Secured Distributed Algorithms Without Hardness Assumptions

Leonid Barenboim
Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel
leonidb@openu.ac.il

Harel Levin
Department of Physics, Nuclear Research Center-Negev, Giv'atayim, Israel
Department of Mathematics and Computer Science, The Open University of Israel, Raanana, Israel
harell@nrcn.org.il

Abstract

We study algorithms in the distributed message-passing model that produce secured output, for an input graph G. Specifically, each vertex computes its part in the output, the entire output is correct, but each vertex cannot discover the output of other vertices, with a certain probability. This is motivated by high-performance processors that are embedded nowadays in a large variety of devices. Furthermore, sensor networks were established to monitor physical areas for scientific research, smart-cities control, and other purposes. In such situations, it no longer makes sense, and in many cases it is not feasible, to leave the whole processing task to a single computer or even a group of central computers. As the extensive research in the distributed algorithms field yielded efficient decentralized algorithms for many classic problems, the discussion about the security of distributed algorithms was somewhat neglected. Nevertheless, many protocols and algorithms were devised in the research area of secure multi-party computation problem (MPC or SMC). However, the notions and terminology of these protocols are quite different than in classic distributed algorithms. As a consequence, the focus in those protocols was to work for every function f at the expense of increasing the round complexity, or the necessity of several computational assumptions. In this work, we present a novel approach, which rather than turning existing algorithms into secure ones, identifies and develops those algorithms that are inherently secure (which means they do not require any further constructions). This approach yields efficient secure algorithms for various locality problems, such as coloring, network decomposition, forest decomposition, and a variety of additional labeling problems. Remarkably, our approach does not require any hardness assumption, but only a private randomness generator in each vertex. This is in contrast to previously known techniques in this setting that are based on public-key encryption schemes.

This paper is eligible for the best student paper award

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Security and privacy → Privacy-preserving protocols; Mathematics of computing → Graph coloring; Mathematics of computing → Graph algorithms

Keywords and phrases distributed algorithms, privacy preserving, graph coloring, generic algorithms, multi-party computation

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2020.32

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/2011.07863`.

1 Introduction

Over the last few decades, computational devices get smaller and are embedded in a wide variety of products. High-performance processors are embedded in smart phones, wearable devices and smart home devices. Furthermore, sensor networks were established to monitor physical areas for scientific research, smart-cities control and other purposes. In such situations, it no longer makes sense, and in many cases it is not feasible, to leave the whole

processing task to a single computer or even a group of central computers. In the distributed algorithms research field, all the processors are employed to solve a problem together. The basic assumption is that all the processors run the same program simultaneously. The network topology is represented by a graph $G = (V, E)$ where each processor (also referred as *node*) is represented by a vertex, $v \in V$. Each communication line between a pair of processors $v, u \in V$ in the network is represented by an edge $(v, u) \in E$.

The time complexity of a distributed algorithm is measured by rounds. Each round consists of three steps: (1) Each processor receives the messages that were sent by its neighbors on the previous round. (2) Each processor performs a local computation. (3) Each processor may send messages to its neighbors. The time complexity of distributed algorithms is measured by the number of rounds necessary to complete an algorithm. Local computations (that is, computations performed inside the nodes) are not taken into account in the running time analysis in this model.

Despite the extensive research in the distributed algorithms field in the last decades, the discussion about the security of distributed algorithms was somewhat neglected. Nevertheless, many protocols and algorithms were devised in the research area of cryptography and network security. The secure multi-party computation problem (MPC or SMC) is one of the main problems in the cryptography research. However, the notions and terminology of these protocols is quite different than in classic distributed algorithms. Moreover, most of these protocols assume the network forms a complete graph. Additionally, the protocols have no restriction on the amount of communication between the nodes.

In this work we devise secure distributed algorithms, in the sense that the output of each processor is not revealed to others, even though the overall solution expressed by all outputs is correct. Our notion of security is the following. Consider a problem where the goal is assigning a label to each vertex or edge of the graph $G = (V, E)$, out of a range $[t]$, for some positive t. A secure algorithm is required to compute a proper labeling, such that for any vertex $v \in V$, (respectively edge $e \in E$) the other vertices in V (resp. edges in E) are not aware of the label of v (resp. e). Moreover, other vertices or edges can guess the label with probability at most $1/\lambda$, for an appropriate parameter $\lambda \leq t$. Note that this requirement can be achieved if each participant v (resp. e) in the network computes a set of labels $\{l^1, l^2, ..., l^\lambda\}$ ($l^i \in [t]$), such that any selection from its set forms a proper solution, no matter which selections are made in the sets of other participants. For example, in a proper coloring problem, if each vertex computes a set of colors (rather than just one color), and the set is disjoint from the sets of all its neighbors, the goal is achieved. In this case each participant draws a solution from its set of labels uniformly at random. The result is kept secret by the participant, and thus others can guess it with probability at most $1/\lambda$. Thus, if the number of labels is small, the possibility of guessing a result of a vertex becomes quite large, inevitably. As we will demonstrate later, one can artificially increase the amount of labels to achieve smaller probabilities. However, when it is impossible to use a large number of labels, other techniques can be taken into account (such as Parter and Yogev's compiler [20]). Nevertheless, our method is applicable to various distributed problems. Moreover, the overhead caused by the privacy preserving is negligible as the round complexity of our algorithms is similar to the best known (non privacy preserving) algorithms. A summary is found in Table 1. The parameter λ is referred to as the *solution domain* in Table 1. The ratio between t and λ is referred to as the *contingency factor*. These terms will be discussed later in Chapter 3.

Table 1 List of inherently secure algorithms and their privacy attributes.

Problem	Type of Graph	Rounds Complexity	Solution Domain Size	Contingency Factor
3Δ-Coloring	Oriented trees	$O(\log^* n)$	Δ	3
$2c \cdot \Delta \log n$-Coloring	General	$O(1)$	$c \cdot \log n/2$	$O(\Delta)$
$O(\Delta^2)$-Coloring	General	$\log^* n + O(1)$	Δ	$O(\Delta)$
p-Defective $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-Coloring	General	$O(\log^* n)$	$O\left(\frac{\Delta}{p}\right)$	$O\left(\frac{\Delta}{p}\right)$
$2a \cdot c \cdot \log n$-Coloring	Bounded Arboricity a	$O(\log n)$	$O(\log n)/2$	$O(a)$
$(O(\log n), O(c \cdot \log n))$-Network Decomposition	General	$O(\log^2 n)$	$c > 1$	$O(\log n)$
Δ-Forest Decomposition	General	$O(1)$	$\binom{\Delta}{\lvert E(v)\rvert}$	1
$(2+\epsilon) \cdot a$-Forest Decomposition	Bounded Arboricity a	$O(\log n)$	$\binom{((2+\epsilon)\cdot a)}{\lvert E(v)\rvert}$	1
$O(\Delta \log n)$-Edge Coloring	General	$O(1)$	$c \cdot \log n$	$O(\Delta)$
$O(\Delta^2)$-Edge Coloring	General	$\log^*(n) + O(1)$	$(2\Delta - 1)$	$O(\Delta)$
p-Defective $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-Edge Coloring	General	$O(1)$	$O\left(\left(\frac{\Delta}{p}\right)^2\right)$	1
$(t \cdot \sqrt{\Delta})$-Edge Coloring of a Dominating Set	General	$\tilde{O}(\log \Delta + \log^3 \log n)$	t	$\sqrt{\Delta}$

2 Background

2.1 Distributed Algorithms

Given a network of n processors (or *nodes*), consider a graph $G = (V, E)$ such that $V = v_1, v_2, ..., v_n$ is a set of vertices, each represents a processor. For each two vertices $u, v \in V$, there is an edge $(u, v) \in E$ if and only if the two processors corresponding to the vertices u, v have a communication link between them. A communication link may be unidirectional or bidirectional, resulting in an undirected or a directed graph (respectively). Unless stated otherwise, the graphs in this work are simple, undirected and unweighted.

Two vertices $u, v \in V$ are *independent* if and only if $(u, v) \notin E$. The *neighbors* set of a vertex $v \in V$, $\Gamma(v)$ consists of all the vertices in V that share a mutual edge with v in E. Formally, $\Gamma(v) = \{u \in V | (u, v) \in E\}$. The *degree* of a vertex $v \in V$, $deg(v) = |\Gamma(v)|$. Note that $0 \leq deg(v) \leq n - 1$. The *maximum degree* of graph G, $\Delta(G)$, is the degree of the vertex $v \in V$ which has the maximum number of neighbors. If the graph G is directed, the out (respectively, in) degree of vertex $v \in V$ ($deg_{out}(v)$ and resp. $deg_{in}(v)$) is the number of edges $(u, v) \in E$ ($u \in V$) with orientation that goes out from (respectively, in to) vertex v.

Throughout this paper, $\mathcal{LOCAL}$ model will be used as the message-passing model. In this model, each communication line can send at each round an unrestricted amount of bits. It means that the primary measure is the number of rounds each node needs to "consult" its neighborhood by sending messages. This is in contrast to $\mathcal{CONGEST}$ model, where the bandwidth on each communication line on each cycle is bounded by $O(\log n)$.

A single bit can pass from one endpoint of the graph to the other endpoint in $D(G)$ rounds (where $D(G)$ is the diameter of graph G). Thus, in the $\mathcal{LOCAL}$ model we usually look for time complexity lower than $O(D(G))$ and even sub-logarithmic (in terms of $|V|$), since all the nodes can learn the entire topology of the graph in $O(D(G))$ rounds and then perform any computation on the entire graph. Consequently, the research in local distributed algorithms is focused on solving those graph theory problems which have solutions that depend on the local neighborhood of each vertex rather than the entire graph topology.

Most of the problems in graph theory may be classified into two types. The first type is a bipartition of the graph (whether the vertices, the edges, or both) into two sets. For some problems both of these sets are of interest, and for other problems only one of the sets, while the other sets may be categorized as "all the rest". Examples of such problems include

Maximal Independent Set and Maximal Matching. The other type of problems partitions the graph into several sets. This type of problems may be referred as "labeling" problems where there is a set of valid labels and every part of the graph is labeled by a unique label. Examples of such problems include Coloring and Network Decomposition.

In the following sections we will present some of the problems in the field of graph theory which exploit the potential of distributed algorithms. While we discuss some of the main bipartition problems, in our model for privacy preserving the labeling problems are more relevant.

2.2 Graph Theory Problems

The definition of the problems is given here briefly, a detailed definition can be found in [5].

A function $\varphi : V \to [\alpha]$ is a legal α-*Coloring* of graph $G = (V, E)$ if and only if, for each $\{v, u\} \in E \to \varphi(v) \neq \varphi(u)$. Similarly, a function $\varphi : E \to [\alpha]$ is a valid α-*edge coloring* of graph $G = (V, E)$ i.f.f. for any vertex $v \in V$ there are no two distinct vertices $u, w \in \Gamma(v)$ such that $\varphi((v, u)) = \varphi((v, w))$. While a deterministic construction of such $(\Delta + 1)$-graph coloring requires at least $O(\log^* n)$ rounds [16]. A randomized $(\Delta + 1)$-graph coloring can be done in poly$(\log \log n)$ rounds [22]. Graph coloring is of special interest due to its applications in many resource management algorithms. In particular, certain resource allocation tasks require a proper coloring (possibly of a power graph) and that each vertex knows its own color, but not the colors of its neighbors. For example, this is the case in certain variants of Time Division Multiple Access schemes.

A *forest* is a graph which contains no cycles. A *forest decomposition* of graph $G = (V, E)$ is an edge-disjoint partition of G, to α sub-graphs $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_\alpha$ such that $\mathcal{F}_i$ is a forest for every $1 \leq i \leq \alpha$. One way to define the *arboricity* of a graph G is as the minimal number of forests which are enough to fully cover G.

Given a graph $G = (V, E)$ and a vertex-disjoint partition of graph $G = (V, E)$ to α clusters $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_\alpha$, we define an auxiliary graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_\alpha\}$ and $(\mathcal{C}_u, \mathcal{C}_v) \in \mathcal{E}$ $(\mathcal{C}_u, \mathcal{C}_v \in \mathcal{V})$ iff $\exists (u, v) \in E$ such that $u \in \mathcal{C}_u$ and $v \in \mathcal{C}_v$. The partition $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_\alpha$ is a valid (d, c)-*network decomposition* [1] if (1) the chromatic number of $\mathcal{G}$ is at most c and (2) the distance between each pair of vertices contained in the same cluster $v, u \in \mathcal{C}_i$ is at most d. In strong network decomposition, the distance is measured with respect to the cluster $\mathcal{C}_i$ (in other words, $dist_{\mathcal{C}_i}(v, u) \leq \mathrm{d}$). In weak network decomposition, the distance is measured with respect to the original graph G (in other words, $dist_G(v, u) \leq \mathrm{d}$). Different algorithms yield different kind of network decompositions which satisfy different values of d and c. One of the most valuable decompositions, which presents good trade-off between the radius of each cluster and the chromatic number of the auxiliary graph is an $(O(\log n), O(\log n)$-network decomposition.

A set $I \subseteq V$ of vertices is called an *Independent Set* (*IS*) if and only if for each pair of vertices $v, u \in I$ there is no edge $(v, u) \in E$. An independent set I is *Maximal Independent Set* (*MIS*) i.f.f. there is no vertex $v \in V \setminus I$ such that $I \cup \{v\}$ is a valid independent set. Similarly, a set of edges $M \in E$ is called a *Matching* i.f.f. there is no pair of vertices $u_1, u_2 \in V$ $(u_1 \neq u_2)$ such that $\exists v \in V$ where $\{(u_1, v), (u_2, v)\} \subseteq M$. A matching M is *Maximal Matching* (*MM*) i.f.f. there is no edge $e \in E \setminus M$ such that $M \cup \{e\}$ is a valid matching.

2.3 Secure Multi-Party Computation

Multi-Party Computation (*MPC*) is the ability of a party consisting of n participants to compute a certain function $f(x_1, x_2, ..., x_n)$ where each participant $i (1 \leq i \leq n)$ holds only its own input x_i. At the research fields of cryptography and networks security, *Secure MPC* [23] protocols enables parties to compute a certain function f without revealing their own input (x_i). Our security model is information-theoretic secure, which means it is not based on any computational assumptions. Furthermore, we base our security notion on the *semi-honest model* (as was devised by [11]) which means that there are possibly curious participants but no *malicious adversary*. In other words, adversary participant can not deviate from the prescripted protocol. However, it may be curious, meaning it may run an additional computation in order to find out private data of another participants. Permitting the existence of malicious adversaries which may collude with t nodes will necessitate the graph to be $(2t + 1)$-connected for security to hold (as shown by [20]), which is not a feasible constraint.

Previous works on secure-MPC ([23], [11]) do not state any assumptions on the nature of neither the function f nor the interactions between the participants. As a consequence, the privacy preserving protocols devised during the past decades are generalized for any kind of mathematical function and not necessarily computation of graph features. Furthermore, each of the participants is assumed to be an equal part of the computation. As such, any pair of participants is assumed to have a private communication line of its own. Translating those protocols to distributed algorithms for graph theory problems, will require a complete graph representing the communication which may be different than the input graph of the problem. While this approach is applicable in many realistic networks and problems, general networks with non-uniform communication topology may benefit from efficient distributed algorithms for computations where the desired function f is local. Other works (such as [13] and [12]) are dedicated to general graphs. However, their goal was not to optimize the rounds complexity as the protocols created by their algorithms will require at least $O(n^2)$ rounds even for a relatively simple function f. Furthermore, their techniques require a heavy setup phase, and based on some computational assumptions. Several other works provide secure protocols for general or sparse graphs ([6] [10] [7]). However, the focus in those protocols was to work for every function f, at the expense of increasing the round complexity, or the necessity of several computational assumptions.

Recently, Parter and Yogev [20] [21] suggested a new kind of privacy notion which they tailored to the $\mathcal{CONGEST}$ distributed model. In their notion, the neighbors of each node v construct a *private neighborhood tree* throughout which they broadcast a shared randomness. This randomness is used in order to encrypt the private variable of each neighbor. The node receives these encrypted private variables $x_1, x_2, ..., x_t$ $(t = |\Gamma(v)|)$ and performs its local computation $f(x_1, x_2, ..., x_t)$. Let $\mathrm{OPT}(G)$ be the best depth possible for private neighborhood trees. Parter and Yogev devised an algorithm which constructs such trees in $O(n + \Delta \cdot \mathrm{OPT}(G))$ rounds, where each tree has depth of $O(\mathrm{OPT}(G) \cdot \mathrm{polylog}(n))$ and each edge $e \in E$ is part of at most $O(\mathrm{OPT}(G) \cdot \mathrm{polylog}(n))$ trees. Using their notion one can turn any r-rounds algorithm into a secure algorithm with an overhead of $\mathrm{poly}(\Delta, \log n) \cdot \mathrm{OPT}(G)$ rounds for each round. Furthermore, they showed that for a specific family of distributed algorithms (to which they referred as "simple"), the round overhead can be reduced to $\mathrm{OPT}(G) \cdot \mathrm{polylog}(n)$. Using their method they have devised a variety of both global and symmetry-breaking local algorithms. However, their notion requires an extensive pre-construction phase additional to the secure computation itself which requires quite high round complexity. Furthermore, their notion assumes a bridgeless graph, meaning that the

graph consists of a single connected component and there is no single edge such that its removal will split the graph into two connected components. The method of Parter and Yogev relies on cryptographic hardness assumptions, e.g., the existence of a shared randomness and a public-key encryption scheme. This allows achieving security both for input and output. In contrast, our current work focuses on output security, but there are no hardness assumptions, and no requirement for bridgeless graphs.

3 Inherently Secure Distributed Algorithms

Most of the classic distributed algorithms models assume that each vertex is aware of its neighbors. This is the case in our paper as well. Usually each node does not have any additional input except for its own ID. The output of the algorithm is a set of labels where each label corresponds to each vertex. Throughout the current work, vertices IDs will not be considered as a private input. Formally, from the perspective of node $v \in V$, a classic distributed algorithm calculates a function $f_v(D_{\Gamma(v)}) = l_v$, where $l_v \in [l]$ (for some constant l) is vertex v's label which was calculated based on the input messages ($D_{\Gamma(v)}$) came from vertex v's neighbors ($\Gamma(v)$). From a global perspective, the algorithm computes: $f : G(V, E) \to [l]^n$. This work considers the following security notion: each vertex v cannot infer the value of l_u (such that $u \in V, u \neq v$) with a certain probability. Our model assumes that each node $v \in V$ holds a private randomness generator r_v.

As an example, consider an algorithm for Δ^2 graph multicoloring of graph $G = (V, E)$ with maximum degree $\Delta = \Delta(G)$ which provides any vertex v with a set of Δ *valid* colors $\varphi(v) = \{x_1, x_2, ..., x_\Delta\}$. By "valid" we mean that any of the colors in $\varphi(v)$ is not contained in any of v's neighbors' sets, i.e. $x_i \notin \bigcup_{u \in \Gamma(v)} \varphi(u)$ (for any $1 \leq i \leq \Delta$). Using this kind of coloring, v can *privately* select a random color out of the Δ valid colors in $\varphi(v)$. Hence, the identity of the exact color of v can be securely hidden from any of the other vertices in G.

We generalize the above idea as follows. Consider the following family of algorithms. Each algorithm Π in the *Inherently-Secure* algorithms family $\mathcal{IS}$ consists of two stages: (1) Calculating a generic set of k possible valid labels. (2) Randomly and privately (using the private randomness generator r_v), each node selects its final label. That is, the first stage of algorithm Π (denoted by $\Pi_{generic}$) calculates the function: $f_1(G(V, E)) = \{\ell_{u_1}, ..., \ell_{u_n}\}$, where $\ell_{u_i} = \{l_i^1, l_i^2, ..., l_i^k\}$ for any $u_i \in V$. Henceforth, $\Pi_{generic}$ will be referred as *generic-algorithm*. The first stage can run without any additional security considerations, meaning any node may know the ℓ_{u_i} of other nodes. Later, we will show algorithms which satisfies even stronger security notion where the identity of ℓ_{u_i} is also kept secret. The second stage (Π_{select}) securely calculates the function $f_2 : [\ell]^{k^n} \to [l]^n$. Overall, algorithm Π indeed calculates $f = f_1 \circ f_2 : G(V, E) \to \{l_1, ..., l_n\}$. Let L be the ground set of valid labels from which the possible labels are being picked, i.e. for any $1 \leq i \leq n$ and $1 \leq j \leq k$, $l_i \in L$ and $l_i^j \in L$.

By increasing the amount of possible values (k) we make the actual labels $\{l_1, l_2, ..., l_n\}$ less predictable. However, in order to do so we may need to increase the ground set of the available labels. For instance, in graph coloring we may want to be able to produce Δ valid possible colors for each vertex. However, an increase of the amount of colors (to Δ^2) may be necessary. On the other hand, one may want to minimize the size of the ground set since large ground sets may lead to trivial algorithms on one hand, and to a higher memory complexity on the other hand.

In order to analyze this kind of algorithms we define several parameters.

▶ **Definition 1.** *The size of the problem domain of a problem $\mathcal{P}$ solved by algorithm Π which calculates the function $f : G(V, E) \to \{l_1, l_2, .., l_n\}$, where $l_i \in L$, is the number of valid labels for any l_i, i.e. $|L|$.*

▶ **Definition 2.** *The size of the solution domain of a generic algorithm $\Pi_{generic}$ which calculates the function $f_1 : G(V, E) \to \{\{l_1^1, l_1^2, ..., l_1^k\}, ...\{l_n^1, l_n^2, ..., l_n^k\}\}$ is the minimal number of valid possible labels for any vertex, i.e. k.*

▶ **Definition 3.** *The contingency factor of generic algorithm $\Pi_{generic}$ used to solve problem $\mathcal{P}$ (as they defined on definitions 1 and 2) is the ratio between the size of the problem domain $|L|$ (Def. 1) and the size of the solution domain k (Def. 2) , i.e. $|L|/k$.*

In order to clarify these definitions, consider the problem of Δ^2-graph coloring. The size of the problem domain is Δ^2. A generic algorithm that calculates Δ possible valid colors for each vertex will provide a solution domain of size Δ. The contingency factor of this algorithm will be $\frac{\Delta^2}{\Delta} = \Delta$.

In many cases, the number of labels (i.e. the size of the problem domain) can be increased artificially by a factor $c > 1$. This artificial increase will lead to an expansion of the problem domain by the same factor c. As a result, the contingency factor will remain the same. That is, the contingency factor is a property of the algorithm itself and not influenced by artificial increases. Small contingency factor indicates that most of the members of the problem domain are valid options on the solution domain, while the generic algorithm did not exclude those members from being considered as valid possible solutions. As such, small contingency factor indicates that the algorithm preserves better security by excluding only a small portion of possible solutions. Problems with small problem-domain will have even smaller solution domain which will lead to a contingency factor that is close to the original size of the problem domain. Therefore, finding a generic algorithm with good contingency factor for these problems is a complicated task. As a consequence, we will focus on finding generic algorithms for problems with relatively large problem-domain, i.e. labeling problems. For problems with small problem domain, other techniques (such as Parter and Yogev's compiler [20]) should be considered.

Note that even though a malicious node may interrupt the validity of the algorithm by picking a solution which is not part of its solution domain, this kind of intrusion will not affect the privacy of the algorithm. However, in our model the nodes are not malicious.

3.1 Generic Algorithms for Graph-Coloring

Considering the problem of graph coloring, we will focus on finding generic algorithms that will provide contingency factor of Δ. This factor is optimal for general graphs, as we prove in Theorem 4.

▶ **Theorem 4.** *For any α-coloring problem ($\alpha > \Delta$), and any generic algorithm Π, there is an infinite family of graphs such that their solution domain must be of size $O(\alpha/\Delta)$ at most. Hence, the contingency factor would be at least $\Omega(\Delta)$.*

Proof. Suppose for contradiction that there is a valid solution domain such that every vertex has more than α/Δ valid options. Consider a graph $G = (V, E)$ with clique $C \subseteq V$ of size $|C| = \Delta + 1$. Each vertex $v \in C$ has Δ neighbors, each of them has α/Δ valid colors. But since v and all its neighbors are part of the clique, each of them has a unique set of colors. It means that there are at least $(\Delta + 1) \cdot (\alpha/\Delta) > \alpha$ colors in the α-coloring, which is a contradiction. ◀

3.1.1 Generic Algorithm for 3Δ-Coloring of Oriented Trees

A well known algorithm for 3-Coloring of oriented trees was devised by Cole and Vishkin [8]. The deterministic algorithm exploits the asymmetric relationship between a vertex v and its parent (in the tree) $\pi(v)$ in order to get a valid coloring in $O(\log^* n)$ rounds.

For any two integers a and b, let $< a, b >$ be a tuple which can be represented in binary as the concatenation of the binary representations of a and b. We can define a generic algorithm that runs Cole Vishkin's algorithm to get a valid coloring $\varphi : V \to [3]$. Later, each vertex will set its solution domain $\hat{\varphi}(v)$ as follows: $\hat{\varphi}(v) = \bigcup_{0 \le i < \Delta} < i, \varphi(v) >$. As a result, we get a generic algorithm where each vertex has Δ valid colors. The validity of this algorithm is provided by the following Lemma:

▶ **Lemma 5.** *For any vertex $v \in V$, for every value $x \in \hat{\varphi}(v)$, x is not a possible color for any other vertex $u \in \Gamma(v)$.*

Proof. Suppose for contradiction that there exists a vertex $u \in \Gamma(v)$ such that $x \in \hat{\varphi}(u)$. Since, $\hat{\varphi}(v) = \bigcup_{0 \le i < \Delta} < i, \varphi(v) >$, there exists a value $1 \le i \le \Delta$ such that $x =< i, \varphi(v) >=< i, \varphi(u) >$. Hence, $\varphi(v) = \varphi(u)$, which is a contradiction since φ is a valid coloring as was proved by [8]. ◀

Lemma 5 leads to the following corollary:

▶ **Corollary 6.** *Given a tree $T = (V, E, v)$, there exists a generic algorithm which provides any vertex $v \in V$ with a set of Δ possible valid colors*

The generic algorithm described above uses a simple approach which achieves privacy by artificially increasing the size of both the problem domain and the solution domain accordingly. Another technique is to run an algorithm d times in parallel. For coloring problems a good d will probably be Δ (as was shown in Theorem 4). While this approach will lead in deterministic algorithms to the same results as the previous technique, applying this technique with random algorithms will lead to solution domain which is somewhat less predictable than the domain we will receive by artificially increasing the size of the solution domain.

While these approaches (artificially increasing the size of the solution domain and run the algorithm multiple times) are useful for problems with very efficient base algorithms (i.e. Cole Vishkin 3-coloring), for many problems such an efficient algorithm is not yet known. However, one still may devise efficient generic-algorithms for some of these problems, as demonstrated in the following sections.

3.1.2 Generic Algorithm for $2\Delta c \cdot \log n$-Coloring of General Graphs

While the best known algorithms for $(\Delta + 1)$-Coloring of generic graphs uses logarithmic number of rounds [14] [18], a reasonable size of contingency factor may yield more efficient algorithms with sub-logarithmic and even constant number of rounds. As an example, consider Algorithm 1 which uses $O(1)$ rounds to achieve a secure coloring of general graphs with contingency factor of $O(\Delta)$.

Algorithm 1 GENERIC-RANDOM-COLORING.

Result: A set of $O(\log n)$ colors for each vertex $v \in V$

1. Every vertex selects independently at random $k = c \cdot \log n$ different numbers (c is a constant, $c > 1$) $I = \{< 1, x_1 >, < 2, x_2 >, ..., < k, x_k >\}$ where $x_i \in [2\Delta]$ is a number selected uniformally at random (for each $1 \le i \le k$).
2. Send I to each neighbor.
3. For each message $\hat{I} = \{< 1, \hat{x_1} >, < 2, \hat{x_2} >, ..., < k, \hat{x_k} >\}$ received, **do** $I \leftarrow I \setminus \hat{I}$.

The fact that Algorithm 1 is privacy preserving is established by the Theorem below. Its proof can be found in [5]. The contingency factor of the algorithm is $\frac{2\Delta c \cdot \log n}{c \cdot \log n/2} = O(\Delta)$.

▶ **Theorem 7.** *For any vertex $v \in V$, executing algorithm GENERIC-RANDOM-COLORING, the algorithm produces a set of at least $k/2$ valid colors in $O(1)$ rounds, with high probability.*

3.1.3 Generic Algorithm for $O(\Delta^2)$-Coloring of General Graphs

Since currently known deterministic algorithms for $(\Delta+1)$-coloring require at least $\sqrt{\log n}$ rounds, applying the simultaneous execution described above with such an algorithm will lead to relatively poor round complexity. Instead, in this section we generalize a construction of [3][16] which provides $O(\Delta^2)$-coloring in $\log^* n + O(1)$ rounds, in order to directly (i.e. without simultaneous executions) obtain secure algorithm for $O(\Delta^2)$-coloring. We employ a Lemma due to Erdös et al. [9].

▶ **Lemma 8.** *For two integers n and Δ, $n > \Delta \geq 4$, there exists a family $\mathcal{J}$ of n subsets of the set $\{1,...,m\}$, $m = \lceil \Delta^2 \cdot \ln n \rceil$, such that if $F_0, F_1, ..., F_\Delta \in \mathcal{J}$ then $F_0 \nsubseteq \bigcup_{i=1}^{\Delta} F_i$.*

A set system $\mathcal{J}$ which satisfies the above is referred as Δ-*cover-free* set.

Erdös et al. [9] also showed an algebraic construction which satisfies Lemma 8. For two integers n and Δ, using a ground set of size $m = O(\Delta^2 \cdot \log^2 n)$, they construct a family $\mathcal{F}$ of n subsets of the set $\{1,...,m\}$, such that $\mathcal{F}$ is a Δ-cover-free. Linial [16] showed that this construction can be utilized for distributed graph coloring. We construct a slightly different family which also provides multiple uncovered elements in each set:

▶ **Theorem 9.** *For two integers n and Δ, using a ground set of size $m = O(\Delta^2 \cdot \log^2 n)$, there exists a family $\mathcal{F}$ of n subsets of the set $\{1,...,m\}$, such that if $F_0, F_1, ..., F_\Delta \in \mathcal{F}$ then $\left| F_0 \setminus \bigcup_{i=1}^{\Delta} F_i \right| \geq \Delta$.*

The proof of Theorem 9 can be found in [5].

These polynomials provides sets of labels such that if every vertex is assigned to a set, each set has at least Δ values which are not contained in any of its neighbors' sets. An illustration of this construction is provided in [5].

Next, we will use the constructions from [9] and Theorem 9 to devise a generic-algorithm for $O(\Delta^2)$-coloring. Our algorithm is similar to Linial's iterative algorithm ([16]), but instead of getting only one color on the last iteration, we get Δ different possible colors (for each vertex).

Starting with a valid n-coloring for some graph $G = (V, E)$ (the color of each vertex is its ID), we can apply the coloring algorithm from [16] which will turn the n-coloring into an $O(\Delta^2 \log^2 n)$-coloring in a single round. After $\log^* n + O(1)$ rounds we will get an $O(\Delta^2 \log^2 \Delta)$-coloring. For a sufficiently large Δ, it holds that $O(\Delta^2 \log^2 \Delta) \leq (3\Delta)^3$. Hence, in order to further reduce the number of the colors to $O(\Delta^3)$ and get Δ valid optional colors we will use the set system from Theorem 9 to reduce the $O(\Delta^2 \log^2 \Delta)$-coloring to a $q^2 = (3\Delta)^2$-coloring of G such that each vertex has at least Δ valid colors. To conclude, the problem domain is of size $9\Delta^2$. The solution domain contains of at least Δ valid colors. Consequently, the contingency factor is $O(\Delta)$, which is proved to be optimal (see Theorem 4).

3.1.4 Generic Algorithm for p-Defective $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-Coloring of General Graphs

For a graph $G = (V, E)$, the function $\varphi : V \to [\alpha]$ is a valid p-defective α-coloring iff for each vertex $v \in V$, the number of neighbors which have the same color as v is at most p, i.e. $|\{u \in \Gamma(v) \mid \varphi(v) = \varphi(u)\}| \leq p$.

First, we shall find the lower bound of contingency factors for generic algorithms of defective coloring:

▶ **Theorem 10.** *For any p-defective α-coloring problem, there is an infinite family of graphs such that their solution domain must be of size $O\left(\alpha \cdot \frac{p}{\Delta}\right)$ at most and the contingency factor will be at least $\Omega\left(\frac{\Delta}{p}\right)$.*

Proof. Suppose for contradiction that there is a valid solution domain such that every vertex has more than $\alpha \cdot \frac{p}{\Delta}$ valid options. Consider a graph $G = (V, E)$ with a clique $C \subseteq V$ of size $|C| = \Delta + 1$. Each vertex $v \in C$ has Δ neighbors, each of them has $\alpha \cdot \frac{p}{\Delta}$. Since v and all its neighbors are part of a clique, each color can be an optional color of at most p different vertices. It means that there are at least $(\Delta + 1) \cdot (\alpha \cdot \frac{p}{\Delta})/p > \alpha$ colors in the p-defective α-coloring, which is a contradiction. ◀

The results from the previous section can be extended and combined with the results of [4], to achieve generic algorithm for ρ-defective $O\left(\left(\frac{\Delta}{\rho}\right)^2\right)$-coloring which provides a solution domain of size $O\left(\frac{\Delta}{\rho}\right)$. Such an algorithm provides an optimal contingency factor. The proof of the next Theorem can be found in [5].

▶ **Theorem 11.** *Given a graph $G = (V, E)$ ($|V| = n$) with maximum degree Δ, and a fixed parameter $1 \leq p \leq \Delta$, there is a generic algorithm that calculates p-defective $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-coloring with a solution domain of size $O(\Delta/p)$ and a contingency factor of at least $\Omega(\Delta/p)$, in $O(\log^* n)$ rounds.*

The contingency factor is optimal by Theorem 10.

3.2 Generic Algorithm for Network Decomposition

As was mentioned before, network decomposition may be referred as a labeling problem where the cluster IDs are the labels and the clusters assignment is the labeling function. Hence, network decomposition problems are good candidates for generic algorithms. However, considering the term of privacy in the network decomposition problem, different definitions may be suggested. One may suggest a permissive notion where all the members of the same cluster are allowed to share their private data with each other. This permissive notion makes sense since network decomposition is frequently used as a building block in other algorithms (such as coloring or finding MIS) where in the first stage each vertex discovers its cluster's topology, calculates private solution for the entire cluster, and then communicates with other clusters in order to generate an overall solution. However, even on a restrictive notion where each vertex may know only its own cluster assignment, some efficient algorithms may be suggested. Hence, during this work we will use the restrictive notion.

In section 3.1.1 we have shown how multiple simultaneous executions of the same random algorithm can expand the solutions domain and provide a generic algorithm for graph coloring problems. This approach can be adopted in order to expand the solution domain

of network decomposition algorithms, However, since the network decomposition should satisfy certain constraints (namely, the depth of the clusters and the chromatic number of the auxiliary graph), this approach should be implemented carefully. We will use the weak-diameter $(O(\log n), O(\log n))$-network decomposition random algorithm devised by Linial and Saks [17]. This algorithm runs in $O(\log^2 n)$ rounds. Our generic algorithm proceeds as follows. Given a graph $G = (V, E)$, and a positive integer $c > 1$, execute Linial and Saks's algorithm for c times simultaneously, in parallel. Each of the execution will have its own serial number $i \in \{1, ..., c\}$. Let $C_i : V \to \{1, ..., O(\log n)\}$ be a set of labeling functions $(1 \leq i \leq c)$, such that $C_i(v) = j$ iff vertex $v \in V$ was assigned by the i-th execution to cluster $\mathcal{C}_j$. For each vertex $v \in V$ we assign a set of $\log n$ different possible labels $C(v) =< 1, C_1(v) >, ..., < c, C_c(v) >$. Each of the labels will represent a distinct cluster ID. These labels are different since even if the independent executions produced the same cluster assignments, the first parameter on each tuple representing the label will be different since it represents the unique ID of each independent execution.

The clusters assignment described above is a privacy preserving $(O(\log n), O(c \cdot \log n))$-network decomposition. The proof of the following Theorem can be found in [5]

▶ **Theorem 12.** *Given a graph $G = (V, E)$, there is a generic algorithm which calculates weak-diameter $(O(\log n), O(c \cdot \log n))$-network decomposition in $O(\log^2 n)$ rounds. The algorithm produces c valid possible cluster assignments for each vertex $v \in V$.*

Since the size of the problem domain is $O(c \cdot \log n)$ and the solution domain is of size c, the contingency factor is $O(\log n)$.

Usually, it is useful to set the parameters of the network decomposition to be polylogarithmic in n. Hence, it may be useful to set $c = \log n$ and get an $(O(\log n), O(\log^2 n))$-network decomposition. On the other hand, in order to preserve privacy, setting $c = \min(\Delta, \log n)$ is sufficient as it allows each of the Δ neighbors of each vertex to have a different set of possible cluster assignments.

3.3 Generic Algorithms for Forest Decomposition

An *oriented tree* is a directed tree $T = (V, E, r)$ where $r \in V$ is the root vertex, where every vertex $v \in V$ knows the identity of its parent $\pi(v)$ and has an oriented edge $(v, \pi(v))$. An *oriented forest* is such a graph that any of its connected components are oriented trees. Any graph $G = (V, E)$ with maximum degree Δ can be decomposed into a set of Δ edge-disjoint forests $F_1, ..., F_\Delta (F_i = (V_{F_i}, E_{F_i})$ such that $E = \bigcup_{1 \leq i \leq \Delta} E_{F_i}$. The problem of how to decompose a graph into forests can be viewed as a labeling problem where each edge should have a label $1 \leq i \leq \Delta$ that represents the forest F_i which it belongs to. Since in every oriented forest, each vertex has at most 1 parent, each vertex will have at most Δ outgoing edges, each belongs to a different forest. Hence, for each vertex $v \in V$ there are $\binom{\Delta}{\deg_{out}(v)}$ different options to associate edges to different forests. From the edge's point of view, each of the Δ labels is a valid possible label.

Panconesi and Rizzi [19] devised an algorithm for Δ-forest decomposition of a general undirected graph in 2 rounds. Their algorithm can be viewed as two separate algorithms, each of a single round. The first algorithm is a simple yet powerful way to decompose a directed acyclic graph with maximum outgoing degree d into d oriented forests. The second is a a way to turn an undirected graph with maximum degree Δ into a directed acyclic graph with maximum outgoing degree Δ. Each of these algorithms run in a single round. Combining these two algorithms produces a 2 round algorithm for Δ-forest decomposition of any undirected graph with maximum degree Δ.

Next we will describe Panconesi and Rizzi's algorithm for forest decomposition of oriented graphs. We will show how this algorithm can be modified in order to preserve privacy while maintaining a contingency factor of 1. Later, we will show two algorithms which produce a directed acyclic graphs. The first is Panconesi and Rizzi's algorithm for orienting any general undirected graph. The second algorithm (due to [2]) performs an acyclic orientation for a graph with bounded arboricity a such that the maximum outgoing degree is $\lfloor 2+\epsilon \rfloor \cdot a$. The combination of these algorithms yields a Δ-forest decomposition for graphs with maximum degree Δ and $\lfloor 2+\epsilon \rfloor \cdot a$-forest decomposition for graphs with bounded arboricity a. Both of them fit the constraint of preserving privacy.

3.3.1 Forest Decomposition of Oriented Graphs

Given a directed acyclic graph $G = (V, E)$, such that each vertex has a set of outgoing edges $E(v) = \{(v, u) \mid (v, u) \in E\}$ the single round algorithm of Panconesi and Rizzi [19] goes as follows. Each vertex $v \in V$, in parallel, assigns a distinct number $1 \leq i \leq |E(v)|$ to each $e \in E(v)$. Let $\hat{E}_i$ be the set of all edges that were assigned with the number i. The forest decomposition is the set of forests $F_1, ..., F_\Delta$ where $F_i = (V, \hat{E}_i)$. The correctness of the algorithm was proved by [19]. While in the original algorithm the nodes do not assign the labels randomly, in our algorithm a random assignment is required. Next, we analyze the privacy of the algorithm. The proof of the following Theorem can be found in [5].

▶ **Theorem 13.** *Panconesi and Rizzi's forest decomposition algorithm with random label assignment is privacy preserving and it has a contingency factor of 1.*

3.3.2 Acyclic Orientation of Graphs

Panconesi and Rizzi [19] showed that the simple orientation where each edge is oriented towards the vertex with the higher ID, is an acyclic orientation. Hence, any undirected graph can achieve an acyclic orientation in a single round, and can be decomposed privately into Δ forests in one additional round. This orientation provides each vertex with up to $\Delta!$ valid options for forest assignments. From the edge's point of view, each of the Δ labels is a valid possible label. Barenboim et al. [2] devised an $O(\log n)$-rounds algorithm that receives a graph with bounded arboricity a and performs an acyclic orientation with maximum outgoing degree of $\lfloor 2+\epsilon \rfloor \cdot a$. This orientation is achieved by partitioning the vertices of a graph G into $l = \lfloor \frac{2}{\epsilon} \log n \rfloor$ sets $H_1, ..., H_l$ such that each vertex $v \in H_i (i \in \{1, ..., l\})$ has at most $(2+\epsilon) \cdot a$ neighbors in $\cup_{j=i}^{l} H_j$. Then, the orientation is done such that each edge $(u, v) \in E$ with endpoints $u \in H_i$ and $v \in H_j$, points towards the vertex that belongs to the higher ranked set (in case the two endpoints belong to the same set, the edge will point towards the vertex with the higher ID). This orientation provides each vertrex with up to $\binom{(\lfloor 2+\epsilon \rfloor \cdot a)}{|E(v)|}$ valid options for forest assignments. From the edge's point of view, each of the $\lfloor 2+\epsilon \rfloor \cdot a$ labels is a valid possible label.

3.4 Generic Algorithms for Graph Coloring of Graphs With Bounded Arboricity a

The forest decomposition algorithms that was described above can be used as building blocks for other distributed algorithms for classic graph theory problems as graph coloring. In the following chapter we will use the $\lfloor 2+\epsilon \rfloor \cdot a$-forest decomposition of [2] that we showed in the previous chapter to achieve an $2a \cdot c \cdot \log n$-coloring for graphs with bounded arboricity a (for any $c > 1$). We will show that this coloring is private and has contingency factor of $O(a)$.

Combining the GENERIC-RANDOM-COLORING algorithm (algorithm 1) with the acyclic orientation algorithm devised by [2] yields a secure algorithm for generic $2a \cdot c \cdot \log n$-Coloring for graphs with bounded arboricity a such that from initial selection of $k = c \cdot \log n$ initial colors (for any $c > 1$), each vertex has, at the end of the execution, at least $k/2$ valid optional colors. The algorithm basically performs the original random generic coloring, but it makes advantage of the acyclic orientation to break the symmetry between each pair of neighbors and make sure that only $O(a)$ neighbors constraint the valid residual colors of each vertex.

The algorithm consists of two steps. On the first step, the algorithm performs an acyclic orientation of graph G such that the maximum outgoing degree is $\lfloor 2 + \epsilon \rfloor \cdot a$. The orientation is done by invoking the first two steps of procedure Forests-Decomposition (algorithm 2 in [2]) with graph G and parameter $0 < \epsilon \leq 2$. On the second step the generic coloring is done. Each vertex v chooses independently at random $k = c \cdot \log n$ numbers from the range $[2 \cdot A]$. These choices form a set of optional colors: $I_v = \{< 1, x_1 >, ..., < k, x_k >\}$. Next, each vertex v sends its set of colors I_v to its children (in correspondence to the orientation). Each vertex u which received a set I_v from one of its parents performs $I_u \leftarrow I_u \setminus I_v$.

▶ **Lemma 14.** *The residual set of colors contains at least $k/2$ colors.*

The proof of this Lemma can be found in [5].

▶ **Lemma 15.** *For any vertex $v \in V$, there is no color $< i, x_i >$ in the residual available colors set I_v such that $< i, x_i > \in \bigcup_{u \in \Gamma(v)} I_u$.*

Proof. Suppose for contradiction that $< i, x_i > \in I_u$ for some $u \in \Gamma(v)$. Let F_j be the forest in $\mathcal{F}$ which includes the edge (u, v). It means that either $u \in \pi_j(v)$ or $v \in \pi_j(u)$, which means that either u or v received it from its parent (v or u, respectively) and should have removed it from its residual set, contradiction. ◀

The time complexity of the algorithm follows from the time complexity of Procedure Forest-Decomposition(a, ϵ), which is $O(\log n)$, plus $O(1)$ for coloring. The size of the problem domain is $2a \cdot c \cdot \log n$ and the size of the solution domain is $\frac{c \cdot \log n}{2}$. Hence, the contingency factor is $O(a)$.

3.5 Generic Algorithms for Edge Coloring

When considering the meaning of privacy in the context of edges, there is a slight difference between vertex coloring and edge coloring. Since the algorithms in both $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ ran on the vertices (rather than the edges, which represents communication lines) the color of each vertex should be known only to the vertex itself. On the other hand, in edge coloring, both edge endpoints are responsible for the coloring of the edge, which means that in terms of privacy preserving we may consider the edge coloring as private when at most the two endpoints of each edge know the color of the edge. However, when the graph is directed, we may demand that only the source endpoint of the edge will be aware of edge's color.

Nevertheless, there is a strong connection between graph vertex coloring to edge coloring. The similarity between the two problems is obvious, but more interestingly, there is a straight reduction between vertex coloring and edge coloring algorithms for general graphs. In the following section we will use this reduction in order to perform privacy preserving generic edge coloring of graphs. This reduction can be used to apply the defective graph coloring we

presented in section 3.1.4 in order to compute a defective edge coloring. There is however, a faster way to get a defective edge coloring. This technique, which is due to [15] will be presented in the later section.

3.5.1 Edge Coloring Using Line Graphs

Given a graph $G = (V, E)$, a *line graph* $L(G)$ of graph G is a graph which is constructed as follows. Each edge $e \in E$ becomes a vertex of the line graph $L(G) = (E, \mathcal{E})$. Each two distinct vertices of the line graph $e_1, e_2 \in E$ are connected ($(e_1, e_2) \in \mathcal{E}$) if they are incident to a single vertex in the original graph, i.e. there exist three vertices $v, u, w \in V$ such that $e_1 = (v, u)$ and $e_2 = (v, w)$. Observe that the line graph has $m \leq n^2$ vertices and a maximum degree of $\Delta(L(G)) = (2\Delta(G) - 1)$. Also observe that a legal vertex coloring in the line graph $L(G)$ is a legal edge coloring in the original graph. Hence, if any vertex of the original graph is responsible for the coloring of part of its incident edges, the vertices can produce a legal graph coloring for the line graph and translate it to a legal edge coloring of the original graph. The assignment of each vertex to any incident edge can be done by specifying that for any edge $(u, v) \in E$, the vertex with the greater ID is responsible for the coloring of the edge in the line graph.

As a result, the algorithms provided in section 3.1 can be applied to the line graph in order to produce a generic edge coloring. Given a graph $G = (V, E)$ with maximum degree Δ, the algorithm for $2\Delta c \cdot \log n$-Coloring, applied on the line graph $L(G)$, produces an $2 \cdot (2\Delta - 1) \cdot c \cdot \log(n^2) = 8\Delta \cdot c \cdot \log n$-edge-coloring, with solution domain of size $c \cdot \log(n^2)/2 = c \cdot \log n$, which yields a contingency factor of $O(\Delta)$. The algorithm for $O(\Delta^2)$-Coloring, applied on the line graph $L(G)$, produces an $O((2\Delta - 1)^2) = O(\Delta^2)$ edge coloring of the original graph, with a solution domain of size $(2\Delta - 1)$, which keeps a contingency factor of $O(\Delta)$.

3.5.2 Generic Defective Edge Coloring

The line graph, which was presented in the previous section, can be used in order to perform a p-defective $O\left(\left(\frac{2\Delta-1}{p}\right)^2\right)$-generic edge coloring of the original graph by applying the algorithm from Theorem 11 on the line graph. Such an algorithm will achieve a solution domain of size $O(\Delta/p)$ and a contingency factor of $O(\Delta/p)$ as well, both in $O(\log^* n)$ rounds. There is however a faster privacy preserving algorithm for p-defective $O\left(\left(\frac{2\Delta-1}{p}\right)^2\right)$-defective coloring based on the algorithm of Kuhn (Algorithm 3 in [15]).

Kuhn's algorithm goes as follows. Suppose we have an undirected graph $G = (V, E)$ with maximum degree Δ, and a constant $i \geq 1$. Each vertex numbers its adjacent edges with numbers between $\{1, ..., \lceil \Delta/i \rceil\}$ such that each number will be assigned to at most i of the vertex's adjacent edges. Then each vertex sends the number of each of its adjacent edges to the vertex on the other endpoint of the edge. Suppose that for a graph $G = (V, E)$, and an edge $(u, v) \in E$, e_u and e_v are the colors that was assigned to edge e by vertex u and v (respectively). The set $\{e_u, e_v\}$ is assigned to be the color of edge e. Kuhn showed that this simple $O(1)$ rounds algorithm achieves a $4i - 2$-defective $\binom{\lceil \Delta/i \rceil + 1}{2}$-Edge Coloring. Setting $p = 4i - 2$ we get an p-defective $O\left(\left(\frac{\Delta}{p}\right)^2\right)$-edge coloring.

Kuhn's algorithm performs communication only between the two endpoints of each edge and only once. Hence the knowledge about the color of each edge is held only by its two endpoints. While in Kuhn's algorithm the nodes do not assign the labels randomly, in our

algorithm a random assignment is required. Therefore, our algorithm preserves privacy. Since every vertex assigns the numbers independently, each of the available colors may be assigned (in certain scenario) to each edge. As a result, we achieve the following theorem.

▶ **Theorem 16.** *There is a privacy preserving algorithm for p-defective* $O\left(\left(\frac{\Delta}{p}\right)^2\right)$*-edge coloring with contingency factor of 1.*

3.6 Generic Algorithm for Edge Dominating Set Colored with $O(\sqrt{\Delta})$ Colors

The problem of constructing an edge domination set, is a bipartition. However, a useful variant of this problem, where the dominating set should be colored with $O(\sqrt{\Delta})$ colors is actually a labeling problem. A privacy preserving algorithm for this problem can be found in [5].

4 Conclusion

The computer-science research fields of secure multi-party computation and distributed algorithms were both highly investigated during the last decades. While both of the fields prosper and yield many theoretical and practical results, the connection between these fields was made only seldom. Nevertheless, as implementation of distributed algorithms becomes common in sensor networks and IoT (Internet of Things) architectures, efficient privacy preserving techniques are essential.

In this work we present a novel approach, which rather than turning existing algorithms into secure ones, identifies and develops those algorithms that are inherently secure. Naturally, our work focuses on labeling problems. The inherently secure algorithms analyzed in this work are listed in

We believe that these results establishes a broad basis for further research of both inherently secure algorithm and efficient techniques to translate distributed algorithms into secure algorithms. Such algorithms will open new possibilities for secure interconnection between machines, eliminating the need to mediate through a central secure server. As a consequence, distributed communication would possibly open a free and secure way to transmit data and solve problems.

References

1. Baruch Awerbuch, Michael Luby, Andrew V Goldberg, and Serge A Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science*, pages 364–369. IEEE, 1989.
2. Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.
3. Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.
4. Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed (δ+1)-coloring in linear (in δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2014.
5. Leonid Barenboim and Harel Levin. Secured distributed algorithms without hardness assumptions, 2020. `arXiv:2011.07863`.
6. Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation. In *Theory of Cryptography Conference*, pages 356–376. Springer, 2013.

7 Nishanth Chandran, Juan Garay, and Rafail Ostrovsky. Improved fault tolerance and secure computation on sparse networks. In *International Colloquium on Automata, Languages, and Programming*, pages 249–260. Springer, 2010.

8 Richard Cole and Uzi Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 206–219. ACM, 1986.

9 Paul Erdös, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union ofr others. *Israel Journal of Mathematics*, 51(1):79–89, 1985.

10 Juan A Garay and Rafail Ostrovsky. Almost-everywhere secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 307–323. Springer, 2008.

11 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proc. of the 19th Symp. on Theory of Computing*, pages 218–229, 1987.

12 Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 181–211. Springer, 2017.

13 Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 157–168. ACM, 2016.

14 Öjvind Johansson. Simple distributed $\delta+$ 1-coloring of graphs. *Information Processing Letters*, 70(5):229–232, 1999.

15 Fabian Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 138–144. ACM, 2009.

16 Nathan Linial. Distributive graph algorithms global solutions from local data. In *Foundations of Computer Science, 1987.*, pages 331–335. IEEE, 1987.

17 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

18 Michael Luby. Removing randomness in parallel computation without a processor penalty. *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

19 Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

20 Merav Parter and Eylon Yogev. Distributed computing made secure: A new cycle cover theorem. *arXiv preprint arXiv:1712.01139*, 2017.

21 Merav Parter and Eylon Yogev. Secure distributed computing made (nearly) optimal. In *Proc. of 38th Symp. on Principles of Distributed Computing*, pages 107–116, 2019.

22 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 350–363, 2020.

23 Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08.*, pages 160–164. IEEE, 1982.