for the adversary to distinguish between encrypted dummy vs. meaningful messages, and therefore it cannot gain any information on the original communication pattern of the non-secure algorithm.

For modularity purposes, we introduce a notion we call a “few-time pad”, where similarly to a one-time pad that allows to (information-theoretically) encrypt a single message, a few-time pad allows to encrypt a small number of messages, see Section 3.1 for details. The logic of having a pool of keys and encrypting with a random subset of keys (as described above) is encapsulated in the interface of this few-time pad machinery. We next demonstrate the compiler’s usefulness in the context of secure distributed optimization.

⁶ I.e., some vertices send their i -th message in round r' and others send their i -th message on round $r'' > r'$. This stands in a strike contrast to having a pool per round, where vertices are fully synced, using the i -th pool in the i -th round.

2.2 Secure Network Optimization

To illustrate our algorithmic framework, we focus on the secure computation of MST. Our starting observation is that the congestion of any MST congest algorithm might be $\Omega(\sqrt{n})$ in the worst case, as demonstrated by the work of Peleg and Rubinfeld [32] and Das-Sarma et al. [33]. Hence, a naïve and black-box implementation of our congestion-sensitive compiler asks for exchanging $\Omega(\sqrt{n})$ secrets, leading to a round complexity of $\Omega(n^{3/4})$, even for $f = O(1)$ (see Lemma 2). Providing a near-optimal solution (with only $\tilde{O}(D + \sqrt{n})$ rounds) asks for a white-box secure simulation of a *particular* (non-secure) MST algorithm, designed in a way that facilitates its secure simulation.

As many other distributed MST algorithms, our algorithm, as well, is based on the well-known Borůvka algorithm [26]. This (meta) algorithm works in $O(\log n)$ phases, where in each phase, from each growable component an outgoing edge is selected. All these outgoing edges are added to the forest, while ignoring cycles. Each such phase reduces the number of growable components by a constant factor, thus within $O(\log n)$ phases, a maximal forest is computed. Our *non-secure* MST algorithm is then identified by two main phases: a *light-weight* phase and a *global* phase. The light-weight phase grows MST-fragments in a controlled manner, by applying the merging iterations of the Borůvka’s algorithm, until each fragment becomes of size $\Omega(\sqrt{n})$ and with diameter of $\tilde{O}(\sqrt{n})$. These fragments are hereafter denoted as *basic-fragments*. We call this phase light-weight as it can be simulated (in the non-secure setting) with $\tilde{O}(1)$ congestion (i.e., sending $\tilde{O}(1)$ messages across every edge).

The global phase completes the MST computation by taking a more global approach: We pick a global leader \mathbf{g}^* and sample a collection of $\Theta(\sqrt{n} \log n)$ vertices $L \subset V$, that we denote as *landmarks*. By Chernoff, we have that w.h.p. each basic-fragment contains at least one landmark. Now, the computation of the remaining $O(\sqrt{n})$ MST edges is completed by alternating between two types of computations:

- (bidirectional) communication between the landmarks L and a global leader \mathbf{g}^* , and
- internal light-weight computation inside each basic-fragment.

Let us focus on the implementation of the *first* merging iteration of the Borůvka’s algorithm given the collection of basic-fragments. Each basic-fragment computes its minimum-weight-outgoing-edge (MWOE). Then, each landmark sends the information on this MWOE to the leader \mathbf{g}^* (along with fragment-ID information). This allows \mathbf{g}^* to locally simulate this one iteration of Borůvka, and merge the basic-fragments along the proposed MWOEs. The output of this merging step can then be sent back to the landmarks, where each landmark can be acknowledged with the new component-ID of its basic fragment, along with the information on the added MST edge. This completes the high-level description of the non-secure MST algorithm. It is also easy to see that it runs in $\tilde{O}(D + \sqrt{n})$ rounds, and admits *no* benefit in the standard congest model.

Our f -secure MST algorithm follows this high-level structure and operates as follows. The light-weight phase is implemented by applying our congestion-sensitive compiler. Using Theorem 2, this is done in $\tilde{O}(D + f\sqrt{n})$ rounds. The distributed secure output of this phase is as follows: each vertex v knows the basic-fragment ID of every $u \in N(v) \cup \{v\}$, and its set of incident edges in its basic-fragment.

In the second global part, we use secure unicast procedures from [20, 18] that allow the collection of landmarks to exchange information with the global leader \mathbf{g}^* , in a bidirectional manner. We show that each landmark is required to exchange $\tilde{O}(1)$ messages with the leader \mathbf{g}^* . Note that since the landmark set is a random sample of vertices, their identities do not leak information on the edge weights of the graph. The internal light-weight computation inside each basic-fragment is implemented in a secure manner by applying our congestion-

sensitive compiler. This part as well is implemented in $\tilde{O}(D + f\sqrt{n})$ rounds. In the security arguments, we show that the combination of several secure subroutines can be implemented in a way that leaks no information to the adversary. Due to the neat guarantees of information theoretic security, the composition of several secure subroutines can be easily shown to be secure as well, in our context. We mainly need to make sure that the duration of each subroutine is set to be a function of only the number of vertices and the graph diameter, so that the adversary cannot deduce any information merely from the duration of a given step. This completes the high-level description of our secure-MST algorithm.

Discussion and Open Problems. We provide f -secure algorithms for a collection of network optimization tasks, nearly matching the existential optimal bounds of the non-secure solutions, for $f = \tilde{O}(1)$. The prior algorithms obtained by the general compilers of e.g., [17] have a super-linear round complexity for $f = \Omega(\log n)$. In a sequence of influential works, it has been shown that the (non-secure) complexity of these distributed graph problem is tightly related to the graph theoretic object of *low-congestion shortcuts* [9]. In particular, the *quality* of these structures fully characterizes the round complexity of these problems (even from a universal optimal perspective) [16]. While for general graph topologies the worst-case existential bounds on the shortcut quality is $\Theta(D + \sqrt{n})$, special graph families admit improved bounds [9, 14, 10, 23, 15, 22]. It will be very interesting to provide f -secure solutions whose complexities depend on the quality of the (optimal) graph shortcuts. Finally, while the current work optimizes the round complexity, it will be interesting to improve upon the message complexity (e.g., by employing ideas from Elkin [5]).

3 Congestion-Sensitive Compilers

In this section, we present our new broadcast-based compilation scheme and establish Theorem 2. We define a new type of encryption scheme that we call a “few-time pad”. We stress that this is an information-theoretic encryption scheme that is based on hiding a b -bit message m by XORing it with a random b -bit key K to get a “cipher” $c = m \oplus K$ that hides all information about m (i.e., one-time pad). No computational assumptions are required. Unlike a one-time pad, our generalization allows encrypting a small number of messages while maintaining a relatively small key. We believe that this encryption scheme is of independent interest. Then, we show how to use the few-time pad to describe our general transformation (the compiler). We first describe the compiler and then analyze its correctness and security.

3.1 Few-Time Pad Encryption

For the purposes of this section, we will use a variant of the “one-time pad” encryption, which lets us encrypt a small number of messages. It is important to remark that it is easy to implement a few-time pad by having many keys, one for each message, and using the corresponding key for each encryption. However, this implementation requires the encryption algorithms to be *stateful* and remember which keys have already been used, so it can ensure that it never uses the same key twice (which will compromise security). Crucially for the use of the general transformation, we need the encryption algorithm to be *stateless*, that is, when we encrypt a message with a key K , we do not know how many message were already encrypted using the same key K (this key K will be used by many different vertices in the graph throughout many rounds).

Below, we give a definition and construction of our few-time pad. The scheme will be parameterized by the following: k is the number of messages we wish to encrypt, B is the number of bits each message consists of, and λ is the security parameter.

► **Definition 7** (Few-time Pad). A few-time pad encryption scheme, for parameters $k, B, \lambda \in \mathbb{N}$, is a symmetric key information-theoretic encryption scheme that consists of three algorithms $(\text{Gen}, \text{Enc}, \text{Dec})$:

- $K \leftarrow \text{Gen}(k, B, \lambda)$. A probabilistic polynomial-time key-generation algorithm Gen that takes as input a security parameter λ , number of messages k and message size B , and outputs a secret key K .
- $c \leftarrow \text{Enc}(K, m)$. A probabilistic polynomial-time encryption algorithm Enc takes as input a key K and a message $m \in \{0, 1\}^B$, and outputs a ciphertext c (possibly of different length).
- $m \leftarrow \text{Dec}(K, c)$. A deterministic polynomial-time decryption algorithm Dec that takes as input a key K and a ciphertext c , and outputs a plaintext m .

We require the encryption scheme to satisfy the following security and completeness properties.

- **Correctness:** The scheme is correct, if for all k, B, λ and all messages $m \in \{0, 1\}^B$, it holds that:

$$\Pr[m = \text{Dec}(K, c) : K \leftarrow \text{Gen}(k, B, \lambda), c \leftarrow \text{Enc}(K, m)] = 1 .$$

We formally define the security of a multi-message symmetric key encryption scheme using an unbounded distinguisher that tries to distinguish between encryption of two vector messages. Formally, we define:

- **Security:** For every choice of vectors (m_0^1, \dots, m_0^k) and (m_1^1, \dots, m_1^k) where $m_b^i \in \{0, 1\}^B$ for all $b \in \{0, 1\}$, $i \in [k]$, for any unbounded distinguisher D the following holds:

$$\begin{aligned} & \left| \Pr[D(\text{Enc}(K, m_0^1), \dots, \text{Enc}(K, m_0^k)) = 1 : K \leftarrow \text{Gen}(k, B, \lambda)] - \right. \\ & \left. \Pr[D(\text{Enc}(K, m_1^1), \dots, \text{Enc}(K, m_1^k)) = 1 : K \leftarrow \text{Gen}(k, B, \lambda)] \right| \leq 2^{-\lambda} . \end{aligned}$$

In the full version, we show that it is possible to implement such an information-theoretic encryption scheme with a relatively small key.

► **Theorem 8.** For any $k, B, \lambda \in \mathbb{N}$, there is a few-time pad with parameters k, B, λ where the size of the generated key $K \leftarrow \text{Gen}(k, B, \lambda)$ satisfies $|K| = 2 \cdot |B| \cdot k \cdot (\lambda + \log k)$, and the size of ciphertexts are $O(B + (\lambda + \log k)^2)$.

Our transformation will rely on two additional properties of our few-time pad that immediately follow from the construction. First, we observe that the encryption has the property that ciphertexts are distributed as random strings, as long as security holds. We also observe that a random ciphertext decrypts to a random message.

► **Observation 9.** With probability $1 - 2^{-\lambda}$, the distribution $\text{Enc}(K, m_0^1), \dots, \text{Enc}(K, m_1^k)$, where $K \leftarrow \text{Gen}(k, B, \lambda)$ is exactly distributed as u_1, \dots, u_k , where u_i is uniformly sampled from $\{0, 1\}^B$.

► **Observation 10.** For any K , a decryption of a random string c is a random message $m \leftarrow \text{Dec}(K, c)$.

3.2 Our General Transformation

Throughout, consider an $r = \text{poly}(n)$ -round (non-secure) algorithm \mathcal{A} with total edge congestion of **cong** and a bound of $B = O(\log n)$ per message. For simplicity, assume that \mathcal{A} is deterministic⁷.

⁷ This assumption can be easily removed by fixing the random coins used by the vertices when simulating \mathcal{A} .

Initialization Phase: Exchanging a Few-Time Key K . Our transformation begins with an initialization phase where the vertices in the graph exchange a key K for a few-time encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$, that remains *hidden from the adversary*. We set the following parameters for the few-time pad: the message length is $B' = B + 3 \log(nr)$ where r is the number of rounds, the number of messages $k = f \cdot \mathbf{cong}$, and the security parameter $\lambda = O(\log n)$ (for a sufficiently large hidden constant such that the error is sufficiently small $1/\text{poly}(n)$). An arbitrary vertex s runs $K \leftarrow \text{Gen}(B', k, \lambda)$ to compute the key K . According to Theorem 8 the key is of size

$$|K| = O(|B'| \cdot k \cdot (\lambda + \log k)) = \tilde{O}(f \cdot \mathbf{cong}).$$

Then, we apply the secure broadcast algorithm of Theorem 5 with K as the message. This phase can then be implemented in $\tilde{O}(D + f \cdot \sqrt{\mathbf{cong} \cdot n} + f \cdot \mathbf{cong})$ rounds. Consequently, all vertices learn K , while the adversary learns nothing.

Main Phase: Secure Simulation of \mathcal{A} . The simulation is done round-by-round and is proved using induction. We assume all vertices receive the same messages as in \mathcal{A} up to round τ , and consider round $\tau + 1$. We show how to securely send all messages for round $\tau + 1$ of the algorithm \mathcal{A} . For any edge $(u, v) \in E$, let $M_{u \rightarrow v}$ be the message that u needs to send to v in round $\tau + 1$ (according to the algorithm \mathcal{A}), while $M_{u \rightarrow v}$ might be *empty*. Observe that we cannot let the adversary know whether $M_{u \rightarrow v}$ is empty or not. We let every vertex u act differently according to whether $M_{u \rightarrow v}$ is empty, but in a way that is indistinguishable to the adversary. If $M_{u \rightarrow v}$ is not empty, then u pads the message $M_{u \rightarrow v}$ with 0's of length B' , $M' = M_{u \rightarrow v} \circ 0^{B'-B}$, uses the key K to encrypt M' , $c \leftarrow \text{Enc}(K, M')$, and sends it to v . Otherwise, if $M_{u \rightarrow v}$ is empty, then u simply sends a uniformly random string length B' .

The adversary will not be able to distinguish between real messages and random messages. However, the receiver v will indeed be able to distinguish between the two. The vertex v use the key K to decrypt c to get $M' \leftarrow \text{Dec}(K, c)$. If M' ends with $B' - B$ 0's, then v knows that the message was not empty, and thus sets the message M to be the first B bits of M' . Otherwise, v knows that a random string was sent, and thus sets M to be an empty message. The summary of this protocol is given below.

Secure simulation of \mathcal{A} :

Initialization.

1. Let an arbitrary vertex s compute $K \leftarrow \text{Gen}(B', k, \lambda)$.
2. All vertices participate in a secure broadcast algorithm to share the (secret) message K .

Sending round τ Let $M_{u \rightarrow v}$ be the message the u needs to send to v in round τ .

1. If $M_{u \rightarrow v}$ is not empty set $c \leftarrow \text{Enc}(K, M_{u \rightarrow v} \circ 0^{B'-B})$.
2. Otherwise, set c to be a uniformly random message of length B' .
3. u sends c to v .

Receiving round τ Let c be the message received by v from u .

1. Compute $M' \leftarrow \text{Dec}(K, c)$.
2. If M' does not end with $B' - B$ 0's, then drop the message (i.e., no message from u has been received).
3. Otherwise, set M to be the first B bits of M' .

This completes the description of our simulation. Observe that the simulation of a single round consists of a single round where each vertex sends a message c . By Theorem 8, the ciphertext c is of size $O(B + (\lambda + \log k)^2) = O(\log^2 n)$. However, we can send this message over $O(\log n)$ rounds by splitting it to blocks of size $O(\log n)$. This incurs a $\log n$ factor in the number of rounds.

Proof of Theorem 2. We next turn to analyze the correctness of the construction.

► **Lemma 11** (Correctness of Simulation). *With high probability, for every rounds $\tau \in [r]$ and every pair of vertices $(u, v) \in E$, it holds that:*

- If $M_{u \rightarrow v}$ is not empty then v will obtain the correct message $M_{u \rightarrow v}$.
- If $M_{u \rightarrow v}$ is empty then v drops the received message.

Proof. Fix a round τ and a pair (u, v) . If $M_{u \rightarrow v}$ is not empty, then u sends $c \leftarrow \text{Enc}(K, M_{u \rightarrow v} \circ 0^{B'-B})$. The vertex v gets c and from the perfect correctness of the few-time pad, we have that it will decrypt $M' \leftarrow \text{Dec}(K, c)$ such that $M' = M_{u \rightarrow v} \circ 0^{B'-B}$. Thus, M' ends with $B' - B$ 0's, and thus v will extract the message $M_{u \rightarrow v}$.

If $M_{u \rightarrow v}$ is empty, then u sends a random string c . By Observation 10, when v decrypts, it will get a uniformly random message M' . Recall that $B' = B + 3 \log(nr)$ and thus $B' - B = 3 \log(nr)$. Thus, M' will end with $(B' - B)$ bits of 0, with probability $2^{-(B'-B)} = (nr)^{-3}$. There are r rounds, and in each round at most n^2 messages are sent. Thus, taking a union bound over all $r \cdot n^2$ messages we get that the error probability is bounded by $r \cdot n^2 \cdot (nr)^{-3} \leq n^{-1}$ as required. (Note that it is easy to set the error probability to be n^{-a} for any constant a by setting $B' = B + (a + 2) \log(nr)$.) ◀

► **Lemma 12** (Security of Simulation). *W.h.p., the adversary controlling f edges F^* learns nothing about the messages exchanged throughout the simulation of \mathcal{A} . That is, Alg. \mathcal{A}' is f -secure.*

Proof. Fix a round τ and a pair $(u, v) \in F^*$. The security of our protocol stems from the security of the few-time pad. Recall that we set the few-time pad to be secure for k messages. All messages in our protocol are encrypted. This means that security follows as long as the adversary did not see more than k encryptions. Moreover, for the other messages, we send random strings. From Observation 9 we know that these are indistinguishable from real encryptions.

Formally, the adversary sees all messages sent over the f edges of F^* during the r rounds of the protocol. For $i \in [f]$ and $\tau \in [r]$, let $c_{i,\tau}$ be the message sent on the i -th edge in F^* in round τ . We know that on each edge, only **cong** messages are not empty. Thus, the set $\{c_{i,j}\}_{i \in [f], \tau \in [r]}$ contains at most $f \cdot \mathbf{cong}$ encryptions. Notice that we have set our message bound k to be exactly $k = f \cdot \mathbf{cong}$. Thus, by Observation 9, we can replace all of these messages by uniformly random strings, except with error probability n^{-c} from the encryption scheme. Since all other messages are by themselves random strings, we get that w.h.p., the set of messages $\{c_{i,j}\}_{i \in [f], \tau \in [r]}$ observed by the adversary is identically distributed as a set of uniformly random strings for which security follows immediately. ◀

The proof of Theorem 2 follows now directly. The number of rounds of the simulated algorithm is

$$\tilde{O}(D + f \cdot \sqrt{\mathbf{cong} \cdot n} + f \mathbf{cong} + r \cdot \log n) = \tilde{O}(r + D + f \cdot \sqrt{\mathbf{cong} \cdot n} + f \cdot \mathbf{cong}),$$

as desired. The correctness and security follow directly from Lemmas 11 and 12, respectively.

4 Secure MST

In this section we present our f -secure MST algorithm for weighted graphs with (unweighted) edge-connectivity at least $(2f + 3)(1 + o(1))$. Recall that the eavesdropper is allowed to know the graph topology, but has no information on any of the edge weights. Our secure MST computation guarantees that the adversary learns nothing on the output MST (and in particular, gains no information on the edge weights). At the end of the computation, each vertex knows its incident edges in the MST. We show:

► **Theorem 13 (Secure MST).** *For every λ edge-connected D -diameter weighted graph $G = (V, E, W)$, there is an f -secure randomized MST algorithm, provided that $\lambda \geq (2f+3)(1+o(1))$. The round complexity is $\tilde{O}(D + f\sqrt{n})$, and the security guarantee holds w.h.p.*

We start by stating the following basic secure primitives that follow immediately by the congestion-sensitive compiler of Theorem 2.

Basic Tool: Secure Parallel Aggregation. We provide the secure analogue of Claim 3, by combining Claim 3 and Theorem 2, we immediately have:

▷ **Claim 14 (Secure Parallel Aggregation).** Given is a $(2f + 3)(1 + o(1))$ edge-connected graph $G = (V, E)$, and a collection of vertex-disjoint rooted subtrees $T_1, \dots, T_k \subseteq G$ known in a secure distributed manner. Each vertex v knows its parent and children in each T_i , a function g and an input value x_v . Additionally, all vertices hold an upper bound \hat{D} on $\max_i D(T_i)$. There exists an f -secure algorithm `SecureParallelAggregate` where each vertex $v \in T_i$ obtains $g(V(T_i))$. The round complexity of the algorithm is $\tilde{O}(D + f \cdot \sqrt{n} + \hat{D})$, and the security holds w.h.p.

In addition, by applying the compiler of Theorem 2 to securely simulate a single (non-secure) round, we have:

▷ **Claim 15 (Secure Single-Round Simulation).** A single simulation of a (non-secure) round can be implemented in an f -secure manner, w.h.p., using $\tilde{O}(D + f \cdot \sqrt{n})$ rounds.

Throughout, we assume, w.l.o.g., that the edge weights are unique and thus the MST is unique. An *MST-fragment* is a subtree of the MST. A fragment is denoted as *small* if it has $O(\sqrt{n})$ vertices, otherwise, it is *large*. We are now ready to describe Alg. `SecureMST`. The algorithm consists of two parts, each implemented in a fixed number of $K = \Theta(D + f \cdot \sqrt{n})$ rounds. Note that K is a function of parameters that are known to the adversary (as it knows the unweighted topology of G), and more specifically, it is independent of the edge weights of the graph (the information that we wish to protect). Part 1 implements the sub-routine `SecureSmallMST`, which grows (small) MST fragments until all fragments become large (and with bounded diameter). Part 2 completes the MST computation by applying Alg. `SecureLargeMST`.

Part 1: Merging Small MST-Fragments. Alg. `SecureSmallMST` applies fragment merging steps in a Borůvka-like manner, as long as there exist fragments of size $O(\sqrt{n})$. The algorithm is based on a secure compilation of the following light-weight (non-secure) algorithm `SmallMST`. The output of this algorithm is a forest \mathcal{F} of MST-fragments, where every tree $T_i \in \mathcal{F}$ is of size $\Omega(\sqrt{n})$, and depth $O(\sqrt{n} \log n)$.

► **Lemma 16.** *There exists an $\tilde{O}(\sqrt{n})$ -round randomized (non-secure) algorithm `SmallMST`(G) that given a weighted graph $G = (V, E, W)$, computes a forest of rooted MST-fragments $\mathcal{F} = \{T_1, \dots, T_\ell\}$ such that $|T_i| = \Omega(\sqrt{n})$ and $D(T_i) = O(\sqrt{n} \log n)$, for every $T_i \in \mathcal{F}$. In*

the distributed output format, each vertex $v \in T_i$ holds a unique identifier $ID(T_i)$ and can identify its children and parent in the tree T_i , as well as the fragment-IDs of all its other neighbors. The congestion is $\tilde{O}(1)$, hence it is light-weight.

By using the compilers of Theorem 2, we immediately obtain:

► **Corollary 17.** *Alg. SmallMST can be compiled into an equivalent f -secure algorithm SecureSmallMST with a round complexity of $\tilde{O}(D + f \cdot \sqrt{n})$.*

Part 2: Merging Large MST-Fragments. We next present Alg. SecureLargeMSTPhase that completes the computation of the MST by merging large MST-fragments. The input to this algorithm is the output of Alg. SecureSmallMST(G) which consists of a forest $\mathcal{F} = \{T_1, \dots, T_k\}$ of MST-fragments, where every tree $T_i \in \mathcal{F}$ is of size $\Omega(\sqrt{n})$ and depth $O(\sqrt{n} \log n)$. Each vertex $v \in T_i$ knows $ID(T_i)$, and its parent and children in T_i . From that point on, we refer to the MST-fragments in \mathcal{F} as *basic-fragments*. Alg. SecureLargeMSTPhase also implements $O(\log n)$ forest growing iterations, but in a somewhat distinct manner than the usual (non-secure) implementation⁸.

Step 0: Selecting Global Leader and Landmarks. The algorithm starts by defining a random sample of landmarks L , by letting each vertex $v \in V$ join L independently with probability $p = \Theta(\log n / \sqrt{n})$. Note that w.h.p. $|L| = O(\sqrt{n} \log n)$, and in addition, each basic-fragment $T_j \in \mathcal{F}$ contains at least one landmark vertex, i.e., $V(T_j) \cap L \neq \emptyset$. The IDs of the sampled vertices L are broadcast over a BFS tree to all the vertices. Note that this can be done in a non-secure manner as these are simply a random sample of the vertices (which does not relate to the edge-weights of the graph). We also select an (arbitrary) global leader \mathbf{g}^* (in a non-secure manner), this can be done e.g., by [30]. We let each landmark $u \in L$ send its basic-fragment ID to the leader \mathbf{g}^* in a *secure* manner. That is, for every u , letting $T(u) \in \mathcal{F}$ be the fragment containing u , we apply Alg. SecureSimUnicast($\{u, \mathbf{g}^*, ID(T(u))\}_{u \in L}$) of Theorem 6. Note that the basic-fragments IDs may leak information to the eavesdropper, and therefore it is essential that they are sent securely⁹. From this point on, the algorithm implements $Q = O(\log n)$ iterations of fragment merging. Throughout this process, the basic-fragments of \mathcal{F} are merged into larger components (of possibly large diameter). To control the round complexity of the algorithm, we restrict the communication to be one of two types: (A) secure-aggregation inside all basic-fragments (in parallel), and (B) secure unicasts for pairs in $\{\mathbf{g}^*\} \times L$. Since the depth of each basic-fragment is $O(\sqrt{n} \log n)$, type (A) will be implemented in $\tilde{O}(D + f\sqrt{n})$ rounds using Claim 14, with diameter bound $\hat{D} = \Theta(\sqrt{n} \log n)$. Since, w.h.p., we have $|L| = \tilde{O}(\sqrt{n})$ landmarks, type (B) will be implemented in $\tilde{O}(D + \sqrt{n})$ rounds, by Theorem 6. The duration of each of the Q iterations is set to be a fixed bound $K' = \tilde{O}(D + \sqrt{n})$.

The input for iteration $i \in \{1, \dots, Q\}$ is a forest $\mathcal{S}_i = \{S_{i,1}, \dots, S_{i,k_i}\}$ of MST-fragments that respects¹⁰ the initial partitioning into basic-fragments \mathcal{F} . For a vertex v , let $S_i(v)$ be the (unique) MST-fragment in \mathcal{S}_i containing v . At the beginning of every iteration i , the following invariant holds w.r.t the input forest \mathcal{S}_i :

⁸ E.g., it is not restricted to star-shaped merges, and it is employed in a more global, centralized manner.

⁹ For example, if two neighboring landmarks have the same fragment ID, the eavesdropper can deduce that the weight of the edge between them is relatively small.

¹⁰ In the sense that for each $T_j \in \mathcal{F}$ there exists a unique $S_{i,\ell} \in \mathcal{S}_i$ that fully contains T_j .

- (1a) Every fragment $S_{i,j}$ is a union of basic-fragments from \mathcal{F} .
- (1b) Each v knows the identifier $ID(S_i(u))$ for every $u \in \{v\} \cup N(v)$.
- (1c) The endpoints of each MST-edge in the fragment $S_{i,j}$ know that this edge is in the MST, for every $S_{i,j} \in \mathcal{S}_i$.
- (1d) The global leader \mathbf{g}^* holds the identifier of $S_i(u) \in \mathcal{S}_i$ for every $u \in L$ (hence, in particular w.h.p., it knows all identifiers of the fragments \mathcal{S}_i).

The initial forest is given by $\mathcal{S}_1 = \mathcal{F}$. By Lemma 16 and Step 0, one can see that the invariant is satisfied w.r.t \mathcal{S}_1 (i.e., the beginning of iteration 1). We next describe the i -th iteration of the algorithm for $i \in \{1, \dots, Q\}$, in which given a forest \mathcal{S}_i that satisfies the i -th invariant, outputs a forest \mathcal{S}_{i+1} that satisfies the $(i+1)$ -th invariant.

Iteration i of SecureLargeMSTPhase. Given the i -th forest $\mathcal{S}_i = \{S_{i,1}, \dots, S_{i,k_i}\}$, an edge (u, v) is called i -outgoing if u and v belong to distinct components in \mathcal{S}_i . For a basic-fragment $T_j \in \mathcal{F}$, its i -MWOE is the edge of minimum weight among all i -outgoing edges that have an endpoint in $V(T_j)$, if such exists. The iteration consists of the following steps.

Step 1: Computing i -MWOEs of Basic-Fragments. In this step, the landmarks learn the i -MWOEs of their basic-fragments (in a secure-manner). By the invariant (1b) for \mathcal{S}_i , each vertex v can locally compute its minimum weight i -outgoing edge, $e_i(v) = (v, w)$, if exists. If such an edge exists, let $x_v = \langle W(e_i(v)), e_i(v), ID(S_i(w)) \rangle$ be the tuple containing the information on that edge: weight, ID and the ID of the i -fragment to which the second endpoint belongs. If $e_i(v)$ does not exist, let $x_v = \langle N, (v, v), ID(S_i(v)) \rangle$ for some very large number $N \in \text{poly}(n, W)$, where W is the maximum edge-weight in G (those tuples will be ignored due to their large weights). By Employing Alg. SecureParallelAggregate on all basic-fragments in \mathcal{F} with the function $g = \min$ and diameter bound $\hat{D} = \Theta(\sqrt{n} \log n)$, the vertices in each T_j learn the tuple of the i -MWOEs of T_j . Hence, each landmark $u \in L$ knows the tuple of the i -MWOE of its basic-fragment (if exists), denoted hereafter¹¹, by $M^\uparrow(u) = \langle W(e), e = (x, y), ID(S_i(y)) \rangle$ where $y \notin S_i(u)$.

Step 2: Secure Unicasts and Local Merging. Apply Alg. SecureSimUnicast of Theorem 6 to allow each landmark $u \in L$ to send $M^\uparrow(u)$ to the global leader \mathbf{g}^* . These $|L|$ secure-unicast procedures can be implemented in $\tilde{O}(D + |L|)$ rounds. We are guaranteed that the adversary learns nothing on these messages. Equipped with the collection of the $\{M^\uparrow(u), u \in L\}$ tuples, the global leader \mathbf{g}^* simulates the merging of the fragments in \mathcal{S}_i along their i -MWOEs. Specifically, \mathbf{g}^* locally computes a maximal spanning forest \mathcal{F}_i in a contracted graph obtained from G by contracting each fragment in \mathcal{S}_i into a super-vertex. Note that each edge of \mathcal{F}_i corresponds to a G -edge. To avoid cumbersome notation, we refer to \mathcal{F}_i as the collection of these G -edges, note that this set corresponds to the MST edges added in that iteration. The output of this local computation provides the partitioning $\mathcal{S}_{i+1} = \{S_{i+1,1}, \dots, S_{i+1,k_{i+1}}\}$ where specifically, each $S_{i+1,j}$ is identified by the global leader \mathbf{g}^* by the set of landmarks that this fragment contains. The global leader \mathbf{g}^* then determines a unique ID to each $S_{i+1,j}$ (e.g., maximum ID of the landmark that it contains). Consequently, \mathbf{g}^* knows the fragment ID of $S_{i+1}(u) \in \mathcal{S}_{i+1}$ for each landmark $u \in L$ (as required by (1c)).

This step terminates by letting the global leader \mathbf{g}^* send the landmarks the information on the output fragment collection \mathcal{S}_{i+1} . This information can be compressed by sending each landmark only $O(\log n)$ bits, in the following manner. The leader \mathbf{g}^* considers the MSF

¹¹We use the notation M^\uparrow to denote messages from landmarks to the global leader, and M^\downarrow to denote messages from the global leader to the landmarks.

\mathcal{F}_i obtained over the fragments of \mathcal{S}_i . Note that each connected component in that MSF corresponds to a merged fragment in \mathcal{S}_{i+1} , and the edges in such connected components correspond to the newly added edges to the MST. The leader \mathbf{g}^* orients all the edges in \mathcal{F}_i towards an arbitrary root, and for every fragment $S_{i,j}$, let $e_{i,j}^*$ be the G -edge connecting $S_{i,j}$ to its parent in \mathcal{F}_i . For a landmark $u \in L$ belonging to $S_{i,j}$, the leader \mathbf{g}^* defines a message $M^\downarrow(u) = \langle ID(S_{i+1}(u)), e_{i,j}^* \rangle$, consisting of the fragment-ID of u in \mathcal{S}_{i+1} , along with the added MST-edge $e_{i,j}^*$. (For the landmarks u of basic-fragments of the root fragment in \mathcal{F}_i , it set $M^\downarrow(u) = \langle ID(S_{i+1}(u)), \bar{0} \rangle$, where $\bar{0}$ is the all-zero message.) The messages $\{M^\downarrow(u), u \in L\}$ are sent from \mathbf{g}^* to the corresponding vertices $u \in L$, by applying Alg `SecureSimUnicast`. Note that for landmarks u, u' with $S_i(u) = S_i(u')$, it holds that $M^\downarrow(u) = M^\downarrow(u')$.

Step 3: Global Merging. By the end of Step 2, each landmark u obtains its new fragment-ID in \mathcal{S}_{i+1} and the identity of one newly added MST-edge $e = (x, y)$ for $y \notin S_{i+1}(u)$ (if exists). The received message $M^\downarrow(u)$ is broadcast over the basic-fragment of u using Alg. `SecureParallelAggregate`. Since the landmarks u of each basic-fragment of $S_{i,j}$ have the same message $M^\downarrow(u)$ for every $S_{i,j} \in \mathcal{S}_i$, it holds that the entire fragment $S_{i,j}$ receives the same information (and in a secure manner).

At this point, each vertex v learns its fragment ID in \mathcal{S}_{i+1} , i.e., $ID(S_{i+1}(v))$. In addition, for each newly added MST-edge $e_{i,j}^* \in \mathcal{F}_i$, there is exactly *one* endpoint of that edge that learns that this edge is in the output MST. We next apply the secure single-round simulation of Claim 15 which simulates the following (non-secure) round: each vertex v sends $ID(S_{i+1}(v))$ to its neighbors, and acknowledges its neighbors on its current incident edges in the MST. This completes the description of iteration i , see Fig. 1 for an illustration.

Correctness. By Lemma 16, the forest constructed in the Part 1 by Alg. `SecureSmallMST` is contained in the MST of G . We next show that the $(i+1)$ -th invariant is satisfied at the end of the i -th iteration. By Step 0, the invariant holds for \mathcal{S}_1 . We next focus on iteration i . Invariant (Ia) follows by the merging process, and noting that the partitioning of \mathcal{S}_{i+1} respects the partitioning of \mathcal{S}_i . By Step 2, the global leader \mathbf{g}^* obtains $ID(S_{i+1}(u))$ for every $u \in L$, and by Step 3, this information arrives to the entire basic-fragment of u . Since, w.h.p. (by Chernoff), each basic-fragment contains a landmark, we get that all vertices v learn the identifier $ID(S_{i+1}(v))$. By the end of Step 3, each vertex also learns the $(i+1)$ -fragment ID of its neighbors, satisfying (Ib). In addition, the endpoints of the newly added MST-edges, i.e., the edges of \mathcal{F}_i hold this information, thus satisfying (Ic). Finally (Id) follows immediately by Step 2 as the global leader \mathbf{g}^* determines these identifiers locally.

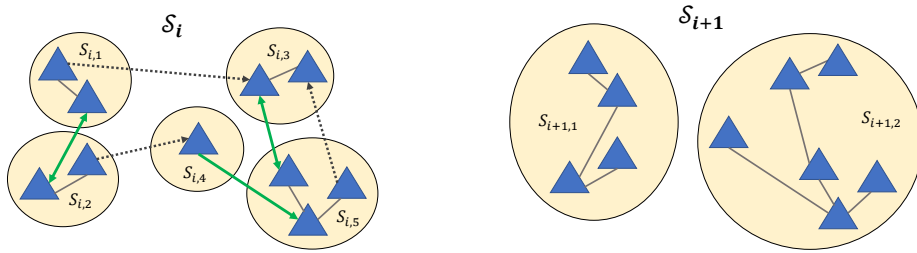
We next show that, w.h.p., in each iteration i the global leader \mathbf{g}^* obtains the i -MWOE of every fragment $S_{i,j} \in \mathcal{S}_i$. By property (Ia) of the invariant, each component $S_{i,j}$ is composed of a union of basic-fragments in \mathcal{F} . Therefore, the i -MWOE of $S_{i,j}$ is taken as the edge of minimum weight among all i -MWOEs of the basic-fragments $T_k \subseteq S_{i,j}$. Note that the global leader can safely ignore any received tuple $M^\uparrow(u)$ for $u \in L$ that contains edge-weight of $N \in \text{poly}(n, W)$. Since w.h.p. each basic-fragment contains a landmark, we get that the global leader obtains the i -MWOEs of all basic-fragments. Since it also knows the i -th fragment-ID of each landmark u , it can compute the i -MWOEs of each fragment in \mathcal{S}_i . Therefore, the forest \mathcal{F}_i consists of MST edges, as desired.

We are left to show that, with high probability, by the end of all $Q = O(\log n)$ iterations of the second part, the final output \mathcal{S}_Q contains a single MST-fragment, and therefore the algorithm outputs an MST of the graph G . In each iteration i , the leader \mathbf{g}^* merges the fragments in \mathcal{S}_i along their i -MWOEs. Since each fragment \mathcal{S}_i is merged along its i -MWOEs, the size of \mathcal{S}_{i+1} decreases by a factor of at least $1/2$ (i.e., $|\mathcal{S}_{i+1}| \leq 1/2 \cdot |\mathcal{S}_i|$). Thus, after $Q = O(\log n)$ iteration, the output consists of a single MST-fragment: the MST of G .

Round Complexity. The round complexity of Part 1 is $\tilde{O}(D + f\sqrt{n})$ by Cor. 17. We next bound the round complexity of `SecureLargeMST`. Since $|L| = O(\sqrt{n} \log n)$, w.h.p., Step 0 is implemented in $\tilde{O}(D + \sqrt{n})$ rounds. We next focus on iteration $i \in \{1, \dots, Q\}$, and show that it can be implemented in $\tilde{O}(D + f\sqrt{n})$ rounds. By Claim 14, Step 1 is implemented in $\tilde{O}(D + f \cdot \sqrt{n})$ rounds, since the depth of each basic-fragment is $\tilde{O}(\sqrt{n})$. By Theorem 6, Step 2 is implemented in $\tilde{O}(D + \sqrt{n})$ rounds as $|L| = O(\sqrt{n} \log n)$. Finally, by Claim 15 and Claim 14, Step 3 is implemented in $\tilde{O}(D + f\sqrt{n})$ rounds, as well.

Security. The security of Part 1 follows by Cor. 17. Consider Alg. `SecureLargeMSTPhase`. During the algorithm, the entire communication is performed using three subroutines: (1) secure-aggregation inside the basic-fragments, using Alg. `SecureParallelAggregate` of Claim 14; (2) secure unicasts for pairs in $\{r\} \times L$ using Alg. `SecureSimUnicast` of Theorem 6, and (3) secure single-round implemented using the secure single-round simulation of Claim 15.

In the high level, each of the subroutines is f -secure, in the sense that the eavesdropper gains no information from the messages exchanged throughout that execution due to Theorem 6, Claim 14, and Claim 15. Additionally, the subroutines used in each step (and their ordering) are fixed and do not depend on the input (i.e., the edge-weights) or output (i.e., the constructed MST) of the algorithm. Moreover, the duration of all subroutines used are independent of the input and output, revealing no additional information to the adversary.



■ **Figure 1** An illustration of iteration i . **Left:** The fragments in \mathcal{S}_i are illustrated by the yellow circles, where the blue triangles are the basic-fragments, computed in the first part. The dashed and green directed edges are the i -MWOEs of the basic-fragments, where the tails of the edges are the proposing fragments. Note that each basic-fragment in a given fragment in \mathcal{S}_i can propose a distinct outgoing edge. The green edges are those selected by the global leader g^* . **Right:** The output of the i -th iteration \mathcal{S}_{i+1} , obtained by merging the fragments in \mathcal{S}_i over the selected green edges.

References

- 1 Ioannis Anagnostides, Christoph Lenzen, Bernhard Haeupler, Goran Zuzic, and Themis Gouleakis. Brief announcement: Almost universally optimal distributed laplacian solver. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 372–374. ACM, 2022.
- 2 Ning Cai and Raymond W Yeung. Secure network coding on a wiretap network. *IEEE Transactions on Information Theory*, 57(1):424–435, 2010.
- 3 Justin Y. Chen, Shyam Narayanan, and Yinzhan Xu. All-pairs shortest path distances with differential privacy: Improved algorithms for bounded and unbounded weights. *CoRR*, abs/2204.02335, 2022.
- 4 Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22–24, 1990, Volume I*, pages 36–45. IEEE Computer Society, 1990.

- 5 Michael Elkin. A simple deterministic distributed MST algorithm with near-optimal time and message complexities. *J. ACM*, 67(2):13:1–13:15, 2020. doi:10.1145/3380546.
- 6 Jon Feldman, Tal Malkin, Cliff Stein, and Rocco A Servedio. On the capacity of secure network coding. In *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, pages 63–68. Cambridge University Press, 2004.
- 7 Juan A. Garay, Shay Kutten, and David Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees (extended abstract). In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 659–668. IEEE Computer Society, 1993. doi:10.1109/SFCS.1993.366821.
- 8 Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015*, pages 3–12. ACM, 2015.
- 9 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219. SIAM, 2016.
- 10 Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 213–221. ACM, 2021.
- 11 Mohsen Ghaffari and Goran Zuzic. Universally-optimal distributed exact min-cut. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 281–291. ACM, 2022.
- 12 Oded Goldreich. *The Foundations of Cryptography – Volume 2: Basic Applications*. Cambridge University Press, 2004. doi:10.1017/CB09780511721656.
- 13 Bernhard Haeupler. The quest for universally-optimal distributed algorithms (invited talk). In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPICs*, pages 1:1–1:1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 14 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing – 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2016.
- 15 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. *Distributed Comput.*, 34(1):79–90, 2021.
- 16 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1166–1179. ACM, 2021.
- 17 Yael Hitron and Merav Parter. General CONGEST compilers against adversarial edges. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPICs*, pages 24:1–24:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 18 Yael Hitron, Merav Parter, and Eylon Yogev. Broadcast CONGEST algorithms against eavesdroppers. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPICs*, pages 27:1–27:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 19 Taisuke Izumi, Naoki Kitamura, Takamasa Naruse, and Gregory Schwartzman. Fully polynomial-time distributed computation in low-treewidth graphs. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11–14, 2022*, pages 11–22. ACM, 2022.

- 20 Kamal Jain. Security based on network topology against the wiretapping attack. *IEEE Wirel. Commun.*, 11(1):68–71, 2004.
- 21 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
- 22 Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. Low-congestion shortcut and graph parameters. *Distributed Comput.*, 34(5):349–365, 2021.
- 23 Shimon Kogan and Merav Parter. Low-congestion shortcuts in constant diameter graphs. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 203–211. ACM, 2021.
- 24 Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in $O(\text{congestion} + \text{dilation})$ steps. *Comb.*, 14(2):167–186, 1994.
- 25 Xiaoye Li, Jing Yang, Zhenlong Sun, and Jianpei Zhang. Differential privacy for edge weights in social networks. *Secur. Commun. Networks*, 2017:4267921:1–4267921:10, 2017.
- 26 Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233:3–36, April 2001. doi:10.1016/S0012-365X(00)00224-7.
- 27 Merav Parter and Eylon Yogev. Distributed algorithms made secure: A graph theoretic approach. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1693–1710. SIAM, 2019.
- 28 Merav Parter and Eylon Yogev. Low congestion cycle covers and their applications. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1673–1692, 2019.
- 29 Merav Parter and Eylon Yogev. Secure distributed computing made (nearly) optimal. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 – August 2, 2019*, pages 107–116, 2019.
- 30 David Peleg. Time-optimal leader election in general networks. *J. Parallel Distributed Comput.*, 8(1):96–99, 1990.
- 31 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.
- 32 David Peleg and Vitaly Rubinfeld. A near-tight lower bound on the time complexity of distributed MST construction. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 253–261. IEEE Computer Society, 1999.
- 33 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 363–372. ACM, 2011.
- 34 Adam Sealfon. Shortest paths and distances with differential privacy. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 – July 01, 2016*, pages 29–41. ACM, 2016.