On Private Computation in Incomplete Networks*

Amos Beimel[1]

Dept. of Computer Science, Ben Gurion University, Beer Sheva 84105, Israel.

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 processor 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 protocol. The question we address in this paper is what functions can be computed privately in a given incomplete network. It is known that if a network is 2-connected then every pair of parties can communicate privately. Thus, the question is interesting only for non-2-connected networks. We first characterize the functions that can be computed privately in simple networks – networks with one separating vertex and two 2-connected components. We then deal with private computations in arbitrary networks: we reduce this question to private computations of related functions on trees, and give sufficient and necessary conditions on the functions that can be computed privately on trees.

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 that some parties are connected by an incomplete network of reliable and private channels and each party has an input. The parties cooperate to honestly execute some protocol computing a given function, but some of them are curious. That is, after the protocol terminates they collude and try to learn information from the communication they heard. A protocol is t-private if any coalition of at most t passively corrupted parties does not learn any information that is not implied by their inputs and the output of the function.

Many papers dealing with private computation, e.g., [7, 3, 8], assume that the communication network is complete, that is, there is a private and reliable communication channel between any pair of parties. The question we address in this paper, following [4], is what functions can be computed privately in a given incomplete network. If the network is sufficiency connected then the situation is simple as proved by [3, 7, 11, 12].

* Partially supported by the Lynn and Wiliam Frankel Center for Computer Sciences.

Theorem 1. *If $n > 2t$ and the network is $(t+1)$-connected, every function can be computed t-privately in G.*

Bläser et al. [4] characterize the Boolean functions that can be 1-privately computed in connected networks with one separating vertex and two 2-connected components. We consider the more general question that naturally arises. Our goal is, given a communication network, characterize which functions can be computed 1-privately in this network.

Our Results. We first consider simple networks with one separating vertex and two 2-connected components. We give an exact characterization of the functions that can be computed 1-privately in such a network. This result generalizes a result of Bläser et al. [4] who characterize the *Boolean* functions that can be computed 1-privately in such a network. While Boolean functions that can be computed privately in such networks are very simple structure ("if then else" functions), the non-Boolean functions that can be computed privately 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 question of private computation in such a network to a question of private computation of a related function with two variables in a simpler model, and then characterize the functions that can be computed in the simpler model.

We next consider 1-private computations in arbitrary networks. We reduce the private computation of a function in an arbitrary network to private computation of a related function in a tree. The idea of this reduction is that we can replace each 2-connected component in the network by a single vertex holding the inputs of the component. We then give sufficient and necessary conditions on the functions that can be computed privately on trees. We do not know the exact characterization of the functions that can be computed privately on trees.

t-privacy. In this work we focus on 1-privacy. Our results in Section 3 generalize to networks with one separating set of size $t-1$ and two t-connected components. However, our results for arbitrary networks do not generalize to t-privacy as the component structure of such networks can be complicated.

Historical Notes. There are a few models of secure computation. One distinction is whether the "bad" parties have unlimited power or they are polynomial-time randomized machines. 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. In this work we consider honest-but-curious parties with unlimited power. We review some previous results concerning this model. Chaum, Crépeau, and Damgård [7] 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 [20] characterizes the functions that can be computed privately in a network with two parties. Chor and Kushilevitz [8] characterize the Boolean functions that can be computed t-privately in complete networks when $n \leq 2t$. 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 [12] 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 computed 1-privately in a network with one separating vertex and two 2-connected components. They also considered the randomness required for private protocols in incomplete networks. Jakoby, Liskiewicz, and Reischuk [18] considered tradeoff 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 [11, 14], approximate Byzantine agreement [13, 27], reliable message transmission [11, 12], and reliable and private message transmission [23, 12, 24–26]. Simple impossibility results and references can be found in [14, 21]. 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 [17, 16, 15, 9]. Secure computation in directed networks has been studied in [10]. Secure communication against general adversarial structures has been studied in [19].

Organization. In Section 2, we describe our model and present some background on connectivity. In Section 3, we characterize the functions that can be computed 1-privately in networks with two 2-connected components. In Section 4, we reduce private computation of functions in arbitrary networks to private computation of related function on trees, and, in Section 5, we give sufficient and necessary conditions on the functions that can be computed privately on trees.

2 Preliminaries

The Model. The communication network is modeled by an undirected graph $G = \langle V, E\rangle$, where (1) 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 sequence we refer to parties as vertices. (2) The edges E describe the communication channels. That is, there is an edge $\langle u, v\rangle$ 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 $\langle u, v\rangle$.

Protocols. We consider an n-party protocol for computing a given function. 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 $(r_1, \ldots, r_n)$ 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[1] and sends a message $m_{j,k}$ (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_k then $m_{j,k}$ is the empty string. The message $m_{j,k}$ 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., $\langle m_{j',i}\rangle_{1\leq j'<j}$. A computation of the protocol ends in a round in which each vertex computes an *output*.

Transcripts, Views, and Outputs. Let $S \subseteq \{v_1,\ldots,v_n\}$. Given an execution of a protocol Π on inputs $(a_1,\ldots,a_n)$ and random inputs $(r_1,\ldots,r_n)$, we define: The *transcript* of S of the execution is the sequence of messages that vertices in S got during the execution; it is denoted by $\mathrm{TRANS}_S(a_1,\ldots,a_n,r_1,\ldots,r_n)$. The *view* of S is the triplet $\langle a_i\rangle_{v_i\in S}$, $\langle r_i\rangle_{v_i\in S}$, and $\mathrm{TRANS}_S(a_1,\ldots,a_n,r_1,\ldots,r_n)$; it is denoted by $\mathrm{VIEW}_S(a_1,\ldots,a_n,r_1,\ldots,r_n)$. We consider the random variables $\mathrm{TRANS}_S(a_1,\ldots,a_n,\langle r_i\rangle_{v_i\in S})$ obtained by randomly selecting $\langle r_i\rangle_{v_i\notin S}$ and outputting $\mathrm{TRANS}_S(a_1,\ldots,a_n,r_1,\ldots,r_n)$. We also consider the similarly defined random variables for $\mathrm{VIEW}_S(a_1,\ldots,a_n,\langle r_i\rangle_{v_i\in S})$.

In the model we consider, the n-party honest-but-curious model, each party is curious, that is, coalitions of parties may try to deduce as much information possible from their 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 $f : A_1\times\ldots\times A_n \rightarrow O$, 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 does 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. Our results remain the same if we only require statistical security. In the following definition we define privacy against an adversarial structure $\mathcal{S} \subseteq 2^V$, that is $\mathcal{S}$ is a collection of subsets of the vertices. We require that if parties in a set in $\mathcal{S}$ collude then, from their view, they do not gain information that is not implied by their inputs and the output of the function. In this work we mainly focus on 1-privacy, that is we want to protect the privacy against each individual. We define the more general case of $\mathcal{S}$-privacy as it is used as a tool to characterize 1-privacy.

Definition 1 (Private Computation). *Let $G = \langle V,E\rangle$ be network with n vertices, $A_1,\ldots,A_n$, and O be finite sets, $f : A_1\times\ldots\times A_n \rightarrow O$ be a function, and $\mathcal{S} \subseteq 2^V$ be an adversarial structure. A protocol Π $\mathcal{S}$-*privately computes f*, if the following conditions hold:*

Correctness. *For every $a_1,\ldots,a_n$ and every $r_1,\ldots,r_n$, the output of each v_i with $\mathrm{VIEW}_{\{v_i\}}(a_1,\ldots,a_n,r_1,\ldots,r_n)$ is $f(a_1,\ldots,a_n)$.*

[1] By adding extra rounds, this assumption is without loss of generality.

Privacy. *For every $S \in \mathcal{S}$, 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 $a_i = a'_i$ for every $v_i \in S$, and every $\langle r_i \rangle_{v_i \in S}$, if $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$ then the random variables $\mathrm{VIEW}_S((a_1, \ldots, a_n, \langle r_i \rangle_{v_i \in S}))$ and $\mathrm{VIEW}_S((a'_1, \ldots, a'_n, \langle r_i \rangle_{v_i \in S}))$ are equally distributed.*

A function f can be computed t-privately in G if there is a protocol Π that $\mathcal{S}$-privately computes f in G, where $\mathcal{S} = \{S \subseteq V : |S| \leq t\}$.

Since each vertex learns the output of f (and knows its input), we require that the privacy is protected only when $f(a_1, \ldots, a_n) = f(a'_1, \ldots, a'_n)$. 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.

In the sequence, we will use the following proposition of [8], which holds for every 2-party protocol (i.e., even without the correctness and privacy requirement). Informally, this proposition says that if changing the inputs of both parties yields the same transcript, then changing the input of only one party also yields the same transcript.

Proposition 1 ([8]). *Consider a two-party protocol. Let a_1, a_2, a'_1, a'_2 be inputs, r_1, r'_1, r_2, r'_2 be random inputs, and h be a transcript such that*

$$\mathrm{TRANS}_{\{1,2\}}(a_1, a_2, r_1, r_2) = \mathrm{TRANS}_{\{1,2\}}(a'_1, a'_2, r'_1, r'_2) \ = \ h.$$

Then, $\mathrm{TRANS}_{\{1,2\}}(a_1, a'_2, r_1, r'_2) = \mathrm{TRANS}_{\{1,2\}}(a'_1, a_2, r'_1, r_2) = h.$

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].

We consider *vertex* connectivity of *undirected* graphs. A graph $G = \langle V, E \rangle$ 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 2-connected if it contains at least 3 vertices and it does not contain a separating vertex. By a result of Menger [22], a graph G with at least 3 vertices is 2-connected if and only if for every vertices $u, v \in V$ either $\langle u, v \rangle \in E$ 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 *component* if it is a maximal 2-connected induced subgraph of G. We next define the component graph of a connected graph, which replaces every component in G by a single vertex.[2]

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

Definition 2 (Component Graph). *Given a connected graph $G = \langle V, E \rangle$, we define its component graph $T_G = \langle V', E' \rangle$ as follows: The vertices in V' are the components of G, the leaves in G, and the separating vertices in G. There is an edge in E' between every separating vertex and every component containing it, between a leaf and its neighboring separating vertex, and between two separating vertices connected by a bridge. For every component W in G, we denote the corresponding vertex in T_G by v_W.*

For example, a graph G and its component graph are described Fig. 1. In G there are two 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 of G has 5 vertices.

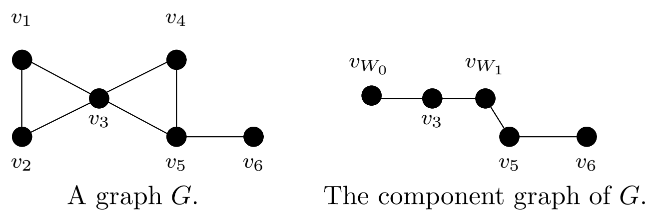

Fig. 1. A graph and its component graph.

By Menger's theorem, every cycle in G is contained in exactly one component. This fact implies the following observation.

Observation 2 *If the graph G is connected, then the graph T_G is a tree.*

3 Incomplete Networks with Two 2-Connected Components

In this section we characterize the functions that can be computed 1-privately in connected networks that contain one separating vertex and two 2-connected components. As an intermediate step, we consider a model we call the two-party and eavesdropper model. Using this intermediate model, we characterize the functions that can be computed privately in connected networks that contain two 2-connected components. That is, we prove that a function can be computed privately in connected networks that contain two 2-connected components if and only if a related function can be computed in the two-party and eavesdropper model. Roughly speaking, the two parties correspond to the 2-connected components and the eavesdropper is the separating vertex. To complete the characterization, we characterize the functions that can be computed privately in the the two-party and eavesdropper model.

The Two-Party and Eavesdropper Model. Consider the following two-party and eavesdropper model of private computation. Alice has a secret input a taken from some finite domain A and Bob has a secret input b taken from some finite domain B; they wish to compute a function $f : A \times B \to O$ such that the eavesdropper Eve, which hears the communication that they exchange, can compute $f(a, b)$; however Eve should not learn any information on a and b that is not implied by $f(a, b)$. Formally, we consider the network with 3 vertices $\{v_1, v_2, v_3\}$ (where v_1 and v_3 are Alice and Bob respectively and v_2 is Eve) and two edges $\langle v_1, v_2\rangle$ and $\langle v_2, v_3\rangle$, consider functions that do not depend on v_2's input, and consider the adversarial structure $\mathcal{S} = \{\{v_2\}\}$.

3.1 Reduction to the Two-Party and Eavesdropper Model

We first use the two-party and eavesdropper model to characterize the functions that can be 1-privately computed in connected networks with two 2-connected components. We consider a network G_{n_0,n_1}, with $n_0 + n_1 + 1$ vertices, which is composed of two 2-connected components. The first component is denoted by W_0 and has $n_0 + 1$ vertices; the second component is denoted by W_1 and has $n_1 + 1$ vertices. The two components share exactly one vertex denoted z. In this paper we assume that $n_0, n_1 \geq 2$, that is, each connected component contains at least 3 vertices. (The cases where $n_0 = 1$ or $n_1 = 1$ is characterized in the full version of this paper). Such a graph is illustrated in Fig. 2. Given an (n_0+n_1+1)-argument function $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$, define, for every $c \in C$, a 2-argument function $f_c : (A_1 \times \ldots \times A_{n_0}) \times (B_1 \times \ldots \times B_{n_1}) \to O$, where for every $\boldsymbol{a} \in A_1 \times \ldots \times A_{n_0}$ and every $\boldsymbol{b} \in B_1 \times \ldots \times B_{n_1}$,

$$f_c(\boldsymbol{a}, \boldsymbol{b}) \stackrel{\text{def}}{=} f(\boldsymbol{a}, \boldsymbol{b}, c). \tag{1}$$

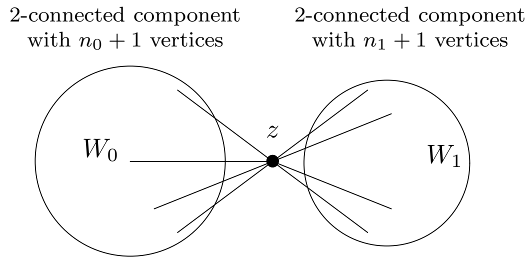

Fig. 2. The Graph G_{n_0,n_1}.

Lemma 1. *Let $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$ be a function, where $n_0, n_1 \geq 2$. The function f can be computed 1-privately in G_{n_0,n_1} if and only if for every $c \in C$ the function f_c can be computed privately in the two-party and eavesdropper model, where Alice's input is $\boldsymbol{a}$ and Bob's input is $\boldsymbol{b}$.*

Proof. First, assume that there is a protocol privately computing f in G_{n_0,n_1}. For every $c \in C$, we construct a private protocol for f_c in the two-party and eavesdropper model. Alice, holding $\boldsymbol{a} \in A_1 \times \ldots \times A_{n_0}$ simulates the $n_0 + 1$ vertices in the component W_0 (including z), and Bob, holding $\boldsymbol{b} \in B_1 \times \ldots \times B_{n_1}$ simulates the n_1 vertices in component W_1 excluding z. At the end of the protocol, Alice sends the output to Bob, 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_1. Thus, the information the eavesdropper learns is at most the information that z learns in the protocol for f, and the 1-privacy of that protocol implies the privacy of the protocol for f_c.

Now assume that, for every $c \in C$, the function f_c can be privately computed in the two-party and eavesdropper model, where Alice's input is $\boldsymbol{a}$ and Bob's input is $\boldsymbol{b}$. By Corollary 1 (appearing in Section 3.2), we can assume that this protocol is deterministic. We construct a (randomized) protocol for f. W.l.o.g., assume that the protocol Π_c for every f_c proceeds in rounds, where in odd rounds Alice sends a one bit message to Bob, and in even rounds Bob sends a one bit message to Alice. Let Π_c^i be the ith message sent in the protocol. Thus, in odd rounds (respectively, even rounds) the bit Π_c^i depends on c, $\boldsymbol{a}$ (respectively, $\boldsymbol{b}$), and the previous messages. 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_i, and the parties in $W_{i \bmod 2}$ (including z) compute the function $\Pi_c^i \oplus r_i$ using a 1-private protocol. Such 1-private protocol exists by Theorem 1, since $1 + n_{i \bmod 2} > 2$ and each component is 2-connected.

We next argue that this protocol is 1-private. As the vertices use 1-private protocols to compute each value $\Pi_c^i \oplus r_i$ and r_i is chosen at random by z, each vertex, except for z, does not learn any information during the protocol. Vertex z knows the random bits, 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. □

3.2 The Two-Party and Eavesdropper Model

The functions that can be computed privately in the two-party and eavesdropper model are a subset of the functions that can be computed in the two-party model (without the eavesdropper) as characterized by Kushilevitz [20]. We first introduce some notation from [20]. We represent a function $f : A \times B \to O$ by a matrix M_f whose rows are labeled by the elements of A, columns are labeled by the elements of B, and $M_f(a, b) = f(a, b)$.

Definition 3 ([20]). *Let M be a matrix whose rows are labeled by the elements of A and columns are labeled by the elements of B. The relation $\sim_C$ on B is defined as follows: $b, b' \in B$ satisfy $b \sim_C b'$ if there exists some $a \in A$ such that $M(a, b) = M(a, b')$. The equivalence relation $\equiv_C$ on B is defined as the transitive closure of the relation $\sim_C$. That is, $b \equiv_C b'$, for $b, b' \in B$, if there are $b_1, \ldots, b_\ell$ such that $b \sim_C b_1 \sim_C b_2 \sim_C \ldots \sim_C b_\ell \sim_C b'$. Similarly, the relations $\sim_R$ and*

$\equiv_R$ are defined on A. That is, $a, a' \in A$ satisfy $a \sim_R a'$ if there exists some $b \in B$ such that $M(a,b) = M(a',b)$, and the relation $\equiv_R$ on A is defined as the transitive closure of the relation $\sim_R$. If $b \equiv_C b'$, then we say that columns b and b' of M are equivalent, and, similarly, if $a \equiv_R a'$, then we say that rows a and a' of M are equivalent.

Definition 4 (Forbidden Matrix [20]). *A matrix M is a forbidden matrix if the following three conditions hold: (1) the matrix is not constant, (2) all the rows of M are equivalent according to $\equiv_R$, and (3) all the columns of M are equivalent according to $\equiv_C$.*

Kushilevitz [20] proved that a function f can be privately computed in the two-party model (without the eavesdropper) if and only if the matrix M_f does not contain a forbidden matrix. We adapt this result to the two-party and eavesdropper model, where there is an additional requirement.

Lemma 2. *A function $f : A \times B \to O$ can be computed privately in the two-party and eavesdropper model if and only if*

1. *The inputs corresponding to any output value form a rectangle. That is, for every $a_0, a_1 \in A$, every $b_0, b_1 \in B$, and every $o \in O$ if $f(a_0, b_0) = f(a_1, b_1) = o$ then $f(a_0, b_1) = f(a_1, b_0) = o$.*
2. *The matrix M_f does not contain a forbidden matrix.*

Proof. First, assume that f satisfies Conditions (1) and (2). We construct a deterministic private protocol computing f. The protocol is identical to the protocol of [20], with a small change in the proof of privacy. In each step of the protocol, Alice, holding an input a, and Bob, holding an input b, maintain a rectangle $A_0 \times B_0 \subseteq A \times B$, known also to Eve, such that $\langle a, b\rangle \in A_0 \times B_0$. In the beginning, $A_0 \leftarrow A$ and $B_0 \leftarrow B$. At the end of the protocol, $A_0 \times B_0$ is constant, so Eve can deduce the value of f. In each step, consider the matrix M which is the matrix M_f restricted to $A_0 \times B_0$. The matrix M becomes smaller in each step, and the equivalence relations $\equiv_R$ and $\equiv_C$ change accordingly. By Condition (2), the matrix M is not forbidden. If M is constant, Eve deduces that this constant is the output and the protocol ends. If not all the rows of M are equivalent according to $\equiv_R$, Alice sends to Bob the equivalence class of a in M, and both parties set A_0 as this equivalence class. Otherwise, not all the columns of M are equivalent according to $\equiv_C$, Bob sends to Alice the equivalence class of b in M, and both parties set B_0 as this equivalence class. As M_f does not contain a forbidden matrix, the protocol must reach a constant rectangle. Since, in each stage of the protocol, $\langle a, b\rangle \in A_0 \times B_0$, this protocol is correct. We next argue that Eve does not learn information on $\langle a, b\rangle$ that is not implied by $f(a, b)$. This follows from the fact that if $f(a, b) = f(a', b')$ then, by Condition (1), $f(a, b) = f(a, b') = f(a', b) = f(a', b')$. Thus, in each stage of the protocol, $a \equiv_R a'$ and $b \equiv_R b'$ in M, and the same communication string is exchanged on $\langle a, b\rangle$ and $\langle a', b'\rangle$, thus Eve does not gain extra information.

We next assume that Conditions (1) and (2) are necessary. It can be shown that if f can be computed privately in the two-party and eavesdropper model, then it can be computed privately in the regular two-party model. Thus, by [20], the matrix M_f does not contain a forbidden sub-matrix.

Suppose that the set of inputs corresponding to some output value is not a rectangle, that is, there are $a_0, a_1 \in A$, and $b_0, b_1 \in B$ such that $f(a_0, b_0) = f(a_1, b_1) = o$ while $f(a_0, b_1) \neq o$. Since Eve does not learn any information on the inputs, the probability distribution on the transcripts that are possible on $\langle a_0, b_0 \rangle$ is equivalent to the probability distribution on the transcripts that are possible on $\langle a_1, b_1 \rangle$. By Proposition 1, this distribution should be the same on the inputs $\langle a_0, b_1 \rangle$, contradicting the requirement that Eve can compute f from the communication. $\square$

Notice that in the proof of Lemma 2 we construct a deterministic protocol.

Corollary 1. *A function $f : A \times B \to O$ can be computed privately in the two-party and eavesdropper model if and only if it can be computed privately in the two-party and eavesdropper model by a deterministic protocol.*

We next describe two examples.

$\mathbf{f_1}$	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	3

$\mathbf{f_2}$	b_0	b_1	b_2
a_0	0	0	3
a_1	2	1	1
a_2	2	4	4

In both examples, Condition (1) holds. For example, in both examples the rectangle corresponding to the output value 2 is $\{a_1, a_2\} \times \{b_0\}$. In the first example, however, the matrix is forbidden and the function f_1 cannot be computed privately. In the second example, we changed the bottom-left entry from 3 to 4; now the matrix does not contain a forbidden sub-matrix, and the function f_2 can be computed privately. The partition induced by the protocol is detailed in the matrix.

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

Lemma 3. *A Boolean function $f : A \times B \to \{0, 1\}$ can be computed privately in the two-party and eavesdropper model iff it depends only on one of its inputs, that is, if there exists a function f' such that at least one of the following conditions hold: (1) $f(a, b) = f'(a)$ for all $a \in A$ and $b \in B$, or (2) $f(a, b) = f'(b)$ for all $a \in A$ and $b \in B$.*

Proof. If a function f (Boolean or non-Boolean) depends only on one of its inputs then it satisfies Conditions (1) and (2) of Lemma 2, thus can be computed privately in the two-party and eavesdropper model.

For the other direction, assume that a Boolean function f can be computed privately in the two-party and eavesdropper model, thus satisfies Condition (1). Assume towards contradiction that f depends on its two input, thus: (1) As f depends on its first input, there exist $b \in B$ and $a', a'' \in B$ such that $f(a', b) = 0$ and $f(a'', b) = 1$, and (2) As f depends on its second input, there exist $a \in A$

and $b', b'' \in B$ such that $f(a, b') = 0$ and $f(a, b'') = 1$. By Condition (1), on one hand, $0 = f(a, b') = f(a', b) = f(a, b)$, and on the other hand $1 = f(a, b'') = f(a'', b) = f(a, b)$, a contradiction. $\square$

Combing Lemma 1 and Lemma 2, we get a combinatorial characterization of the functions that can be computed 1-privately in networks with two components.

Theorem 3. *Let $f : A_1 \times \ldots \times A_{n_0} \times B_1 \times \ldots \times B_{n_1} \times C \to O$ be a function, where $n_0, n_1 \geq 2$. The function f can be computed 1-privately in G_{n_0,n_1} iff for every $c \in C$:*

1. *The inputs of f_c (as defined in (1)) corresponding to any output value form a rectangle. That is, for every $\boldsymbol{a_0}, \boldsymbol{a_1}$ and $\boldsymbol{b_0}, \boldsymbol{b_1}$, if $f_c(\boldsymbol{a_0}, \boldsymbol{b_0}) = f_c(\boldsymbol{a_1}, \boldsymbol{b_1}) = o$ then $f_c(\boldsymbol{a_0}, \boldsymbol{b_1}) = f_c(\boldsymbol{a_1}, \boldsymbol{b_0}) = o$.*
2. *The matrix M_{f_c} does not contain a forbidden matrix.*

4 Networks with Many Connected Components

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 computed privately 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 necessary and sufficient conditions for computing a function privately on trees. However, we do not give an exact characterization of these functions.

4.1 Reduction to Private Computation in Trees

In this section we 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. 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). We say that a vertex in G is curious if it is either a separating vertex in G or a leaf in G. Notice that curious vertices in G are also vertices in T_G. Given a function $f : A_1 \times \ldots \times A_n \to O$ we define an n'-argument function f', where the input of a curious vertex is the same as before and the input of a vertex v_W, for a component W in G, is the inputs of the non-separating vertices in W.

In the component graph we replaced every component W in G by one vertex v_W in T_G holding the inputs of the non-separating vertices in the 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. *Let $\mathcal{S} = \{\{v\} : v$ is a curious vertex in $G\}$. A function f can be computed 1-privately in G iff f' can be computed $\mathcal{S}$-privately in T_G.*

Proof. First, assume that there is a protocol Π privately computing f in G. We construct an $\mathcal{S}$-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 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 1-private and in Π' we require privacy only for the curious vertices, Protocol Π' is $\mathcal{S}$-private.

Now, assume that there is an $\mathcal{S}$-private protocol Π' computing f' in T_G. We construct a 1-private protocol Π computing f in G. In this protocol, every curious vertex will effectively have the same information as in Π', and the messages received by a vertex v_W will be secret-shared by all vertices in W. 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 $r_{w,0}$ distributed uniformly in R, and the parties define $r_W = \bigoplus_{w \in W} r_{w,0}$. Protocol Π' has rounds and in each round only one vertex sends messages. W.l.o.g., assume that every message in Π' is one bit. Protocol Π will have a virtual round for every round of Π'. There are three cases to consider:

If the sender of a message m in Π' is u and the receiver is v, where u and v are curious vertices, then u sends the message m to v in Π.

If the sender of a message m in Π' is u, where u is a separating vertex in G, and the receiver is v_W, where W is a connected component in G, then each vertex w in W chooses at random, with uniform distribution, a bit r_w and the vertices in W compute the function $m \oplus \bigoplus_{w \in W} r_w$ using a 1-private protocol. By Theorem 1 such protocol exists since each 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 last case is when the sender of a message m in Π' is v_W, where W is a connected component in G, and the receiver is v, where v is a separating vertex in G. In this case, the message m in Π' is 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. In Protocol Π, the receiver v chooses a random bit r_v with uniform distribution, and the vertices in W compute the function $m \oplus r_v$ using a 1-private protocol. On one hand, vertex v learns the message m, but no additional information. On the other hand, each vertex in $W \setminus \{v\}$ gains no information from this virtual round.

We next argue that this protocol is 1-private. First we argue that every non-curious vertex in G learns no information in this protocol. The messages such a vertex gets during the execution of Protocol Π are messages in 1-private protocols computing a function masked by r_v for a separating vertex v in the connected component. Thus, each non-curious vertex does not learn any information during the protocol for f. Every curious vertex learns only the messages it got in Π', and, since Π' is $\mathcal{S}$-private, the curious vertex gains no information that is not implied by its input and the output of the function. $\square$

5 Private Computation on Trees

By Lemma 4, to characterize which functions can be computed 1-privately on G, we need to characterize which functions can be computed $\mathcal{S}$-privately in T_G. We do not have an exact characterization of these functions. We only give necessary and sufficient conditions for this task. In the sequence, we say that a vertex v is curious if $\{v\} \in \mathcal{S}$.

5.1 Sufficient Condition

In this section we give a sufficient condition for computing a function $\mathcal{S}$-privately in a tree. Using Lemma 4, the results of this section give a sufficient condition for computing a function 1-privately in arbitrary networks. The sufficient condition is a simple generalization of the condition of [20]. We next introduce some notation generalizing Definitions 3 and 4 (taken from [20]). We represent a function $f : A_1 \times \ldots A_n \to O$ 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 5 (Forbidden Array). *Let M be an n-dimensional array whose ith-dimension is labeled by the elements of A_i, and $\mathcal{S}$ be a the collection of curious vertices (where $|S| = 1$ for every $S \in \mathcal{S}$). The relation $\sim_i$ on A_i is defined as follows: $a, b \in A_i$ satisfy $a \sim_i b$ if there exist some $\boldsymbol{a}, \boldsymbol{b} \in A_1 \times \cdots \times A_n$ and an index $j \neq i$ such that (1) $\{v_j\} \in \mathcal{S}$, (2) $a_i = a$, (3) $b_i = b$, (4) $a_j = b_j$, and (5) $M(\boldsymbol{a}) = M(\boldsymbol{b})$. The equivalence relation $\equiv_i$ on A_i is defined as the transitive closure of the relation $\sim_i$.*

An array M is a forbidden array iff (1) the array is not constant, (2) for all i, all the elements of A_i are equivalent in M according to $\equiv_i$.

Lemma 5. *Let f be a function. If the array M_f does not contain a forbidden array, then f can be computed $\mathcal{S}$-privately on any tree with n vertices.*

Proof. The protocol is a simple generalization of the protocol of [20]. In each step of the protocol, the parties, maintain a cube $R_1 \times \ldots \times R_n \subseteq A_1 \times \ldots \times A_n$, such that $\langle a_1, \ldots, a_n \rangle \in R_1 \times \ldots \times R_n$. In the beginning, $R_i \leftarrow A_i$ for $i \in \{1, \ldots, n\}$. At the end of the protocol, $R_1 \times \ldots \times R_n$ is constant, so each party can deduce the value of f. In each step, consider the array M which is the array M_f restricted to $R_1 \times \ldots \times R_n$. As M_f does not contain a forbidden array, the array M is not forbidden. If M is constant, then all the vertices know that this constant is the output, and the protocol ends. Otherwise, for some $i \in \{1, \ldots, n\}$, not all the elements of A_i are equivalent in M according to $\equiv_i$. Vertex v_i sends to its neighbors the equivalence class of a_i in M, and this information is propagated to all vertices in the tree. Thereafter, all parties set R_i as this equivalence class.

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. We next argue that 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 $f(\boldsymbol{a}) = f(\boldsymbol{b})$ where $a_j = b_j$, then in each stage of the protocol $a_i \equiv_i b_i$ in M for every i, and the same communication string is exchanged on $\boldsymbol{a}$ and $\boldsymbol{b}$, thus v_j does not gain extra information. $\square$

In the protocol described in the proof of Lemma 5, each message is propagated 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.

5.2 Necessary Condition

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: We show that if a function can be computed in a tree then a related function can be computed in the two-party and eavesdropper model, where Alice's input is the inputs of one side of the tree, Bob's input is the inputs of other 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 sufficiency in Lemma 1.

Lemma 6. *Let $T = \langle V, E \rangle$ be a tree and $\mathcal{S} \subseteq 2^V$, where $|S| = 1$ for every $S \in \mathcal{S}$. Let v_i be a curious vertex in T which is not a leaf. W.l.o.g. assume that for every j, k such that $j < i < k$, vertex v_i separates vertex v_j and vertex v_k. Furthermore, let f be a function that can be computed $\mathcal{S}$-privately in T. For every $c \in A_i$, define f_c as $f_c(\langle a_1, \ldots, a_{i-1} \rangle, \langle a_{i+1}, \ldots, a_n \rangle) = f(a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_n)$. Then, for every every $c \in A_i$, the function f_c can be computed privately in the two-party and eavesdropper model.*

Using Lemma 2 we can deduce the following necessary condition. Roughly speaking, the condition is that the inputs corresponding to each output value are a union of certain n-dimensional cubes. For the lemma, 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 $\boldsymbol{c} = \langle c_i \rangle_{i \in I}$ be a vector where $c_i \in A_i$ for every $i \in I$. Denote $f_{I,\boldsymbol{c}} : \Pi_{i \notin I} A_i \to O$, the restriction of f to $\{1, \ldots, n\} \setminus I$, as follows $f_{I,\boldsymbol{c}}(\langle a_i \rangle_{i \notin I}) = f(\langle b \rangle_{i \in \{1,\ldots,n\}})$ where $b_i = c_i$ if $i \in I$ and $b_i = a_i$ otherwise. The proof of the following lemma is omitted for lack of space.

Lemma 7. *Let $T = \langle V, E \rangle$ be a tree and $\mathcal{S} \subseteq 2^V$, where $|S| = 1$ for every $S \in \mathcal{S}$ and $\langle u, v \rangle \notin E$ for every two non-curious vertices u, v.*[3] *Assume that a function $f : A_1 \times \cdots \times A_n \to O$ can be computed $\mathcal{S}$-privately in T, and define $I \stackrel{def}{=} \{i : v_i \text{ is a curious vertex}\}$, and $n' = n - |I|$. Then, for every $\boldsymbol{c} \in \Pi_{i \in I} A_i$*

[3] This is a technical requirement as we can replace such non-curious neighbors by a new vertex holding the inputs of the two neighbor.

and every output value $o \in O$, the inputs of $f_{I,\mathbf{c}}$ corresponding to o are an n'-dimensional cube, that is, there exist sets $\langle R_i \rangle_{i \notin I}$ such that $R_i \subseteq A_i$ and $f_{I,\mathbf{c}}(\boldsymbol{a}) = o$ if and only if $a_i \in R_i$ for every $i \notin I$.

Acknowledgement. We would like to thank Enav Weinreb for valuable comments that greatly improved this write-up.

References

1. A. Beimel, M. Franklin. Reliable communication over partially authenticated networks. *Theoretical Computer Science*, 220:185–210, 1999.
2. A. Beimel, L. Malka. Efficient reliable communication over partially authenticated networks. In *the 22nd PODC*, pages 233–242, 2003.
3. M. Ben-Or, S. Goldwasser, A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *the 20th STOC*, pages 1–10, 1988.
4. M. Bläser, A. Jakoby, M. Liśkiewicz, B. Manthey. Private computation – k-connected vs. 1-connected networks. In *CRYPTO 2002*, vol. 2442 of *LNCS*, pages 194–209, 2002.
5. M. Bläser, A. Jakoby, M. Liśkiewicz, B. Manthey. Privacy in non-private environments. In *ASIACRYPT 2004*, vol. 3329 of *LNCS*, pages 137 – 151. 2004.
6. B. Bollobás. *Modern Graph Theory.* 1998.
7. D. Chaum, C. Crépeau, I. Damgård. Multiparty unconditionally secure protocols. In *the 20th STOC*, pages 11–19, 1988.
8. B. Chor, E. Kushilevitz. A zero-one law for Boolean privacy. *SIDMA*, 4(1):36–47, 1991.
9. Y. Desmedt, Y. Wang. Secure communication in multicast channels: The answer to Franklin and Wright's question. *J. of Cryptology*, 14(2):121–135, 2001.
10. Y. G. Desmedt, Y. Wang. Perfectly secure message transmission revisited. In *EUROCRYPT 2002*, vol. 2332 of *LNCS*, pages 502–517. 2002.
11. D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, 3:14–30, 1982.
12. D. Dolev, C. Dwork, O. Waarts, M. Yung. Perfectly secure message transmission. *J. of the ACM*, 40(1):17–47, 1993.
13. C. Dwork, D. Peleg, N. Pippenger, E. Upfal. Fault tolerance in networks of bounded degree. *SIAM J. on Computing*, 17(5):975–988, 1988.
14. M. J. Fischer, N. A. Lynch, M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.
15. M. Franklin, R. N. Wright. Secure communication in minimal connectivity models. *J. of Cryptology*, 13(1):9–30, 2000.
16. M. Franklin, M. Yung. Secure hypergraphs: privacy from partial broadcast. In *the 25th STOC*, pages 36–44, 1993.
17. O. Goldreich, S. Goldwasser, N. Linial. Fault-tolerant computation in the full information model. In *the 32nd FOCS*, pages 447–457, 1991.
18. A. Jakoby, M. Liskiewicz, R. Reischuk. Private computations in networks: Topology versus randomness. In *the 20th STACS*, vol. 2607 of *LNCS*, pages 121–132. 2003.
19. M. V. N. A. Kumar, P. R. Goundan, K. Srinathan, C. Pandu Rangan. On perfectly secure communication over arbitrary networks. In *the 21st PODC*, pages 193–202, 2002.
20. E. Kushilevitz. Privacy and communication complexity. *SIDMA*, 5(2):273–284, 1992.
21. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufman Publishers, 1997.
22. K. Menger. Allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.
23. T. Rabin, M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *the 21st STOC*, pages 73–85, 1989.
24. H. M. Sayeed, H. Abu-Amara. Efficient perfectly secure message transmission in synchronous networks. *Information and Computation*, 126:53–61, 1996.
25. K. Srinathan, V. Vinod, C. Pandu Rangan. Efficient perfectly secure communication over synchronous networks. In *the 22nd PODC*, pages 252–252, 2003.
26. K. Srinathan, V. Vinod, C. Pandu Rangan. Optimal perfectly secure message transmission. In *CRYPTO 2004*, vol. 3152 of *LNCS*, pages 545 – 561, 2004.
27. E. Upfal. Tolerating a linear number of faults in networks of bounded degree. *Information and Computation*, 115(2):312–320, 1994.