On Private Computation in Incomplete Networks*

Amos Beimel
Dept. of Computer Science,
Ben Gurion University, Beer Sheva 84105, Israel.
Email: beimel@cs.bgu.ac.il.

July 4, 2006

Abstract

Suppose that some parties are connected by an incomplete network of reliable and private channels. The parties cooperate to execute some protocol. However, the parties are curious – after the protocol terminates each party tries to learn information from the communication it heard. We say that a function can be computed privately in a network if there is a protocol in which each processor learns only the information implied by its input and the output of the function (in the information theoretic sense).

The question we address in this paper is what functions can be privately computed in a given incomplete network. Every function can be privately computed in two-connected networks with at least three parties. Thus, the question is interesting only for non two-connected networks. Generalizing results of [Bläser et al. CRYPTO 2002], we characterize the functions that can be computed privately in simple networks – networks with one separating vertex and no leaves. We then deal with private computations in arbitrary non two-connected networks: we reduce this question to private computations of related functions on trees, and give some sufficient conditions and necessary conditions on the functions that can be privately computed on trees.

Key Words. Private computation, Incomplete communication networks, Connectivity.

*This work was partially supported by the Frankel Center for Computer Science. Part of this work was done while the author was on sabbatical at the University of California, Davis, partially supported by the David and Lucile Packard Foundation. A preliminary version of this paper was published in *Proc. of the 12th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3499 of *Lecture Notes in Computer Science*, pages 18–33, 2005.

1 Introduction

The question of private computation of functions on communication networks is a fundamental question. For example, we would like to compute the output of an electronic election without revealing the votes of individuals. The general scenario we consider is of some parties connected by a synchronous incomplete network of reliable and private channels, where each party has a secret input. The parties cooperate to honestly execute some protocol computing a given function, but they are curious. That is, after the protocol terminates, each party tries to learn information from the communication it heard. A protocol is private if each curious party does not gain any information that is not implied by its input and the output of the function (in the information theoretic sense). This is a special case of t-privacy which requires that any colluding coalition of at most t parties cannot learn additional information. For brevity, in this paper we use the term privacy to denote 1-privacy.

Many papers dealing with private computation, e.g., [3, 8, 9], assume that the communication network is complete, that is, there is a private and reliable communication channel between each pair of parties. The question we address in this paper is what functions can be privately computed in a given incomplete network. If the network is sufficiently connected, then the situation is simple as proved by [3, 8, 12, 13].

Theorem 1.1 *If $n > 3$ and the network G is two-connected, then every function can be privately computed in G.*

Bläser et al. [4] characterize the Boolean functions that can be privately computed in simple non two-connected networks, that is, in connected networks with one separating vertex and *2* two-connected components. We consider the more general question that naturally arises.

Our Goal. *Given a communication network, characterize which functions can be privately computed in this network.*

1.1 Our Results

We first consider simple networks with one separating vertex, arbitrary number of two-connected components, and no leaves. We give an exact characterization of the functions that can be computed privately in such a network. This result generalizes the result of Bläser et al. [4] characterizing the *Boolean* functions that can be privately computed in such a network with 2 two-connected components. While Boolean functions that can be privately computed in such networks have a very simple structure ("if then else" functions), the non-Boolean functions that can be privately computed in such networks have a richer structure. Our proof is somewhat simpler than the proof of [4], and has two stages: We first reduce the private computation in such a network with n two-connected components to private computation of a related function with n variables in a simpler model which we call the eavesdropper model; this reduction uses ideas similar to the player substitution method of [20]. We then characterize the functions that can be privately computed in the eavesdropper model.

We next consider private computations in arbitrary non two-connected networks. In this case, characterizing the functions that can be privately computed is more complicated. We reduce the private computation of a function in arbitrary networks to private computation of a related function on a tree. The idea of this reduction is that we can replace each two-connected component in the network by a single vertex holding the inputs of the component. We then give sufficient conditions and necessary conditions on the functions that can be privately computed on trees. However, the conditions are not tight and we do not know the exact characterization of the functions that can be privately computed on trees. As an example of the difficulty of the characterization, we characterize the functions that can be privately computed in a simple network with one two-connected component and one leaf; this characterization is already more complicated than the characterization for networks with 2 two-connected components.

1.2 Historical Notes

There are a few models of secure computation. One distinction is whether the "bad" parties have unlimited power (the "information theoretic model") or they are polynomial-time randomized machines (the "computational model"). The other distinction is whether the "bad" parties are honest-but-curious, or they are malicious, that is, they deviate from their protocol to gain more information or to disrupt the computation. In this work we consider honest-but-curious parties with unlimited power. We review some previous results concerning this model. Chaum, Crépeau, and Damgård [8] and Ben-or, Goldwasser, and Wigderson [3] proved that in a complete network with n parties, if $n > 2t$, then every function can be computed t-privately. Kushilevitz [23] characterizes the functions that can be privately computed in a network with two parties. Chor and Kushilevitz [9] characterize the Boolean functions that can be computed t-privately in complete networks when $n \leq 2t$. The question of characterizing the (non-Boolean) functions that can be computed t-privately in complete networks when $n \leq 2t$ is still open. All these works, as well as our work, assume that the network is synchronous.

We next consider private computation in incomplete networks. Dolev, Dwork, Waarts, and Yung [13] have proved that if there are at most t honest-but-curious parties, then every pair of parties can communicate privately if and only if the network is $(t+1)$-connected. Bläser, Jakoby, Liśkiewicz, and Manthey [4], in a work that inspired the current work, characterize the *Boolean* functions that can be privately computed in a network with one separating vertex and 2 two-connected components. They also considered the randomness required for private protocols in incomplete networks. Jakoby, Liśkiewicz, and Reischuk [21] considered tradeoffs between randomness and connectivity in private computation. Finally, Bläser et al. [5] consider protocols that reveal minimum information for functions that cannot be computed privately in a given incomplete network.

The connectivity requirements for several distributed tasks in several models has been studied in many papers; for example Byzantine agreement [12, 16], approximate Byzantine agreement [14, 31], reliable message transmission [12, 13], and reliable and private message transmission [27, 13, 28, 29, 30]. Simple impossibility results and references can be found in [16, 24]. Connectivity requirements in partially authenticated networks has been considered in [1, 2]. Secure communication and secure computation in multi-recipient (multi-cast) models have been studied in [19, 18, 17, 10]. Secure computation in directed networks has been studied in [11]. Secure communication against general adversarial structures has been studied in [22].

Organization. In Section 2, we describe our model and present some background on connectivity. In Section 3, we characterize the functions that can be privately computed in networks with one separating vertex and no leaves. In Section 4, we reduce private computation of functions in arbitrary networks to private computation on trees of related functions, and, in Section 5, we give sufficient conditions and necessary conditions for the functions that can be privately computed on trees. Finally, in Section 6 we conclude and mention some open problems.

2 Preliminaries

2.1 The Model

The communication network is modeled by an undirected graph $G = (V, E)$, where

- The vertices $V = \{v_1, v_2, \ldots, v_n\}$ are the parties in the network. We denote their number by n (i.e., $|V| = n$); in the sequel we refer to parties as vertices.
- The edges E describe the communication channels. That is, there is an edge (u, v) in E if and only if there is a communication channel between u and v. We assume that these communication channels are reliable and private: an adversary that does not control u or v (but might control all other vertices in the network) cannot read, change, delete, or insert messages sent on the edge (u, v).

Protocols. We consider an n-party protocol for computing a given function executed in a *synchronous* network. Briefly, in the beginning of the protocol, each vertex v_i has a private *input* a_i and a private *random input* r_i, where r_i is distributed uniformly in some finite domain (the random inputs $\langle r_1, \ldots, r_n\rangle$ are independent). A protocol π computes its output in a sequence of rounds. For a round j, let $i \leftarrow (j \bmod n) + 1$. In Round j, only Vertex v_i is active and sends a message $h_{j,k}$ (i.e., a string) to v_k for each of its neighbors; this message will become an available input to v_k in the next round. If v_k is not a neighbor of v_j, then $h_{j,k}$ is the empty string. The message $h_{j,k}$ is a function of the round number j, the receiver k, the sender's input a_i, the sender's random input r_i, and the previous messages v_i got, i.e., $\langle h_{j',i}\rangle_{1\le j'<j}$. A computation of the protocol ends in a round in which each vertex computes an *output*. The assumption that only one vertex is active in each round is made only to simplify notations and does not affect the generality of our results. We next define the view of a set of parties after an execution of a protocol as all the information they have after the execution, namely, their inputs, random inputs, and the messages they heard during the execution.

Definition 2.1 (Transcripts and Views) *Let $C \subseteq \{v_1, \ldots, v_n\}$ be a subset of the parties. Considering an execution of a protocol π on inputs $\langle a_1, \ldots, a_n\rangle$ and random inputs $\langle r_1, \ldots, r_n\rangle$, we make the following definitions:*

- *The* transcript *of C in the execution is the sequence of messages that vertices in C get during the execution; it is denoted by $\mathrm{T}_C(a_1, \ldots, a_n, r_1, \ldots, r_n)$.*
- *The* view *of C is the triplet $\langle \langle a_i\rangle_{v_i\in C}, \langle r_i\rangle_{v_i\in C}, \mathrm{T}_C(a_1, \ldots, a_n, r_1, \ldots, r_n)\rangle$; it is denoted by*

$$\mathrm{VIEW}_C(a_1, \ldots, a_n, r_1, \ldots, r_n).$$

We consider the random variables $\mathrm{T}_C(a_1, \ldots, a_n, \langle r_i\rangle_{v_i\in C})$ obtained by randomly selecting $\langle r_i\rangle_{v_i\notin C}$ and outputting $\mathrm{T}_C(a_1, \ldots, a_n, r_1, \ldots, r_n)$. We also consider the similarly defined random variables for

$$\mathrm{VIEW}_C(a_1, \ldots, a_n, \langle r_i\rangle_{v_i\in C}).$$

In the model we consider, the n-party honest-but-curious model, parties are curious, that is, each party may try to deduce as much information as possible from its own view of an execution about the private inputs of the other parties. However, each party is honest, that is, it scrupulously follows the instructions of the protocol. In such conditions, it is easy to enforce the correctness condition (for securely computing a function f), but not necessarily the privacy conditions.

In the following definition we consider functions $f : A_1 \times \ldots \times A_n \to O$, where $A_1, \ldots, A_n$ and O are some finite sets, and the ith input of f is the input of v_i. The privacy requirement we consider is unconditional, that is, even a curious adversary with unlimited power will not gain information. Furthermore, we consider perfect security, that is, we require no error in the correctness, and exactly the same distributions in the privacy requirement. In the following definition we define privacy against an adversarial structure $S \subseteq V$, that is, S is a subset of the vertices. We require that each party in S will not gain information from its view that is not implied by its input and the output of the function. (Parties not in S and sets of parties of size at least two might learn information.) In this work we mainly want to protect the privacy against each individual, namely achieve privacy. We define the more general case of S-privacy as it is used as a tool to characterize privacy. We note that our notion of adversarial structure is a simplification of the notion used in other papers. In the more common notion, an adversarial structure is a collection of subsets of the parties and the protocol should "protect the privacy" even if parties of a set in the adversarial structure collude and try to gain information.

Definition 2.2 (Private Computation) *Let $G = (V, E)$ be network with n vertices, $A_1, \ldots, A_n$, and O be finite sets, $f : A_1 \times \ldots \times A_n \to O$ be a function, and $S \subseteq V$ be an adversarial structure. A protocol π* S-privately computes *f, if the following conditions hold:*

CORRECTNESS. *For every vector of inputs $\langle a_1, \ldots, a_n \rangle$ and every vector of random inputs $\langle r_1, \ldots, r_n \rangle$, the output of each v_i with $\text{VIEW}_{\{v_i\}}(a_1, \ldots, a_n, r_1, \ldots, r_n)$ is $f(a_1, \ldots, a_n)$.*

PRIVACY. *For every $v_i \in S$, for every $\langle a_1, \ldots, a_n \rangle \in A_1 \times \ldots \times A_n$ and every $\langle a'_1, \ldots, a'_n \rangle \in A_1 \times \ldots \times A_n$ such that $a_i = a'_i$, and every r_i, if $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$, then the random variables $\text{VIEW}_{\{v_i\}}(a_1, \ldots, a_n, r_i)$ and $\text{VIEW}_{\{v_i\}}(a'_1, \ldots, a'_n, r_i)$ are equally distributed.*

A function f can be computed privately in G if there is a protocol π that V-privately computes f in G.

We require that the privacy is protected only when $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$ since each vertex learns the output of f (and knows its input). We assume that all parties in the system know the topology of the graph G. Furthermore, we assume that the system is synchronous and all the parties in the system know in which round the protocol starts. We note that our definition of privacy is a special case of t-privacy which requires that any colluding coalition of at most t parties cannot learn additional information (for a formal definition of t-privacy see, e.g., [9]). For brevity, in this paper we use the term privacy to denote 1-privacy.

2.2 Modular Composition of Private Protocols

To prove the privacy of protocols in this paper we use the modular composition paradigm [26, 7]. We want to be able to design private protocols for simple tasks and then use them in more complex protocols as subroutines; the goal is to prove the privacy of the complex protocols while relying on the privacy of the subroutines. This approach, which is the natural approach in the design of non-private protocols, has been formulated and proved for several models of secure computation in, e.g., [26, 7]. Specifically, for the model considered in this paper, the proof of the following theorem is quite simple.

Theorem 2.3 (Modular Composition Theorem [26, 7]) *Assume that there are protocols $\pi_1, \ldots, \pi_m$ privately computing functions $f_1, \ldots, f_m$ respectively. Furthermore, there is a protocol π privately computing a function g while using calls to ideal evaluations of $f_1, \ldots, f_m$. Then, the protocol $\pi^{\pi_1, \ldots, \pi_m}$, obtained by replacing each subroutine call to the ideal evaluation of f_i by the protocol π_i, privately computes g.*

2.3 Connectivity

The reliability of a network is closely related to its connectivity. In this section, we review the relevant concepts related to connectivity. For more details, the reader can consult, e.g., [6, 15].

We consider *vertex* connectivity of *undirected* graphs. A graph $G = (V, E)$ is connected if for every two vertices u, v there is a path connecting them in G. In this paper, we only consider connected graphs. A vertex $z \in V$ is called a *separating vertex* (or a cut-vertex) if, for some $u, v \in V \setminus \{z\}$, every path between u and v passes through z. For a connected graph, a vertex z is a separating vertex if and only if removing z from G results in an unconnected graph. A connected graph is two-connected if it contains at least 3 vertices and it does not contain a separating vertex. By a result of Menger [25], a graph G with at least 3 vertices is two-connected if and only if for every vertices $u, v \in V$ either $(u, v) \in E$ or there exist two vertex-disjoint paths between u and v in G. An edge e is a *bridge* if for some $u, v \in V$, every path between u and v passes through e.

A subgraph B of G is a *two-connected component* if it is a maximal two-connected induced subgraph of G. We next define the component graph of a connected graph, which replaces every two-connected component in G by a single vertex.[1]

Definition 2.4 (Component Graph) *Given a connected graph $G = (V, E)$, we define its component graph $T_G = (V', E')$ as follows. The vertices in V' are:*

- *The two-connected components of G. For a two-connected component W in G, we denote the corresponding vertex in T_G by v_W.*
- *The leaves in G.*
- *The separating vertices in G.*

There is an edge in E' between every separating vertex and every two-connected component containing it, between a leaf and its neighboring separating vertex, and between two separating vertices connected by a bridge.

For example, a graph G_0 and its component graph are described Figure 1. In G_0, there are 2 two-connected components $W_0 = \{v_1, v_2, v_3\}$ and $W_1 = \{v_3, v_4, v_5\}$, two separating vertices v_3 and v_5, and one leaf v_6. Thus, the component graph T_{G_0} of G_0 has 5 vertices.

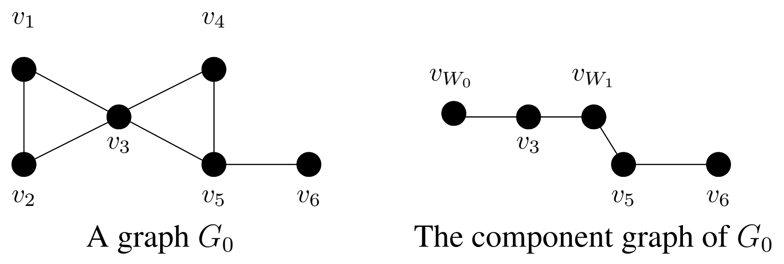


Figure 1: An example of a graph and its component graph.

By Menger's Theorem, every cycle in a graph G is contained in exactly one two-connected component. This fact implies the following observation.

[1] The component graph we define is similar to the block-cutvertex graph as defined in [6].

Observation 2.5 *If the graph G is connected, then the graph T_G is a tree.*

3 Incomplete Networks with One Separating Vertex and No leaves

In this section we characterize the functions that can be computed privately in connected networks that contain one separating vertex and no leaves. As an intermediate step, we consider a model we call the eavesdropper model. Using this intermediate model, we characterize the functions that can be privately computed in connected networks that contain one separating vertex and no leaves. That is, we prove that a function can be privately computed in connected networks that contain one separating vertex and no leaves if and only if a related function can be computed in the eavesdropper model. Roughly speaking, the parties correspond to the two-connected components in the network and the eavesdropper is the separating vertex. To complete the characterization, we characterize the functions that can be privately computed in the eavesdropper model. Informally, the eavesdropper model corresponds to computing a function in a star – a tree with n leaves and one common central vertex – where the central vertex acts as a relay and hears all the exchanged communication. This informal definition is exactly captured in the next definition, where we avoid the requirement that the central vertex acts as a relay by replacing it with an eavesdropper that hears all communication and cannot send messages.

Definition 3.1 (The Eavesdropper Model) *Consider a network with n vertices; each vertex has a secret input taken from some finite domain. In addition there is an eavesdropper Eve. The vertices execute a protocol to compute a function $f(a_1, \dots, a_n)$. The parties can communicate only via a public broadcast channel heard by all vertices and by the eavesdropper Eve.*

We require that at the end of the protocol Eve computes $f(a_1, \dots, a_n)$; however, Eve should not learn any information on the inputs that is not implied by the output $f(a_1, \dots, a_n)$. (The vertices in the network are allowed to learn information from the protocol.) Formally, the privacy requirement is that for every two vectors of inputs $\langle a_1, \dots, a_n \rangle$ and $\langle b_1, \dots, b_n \rangle$ such that $f(a_1, \dots, a_n) = f(b_1, \dots, b_n)$ the random variables $\mathrm{T}_{\{v_1,\dots,v_n\}}(a_1, \dots, a_n)$ and $\mathrm{T}_{\{v_1,\dots,v_n\}}(b_1, \dots, b_n)$ are equally distributed.

For the impossibility results, we also consider a weaker notion of privacy.

Definition 3.2 (Weak Privacy) *We say that a protocol is weakly private if for every two vectors of inputs $\langle a_1, \dots, a_n \rangle$ and $\langle b_1, \dots, b_n \rangle$ such that $f(a_1, \dots, a_n) = f(b_1, \dots, b_n)$ and any transcript h,*

$$\Pr[\mathrm{T}_{\{v_1,\dots,v_n\}}(a_1, \dots, a_n) = h] > 0 \quad \textit{if and only if} \quad \Pr[\mathrm{T}_{\{v_1,\dots,v_n\}}(b_1, \dots, b_n) = h] > 0.$$

That is, the requirement is that a transcript is possible given $\langle a_1, \dots, a_n \rangle$ if and only if it is possible given $\langle b_1, \dots, b_n \rangle$.

3.1 Reduction to the Eavesdropper Model

We first use the eavesdropper model to characterize the functions that can be privately computed in connected networks with one separating vertex and no leaves. We consider a network $G_{k_1,\dots,k_n}$ with $(\sum_{i=1}^{n} k_i)+1$ vertices, which is composed of n two-connected components; the ith two-connected component is denoted by W_i and has $k_i + 1$ vertices. The n two-connected components share exactly one vertex denoted z. By definition, a two-connected component contains at least 3 vertices, thus, $k_i \geq 2$ for $i = 1, \dots, n$.[2] Such

[2]We assume that there are no leaves in the network; networks with one separating vertex that contain leaves are dealt in the following sections. Especially, networks with one two-connected component and one leaf are considered in Section 5.3.

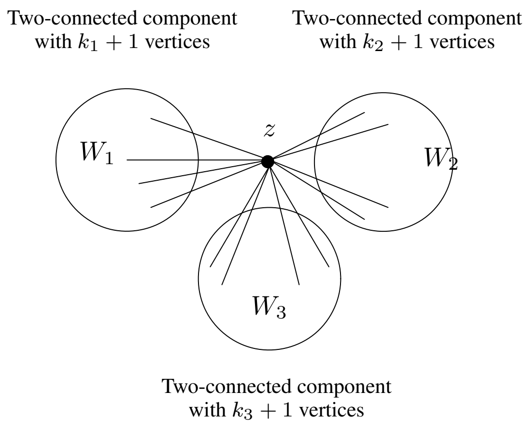

Two-connected component with k_1+1 vertices
Two-connected component with k_2+1 vertices
z
W_1
W_2
W_3
Two-connected component with k_3+1 vertices

Figure 2: A Graph G_{k_1,k_2,k_3} with three two-connected components.

a graph with 3 two-connected components is illustrated in Figure 2. Given a $((\sum_{i=1}^{n} k_i) + 1)$-argument function $f : \Pi_{i=1}^{n}\Pi_{j=1}^{k_i} A_{i,j} \times C \to O$, define for every $c \in C$, a possible value of the input of the separating vertex, an n-argument function $f_c : (\Pi_{j=1}^{k_1} A_{1,j}) \times \cdots \times (\Pi_{j=1}^{k_n} A_{n,j}) \to O$, where for every $a_1 \in \Pi_{j=1}^{k_1} A_{1,j}, \ldots, a_n \in \Pi_{j=1}^{k_n} A_{n,j}$ (that is, a_i consists of the inputs of all non-separating vertices in the ith component)

$$f_c(a_1, \ldots, a_n) \overset{\text{def}}{=} f(a_1, \ldots, a_n, c). \tag{1}$$

That is, f_c is obtained from f by fixing the input of the separating vertex to c and grouping all the inputs of the non-separating vertices in each component to one input of f_c. We next prove the reduction lemma.

Lemma 3.3 *Let $f : \Pi_{i=1}^{n}\Pi_{j=1}^{k_i} A_{i,j} \times C \to O$ be a function, where $k_i \geq 2$ for $i = 1, \ldots, n$. The function f can be privately computed in $G_{k_1,\ldots,k_n}$ if and only if for every $c \in C$ the function f_c can be privately computed in the eavesdropper model.*

The lemma is proved in the following two claims. The first claim holds for every network with one separating vertex (possibly, with leaves), while the second requires that the network contains no leaves.

Claim 3.4 *Let $f : \Pi_{i=1}^{n}\Pi_{j=1}^{k_i} A_{i,j} \times C \to O$ be a function. If the function f can be privately computed in $G_{k_1,\ldots,k_n}$, then for every $c \in C$ the function f_c can be privately computed in the eavesdropper model.*

Proof: Assume that there is a protocol π privately computing f in $G_{k_1,\ldots,k_n}$. For every $c \in C$, we construct a private protocol π_c for f_c in the eavesdropper model. The idea of the protocol for f_c is that the vertices of each two-connected component are simulated by one vertex in the eavesdropping model. Specifically, Vertex v_1, holding $a_1 \in A_{1,1} \times \ldots \times A_{1,k_1}$ simulates the $k_1 + 1$ vertices in the two-connected component W_1 including z with input c, and for $i = 2, \ldots, n$, Vertex v_i, holding $a_i \in A_{i,1} \times \ldots \times A_{i,k_i}$ simulates the k_i vertices in two-connected component W_i excluding z. Notice that only v_1 simulates the separating vertex z. For every i, Vertex v_i simulates a vertex w in W_i, where $w \neq z$, as follows:

- In the initialization stage, v_i chooses a random input for w.

- In each round that w sends messages in π
  - Vertex v_i knows all previous messages w has gotten in previous rounds,
  - it computes the messages that w sends,
  - it records, for future use, the messages that w sends to vertices in $W_i \setminus \{z\}$, and
  - it broadcasts the message that w sends to z.

Vertex v_i simulates the separating vertex z in W_i similarly; the only difference is that v_1 broadcasts all messages that z sends. At the end of the protocol, Vertex v_1 sends the output to the other vertices, thus, the eavesdropper knows the output.

The eavesdropper knows that the input of z is c, as c is fixed, and hears the messages exchanged between z and the vertices in W_i for $i = 1, \ldots, n$. Thus, the information the eavesdropper learns is at most the information that z learns in the protocol π computing f, and the privacy of the protocol π implies the privacy in the eavesdropper model of the protocol π_c for f_c. □

Claim 3.5 *Let $f : \Pi_{i=1}^{n}\Pi_{j=1}^{k_i} A_{i,j} \times C \to O$ be a function, where $k_i \geq 2$ for $i = 1, \ldots, n$. If for every $c \in C$ the function f_c can be privately computed in the eavesdropper model, then the function f can be privately computed in $G_{k_1,\ldots,k_n}$.*

Proof: Assume that, for every $c \in C$, there is a protocol π_c privately computing the function f_c in the eavesdropper model, where the input of v_i is a_i. By Corollary 3.12 (appearing in Section 3.2), we can assume that Protocol π_c is deterministic. We construct a (randomized) private protocol for f in $G_{k_1,\ldots,k_n}$ using the modular composition paradigm (explained in Section 2.2). The idea of Protocol π is that each vertex in the eavesdropping model is simulated by the vertices of a two-connected component. Since in the eavesdropper model there are no privacy requirement on the parties, the simulation needs to ensure that each non-separating vertex in the two-connected component does not learn the messages exchanged in π_c. This is achieved by using a private protocol for computing each message and masking the output of the private protocol (i.e., the message) by random bits held by the separating vertex. This ensures that each non-separating vertex does not learn the message. In contrast, the separating vertex learns the message, which enables the computation of future messages.

We next describe the protocol π more formally. Without loss of generality, assume that for every value of c the protocol π_c proceeds in rounds, where in Round j Vertex $v_{(j \bmod n)+1}$ sends a one bit message to the other vertices. Furthermore, assume without loss of generality that the protocols π_c, for all values of c, have the same communication complexity. Let π_c^j be the jth message sent in the protocol by v_i, where $i = (j \bmod n) + 1$. The protocol for f will have a virtual round for each round of the protocol for f_c. In each virtual round, vertex z picks a random bit r_j, and the parties in W_i (including z) use a subroutine call to the function $\pi_c^j \oplus r_j$. Recall that for every c, in Round j the bit π_c^j depends on a_i and the previous messages. Alternatively, we view the bit π_c^j (from the various protocols) as a single function of c, a_i, and the previous messages. As the vertices in $W_i \setminus \{z\}$ hold a_i, and the separating vertex z holds r_j and c and knows all previous messages, the bit $\pi_c^j \oplus r_j$ is indeed a function of inputs held by the parties in W_i. In the protocol computing f in $G_{k_1,\ldots,k_n}$, we replace each subroutine call by a private protocol. Such private protocol exists by Theorem 1.1, since $1 + k_i > 2$ and each component is two-connected.

We next argue that this protocol is private. By Theorem 2.3, we only need to argue that the protocol using the ideal subroutine calls is private. For every $i \in \{1, \ldots, n\}$, each vertex in the two-connected component W_i, except for z, learns only the values $\pi_c^j \oplus r_j$ for every $i = (j \bmod n) + 1$, where r_j is chosen at random by z, thus, the vertex does not learn any information during the protocol. Vertex z knows the random bits $r_1, r_2, \ldots, r_m$ and its input, thus, it knows the communication exchanged in the protocol for f_c. However, the information it gets is exactly the information the eavesdropper gets in the protocol for f_c, thus, z gains no information. $\square$

It should be mentioned that the transformation in the proof of Claim 3.5 can substantially increase the communication complexity of the resulting protocol π. In Protocol π, the parties in W_i need to compute the bit $\pi_c^j \oplus r_j$ using a private protocol; this private protocol is not necessarily efficient (as, for example, the size of the circuit computing π_c^j can be large).

3.2 The Eavesdropper Model

To characterize the functions that can be privately computed in the eavesdropper model we use ideas similar to the characterization of the functions that can be computed in the two-party model as characterized by Kushilevitz [23]. We first introduce some notation similar to [23]. We represent a function $f : A_1 \times \cdots \times A_n \to O$ by an n-dimensional array M_f whose ith dimension is labeled by the elements of A_i and $M_f(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$.

Definition 3.6 (The Equivalence Relations $\equiv_i$) *Let M be an n-dimensional array whose ith dimension is labeled by the elements of A_i for $i = 1, \ldots, n$. The relation $\sim_i$ on the elements of A_i is defined as follows: $a_i, b_i \in A_i$ satisfy $a_i \sim_i b_i$ if there exist two vectors $\langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \rangle \in A_1 \times \ldots \times A_{i-1} \times A_{i+1} \times \ldots \times A_n$ and $\langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n \rangle \in A_1 \times \ldots \times A_{i-1} \times A_{i+1} \times \ldots \times A_n$ such that $M(a_1, \ldots, a_n) = M(b_1, \ldots, b_n)$. The equivalence relation $\equiv_i$ on A_i is defined as the transitive closure of the relation $\sim_i$. That is, $a_i \equiv_i b_i$ for $a_i, b_i \in A_i$, if there are $\alpha_1, \ldots, \alpha_\ell$ such that $a_i \sim_i \alpha_1 \sim_i \alpha_2 \sim_i \cdots \sim_i \alpha_\ell \sim_i b_i$.*

Kushilevitz [23] defines similar relations for $n = 2$. However, the relation there requires that $a_1 \sim b_1$ if there exists a_2 such that $M(a_1, a_2) = M(b_1, a_2)$. To motivate the above definition, assume that $a_1 \sim_1 b_1$, thus there are $\langle a_2, \ldots, a_n \rangle$ and $\langle b_2, \ldots, b_n \rangle$ such that $f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$. Assume that v_1 sends the first message in a deterministic protocol computing f. Thus, not to violate the privacy requirement, in the first round of the protocol, v_1 has to send the same message while holding a_1 and b_1 (otherwise Eve can distinguish between $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_n \rangle$).

Definition 3.7 (Forbidden Array) *An array M is a forbidden array if the following two conditions hold:*

- *The array is not constant, and*
- *For every $i \in \{1, \ldots, n\}$, all the elements of A_i are equivalent according to $\equiv_i$.*

To understand this definition, consider a non-constant array that is not forbidden. Thus, for some i we can partition A_i into non-empty equivalence classes $A_i^1, \ldots, A_i^\ell$ (where $\ell \geq 2$). We now consider the arrays $M_1, \ldots, M_\ell$, where M_j is the restriction of M to $A_1 \times \ldots \times A_{i-1} \times A_i^j \times A_{i+1} \times \ldots \times A_n$. Then, each value that appears in M appears in exactly one M_j.

We say that an array M labeled by $A_1 \times \cdots \times A_n$ contains a forbidden array if there is a rectangle $A'_1 \times \cdots \times A'_n \subseteq A_1 \times \cdots \times A_n$ such that the array M restricted to this rectangle is forbidden. Similar to Kushilevitz [23], we prove that a function f can be privately computed in the eavesdropper model if and only if the array M_f does not contain a forbidden array.

Lemma 3.8 *A function $f : A_1 \times \cdots \times A_n \to O$ can be computed privately in the eavesdropper model if and only if the array M_f does not contain a forbidden array.*

The lemma is proved in the following two claims.

Claim 3.9 *Let $f : A_1 \times \cdots \times A_n \to O$ be a function. If the array M_f does not contain a forbidden array, then f can be privately computed in the eavesdropper model.*

A Private Protocol for f

Initialization. $R_i \leftarrow A_i$ for $i = 1, \ldots, n$.

Step. Let M be the array M_f restricted to $R_1 \times \cdots \times R_n$. As M_f does not contain a forbidden array, the array M is not forbidden.

1. If M is constant, that is, there is some $o \in O$ such that $f(a'_1, \ldots, a'_n) = o$ for every $\langle a'_1, \ldots, a'_n \rangle \in R_1 \times \cdots \times R_n$, then Eve deduces that this constant o is the output and the protocol ends.
2. Otherwise, for some $i \in \{1, \ldots, n\}$, not all the elements of M are equivalent according to $\equiv_i$. Vertex v_i broadcasts the equivalence class of a_i in R_i, and all vertices set R_i as this equivalence class.
3. Goto **Step**.

Figure 3: A private protocol computing f in the eavesdropping model.

Proof: We construct a deterministic private protocol computing f. Let $\langle a_1, \ldots, a_n \rangle$ be the vector of inputs of the vertices. In each step of the protocol, the parties maintain an n-dimensional rectangle $R_1 \times \cdots \times R_n \subseteq A_1 \times \cdots \times A_n$, known also to the eavesdropper Eve, which contains the input, that is, $\langle a_1, \ldots, a_n \rangle \in R_1 \times \cdots \times R_n$. The protocol is described in Figure 3. Notice that the array M becomes smaller in each step, and the equivalence relations $\equiv_i$ change accordingly. As M_f does not contain a forbidden array and the sets $A_1, \ldots, A_n$ are finite, the protocol must reach a constant rectangle and terminate. Since, in each stage of the protocol, $\langle a_1, \ldots, a_n \rangle \in R_1 \times \cdots \times R_n$, this protocol is correct. We next argue that Eve does not learn information on $\langle a_1, \ldots, a_n \rangle$ not implied by $f(a_1, \ldots, a_n)$, that is, if $f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$, then the same communication is exchanged on $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_n \rangle$. This follows from the fact that if $f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$, then, in each stage of the protocol $a_i \sim_i b_i$ for every $i \in \{1, \ldots, n\}$, and a_i is in the same equivalence class as b_i. $\square$

Claim 3.10 *Let $f : A_1 \times \cdots \times A_n \to O$ be a function. If the array M_f contains a forbidden array, then f cannot be weakly-privately computed in the eavesdropper model.*

Proof: Let $A'_1 \times \cdots \times A'_n$ be a forbidden array in M_f. To prove this claim, we will prove that, for every input in the forbidden array, the same communication transcripts are possible. However, since the array is not constant, for at least one input Eve errs with positive probability contradicting the correctness requirement.

Fix any transcript h that is possible given some input in $A'_1 \times \cdots \times A'_n$. Let $h = h_1 \circ h_2 \circ \cdots \circ h_m$, where h_j is the message sent by vertex $v_{(j \bmod n)+1}$ in Round j. We prove, using induction, that for every j, where $1 \leq j \leq m$, the communication $h_1 \circ h_2 \circ \cdots \circ h_j$ is possible given every input in $A'_1 \times \cdots \times A'_n$. To simplify the notations, we assume, without loss of generality, that $j = 0 \bmod n$, that is, Vertex v_1 sends the message h_j in Round j. Given $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$, the message sent by v_1 in Round j does not depend on the inputs of $v_2, \ldots, v_n$. Thus, we need to prove that for every $a \in A'_1$, the message h_j is a possible message that v_1 sends on input a and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$. Since h is possible given some input in $A'_1 \times \cdots \times A'_n$, there exists some $a_1 \in A'_1$ such that the message h_j is a possible message that v_1 sends on input a_1 and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$. By transitivity and the fact that all elements of A'_1 are equivalent according to $\equiv_i$, it is enough to prove that for every b_1 such that $a_1 \sim_i b_1$, the message h_j is a possible message that v_1 sends on input b_1 and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$.

Thus, assume that $a_1 \sim_i b_1$ and the message h_j is a possible message that v_1 sends on input a_1 and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$. We will prove that h_j is a possible message that v_1 sends on input b_1 and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$. Let $\langle a_2, \ldots, a_n \rangle$ and $\langle b_2, \ldots, b_n \rangle$ be such that $f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$. By the induction hypothesis and the assumption on a_1, the transcript $h_1 \circ h_2 \circ \cdots \circ h_{j-1} \circ h_j$ is possible given $\langle a_1, \ldots, a_n \rangle$. By the weak privacy, $h_1 \circ \cdots \circ h_j$ is a possible transcript given $\langle b_1, \ldots, b_n \rangle$, and in particular, h_j is a possible message that v_1 sends on input b_1 and communication $h_1 \circ h_2 \circ \cdots \circ h_{j-1}$. $\Box$

Lemma 3.8 implies the following simple necessary condition on the functions that can be privately computed in the eavesdropper model. This condition is not sufficient as shown in Example 3.13.

Corollary 3.11 *Let $f : A_1 \times \cdots \times A_n \to O$ be a function. If the function f can be privately computed in the eavesdropper model, then for every output value $o \in O$ there is some rectangle $A^o_1 \times \cdots \times A^o_n \subseteq A_1 \times \cdots \times A_n$ such that $f(a_1, \ldots, a_n) = o$ if and only if $\langle a_1, \ldots, a_n \rangle \in A^o_1 \times \cdots \times A^o_n$.*

Proof: For every i, let $A^o_i \stackrel{\text{def}}{=} \{a \in A_i : \exists_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n} f(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) = o\}$. Thus, for every i all elements of A_i are equivalent according to $\equiv_i$ (and even according to $\sim_i$). Since, f can be privately computed in the eavesdropper model, the array M_f does not contain a forbidden array, and f restricted to $A^o_1 \times \cdots \times A^o_n$ must be constant. $\Box$

Notice that in the proof of Claim 3.9 we construct a deterministic protocol. Thus,

Corollary 3.12 *A function $f : A_1 \times \cdots \times A_n \to O$ can be privately computed in the eavesdropper model if and only if it can be privately computed in the eavesdropper model by a deterministic protocol.*

Example 3.13 We next describe two examples of the possibility of private computation in the eavesdropper model with two parties. We considers the two functions $f_i : \{a_0, a_1, a_2\} \times \{b_0, b_1, b_2\} \to \{0, \ldots, 4\}$ for $i \in \{1, 2\}$ described below.

f_1	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	3

f_2	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	4

In both examples, the inputs corresponding to each output value are a rectangle, thus, they satisfy the necessary condition of Corollary 3.11. For example, in both examples the rectangle corresponding to the output value two is $\{a_1, a_2\} \times \{b_0\}$. For f_1, however, the array is forbidden and the function f_1 cannot be privately computed . In f_2, we changed the bottom-right entry from 3 to 4; now the array does not contain a forbidden sub-array, and the function f_2 can be privately computed . The partition induced by the private protocol is detailed in the array.

The next lemma, which is implicit in [4] for $n = 2$, states that the characterization for Boolean functions is much simpler.

Lemma 3.14 *A Boolean function $f : A_1 \times \ldots \times A_n \rightarrow \{0, 1\}$ can be privately computed in the eavesdropper model iff it depends only on one of its inputs, that is, if there exist an index i and a function f' such that $f(a_1, \ldots, a_n) = f'(a_i)$ for all $\langle a_1, \ldots, a_n \rangle \in A_1 \times \ldots \times A_n$ (in particular, f' can be constant).*

Proof: If a function f (Boolean or non-Boolean) depends only on one of its inputs, then it does not contain a forbidden array, thus, it can be privately computed in the eavesdropper model.

For the other direction, assume that a Boolean function f can be privately computed in the eavesdropper model, thus satisfies the necessary condition of Corollary 3.11. That is, the input values corresponding to each output value form a rectangle, and, as the function is Boolean, there are at most two values in the array M_f. The only possible way to partition an array to two rectangles is to partition the ith-dimension for some $i \in \{1, \ldots, n\}$, that is, if f depends only on its ith variable. $\square$

Combining Lemma 3.3 and Lemma 3.8, we get a combinatorial characterization of the functions that can be computed privately in networks with one separating vertex and no leaves.

Theorem 3.15 *Let $f : \Pi_{i=1}^{n} \Pi_{j=1}^{k_i} A_{i,j} \times C \rightarrow O$ be a function, where $k_i \geq 2$ for $i = 1, \ldots, n$. The function f can be privately computed in $G_{k_1,\ldots,k_n}$ if and only if for every $c \in C$ the array M_{f_c} does not contain a forbidden array.*

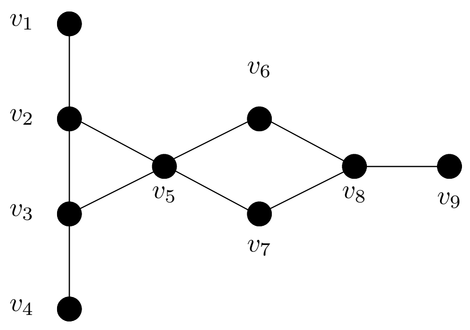
4 Networks with Many Separating Vertices

In this section we consider private computation of functions in arbitrary connected networks. As in the previous section, the characterization of the functions that can be privately computed has two stages. We first reduce the problem of private computation in the network to private computation of a related function in the component graph of G, which is a tree. In Section 5, we give some necessary conditions and sufficient conditions for computing a function privately on trees. However, the conditions are not tight and we do not give an exact characterization of these functions.

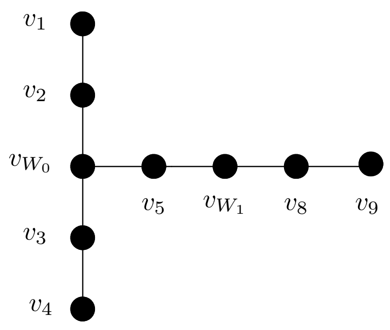
We next reduce private computation in an arbitrary connected network to private computation of a related function in a tree, namely, the component graph of the network. The reduction is similar to the reduction in Section 3.1, however, there is an important difference. In the reduction in Section 3.1 we replaced each two-connected component with a vertex (without any privacy requirement from these vertices), and then published the input of the separating vertex and replaced it by the eavesdropper. In this section we replace each two-connected component with a vertex (without any privacy requirement from these vertices). However, we cannot publish the inputs of the separating vertices since other separating vertices might learn information that they should not learn. Thus, in the tree we construct, we keep the separating vertices with the same input they had in the original network. Furthermore, we have to deal with leaves, and we keep them in the tree with their original inputs.

Formally, let G be a graph with n vertices and T_G be the component graph of G with n' vertices (as defined in Definition 2.4). We say that a vertex in G is curious if it is either a separating vertex in G or a leaf in G. Recall that the vertices in T_G are the curious vertices in G and the two-connected components. Given a function $f : A_1 \times \ldots \times A_n \rightarrow O$ we define an n'-argument function f', to be computed in T_G. In f', the input of each curious vertex is the same as the input in G and the input of each vertex v_W, for a two-connected component W in G, is the vector of inputs of the non-separating vertices in W.

Example 4.1 Consider the graph G_1 and its component graph T_{G_1} described in Figure 4. In G_1, there are 2 two-connected components $W_0 = \{v_2, v_3, v_5\}$ and $W_1 = \{v_5, v_6, v_7, v_8\}$, four separating vertices



A graph G_1.



The component graph of G_1.

Figure 4: Another graph and its component graph.

v_2, v_3, v_5, and v_8, and three leaves v_1, v_4, and v_9. Thus, the component graph of G_1 has 9 vertices. In the two-connected component W_0, all the vertices are separating vertices. We still need to have the vertex v_{W_0} in T_{G_1}, however this vertex has no input. In W_1, the non-separating vertices are v_6 and v_7, thus, the vertex v_{W_1} in T_{G_1} holds the inputs of v_6 and v_7. Note that v_6 and v_7 are separated by $\{v_5, v_8\}$; however this fact does not cause any difficulties as all the vertices of W_1 simulate v_{W_1}.

In the component graph we replaced every two-connected component W in G by one vertex v_W in T_G holding the inputs of the non-separating vertices in the two-connected component. The idea of the reduction is that in G we can compute by a private protocol the messages sent by v_W in the tree. Hence, we do not need any privacy requirements for such v_W.

Lemma 4.2 *Let $S = \{v : v \text{ is a curious vertex in } G\}$. A function f can be computed privately in G iff f' can be S-privately computed in T_G.*

We prove the lemma in the next two claims.

Claim 4.3 *If f can be computed privately in G, then f' can be S-privately computed in T_G.*

Proof: Assume that there is a protocol π privately computing f in G. We construct an S-private protocol π' computing f' in T_G. The protocol π' simulates the protocol π: (1) Every curious vertex in G, which is a vertex in T_G having the same input, sends and receives the same messages in both protocols. (2) Every vertex v_W simulates all the non-separating vertices in W. (3) Every message sent between two separating vertices in the same two-connected component W in G, is sent via v_W in π'. Thus, every curious vertex has the same view in π' as it had in π. Since Protocol π is private and in π' we require privacy only for the curious vertices, Protocol π' is S-private. $\square$

Claim 4.4 *If f' can be S-privately computed in T_G, then f can be computed privately in G.*

Proof: Assume that there is an S-private protocol π' computing f' in T_G. We construct a private protocol π computing f in G. The construction is similar to the construction in the proof of Claim 3.5. Specifically, Protocol π will have a virtual round for every round of π'. In Protocol π, every curious vertex will effectively have the same information as in π'. Specifically, each curious vertex has the same input, and it will learn only the messages sent to it in π'. The non-curious vertices will not learn any information during the protocol. This is achieved by using a private protocol for computing each message sent to a vertex v_W and masking the output of the private protocol (i.e., the message) by random bits held by the all vertices in W. That is, this message is secret-shared by all vertices in W. In other words, in π all separating and non-separating vertices of the two-connected component W simulate the vertex v_W in π'.

We next describe Protocol π more formally. First, there is some initialization. In π', every vertex v_W has a random input r_W distributed uniformly in some finite set R. In the beginning of Protocol π, each vertex $w \in W$ chooses a random input $r_{W,w}$ distributed uniformly in R, and the parties define $r_W = \bigoplus_{w \in W} r_{W,w}$ (a separating vertex chooses an independent random value for every two-connected component containing it). In addition, each curious vertex w chooses a random string r_w as it chooses it in π'.

Protocol π' has rounds and in each round only one vertex sends messages. Without loss of generality, assume that every message in π' is one bit. Protocol π will have a virtual round for every round of π'. Consider Round j of π' in which a message m is sent by a vertex in T_G to a vertex in T_G. Notice that at least one of these two vertices is curious. The messages sent in the virtual round depend on which of these vertices is curious.

Both sender and receiver are curious. Denote the sender by u and the receiver by v. Both vertices are vertices in both T_G and G. This is the simple case, where Vertex u sends the message m to v in π.

The sender is curious and the receiver is non-curious. In this case the sender is some separating vertex u and the receiver is a "component vertex" v_W, where W is a two-connected component in G such that $u \in W$. The virtual round in Protocol π is as follows:

- Each vertex w in W (including the separating vertices) chooses at random, with uniform distribution, a bit r^j_w,
- The vertices in W compute the function $m \oplus \bigoplus_{w \in W} r^j_w$ using a private protocol. By Theorem 1.1 such protocol exists since each two-connected component has size at least 3.

On one hand, the vertices in W collectively know the message m. On the other hand, each vertex gains no information from this virtual round.

The sender is non-curious and the receiver is curious. In this case the sender is a "component vertex" v_W, where W is a two-connected component in G, and the receiver is some separating vertex v such that $v \in W$. The message m sent in Round j in π' is a function of the inputs of the non-separating vertices in W, the random input r_W, and the messages v_W got in previous rounds. In Protocol π, the vertices in W know the inputs of the non-separating vertices in W, and collectively know the random input r_W and the messages v_W got in previous rounds. Thus, m is a function of inputs known to vertices in W. The virtual round in Protocol π is as follows:

- The receiver v chooses a random bit r^j_v with uniform distribution, and
- The vertices in W compute the function $m \oplus r^j_v$ using a private protocol.

On one hand, Vertex v learns the message m (since it learns the output $m \oplus r^j_v$ and knows r^j_v), but no additional information. On the other hand, each vertex in $W \setminus \{v\}$ gains no information from this virtual round, since the output is masked by the random bit r^j_m.

We next argue that this protocol is private using the modular composition paradigm (explained in Section 2.2). That is, we can assume that each invocation of a private protocol is replaced by an ideal invocation. We next prove that every vertex v in G learns no information in Protocol π not implied by its input and the output of f. There are three cases.

v is a leaf. The view of a leaf v in π is exactly its view in π'. Since π' is S-private and each leaf is curious, in π the leaf v gains no information not implied by its input and the output of the function f.

v is non-curious. The view that a non-curious vertex v sees during the execution of Protocol π is its input, the random inputs it chooses, and the outputs of some calls to the ideal invocations, each one masked by at least one random input r_v^j for a separating vertex v in the two-connected component. Thus, in π the non-curious vertex v gains no information not implied by its input and the output of the function f.

v is a separating vertex. The view that a separating vertex v sees during the execution of Protocol π is its input, the random inputs it chooses, and the outputs of some calls to the ideal invocations. For every such invocation for a message sent in π' to v, Vertex v in π learns the same message it learned in π'. For every such invocation for a message sent in π' to a different vertex in T_G, the output is masked by at least one random bit of a vertex different than v. Thus, the separating vertex v learns only the messages it got in π'. Since π' is S-private and v is curious, in π Vertex v gains no information not implied by its input and the output of the function f.

□

5 Private Computation on Trees

By Lemma 4.2, to characterize which functions can be privately computed on G, we need to characterize which functions can be S-privately computed in T_G. We do not have an exact characterization of these functions. We only give necessary conditions and sufficient conditions for this task. In the sequel, we say that a vertex v is curious if $v \in S$.

5.1 Sufficient Condition

In this section we give a sufficient condition for computing a function S-privately in a tree. Using Lemma 4.2, the results of this section give a sufficient condition for computing a function privately in arbitrary networks. The protocol we construct to prove that this condition is sufficient is deterministic and uses only a broadcast channel. As the parties we consider are honest-but-curios, if a vertex wants to broadcast a message, then it sends this message to its neighbors, and this message is propagated to all vertices in the tree. Thus, our private protocol can be implemented in any tree using the regular communication channels.

The sufficient condition is a simple generalization of the condition of [23]. We next introduce some notation and definitions generalizing Definitions 3.6 and 3.7. We represent a function $f : A_1 \times \dots A_n \to O$ by an n-dimensional array M_f whose ith-dimension is labeled by the elements of A_i, and $M_f(a_1, \dots, a_n) = f(a_1, \dots, a_n)$.

Definition 5.1 (The Equivalence Relations $\equiv_i^S$) *Let M be an n-dimensional array whose ith-dimension is labeled by the elements of A_i, and S be the set of curious vertices. The relation $\sim_i^S$ on A_i is defined as follows: $a, b \in A_i$ satisfy $a \sim_i^S b$ if there exist some $\vec{a}, \vec{b} \in A_1 \times \cdots \times A_n$ such that the following conditions hold:*

1. There exists an index $j \neq i$ such that $v_j \in S$ and $a_j = b_j$,

2. $a_i = a$ and $b_i = b$,

3. $M(\vec{a}) = M(\vec{b})$.

The equivalence relation $\equiv_i^S$ on A_i is defined as the transitive closure of the relation $\sim_i^S$.[3]

Example 5.2 For example, consider the function $f : \{0,1\}^3 \to \{0,1\}$, where $f(a_1, a_2, a_3) = a_1 \oplus a_2$ and consider the array M_f.

- If $S = \{v_1, v_3\}$, then $0 \sim_1^S 1$ as we can take $\vec{a} = \langle 0,0,0 \rangle$ and $\vec{b} = \langle 1,1,0 \rangle$. Notice that $f(0,0,0) = f(1,1,0) = 0$ and $a_3 = b_3$.
- However, if $S = \{v_1, v_2\}$, then $0 \not\sim_1^S 1$ as we must take $j = 2$ and if $a_2 = b_2$ then $f(0, a_2, a_3) \neq f(1, a_2, b_3)$ for every a_3, b_3.

To gain some intuition on $\sim_i^S$, we mention that if $v_j \in S$, $a_j = b_j$, and $f(\vec{a}) = f(\vec{b})$ (that is, v_j has the same input and output in the two cases), then, informally, v_i has to broadcast the same messages on a_i and b_i to guarantee that v_j does not any information (recall that our protocol uses only a broadcast channel).

Definition 5.3 (S-Forbidden Array) *An array M is an S-forbidden array iff (1) the array is not constant, and (2) for all i, all the elements of A_i are equivalent in M according to $\equiv_i^S$.*

Lemma 5.4 *Let $f : A_n \times \ldots \times A_n \to O$ be a function. If the array M_f does not contain an S-forbidden array, then f can be S-privately computed on any tree with n vertices.*

Proof: The protocol is a simple generalization of the protocol of [23]. In each step of the protocol, the parties maintain a rectangle $R_1 \times \ldots \times R_n \subseteq A_1 \times \ldots \times A_n$, such that $\langle a_1, \ldots, a_n \rangle \in R_1 \times \ldots \times R_n$. The protocol is described in Figure 3. Since, in each stage of the protocol, $\langle a_1, \ldots, a_n \rangle \in R_1 \times \ldots \times R_n$, this protocol is correct. As M_f does not contain a forbidden array and the sets $A_1, \ldots, A_n$ are finite, the protocol must reach a constant rectangle and terminate.

We next argue that this protocol is S-private, that is, each curious vertex v_j does not learn information on $\langle a_1, \ldots, a_n \rangle$ that is not implied by a_j and $f(a_1, \ldots, a_n)$. This follows from the fact that if v_j is curious and $f(\vec{a}) = f(\vec{b})$ where $a_j = b_j$, then in each stage of the protocol $a_i \equiv_i^S b_i$ in M for every i, and the same communication transcript is exchanged on $\vec{a}$ and $\vec{b}$, thus, v_j does not gain extra information. □

In the protocol described in the proof of Lemma 5.4, each message is broadcasted to all the vertices in the tree. This was possible since the sufficient condition has strong requirements, and this explains why the sufficient condition is not necessary. For example, consider a path v_1, v_2, v_3, where the input of v_1 is a bit a_1, the input of v_2 is a bit a_2, and the v_3 has no input. The vertices want to compute the function $a_1 \oplus a_2$ (that is, the function considered in Example 5.2), where the curious vertices are $S = \{v_1, v_2, v_3\}$. This function can be privately computed on the path since v_1 can send its input to v_2 and v_2 sends the output to v_1 and v_3. However, this function does not satisfy the sufficient condition as its array is S-forbidden.

[3]The equivalence relation $\equiv_i$ as defined in Definition 3.6 can be viewed as a special case of $\equiv_i^S$ where S is the eavesdropper which has some fixed input.

A Private Protocol for f

Initialization. $R_i \leftarrow A_i$ for $i \in \{1, \ldots, n\}$.

Step. Let M be the array M_f restricted to $R_1 \times \ldots \times R_n$. As M_f does not contain an S-forbidden array, the array M is not S-forbidden.

1. If M is constant, then all the vertices know that this constant is the output, and the protocol ends.
2. Otherwise, for some $i \in \{1, \ldots, n\}$, not all the elements of A_i are equivalent in M according to $\equiv_i^S$. Vertex v_i broadcasts the equivalence class of a_i in M. Thereafter, all parties set R_i as this equivalence class.
3. Goto **Step**.

Figure 5: A private protocol computing f in T_G.

5.2 Necessary Conditions

In a tree, every vertex that is not a leaf is a separating vertex. Informally, this means that such vertex can learn all the information sent from one side of a tree to the other side. Formulating this intuition is simple: Let v be a vertex of degree d in the tree. We claim that if a function can be computed in a tree, then a related function can be computed in the eavesdropper model in a network with d vertices; the input of each vertex in the eavesdropper model is the vectors of inputs of one side of the tree, and Eve is the separating vertex. This is formulated in the next lemma, whose proof is similar to the proof of Claim 3.4.

Lemma 5.5 *Let $T = (V, E)$ be a tree and S be a set of curious parties. Let v_n be a curious vertex in T of degree $d \geq 2$ (that is, v is not a leaf). Let $V_1, \ldots, V_d$ be the connected components in $T \setminus \{v_n\}$, where $V_j = \left\{v_{i_{j-1}+1}, \ldots, v_{i_j}\right\}$ for some indices $0 = i_0 < i_1 < \cdots < i_d = n-1$. Furthermore, let $f : A_1 \times \cdots \times A_n \to O$ be a function. For every $c \in A_i$, define the d-argument function f_c as*

$$f_c(\langle a_1, \ldots, a_{i_1}\rangle, \langle a_{i_1+1}, \ldots, a_{i_2}\rangle, \ldots, \langle a_{i_{d-1}+1}, \ldots, a_{n-1}\rangle) = f(a_1, \ldots, a_{n-1}, c).$$

If f can be S-privately computed in T, then, for every $c \in A_n$, the function f_c can be privately computed in the eavesdropper model.

Using Corollary 3.11, we deduce in Lemma 5.8 a simpler (and weaker) necessary condition. Roughly speaking, the condition is that the inputs corresponding to each output value are a union of certain n-dimensional rectangles. For the lemma and its proof we need the following notation: Let $f : A_1 \times \cdots \times A_n \to O$ be a function, $I \subseteq \{1, \ldots, n\}$ be a set, and $\vec{c} = \langle c_i \rangle_{i \in I}$ be a vector where $c_i \in A_i$ for every $i \in I$. Denote $f_{I,\vec{c}} : \Pi_{i \notin I} A_i \to O$, the restriction of f to $\{1, \ldots, n\} \setminus I$, as the following function

$$f_{I,\vec{c}}(\langle a_i \rangle_{i \notin I}) = f(\langle b \rangle_{i \in \{1,\ldots,n\}}) \quad \text{where} \quad b_i = \begin{cases} c_i & \text{if } i \in I \\ a_i & \text{otherwise.} \end{cases}$$

Every curious vertex does not have an input of $f_{I,\vec{c}}$. We first claim that, without loss of generality, we can assume that every leaf in T is non-curious. This observation is important since we will use an inductive proof in which we remove vertices from the tree and a curious vertex can become a leaf.

Claim 5.6 *Let T be a tree with a leaf v that does not have an input. If a function g can be S-privately computed in T, then the function g can be $(S \setminus \{v\})$-privately computed in $T \setminus \{v\}$.*

Proof: Let w be the neighbor of v in T, and let π be a protocol S-privately computing g in T. First, consider an execution of the protocol π on T. At the end of the execution, w chooses a random string r_v with uniform distribution from all the random strings that are consistent with the messages that v received and sent. Second, consider an execution of the protocol π on $T' \stackrel{\text{def}}{=} T \setminus \{v\}$, where in the beginning of the execution w chooses a random string r_v with uniform distribution from all the possible random strings, and simulates v with this random string. Since w is the only neighbor of v, the views of all vertices in T' are equally distributed in the two scenarios. Thus, the protocol $(S \setminus \{v\})$-privately computes g in T'. $\square$

Furthermore, if there are two non-curious neighbors in T, we can replace them by a new vertex holding the inputs of the two neighbors.

Observation 5.7 *We can assume, without loss of generality, that $(u, v) \notin E$ for every two non-curious vertices u, v.*

Lemma 5.8 *Let $T = (V, E)$ be a tree and S be a set of curious parties such that there are no adjacent non-curious vertices. Denote $I \stackrel{\text{def}}{=} \{i : v_i \in S\}$. Assume that a function $f : A_1 \times \cdots \times A_n \to O$ can be S-privately computed in T. Then, for every $\vec{c} \in \Pi_{i \in I} A_i$ and every output value $o \in O$, there exist sets $\langle R_i \rangle_{i \notin I}$ such that $R_i \subseteq A_i$ and*

$$f_{I,\vec{c}}(\vec{a}) = o \quad \text{if and only if} \quad a_i \in R_i \text{ for every } i \notin I.$$

Proof: Define $n' \stackrel{\text{def}}{=} n - |I|$. Fix some output value $o \in O$. The proof is by induction on n', i.e., on the number of variables of $f_{I,\vec{c}}$. If $n' = 0$, that is, there are 0 non-curious vertices in the network, then $f_{I,\vec{c}}$ is constant, and the claim is trivial.

For the induction step assume that there are n' non-curious vertices in the tree T. By Claim 5.6 and by Observation 5.7, we can assume that every leaf in T is non curious and the neighbor of each leaf is curious. Furthermore, by renumbering the vertices, we can assume that v_n is a non-curious leaf of T and v_{n-1} is its curious neighbor.

By Lemma 5.5 and Corollary 3.11, there exist sets $R_n \subseteq A_n$ and $R \subseteq A_1 \times \cdots A_{n-2}$ such that the inputs of $f_{I,\vec{c}}$ corresponding to o are $R \times R_n$. Fix any $c_n \in R_n$ and consider the function $f_{I \cup \{n\},\vec{c},c_n}$. By the construction, the inputs of $f_{I \cup \{n\},\vec{c},c_n}$ corresponding to o are R. On the other hand, the function $f_{I \cup \{n\},\vec{c},c_n}$ can be computed on a tree obtained from T by removing v_n. Thus, by the induction hypothesis applied to $f_{I \cup \{n\},\vec{c},c_n}$, there are sets $\langle R_i \rangle_{\{i : i \notin I, i \neq n\}}$ such that $f_{I \cup \{n\},\vec{c},c_n}(\vec{a}) = o$ if and only if $a_i \in R_i$ for every $i \notin I, i \neq n$. Thus, the inputs of $f_{I,\vec{c}}$ corresponding to o are $\left(\Pi_{\{i : i \notin I, i \neq n\}} R_i\right) \times R_n$, as required. $\square$

5.3 An Example: A Network with One Two-Connected Component and One Leaf

We next consider a simple example of a network G with one two-connected component and one leaf, and the tree T_G which is a path of length three. This might seem to be a simpler network than the networks considered in Section 3, however, it turns out that the characterization is more complicated in this case. This example demonstrates the difficulties in characterizing the functions that can be privately computed in arbitrary networks.

The tree T_G constructed in the proof of Lemma 4.2 for G is a path of length 3, that is, the vertices in T_G are $\{v_1, v_2, v_3\}$ and the edges in T_G are (v_1, v_2) and (v_2, v_3). The adversarial structure is $S = \{v_2, v_3\}$, that

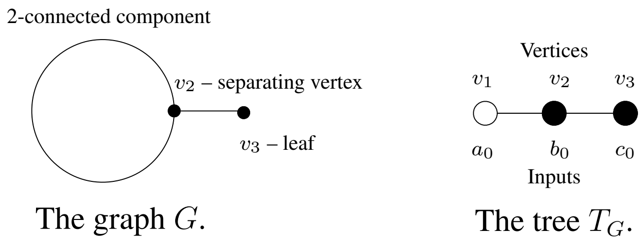


Figure 6: The graph G and the tree T_G.

is, the curious vertices are v_2 (the separating vertex in G) and v_3 (the leaf in G). See Figure 6. We want to S-privately compute a function $f : A \times B \times C \to O$. In [4], it is proved that every Boolean function that depends on its three variables cannot be S-privately computed in T_G. Our characterization shows that the structure of the non-Boolean functions that can be S-privately computed in T_G is much richer.

To gain some intuition, consider a deterministic protocol S-privately computing f in T_G. Let a_0, b_0, and c_0 be the inputs of v_1, v_2, and v_3 respectively. As v_1 is not curious, we can assume that v_2 sends its input b_0 to v_1 and thereafter v_2 acts as a relay. In each stage of the protocol, the vertices will maintain two sets, $R_{1,2} \subseteq A_1 \times A_2$ and $R_3 \subseteq A_3$ such that $\langle a_0, b_0 \rangle \in R_{1,2}$ and $c_0 \in R_3$.

Let up first consider the messages v_3 sends to v_1 via v_2. Vertex v_3 knows c_0 and that $\langle a_0, b_0 \rangle \in R_{1,2}$, and it wants to send a message that does not leak information to v_2 not implied by the input of v_2 and the output of f. This motivates the following notation, similar to the notation of Section 3.2. We say that $c \sim_3 c'$, for $c, c' \in R_3$, if there are a, a', and b such that $\langle a, b \rangle, \langle a', b \rangle \in R_{1,2}$ and $f(a, b, c) = f(a', b, c')$. The equivalence relations $\equiv_3$ is defined as the transitive closure of $\sim_3$. Vertex v_3 can send a message to v_1 maintaining privacy iff not all elements of R_3 are equivalent according to $\equiv_3$ (with the current $R_{1,2}$).

The messages that v_1 can send to v_3 via v_2 without violating the privacy are more complicated, as explained below. First, Vertex v_3 should not gain information from the message. Furthermore, the message v_1 sends to v_3 via v_2 should not leak information to v_2. We say that $\langle a, b \rangle \sim_{1,2} \langle a', b' \rangle$ for $\langle a, b \rangle, \langle a', b' \rangle \in R_{1,2}$ if at least one of the following conditions hold:

- There exists $c \in R_3$ such that $f(a, b, c) = f(a', b', c)$.
- $b = b'$ and there exists $c, c' \in R_3$ such that $f(a, b, c) = f(a', b, c')$.

If $\langle a, b \rangle \sim_{1,2} \langle a', b' \rangle$, then v_1 must send the same message to v_3 on $\langle a, b \rangle$ and on $\langle a', b' \rangle$. The equivalence relations $\equiv_{1,2}$ on $R_{1,2}$ is defined as the transitive closure of $\sim_{1,2}$. By transitivity, if $\langle a, b \rangle \equiv_{1,2} \langle a', b' \rangle$, then v_1 must send the same message to v_3 on $\langle a, b \rangle$ and on $\langle a', b' \rangle$. Given a function f, consider a two-dimensional array M_f whose first dimension is labeled by elements of $A \times B$, the second dimension is labeled by elements of C, and $M_f(\langle a, b \rangle, c) = f(a, b, c)$. As in previous sections, a sub-array of M_f is forbidden if it is not constant, all its rows are equivalent with respect to $\equiv_{1,2}$, and all its columns are equivalent with respect to $\equiv_3$. We next give a characterization of the function that can be S-privately computed in T_G; the proof of the characterization is similar to the proof of Lemma 3.8.

Claim 5.9 *A function f can be computed S-privately in T_G iff the array M_f does not contain a forbidden array with respect to $\equiv_{1,2}$ and $\equiv_3$. Furthermore, if a function f can be computed S-privately in T_G, then f can be computed S-privately in T_G by a deterministic protocol.*

6 Conclusions and Open Problems

In this paper we address the question of characterizing the functions that can be privately computed in incomplete networks. This question was addressed previously in [4, 21, 5]; however, it was addressed only for Boolean functions and very simple networks (networks with one separating vertex and 2 two-connected components). Our first result is an exact characterization of the functions that can be privately computed in networks with one separating vertex, arbitrary number of two-connected components, and without leaves. Our characterization implies that the class of non-Boolean functions that can be privately computed is much richer than the class of Boolean functions. For example, if a Boolean function can be privately computed in a network with one two-connected component and one leaf, then it cannot depend on inputs of non-separating vertices in the two-connected component and on the input of the leaf [4]; this is not true for non-Boolean functions as shown in Section 5.3.

For general networks, we take a major step towards the characterization of the functions that can be privately computed in them. We first reduce the question to private computation on trees, and then give necessary conditions and sufficient conditions for private computation on trees. The exact characterization is still open. As an example for the difficulty of the characterization, we characterize the functions that can be privately computed in a simple network with one two-connected component and one leaf, and show that the characterization is already more complicated.

Some discussion on the privacy requirements is due. All the protocols we construct in this paper have *perfect* privacy (that is, the distribution of VIEW has to be exactly the same). In contrast, in the necessary conditions (e.g., Claim 3.10 and Lemma 5.8) we only require *weak* privacy (that is, the distribution of VIEW has to have the same support, possibly with different probabilities). Another notion of privacy is *statistical* privacy (that is, the distribution of VIEW has to statistically close). We do not know how to characterize the function that can be computed with statistical privacy. We note that the characterization of [23] of the functions that can be computed privately in the two party model is proven only for protocols with perfect privacy (or protocols with very small statistical distance).

In this work we focus on 1-privacy. The obvious generalization is to characterize the functions that can be computed t-privately in incomplete networks. Our results in Section 3 generalize to networks with one separating set of size $t-1$ and an arbitrary number of t-connected components where the size of each t-connected component is greater than $2t$. However, our results for arbitrary networks do not generalize to t-privacy as the component structure of such networks can be complicated. As we mentioned in the introduction, even the characterization of the functions that can be computed t-privately in a complete network with at most $2t$ vertices is still open.

Another related question is what functions can be computed securely in incomplete networks where the "bad" parties are malicious. If a network is $(2t+1)$-connected and contains more than $3t$ vertices, then every function can be computed securely in the presence of t malicious vertices. The open question is to characterize what function can be computed in the presence of t malicious vertices in a network that is not $(2t+1)$-connected.

Acknowledgment. I would like to thank Enav Weinreb and the anonymous referees for valuable comments that greatly improved this write-up. I would also like to thank Matt Franklin for his hospitality and support while I was on sabbatical at the University of California, Davis.

References

[1] A. Beimel and M. Franklin. Reliable communication over partially authenticated networks. *Theoretical Computer Science*, 220:185–210, 1999.

[2] A. Beimel and L. Malka. Efficient reliable communication over partially authenticated networks. *Distributed Computing*, 18(1):1 – 19, 2005.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 1–10, 1988.

[4] M. Bläser, A. Jakoby, M. Liśkiewicz, and B. Manthey. Private computation – k-connected versus 1-connected networks. In *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 194–209. Springer-Verlag, 2002. Journal version: in *J. of Cryptology*, 19(3):341–357, 2006.

[5] M. Bläser, A. Jakoby, M. Liśkiewicz, and B. Manthey. Privacy in non-private environments. In *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 137 – 151. Springer-Verlag, 2004.

[6] B. Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer-Verlag, 1998.

[7] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. of Cryptology*, 13(1):143–202, 2000.

[8] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of the 20th ACM Symp. on the Theory of Computing*, pages 11–19, 1988.

[9] B. Chor and E. Kushilevitz. A zero-one law for Boolean privacy. *SIAM J. on Discrete Mathematics*, 4(1):36–47, 1991.

[10] Y. Desmedt and Y. Wang. Secure communication in multicast channels: The answer to Franklin and Wright's question. *J. of Cryptology*, 14(2):121–135, 2001.

[11] Y. G. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 502–517. Springer-Verlag, 2002.

[12] D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, 3:14–30, 1982.

[13] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. of the ACM*, 40(1):17–47, 1993.

[14] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. *SIAM J. on Computing*, 17(5):975–988, 1988.

[15] S. Even. *Graph Algorithms*. Computer Science press, 1979.

[16] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[17] M. Franklin and R. N. Wright. Secure communication in minimal connectivity models. *J. of Cryptology*, 13(1):9–30, 2000.

[18] M. Franklin and M. Yung. Secure hypergraphs: Privacy from partial broadcast. In *Proc. of the 27th ACM Symp. on the Theory of Computing*, pages 36–44, 1995.

[19] O. Goldreich, S. Goldwasser, and N. Linial. Fault-tolerant computation in the full information model. In *Proc. of the 32nd IEEE Symp. on Foundations of Computer Science*, pages 447–457, 1991.

[20] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. of Cryptology*, 13(1):31–60, 2000.

[21] A. Jakoby, M. Liśkiewicz, and R. Reischuk. Private computations in networks: Topology versus randomness. In *Proc. of the 20th International Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *LNCS*, pages 121–132. Springer-Verlag, 2003.

[22] M. V. N. A. Kumar, P. R. Goundan, K. Srinathan, and C. Pandu Rangan. On perfectly secure communication over arbitrary networks. In *Proc. of the 21st ACM Symp. on Principles of Distributed Computing*, pages 193–202, 2002.

[23] E. Kushilevitz. Privacy and communication complexity. *SIAM J. on Discrete Mathematics*, 5(2):273–284, 1992.

[24] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, 1997.

[25] K. Menger. Allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[26] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer-Verlag, 1992. An updated version presented at the workshop on multi-party computation, Weizmann Inst., 1998.

[27] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. of the 21st ACM Symp. on the Theory of Computing*, pages 73–85, 1989.

[28] H. M. Sayeed and H. Abu-Amara. Efficient perfectly secure message transmission in synchronous networks. *Information and Computation*, 126:53–61, 1996.

[29] K. Srinathan, V. Vinod, and C. Pandu Rangan. Efficient perfectly secure communication over synchronous networks. In *Proc. of the 22nd ACM Symp. on Principles of Distributed Computing*, pages 252–252, 2003.

[30] K. Srinathan, V. Vinod, and C. Pandu Rangan. Optimal perfectly secure message transmission. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 545 – 561, 2004.

[31] E. Upfal. Tolerating a linear number of faults in networks of bounded degree. *Information and Computation*, 115(2):312–320, 1994.