Multi-level models for data security in networks and in the Internet of things

Luigi Logrippo
Université du Québec en Outaouais, University of Ottawa
luigi@uqo.ca

Abstract. Data flow control for security is a mature research area in computer security, and its established results can be adapted to the newer area of data security in the Internet of things or the Cloud. This paper takes a fundamental approach to the problem. It shows that, under reflexivity and transitivity assumptions, *any* network of communicating entities can be seen as a *partial order* of equivalence classes of entities, which is a simplification and generalization of current theory based on the *lattice* concept, where lattices are generated by labelling. Networks of communicating entities can be created in many ways, including routing, access control policies (possibly involving labeling), etc. Their intrinsic partial orders are necessary and sufficient for data security, and in any such network entities will have greater or lower secrecy or integrity according to their position in the partial order. It is shown how labeling systems, capable of expressing many types of security requirements, can be constructed to assign entities to their appropriate positions in network partial orders. Established paradigms in data security, such as conflicts, conglomeration, aggregation, are introduced in examples. Then it is shown how entities can be added, removed or relocated in partial orders, as a result of events such as user or administrative action. A label-based method is described to maintain security requirements through such transformations. Efficient algorithms exist to implement these concepts, they are applications of transitive closure algorithms and strongly connected component algorithms.

Keywords: Data flow control; data security; data secrecy; data confidentiality; data integrity; multi-layer systems; mandatory access control; security labeling systems; Chinese-wall; Brewer-Nash; data aggregation; data conglomeration, data provenance, Internet of things.

1. Introduction

Data networks are defined in this paper as networks of *entities* that can model subjects, users, processes, or 'things' in the Internet of things (IoT), or, more in general, any entities that can hold data, send or receive them through data *channels,* which can be any means by which data can be transferred. A reflexive, transitive *CanFlow (CF)* relation models the fact that data can reach certain entities in a network from certain others through channels, directly or indirectly.

Various data security questions can be asked on the basis of these definitions, namely, given an entity x and a network N:

- the *secrecy* question asks what entities of N the data of x can reach (secrecy is also called *confidentiality* in the literature);
- the *integrity* question asks what data of entities of N can reach x.

Closely related questions are:

- how to classify entities in networks by levels of secrecy and integrity
- how to control the flow of data in networks, so that given secrecy or integrity policies or requirements, concerning the reach of data or the secrecy levels, are respected.

We consider also network *state changes* or *transformations*, which result from the introduction of new entities, removal of entities, or relocation of entities, possibly resulting from administrative, policy or user decisions that change the *CF* relation. Such transformations are shown to lead from partial orders to partial orders, and thus the same principles continue to apply through them.

First submitted March 2020, accepted by the Journal of Information Security and Applications (Elsevier) Jan. 2021.

Following the view that "the fundamental nature of a privacy violation is an improper information flow"(Landwehr [34]), secrecy policies can address at least some *privacy* requirements.

Today's prevailing theory on data flow security was established by Denning in 1976 [17] and is based on the principle that the *CF* relation must form a *lattice of security levels* in order to be able to define and satisfy secrecy and integrity requirements. We show in this paper that a simpler and more general theory can be based on the more general concept of *partial order*. *Any* network can be seen as a partial order of equivalence classes of entities, where data secrecy increases as we go up in the partial order, and data integrity increases as we go down. On this basis, solutions can be found for the questions above.

Further, we show that efficient algorithms exist, to find answers to the security and integrity questions, as well as to implement related policies.

Our method considers both networks defined as entities communicating through channels, and networks defined by associating security labels to entities. We show that the two methods are essentially equivalent.

Although ours is a theoretical model, we show by examples that it can be considered for practical applications and we propose it for further development in areas where the lattice model has been considered to be too restrictive, including the new area of data security in the IoT.

After the literature review in Sect. 2, Sect. 3 presents our basic mathematical concept, the partial order of equivalence classes of entities. Our Property 1, showing the relation between the said partial order and the data flow direction, holds in any reflexive, transitive network (Sect. 4). We then show how basic secrecy and integrity requirements can be defined in terms of data flows and partial orders (Sect. 5); a method is given to design networks that satisfy such requirements. When entity labeling is brought in in our model (Sect. 6), we will see that secrecy and integrity requirements, together with related requirements such as conflicts, conglomerates, aggregates, can be represented by using composite labels that determine the position of entities in partial orders. Several examples are introduced here, together with the concept of 'canonical labeling' which is related to an elementary notion of 'provenance'. One of the purposes of this section is to confirm by using examples that our new theory is consistent with, but also more general than, established theory, and can deal with classical data security problems. Sect. 7 presents our theory of network transformations. After the basic definitions and result in Sect. 7.1, we will see in Sect. 7.2 how data can flow through entities in transformations. Sect.7.3 presents a label-based, IoT inspired example, showing that security requirements can be maintained through transformations by allowing only certain labels. The algorithms that can be used to support this theory are reviewed in Sect. 8. Sect. 9 is a discussion of implications and impacts.

2. Previous work

The literature in this subject is extensive, in the order of hundreds of significant papers. It appeared starting in the 1970s, and especially in the 1980s and 90s. Much of it is now textbook material (Bishop [13 Chapter 5]). This literature cannot be cited here properly, and so we limit ourselves to reviewing the papers that have most directly influenced our work. At that time, the main method for data flow security in organizations was Mandatory access control (MAC) (Sandhu [51]) implemented by Multi-level access control methods (MLS) based on subject and object labeling. These methods were found to be applicable also to operating systems and, to some extent, to programming languages. However MLS methods were considered to be too restrictive and their usefulness was considered to be limited to high-security applications, such as the military, and so interest in them waned. We show in our papers [36,37] that the applicability of MLS, if defined in terms of partial orders, is general and we use the early literature as the basis of our work. More recently, interest in this topic has been revived because of the issue of data flow security in the Cloud and, most recently, in the Internet of things (IoT). Several basic concepts are shared among these topics, see Singh et al. [55], Botta et al. [14].

Bell and La Padula [8] developed a fairly complex theory of data flow security based on partial orders of subjects and objects, system states, and state transitions caused by rules triggered by data access requests. Denning [17] simplified and generalized this and other early models by proposing a theory based on lattices of security classes (or labels) assigned to processes and objects, and on restricting data flows according to the order relations in the lattices. Our work belongs to her line of thought, except that we show that the concept of partial order is sufficient to develop a data flow security theory, independent of labeling.

Foley is the author who has most extensively written on data flow policies. Some of his papers on the subject are references [20 to 25], with [20,21] being two research reports summarizing his early research. These papers include much useful theory, examples and discussion that have inspired our work. Using the lattice model with labelling, Foley considered both intransitive data flow relations, noted $\rightsquigarrow$, and transitive ones, noted $\mapsto$. Foley also considered many types of specialized data security requirements, a few of which will be considered here, especially in Sect. 6.

Sandhu [51,50] presented the applicability of the lattice-based model with labelling for information flow security, considering secrecy, integrity and conflict requirements. He demonstrated the use of his model for defining the related data flow security policies. In this paper, we generalize his concepts by showing that they can be expressed in terms of partial orders rather than lattices.

This body of work remains to be revisited in the context of the IoT and the Cloud.

Coming to our times, although the literature on security in the IoT and the Cloud has been abundant, few papers address the specific problems of data security, i.e. secrecy and integrity in those contexts. Several authors propose the use of variants of Role-based access control (RBAC)(Ferraiolo et al. [19]) or Attribute-based access control (ABAC) (Hu et al. [30]). These methods address access control and not data flow control directly, and for this reason we do not review this literature. Data flow security is the main objective of this paper, because for real data security it is necessary to control where data can end up eventually (Denning [17]).

An interactive tool for model-based security engineering, including, among many other functionalities, the capability of doing data flow analysis, was reported by Amthor et al. [5]. This tool makes it possible to analyze real Unix-like access control systems. It does the transitive closure of reading and writing authorizations. It does have the concept of equivalence classes of entities and applies graph reduction methods. It implements efficient graph algorithms searching for paths according to specific analysis needs. We improve on this work with our theoretical foundation on partial order theory and with the use of strongly connected components algorithms, in addition to considering network state changes.

Bacon and her team [6,45,46,55] correctly insist on the importance of data flow control in the Cloud and the IoT. They present a IFC (information flow control) architecture and a tool, *CamFlow* (Cambridge flow control architecture), that implements it, including support for application management and auditing, through operating-system and middleware support. Their work is based on labeling methods to represent both secrecy and integrity requirements and includes methods for label upgrades and downgrades. The concepts presented here are consistent with their results but constitute a more general theory of data flow security that they could use.

Khobragade et al. [33] present a method for data flow security in the IoT based on the lattice model, and concepts of readers and writers. Related research by the same group is a paper by Narendra Kumar and Shyamasundar [42], where a method is developed to generate security labels in terms of the desired information flow policies. Our method is consistent with theirs, but simpler and more general, in fact their examples are easily resolved in our formalism.

Al-Haj and Aziz [4] propose a data-base method for configuring Software Defined Networks (SDN) with data flow paths that comply with given secrecy and integrity policies. We conjecture that a similar method can be used to implement the method proposed here.

This paper should be read with some background knowledge of previous work by the author and collaborators. Briefly, paper [36] uses elementary graph theory to show that partial orders and multi-level models are necessary and sufficient models for designing data flow control systems in networks that can be represented by directed graphs. Paper [56] shows, by theory and simulation, that efficient algorithms exist to obtain partial order models for given access control matrices or Role-based access control (RBAC) [19] permission lists; as a consequence, simple Label-based access control models can be obtained for any network with access control rules to data that can be specified (or translated into) access control matrices. These two papers use the concepts of subjects and objects and the *CanRead* and *CanWrite* relations [35]. Paper [37] unifies the concepts of subjects and objects into the concept of entity, and the *CanRead* and *CanWrite* relations into the *CanFlow* relation. It shows, by using two examples, how the partial order this last relation implies can be used in order to design multi-layer secure networks in the IoT.

This paper includes a synthesis of the work cited in the previous paragraph and adds several results and examples, mostly in Sect. 5, 6 and 7.

3. Preorders, partial orders and Schröder's theorem

A *preorder* (also known as *quasi-order)* is a transitive, reflexive relation. A *partial order* is a transitive, reflexive, antisymmetric relation. An *equivalence* is a transitive, reflexive, symmetric relation. Binary relations are represented here as digraphs, using bidirectional arrows where relations are symmetric. Partial orders are represented as directed acyclic graphs (DAGs) although by definition these should be antireflexive, as well as antisymmetric. To reduce cluttering, usually graphs will be shown transitively reduced and without self-loops.

A crucial role will be played in this work by a basic result well known in order theory, relation theory and graph theory. Since this result is attributed to Ernst Schröder by Birkhoff [12 footnote to Th. 3], we call it *Schröder's theorem*. We give here an informal account of its proofs, in the two theories that will be used in this paper.

In order or relation theory, given a preorder relation $\lesssim$ over a set, consider the set of equivalence classes generated by the symmetries in $\lesssim$. These equivalence classes are in a partial order relation (since the symmetries have been encapsulated in equivalence classes).

Further, let us denote this partial order relation with the symbol $\sqsubseteq$, and let *[x]* be the equivalence class of set elements equivalent to *x* under $\lesssim$: we have $x \lesssim y$ iff $[x] \sqsubseteq [y]$ (Birkhoff [13 Th. 3], Fraïssé [26 Sect. 2.2]).

Similarly, **in digraph theory**, if $\lesssim$ is represented by the edges of a digraph, then consider the maximally strongly connected subgraphs of this digraph (the *components*) and condense each of them in a single node. The resulting *condensed digraph* is acyclic. There is a directed path between nodes in two components in the original digraph iff there is a directed path between the two components in the condensed digraph (Harary et al. [29 Chapt.3], Bang-Jensen, Gutin [7 Sect. 1.5]). Components correspond to equivalence classes and paths in the compressed digraph represent the relation $\sqsubseteq$.

There are algorithms which, given a partial order or digraph, can calculate the partial order $\sqsubseteq$ or the corresponding condensed digraph, see Sect. 8. These are called *strongly connected components algorithms*. Since the set of equivalence classes in a preorder is uniquely defined, the result of these algorithms is uniquely defined.

We write $[x] \equiv [y]$ if $[x] \sqsubseteq [y]$ and $[y] \sqsubseteq [x]$. We write $[x] \sqsubset [y]$ if $[x] \sqsubseteq [y]$ but $[y] \sqsubseteq [x]$ is false. Comparing set elements in partial orders, we use the term *level*, where $level(x) < level(y)$ iff $[x] \sqsubset [y]$. When partial orders are represented by digraphs, then $level(x) < level(y)$ iff there is a directed path from node *[x]* to node *[y]* in the condensed graph. We say that *[x]* is *maximal* in a partial order (a *sink*) if for all *[y]*, $[x] \sqsubseteq [y]$ implies $[x] \equiv [y]$ and is *minimal* (a *source*) if for all *[y]*, $[y] \sqsubseteq [x]$ implies $[x] \equiv [y]$. Following the established terminology for MLS models (Bishop [13 Sect. 5.2.1]) we write that equivalence class *[y] dominates* equivalence class *[x]* iff $[x] \sqsubseteq [y]$ and in this case we also say that entity *y dominates* entity *x*.

The notion of lattice is not used in our theory, but is often mentioned here for comparison purposes. A *lattice* is a partial order where for every two elements *x* and *y* there are:

- A unique element *z* (the *join*) such that $x \sqsubseteq z$ and $y \sqsubseteq z$
- A unique element *u* (the *meet*) such that $u \sqsubseteq x$ and $u \sqsubseteq y$

A review of the literature on lattice models for data security shows that, although the examples given are lattices, the existence of unique joins and meets is seldom used in the reasoning.

There will be some mention of *efficient algorithms* in this paper. This term will be taken in its usual meaning in complexity theory: algorithms that run in linear or polynomial time are considered to be efficient, see Aho, Ullman [3 Introduction].

4. Networks, preorders and partial orders

Definition 1: A *network N* is a finite set of *entities* with a binary relation *Channel*.

Thus a network can be fully defined by a Boolean matrix. Variables for entities will be written with the letters *x, y, z…*, constants (used in the examples) by names with upper-case initials. In the first part of this paper, networks are considered to be fixed at a given *state* (Bishop [13 Sect. 2.1]).

The primitive concept of *entity* is generic and can be used, according to modeling needs, to model any entity capable of containing and transmitting data, by way of sending them, writing them, or being read, and for which access or data flow can be controlled at its level of granularity: subjects, objects, users, roles, hardware or software components such as databases or sensors, or also folders, files or data sets that can contain data of certain types.

The primitive relation *Channel(x,y)* intuitively expresses the fact that, in the network under consideration, there is a relation from entity *x* to entity *y,* expressing the possibility that data can be directly transmitted from *x* to *y*, because of the existence of data transfer authorizations or of communication links, e.g.:

1. Read and write authorizations that can be expressed in Access control matrices (Bishop [13 Chapter 2]) or capability lists (Stambouli and Logrippo [56]) or RBAC permission lists (Ferraiolo et al. [19 Sect. 3.2.2], Osborn [43]) can be represented as *Channel* relations. If, in a given state, entity (subject) *x* can read or directly receive data from entity (object) *y*, then *Channel(y,x)* can be taken to be true in the network representing that state, while *Channel(x,y)* can be taken to be true if *x* can write on, or directly send data to *y.*
2. In particular, in access control systems of the Multi-level security family (MLS) based on labelling, the sets of entities (normally subjects and objects) are mapped into partially ordered sets of *labels*. For a given mapping, access control authorizations, thus *Channel* relations, between entities exist according to the ordering of labels. Conventionally, the direction of channels is towards the top of the partial order. There are channels from entities at a secrecy level to entities at the same or higher level, but not in the opposite direction. Similarly, in integrity models (Biba [11]) there are channels from entities at a certain level of integrity to entities at an equal or lower level of integrity, however we shall show that only one direction needs to be considered, see Sandhu [51].
3. Distributed networks have routing relations that can be defined as *Channel* relations in our sense. *Software defined networks (SDN)* (Bera et al. [9], Al-Haj, Aziz [4]) provide an ideal method for implementing the theory presented in this paper, because the routing is controlled by centralized software.
4. *Encryption* can be interpreted as creating channels between entities. For example, it could be said that *Channel(x,y)* is true iff *y* can decrypt what it receives from *x*.
5. *Covert channels* (Bishop [13 Sect. 18.1], Jaskolka [32]) can also be represented as *Channel* relations, if it is desired to examine the effects of their possible existence on data secrecy and integrity in a network.
6. Communication in *social networks*: for example, a user can have a folder of photos defined as an entity with channels towards certain friends.
7. Communication possibilities involving entities representing human users can be modeled by *Channel* relations.

Variations and combinations are known of these methods, and the literature is vast.

We assume that, for any given method in any given network state, the *Channel* relation is well defined. In our *network digraphs*, this relation is represented by directed arrows, and bidirectional arrows represent symmetrical relations, see Fig. 1(a).

Definition 2: The *CanFlow* relation for a network, written *CF(x,y),* is the reflexive and transitive closure of the *Channel* relation*.*

Thus *CF* can be fully defined by a Boolean matrix. *CF(x,y)* means that data can flow from entity *x* to entity *y.* In principle, the *Channel* and *CF* relations could be identical, however in practice the first could be more easily determined from the data communication relations mentioned in the points above. Also, at the network design stage, channels may have to be physically implemented, thus may have costs and so their number may have to be kept low;

so a single *CF* relation may be implemented by several *Channel*s, taking advantage of transitivity.

If data can flow in both directions between any two entities in a set of entities, then all the entities in the set can be considered to be equivalent, in the sense that they can contain the same data.

Definition 3: We say that entities *x* and *y* are *data equivalent* if *CF(x,y)* and *CF(y,x).*

We continue to use the notation of Sect. 3. By its reflexivity and transitivity, *CF* is a preorder relation, as the mentioned relation $\lesssim$, and partitions the set of entities into equivalence classes by the data equivalence relation. By *Schröder's theorem* there is a partial order relation between these equivalence classes. Formally:

Property 1: *CF(x,y) iff* $[x]\sqsubseteq[y]$

Proof. Directly by Schröder's theorem and Defs. 2 and 3.

In this sense, in any network data can flow (*CF* relation) *upwards* only, starting from the lowest-level entities that can hold them to the highest-level entities, or from sources to sinks.

Our graphical representation of equivalence classes will be by double-sided rectangles, and arrows between them will represent the $\sqsubseteq$ relation, see Fig.1(b).

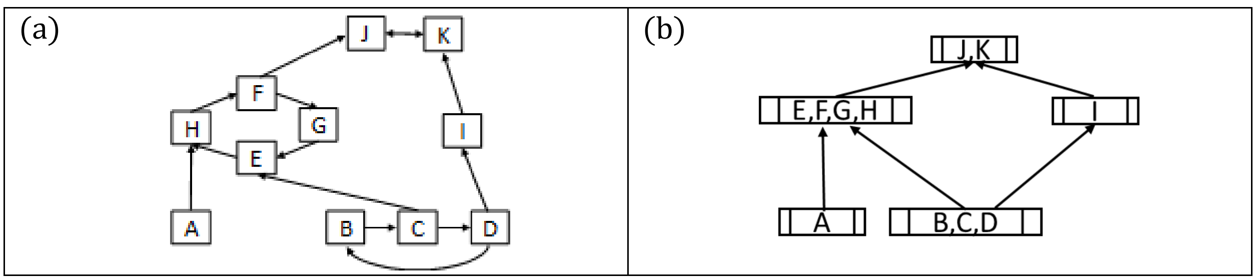

Figure 1 (Ex.1): (a) A network of entities and (b) its partial order of equivalence classes

Example 1. Fig. 1(a) could be read as a IoT network, perhaps implemented by a combination of the methods mentioned above, with two classes of sensors: Sensor *A* and sensors *B,C,D*. The first works alone, but the last three work together, perhaps to complement each other's data. The nodes above might represent various types of processing nodes, data bases or whatever. We see here several structural concerns, some of which could be:

- Initially, the data from *A* and *B,C,D* should be kept separate;
- *I* is supposed to work only on *B,C,D*'s data while *E,F,G,H* use data from all sensors;
- *J,K* are the final users of the data gathered.

The partial order or condensed digraph of Fig. 1(b) can be obtained from the network of Fig. 1(a) by executing a strongly connected component algorithm.

To go from a partial order such as (b) to a *Channel* relation or a graph such as (a) different methods can be used, such as:

Partial order implementation Method 1:

1) According to **Property 1,** define a *CF* relation between entities *x* and *y* such that $[x]\sqsubseteq[y]$.
2) This *CF* relation can be directly used as a *Channel* relation; it can also be transitively reduced to obtain a reduced *Channel* relation.

Partial order implementation Method 2:

1) For each equivalence class in a partial order, create channels between its entities in any way that establishes a *CF* relation between all pairs of them: this results in a set of strongly connected digraphs.
2) For each pair of equivalence classes such as $[x] \sqsubset [y]$, create the additional channels that are sufficient in order to establish a *CF* relation between at least one entity in $[x]$ and one entity in $[y]$.

Method 1 is short and elegant, but may lead to channel configurations that are impossible to implement, given the physical network characteristics (e.g. in the physical network there might be a direct channel from *F* to *J*, as shown in the figure, but it might be impossible to create one between *F* and *K*). So in practice, it may be necessary to take into consideration pre-existing channels, cost and efficiency constraints etc. and this will be possible with method 2. Once the construction is complete, especially if it is done manually, it can be validated by using a strongly connected component algorithm to obtain the partial order of the constructed digraph and to check that it is isomorphic to the initially given partial order digraph.

Canonical label-based networks, defined in Sect. 6, are also implementations of partial orders with label-based access control rules. This implementation method relies on the fact that the $\sqsubseteq$ relation can be represented in terms of labels.

5. Secrecy (confidentiality), integrity and design from requirements

Data security is often defined as having (at least) the two aspects of *secrecy* and *integrity*, see Sandhu [50]. The relation *CF* can be used to express a concept of secrecy, taking *CF(x,y)* to mean that *x is a secret of x* and *y,* or that *x is visible to x* and *y* (equivalently, *y can know x* or *y is in the area of x* in the terminology of Logrippo and Stambouli [36, 56]). Similarly, *CF* can be used to express a basic concept of integrity. If *CF(x,y)* we can say that the *integrity* of *y* can be affected by data from *x*. *Data integrity* in security has had several, but related, definitions (Sandhu and Jajodia [52]). Our definition corresponds to what Sandhu [53] calls *information flow integrity (*or *data flow integrity* in our terminology*)*, for which the main historical reference is Biba [11].

Definition 4: **Secrecy and integrity of entities.** We say that *x is less secret but has more integrity than y* if $[x] \sqsubset [y]$, and we say that *x is more secret and has less integrity than y* if $[y] \sqsubset [x]$.

On this basis, for any network, it is possible to determine:

- What are the most secret and least secret entities: the most secret are those in the maximal equivalence classes or levels in the partial order (the sinks), because data cannot escape from them; the least secret are those in the minimal equivalence classes or levels (the sources), because their data can flow to levels above them.
- What are the entities with the highest and lowest integrity: the highest integrity belongs to the entities in the sources, since no extraneous data can flow into them; the lowest are in the sinks, for the converse reason.

So, by our definitions, the levels of secrecy and integrity are inversely correlated.

By following these principles, some related problems can be solved:

- Given a number of datasets with secrecy and integrity requirements, it is possible to design a network where these are satisfied if the datasets can be placed in a partial order that satisfies the requirements. Using this partial order, a suitable network can be constructed, possibly with one entity per data set. An example, based on slightly different definitions, is in Logrippo [36].

- If a network already exists and the partial order is given, then it must be checked whether the entities and channels in the network can implement the required partial order, or a larger one. If so, the datasets can be placed in the appropriate entities.

For secrecy, this means necessarily placing data in entities that are dominated only by entities that are authorized to get them. For integrity, this means necessarily placing data in entities that dominate only entities that are authorized to receive them. For example, if only two levels are available, then the lower should contain the data that must have higher integrity and lower secrecy, and the upper the data that must have lower integrity and higher secrecy.

Example 2. The network of Fig.1 can be seen as a solution for the following security requirements:

- Secrecy: *A'*s data should only be visible to *E,F,G,H* and *J,K*; *B,C,D'*s data should be visible to all except *A*; *I'*s data should only be visible to *J,K,* and the same holds for the data from *E,F,G,H; J,K'*s data should be top-secret.
- Integrity: *A'*s data*,* as well as data from *B,C,D* should be top-integrity (as appropriate for data coming from sensors); *A'*s data can affect only the integrity of *E,F,G,H* or *J,K,* etc.; *J,K* will by consequence have bottom integrity.

Fig. 2 shows secrecy and integrity diagrams for this network. They can be read as superposed to the digraphs of Figs. 1(a) or (b). All these diagrams contain the same information, however the secrecy diagram shows more clearly where the data originate, while the integrity diagram shows more clearly where they can go. For complex networks, it will be useful to create them.

Other examples can be found in Logrippo, Stambouli [36,37].

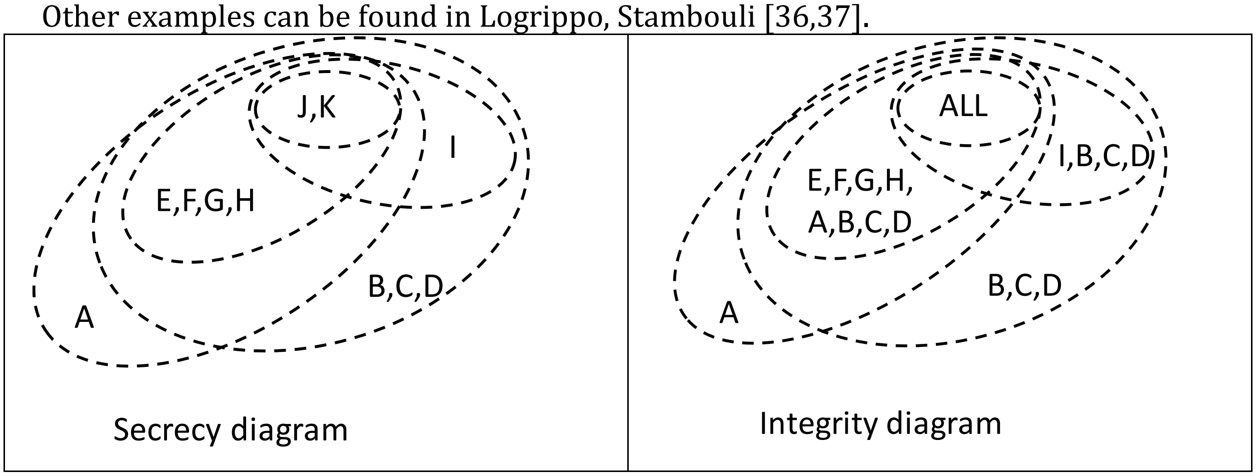

Figure 2 (Ex.2): Secrecy and integrity diagrams for the network of Fig. 1.

6. Label-based access control and requirements

In this section, we illustrate the equivalence between definitions of networks based on channels or on labels. We also show how some concepts well-known in the literature on label-based access control can be expressed in terms of partial orders. At first, this might seem trivial, because lattices are partial orders. However by basing the definitions on partial order of labels instead of lattices of labels the theory is simplified and generalized. None of the examples below are lattice-based.

The use of labels is a time-honored method for assigning entities to security levels. It well predates the Bell-La Padula model [8], since the latter was devised to formalize practices well established in the military and other high-security enterprises such as banks and government. This method is still used, informally and formally, for access control in organizations. Network *requirements* or *policies* define partially ordered sets of labels and the mapping from the set of entities to the set of labels, thus the allowed data flows. Labels are tuples whose elements come from partially ordered *domains*. Domains can be any partially ordered sets. In our examples, according to established data flow security theory (Bishop [13 Sect. 5.2.1], Gollmann [27] Sect. 5.8.4), we consider two types of domains:

- Elementary domains, such as secrecy or integrity levels, with partial order relations understood in security theory
- Category sets, for which the partial order relation is set inclusion.

By the following 'canonical' construction and in Sect. 6 we will see that the former can be reduced to the latter. For now, we define a partial order between labels which is the coordinate wise partial order of the product of the partial orders of the domains.

Definition 5: A *label* is a tuple of elements, each taken from a partially ordered set. A set of labels is *uniform* if all tuples are of the same cardinality and corresponding elements in the tuples are taken from the same domains. A network is said to be *label-based* if for all entities x in the network, the function $Lab(x)$ is defined in the same uniform set of labels.

We assume that in every label-based network and for any two x, y, $Lab(x) \leq Lab(y)$ is decidable, which surely must be the case in practice.

The set of labels, being the product of partial orders, is also a partial order, which induces a partial order on the set of entities. In order to create a relation between the theory based on channels and the theory based on labels, we adopt the following axiom, which is well-established in data flow security theory, see Sects. 3 and 4[1]:

Axiom: In label-based networks, $Channel(x,y)$ *iff* $Lab(x) \leq Lab(y)$

Since the $\leq$ relation is transitively closed, the *Channel* and *CF* relations coincide here, hence $CF(x,y)$ *iff* $Lab(x) \leq Lab(y)$, and $[x] \sqsubseteq [y]$ *iff* $Lab(x) \leq Lab(y)$, $[x] \equiv [y]$ *iff* $Lab(x) = Lab(y)$. If labels are simple category sets, $Lab(x) \leq Lab(y)$ *iff* $Lab(x) \subseteq Lab(y)$, and so $CF(x,y)$ *iff* $Lab(x) \subseteq Lab(y)$.

An entity x will be written with its label as $x{:}\langle a_1,\ldots,a_n,\{y_1,\ldots,y_n\}\rangle$ where $a_1,\ldots a_n$ come from elementary domains and $\{y_1,\ldots,y_n\}$ is a category set (we will only need one such set in our examples); empty elements will not be shown. Graphically, partially ordered label sets will be represented as DAGs with hexagonal nodes. Edges represent the $\leq$ relation between labels, and also the $\sqsubseteq$ relation between equivalence classes of entities with those labels.

Categories can be used to show the *provenance* of the data that can end up in entities. To do this, every source entity is labeled with the set of the names of the entities in its equivalence class, e.g. $x{:}\ \langle\{x,y,z\}\rangle$. Then every entity to which x can flow will contain the names of

[1] In Bishop [13 Sect. 5.2.1] it is said that *"S can read O if and only if $l_o \leq l_s$ and S has discretionary write access to O"*, where l_o and l_s are the labels of *O* and *S* and discretionary access is defined by an access control matrix. A converse rule applies to the *can write* relation. We have shown in [56] that access control matrices can be expressed as labels, and so both parts of the condition can be represented together by combining labels.

x,y,z in its label, see following examples. Note that we use here a very basic concept of provenance; for more complete views, that could be used to refine this concept, see Moreau [40], Park et al. [44], Pasquier et al. [47].

For any network (whether label-based or not), it is possible to construct a label-based network that has the same partial order of equivalence classes, and where labels are sets of entity names, showing the provenance of the data in each entity, as follows:

Definition 6: A *canonical label-based network* is a label-based network where each label is a set of entity names in the network. For an entity x, $y \in Lab(x)$ *iff* $[y] \sqsubseteq [x]$.

Such label-based networks are said to be 'canonical' because they exist and are unique for any network (because of the uniqueness of the partial order of entities in any network). Their labels can be computed efficiently using strongly connected component algorithms, see Sect. 8.

Note that this construction proves that in any network, each entity's incoming and outgoing flows can be defined by using labels that are simple sets. This result will be confirmed by other examples at the end of this section.

Note also that, since any label-based network can be defined in terms of channels (by the Axiom) and since there is a similar correspondence between capability lists or access control lists and label-based networks [56], we have established the equivalence of these different ways of defining networks.

In a canonical label-based network, one can see easily the secrecy and integrity of each entity, since the most secret entities are those whose names are not found in other entities, and the entities that have the most integrity are the ones that have only the names of the entities in their equivalence class in their labels.

Example 3. The canonical label-based network for the network of Fig. 1 is given in Fig. 3. The notation *B,C,D: <{B,C,D}>* in a double-sided rectangle means that entities *B,C,D* are in the same data equivalence class and for each the label is the set *{B,C,D}.* Note the correspondence of this diagram with the integrity diagram of Fig. 2.

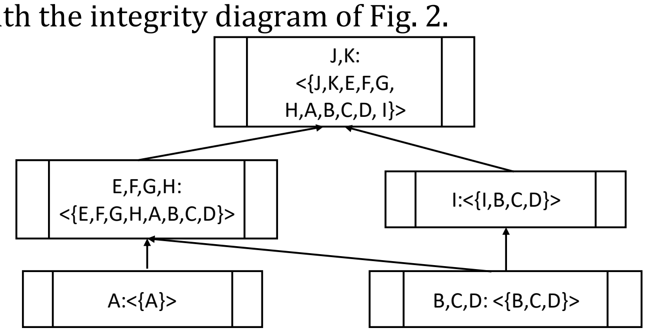

Figure 3 (Ex.3): Canonical label-based network for the network of Fig. 1

The following examples show how requirements specifying conflicts, conglomeration, aggregations, numerical constraints, and combinations of integrity, security and data categories, can be represented with different types of labeling. These are types of requirements that have been studied in the literature on data flow security and have practical applications; hence it is important to show that they can be expressed in our theory.

Example 4: Conflicts. If two data categories are in conflict, then no entity is allowed to contain data from both of them (Sandhu [50],Foley [20,24]). In other words, if data categories *x,y* are in conflict, then the sets of entities for which *x* or *y* are a secret must be disjoint.

This can be expressed using labels with sets showing categories: then no label can be allowed to contain the subset *{x,y}* or, in a practical example of two banks in conflict, *{Bank1,Bank2}*. This could be said to be a ‘static Brewer-Nash or Chinese-wall policy’, whereas the real Chinese-wall policy is of a dynamic type, having been devised in order to prevent reaching such labels as a result of network transformations, this will be demonstrated in Ex. 11. As a variation, it can be specified that there is no conflict in the presence of other categories, e.g. *{Bank1,Bank2,CentralBank}* could be allowed. One practical situation for this is the case where the Central Bank has to investigate possible collusion between the two banks, then some employees of the Central Bank may have to be assigned this label.

Example 5: Conglomeration. We have conglomerates when several combinations of data categories should be considered to be bound together, in the sense that if one of them is part of a flow, then the others must be included also. This was considered in Foley [22,24]. In our model, conglomerates can be taken care of by the opposite mechanism as conflicts, i.e. by the requirement that whenever an entity name appears in a label, then all its conglomerates should also appear. For example, if entities *Company1* and *Company2* are the two parts of a conglomerate, then each of them could be labeled *<{Com1,Com2}>* and this pair should appear together in all labels. Another type of conglomeration is the asymmetrical one where *Company1* can appear alone, while *Company2* must appear in combination with *Company1*, then *Company1* could be labeled *<{Com1}>* while *Company2* could be labeled *<{Com1,Com2}>.* Flow is allowed from the first to the second, but not vice-versa. This can be useful if *Company2* controls *Company1*, see Ex. 11.

Example 6: Aggregation. With aggregation it is possible to specify that certain combinations of data categories have higher secrecy classification than others; usually this is a consequence of the fact that certain inferences are possible with those combinations (Sandhu, Jajodia [52],Foley [22,24],Meadows [39], Cuppens [16]). The following example shows how aggregations can be specified with labels. Consider a network with five levels of secrecy*: Public (P), Unclassified (U), Confidential (C), Secret (S), TopSecret (T),* with $P<U<C<S<T$. There are three data categories, *X, Y, Z*. Taken by itself, each data category is at secrecy level *P*. However *{X,Y}* is *U*, *{Y,Z}* is *C*, *{X,Z}* is *S* and *{X,Y,Z}* is *T*. The label set and partial order corresponding to these classifications are shown in Fig. 4.

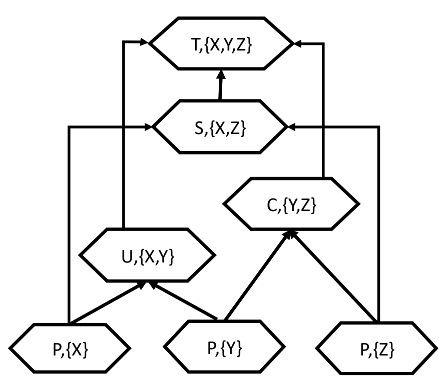

Figure 4 (Ex.6): Partially ordered label set for aggregation example

Example 7: Cardinality requirements. Typical is a requirement that no entity should have in its label more than *n* different categories (Foley [22]). This can be immediately implemented.

Example 8: Labels with both secrecy and integrity levels. For secrecy we use two classifications: *Public, Secret*, abbreviated *P* and *S* with *P<S*. For integrity, we have three classifications: *I1, I2, I3*. *I1* is the highest integrity level while *I2* and *I3* are lower but are mutually incomparable. Flow is allowed from high to low integrity level, thus we have: *I1<I2* and *I1< I3.* The set of labels for this example is the product of these two partial orders. Fig. 5 shows all possible labels for these policies. So each entity in the network will have a label indicating its secrecy and integrity level and the partial order shown in the figure describes the data flow relationships between the entities. A network's policies could use only some of the six labels.

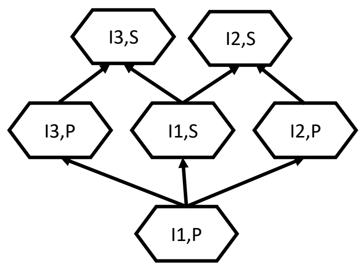

Figure 5 (Ex.8): Partially ordered label set for combined secrecy and integrity example

Example 9: Labels with secrecy, integrity and categories. We consider here a combination of secrecy and integrity requirements with requirements on data categories. Fig. 6 shows six entities with their labels. We consider three secrecy levels ordered *Public<Classified<Secret* and two integrity levels ordered *Certified<Generic.* We also have three data categories, *Financial, Medical, Other* (we use some obvious abbreviations henceforth). Labels can contain subsets of this set, according to the allowed content of the corresponding entities. The set of all possible labels has 48 elements, and we will not show it. We only show the equivalence classes of the entities in the table with their data flows. The equivalence classes have only one element, with the exception of the equivalence class of the two elements *E1* and *E6*, both mapping on the label *<Pub,Gen,{Fin}>*. For example, we see that data of category *Oth* are visible only to (are a secret of) entities *E3* and *E5*. However *E3* can only get *Pub* or *Cert* data of *Oth* category, while *E5* can get *Pub* or *Clas* data, also *Cert* or *Gen* data of *Oth* category. Since the set *{Med,Oth}* does not appear, these two categories could be in conflict.

Entity	Secrecy Level	Integrity Level	Data categories
E1	Pub	Gen	Fin
E2	Sec	Cert	Fin, Med
E3	Pub	Cert	Oth
E4	Clas	Cert	Fin
E5	Clas	Gen	Fin, Oth
E6	Pub	Gen	Fin

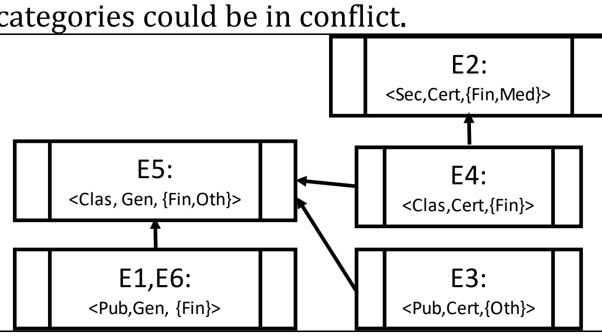

Figure 6 (Ex.9): Labels and data flows for secrecy, integrity and categories

Much more sophisticated options are possible, because of the many possibilities of label purposes and combinations. Going back to Ex. 4, for *Bank1* two entities could be created, one with *Bank1*'s secret data, not to be shared with anyone, and one with *Bank1*'s public data, to be shared with all other entities, including *Bank2*. A more complicated scheme would be to split a bank in three entities: one secret, one confidential for data to share with allied companies, and one public. The confidential level could be split further in several sub-levels, one for each collaborating company, and so on. Each of these possibilities yields a partial order of labels. Labeling can be complex, allowing to express complex policies.

The product of different partial orders used above can be uniformed to only one partially ordered set by using only the concept of category. In Ex. 9 one can replace 'secrecy levels' by 'secrecy categories', with *{Pub}⊂{Pub,Clas}⊂{Pub,Clas,Sec}*. For integrity, we have *{Cert}⊂{Cert,Gen}*. This is justified because *Classified* entities can also contain *Public* data and so on, and similarly for integrity. Then we have a product of three partial orders that are all sets of data categories with inclusion. These can be merged into one single partial order of sets of categories with inclusion. For example, the label for *E2* becomes *<{Pub,Clas,Sec,Cert,Fin,Med}>*. These new labels yield the same partial order as the one in Fig. 6.

As a second example for the reader to check, take the example of Fig. 5 with the following substitutions: *{P}* for *P*, *{P,S}* for *S*, *{I1}* for *I1*, *{I1,I2}* for *I2*, *{I1,I3}* for *I3*. The resulting partial order is isomorphic with the one of Fig. 5.

The construction of canonical label-based networks and the construction above are different, but they agree on the fact that, for the data security model presented here, labels consisting of simple category sets are sufficient. Then data flow control rules or access control rules can be reduced to simple inclusion relations between category sets. This fact is a simplification of established theory, as described in (Bishop [13 Chap. 5], Gollmann [27 Sect. 5.8.4]).

Note that, in the examples above, the use of the lattice model would require adding unnecessary and even unwanted entities and labels to serve as meets and joins, such as labels containing conflicting data categories (Ex. 4, where a label containing *{Bank1,Bank2}* would be necessary) or incomparable ones (Ex. 8, where a label containing *{I2,I3}* would be necessary).

Label-based requirements can be enforced through network transformations by ensuring that only allowed labels are reachable, we will see this in Sect. 7.3.

7. State changes or transformations in networks

7.1 Addition, removal, relocation of entities

In this section, we show that in our theory it is possible to describe fairly naturally state changes, i.e. state transitions or *transformations* (Bishop [13 Sect. 2.3 and 5.2.3]). These are changes in the *Channel* relation, that can occur in networks for a variety of events, such as user or administrative action (i.e. for changing labels or levels of integrity or security), or changes of environmental variables including time- or location- dependent ones, according to policies. There is abundant literature showing that state changes can lead to security breaches, not only in Discretionary access control models, but also in MLS. A controversy on the Bell-La Padula model led to the definition of 'tranquility principle' [13 Sect. 5.3]. In general terms, we only say that, for any network, rigorous policies and auditing must exist, establishing mechanisms and prerequisites for safe transformations. Data purging, an important mechanism, will be mentioned in Sects. 7.2 and 7.3.

We consider an unbounded set of network *states* N_0, N_1 ... Each N_i is a network with its entities x_i, y_i ,..., $Channel_i$ or Lab_i relation or function, and the derived CF_i and $\sqsubseteq_i$ relations. So entities keep their names through state changes but their indexes represent the current state. Three transformations are considered, as follows:

Definition 7: Network transformations.

We say that

1) x *is created* in N_{i+1} if there is no $[x_i]$ but there is $[x_{i+1}]$
2) x *is removed* in N_{i+1} if there is an $[x_i]$ but no $[x_{i+1}]$
3) x *is relocated* in N_{i+1} with respect to N_i if there are $[x_i]$,$[x_{i+1}]$,$[y_i]$, $[y_{i+1}]$ such that the relation between $[x_i]$ and $[y_i]$ is different as the relation between $[x_{i+1}]$ and $[y_{i+1}]$.

In Ex. 10 we will see relocations caused by changes of the *Channel* relation, while in Ex. 11 we will see the creation of entities with known labels and label changes, leading to relocations. By extension of Property 1, within each network we can only have 'upward' data flows in the network's partial order:

Property 2: For all i, $CF_i(x_i,y_i)$ *iff* $[x_i] \sqsubseteq_i [y_i]$.

Proof: The proof of Property 1 holds for each N_i.

So, since the result of each transformation is a network, each transformation yields a partial order of equivalence classes of entities, with different *Channel, CF* and $\sqsubseteq$ relations.

Note that if lattices instead of partial orders are used to model networks and transformations, the transformation of a lattice will not necessarily yield a lattice, and then the lattice structure will have to be recovered in some way. For example, suppose that we have a network with only two entities, x and y, with $CF(x,y)$, and suppose that we create z such that $CF(x,z)$ but without any flow between y and z. Nothing else needs to be done in our method, since both the initial and the resulting networks are partial orders. From the point of view of lattice theory, however, the initial network was a lattice but the resulting network isn't, so something must be done. In this case, the simplest way to recover the lattice structure is to create a fictitious entity u such that $CF(y,u)$ and $CF(z,u)$. In general, however, the method is not straightforward, if lattice size is a consideration (Bertet et al. [10]). Such fictitious entities and their related flows are unnecessary in practice and have no reason to exist in our model.

Because of the global nature of partial order relations, creations and removals of entities may cause relocations of other entities. Relocations can also be caused by the creation or removal of channels, since removing channels or flows towards (from) an entity may move it down (up) in the partial order, increasing (decreasing) its integrity but decreasing (increasing) its secrecy, and vice-versa for adding channels or flows. Relocation of one entity causes relocation of others. All this may deserve to be studied within partial order theory, but the above observations are sufficient for this paper.

Example 10. We go back to Fig. 1, and we take it as our N_0. In Fig. 7(c) we have removed all outgoing flows from A and thus we have increased its secrecy to the maximum; however by adding incoming flows to it, its integrity has been decreased to the minimum. The converse has been done for I. So two entities were relocated. Fig. 7(d) presents a possible implementation of this partial order. It was obtained by the *partial order implementation method 2* of Sect. 4, reusing most channels in the initial configurations and adding some. The channel from D to K has become optional because of transitivity. This is our N_i for $i>0$. (Unfortunately this example is not consistent with the interpretation we have given of Fig. 1(a) earlier, but we use it to reduce the number of figures.)

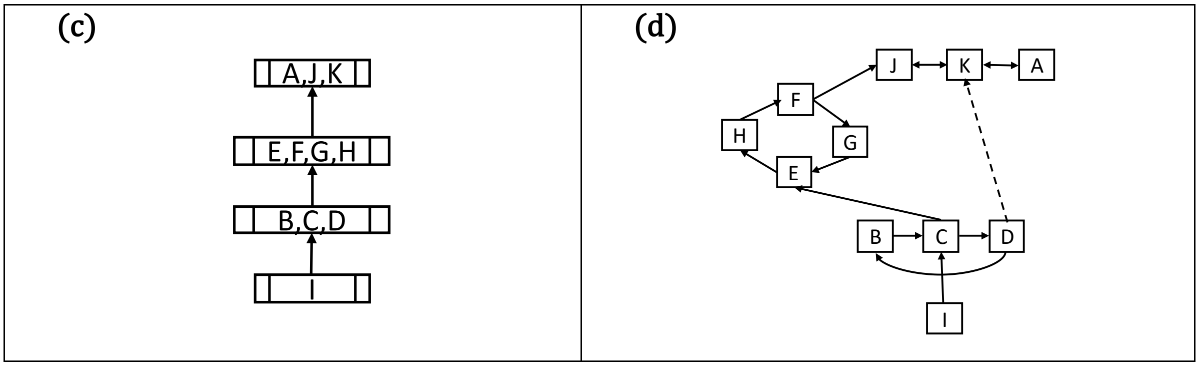

Figure 7 (Ex. 10): Network transformations through partial orders

Note that the figures can be read in the opposite direction, to illustrate transformations from Fig. 7(d) to 1(a).

7.2 Data flows over transformations

We assume that each entity can 'remember' data from a state to the next, thus causing interstate flows. Note that we need not be concerned about entities that have been created, nor about entities that have been removed: the former have no memory of previous data flows and the latter cannot pass it on.

We denote the interstate flow relation as *CFS*. We take this to be a transitive relation, but anti-reflexive and anti-symmetric.

In particular, data can flow between two entities in adjacent states

- if they can flow between the entities in the first state (the destination entity will then remember the data in the next state),
- or if they can flow to an intermediate entity in the first state, from which they can flow to the other one in the next state.

Intuitively, the flow can be direct for an entity by memory, or indirect through other entities that can carry data from a state to the next. Fig. 8 illustrates the two possibilities, one in continuous lines and the other in dashed lines. Thinner lines represent 'memory through states' (relation *CFS*) and thicker lines represent flows within states (relation *CF*).

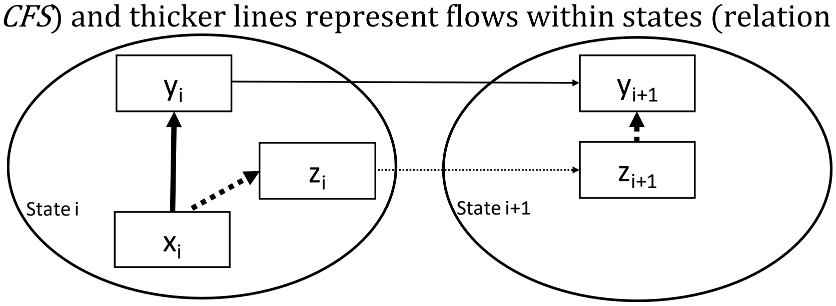

Figure 8 (Def. 8): Interstate data flow

These intuitions are captured by the following:

Definition 8: $CFS(x_i,y_{i+1}) =_{def} CF_i(x_i,y_i)$ *or for some* z $(CF_i(x_i,z_i)$ *and* $CF_{i+1}(z_{i+1},y_{i+1}))$, or equivalently: $CFS(x_i,y_{i+1}) =_{def} [x_i] \sqsubseteq_i [y_i]$ *or for some* z $([x_i] \sqsubseteq_i [z_i]$ *and* $[z_{i+1}] \sqsubseteq_{i+1} [y_{i+1}])$

Definition 8 holds not only for distinct *x,y,z* but also for all combinations of: $x=y$ or $x=z$ or $y=z$.

Transformations are risky for security because entities may be relocated by transformations, and the data they contain may be exposed by new data flows in following states. E.g. $[x_i] \sqsubseteq_i [y_i]$ could be true but $[x_{i+1}] \sqsubseteq_{i+1} [y_{i+1}]$ false. Security policies may require that, if a transformation creates the possibility of certain new data flows for an entity, then the desired security properties should be preserved by using known mechanisms called data purging, sanitizing or declassification. This is a complex problem, among others in practice it is not sufficient to remove data with certain labels, since entities may have the capability of processing the data received, so that these are no longer easily traceable to their source, unless full-fledged provenance labeling is used (Moreau [40], Park et al. [44]). A survey on this problem is in Sabelfeld, Sands [49], an early discussion on it with theory and examples can be found in Foley [24], Pasquier et al. [46] discuss it in Cloud and IoT contexts, while Myers and Liskov [41] provide interesting examples. With reference to the example of Figs. 1 and 7, we see that the flow from entities *B,C,D,* to entity *I* is true before and false after the transformation. So the data that may have flown to *I* from these entities might have to be purged from *I*. We say 'might' because in some situations this might not be required, e.g. if we suppose that *I* will henceforth only flow to *B,C,D*. In some cases, the policy might be to simply require that a record be kept of data that might have been brought in from entities that are no longer accessible, possibly in order to identify future conflicts, see Ex. 11. Also, there is no danger if the data in *I* have been declassified. As another example, in RBAC adding read permissions from an object may cause it to move down in the partial order, however there are no requirements regarding purging or sanitizing.

We consider only purging in this paper, see Sect. 7.3. It can be supposed that purging is done instantly in state transitions.

7.3 State transformations in label-based systems

State changes or transformations in label-based systems with the lattice model have been considered by many authors, starting with Sandhu [50],Foley[22,23,25], Bishop[13 Sect.5.2]. For consideration of the Cloud context, see Pasquier et al. [45]. For the transformation in Figs. 1 and 7, if a canonical label-based system is used, the names of entities *B,C,D* disappear from the label of *I* ; on the other hand, the label of *A* goes to contain the names of all entities.

Our theory is consistent with established theory, which considers that new flows are related to changes in entity's labels, see the 'high and low water mark' policies (Weissman[58], Biba[11], Foley [23,24], Sandhu [51]). We have seen that in label-based networks, the *CF* relations are determined by the partial orders of label sets. If $CF_i(x_i,y_i)$ is false but $CF_{i+1}(x_{i+1},y_{i+1})$ must be true, then $Lab(x_i) \leq Lab(y_i)$ is false and $Lab(x_{i+1}) \leq Lab(y_{i+1})$ must be made true. If labels are simple category sets, then $Lab(x_i) \subseteq Lab(y_i)$ is false and $Lab(x_{i+1}) \subseteq Lab(y_{i+1})$ must be made true. The reverse reasoning holds for removing flows. As mentioned, purging policies may require that an entity be purged of the data of the removed categories. This is particularly clear with canonical labeling. However note that label changes do not necessarily change the partial order, if so they won't cause flow changes or entity relocations.

Policies or requirements may dictate that, as data flows change, certain network properties must be kept invariant. In our method there is a single mechanism to do this, as long as the properties can be expressed in terms on reachable labels: *First, an appropriate set of labels is defined, and then the transformations can only use labels in this set*. In Sect. 6 we have seen how different label sets can be defined to enforce certain requirements. If the requirements must hold over state changes, the label changes must use only labels in those sets. If labels are used to indicate data provenance, the partial order defines the minimum and maximum levels of secrecy and integrity allowed for data of each provenance, and which combinations of data provenances are allowed or required. In succeeding states, labels will show the provenance of the data that can have flown into the entities.

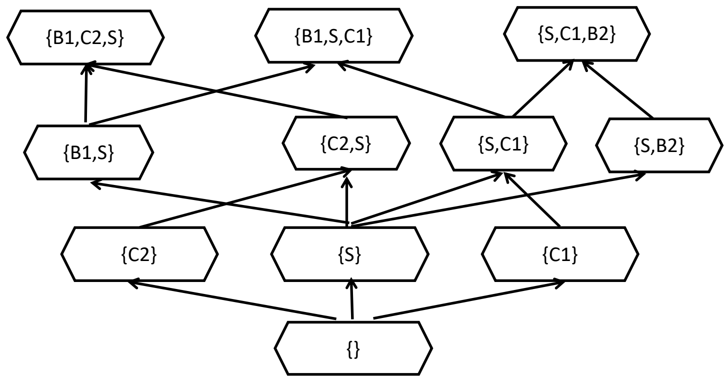

Figure 9 (Ex.11): Partially ordered set of allowed labels

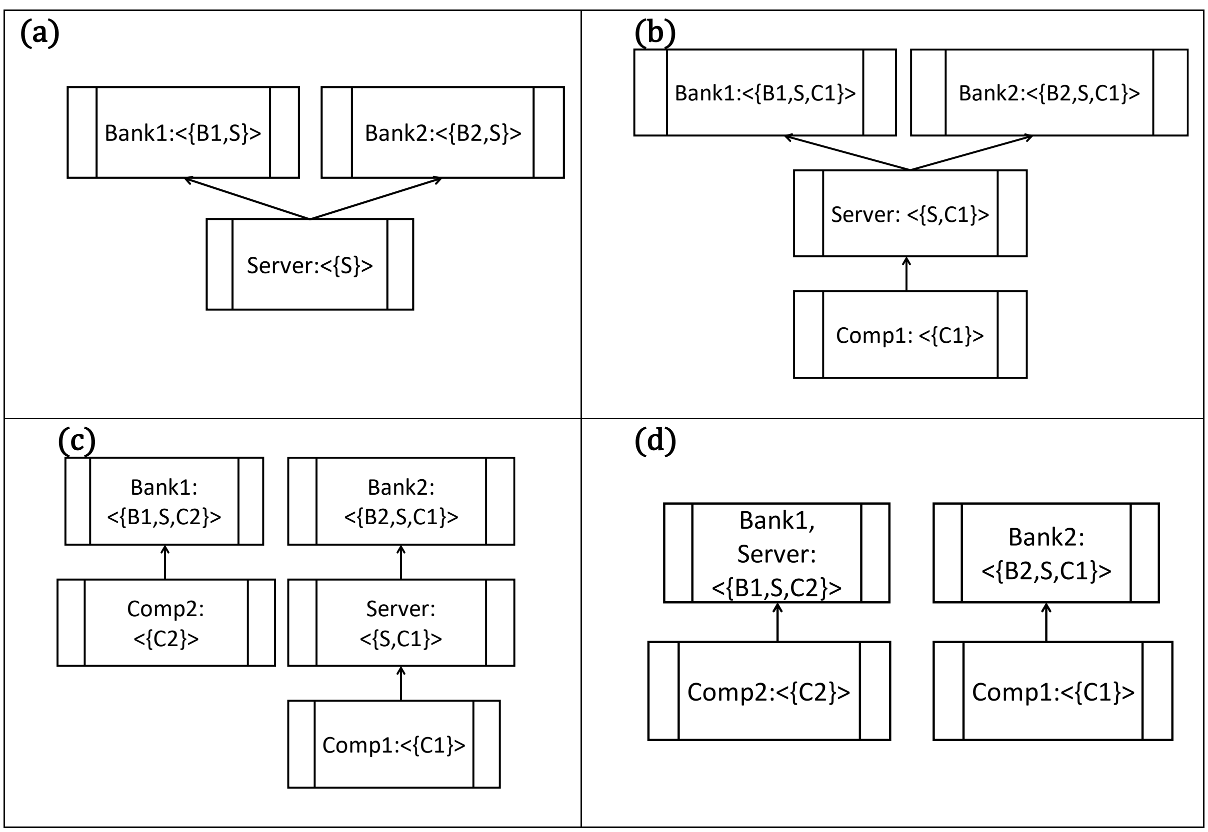

Figure 10 (Ex.11): Transformation Scenario

In particular, Chinese-wall (Brewer-Nash) conflict policies can be implemented by forbidding labels allowing flows from conflicting entities into any entity. This is not the usual definition of these policies, but it addresses the same concerns, see Ex. 4 and Ex. 11. Essentially, our model is consistent with the models of Sandhu [50] and Sharifi, Tripunitara [54], although we do not distinguish here between subjects and objects.

Example 11. We will see how two types of security requirements can be expressed simultaneously by using our method: conflicts and conglomerates, see Sect. 6. In this example, labels are simple category sets, indicating data provenance, partially ordered by the set inclusion relation.

We have two banks, two companies and a data server. The following security requirements (policies) must be observed:

1) Separation of data (conflict) must be observed between the two banks, the two companies, and *Bank2* with *Company2*. With some obvious abbreviations, these requirements are expressed by forbidding labels containing the following subsets: *{B1,B2}, {C1,C2}, {B2,C2}*.
2) In addition, both banks need data from the server, thus whenever one of *B1* or *B2* appears in a label, this must be in combination (conglomerate) with *S*.

The resulting partial order of allowed labels is shown in Fig. 9. Note that if the lattice model was used, some labels contradicting the requirements would have to be included, and then it would have to be said that they cannot be used.

The *purging policy* will be as follows: if and when *CF(x,y)* becomes false, then one of the following has to be done:

- purging from *y* data that might have flown from *x* and removing the name of *x* from the label of *y*
- or not purging and leaving the name of *x* in the label of *y*. In this second case, the resulting labels will not be canonical.

A possible sequence of transformations for this network is as follows, see Fig. 10 (mentioning only some significant steps).

(a) We take this as the starting state: we have a *Server* with data flows towards two banks in conflict of interests.

(b) *Company1* comes in (is created) and a data flow is created from it to *Server* and extending to the two banks. To do this, *C1* is added to the labels of *Server* and the two banks.

(c) *Company2* comes in and is in conflict of interest with *Company1* and *Bank2*, but not with *Bank1*. It is decided that *Company2* should work with *Bank1*. To do this while avoiding creating a label containing the forbidden combination *{C1,C2}*, *C1* is removed from the label of *Bank1*, the flow from *Company1* to *Bank1* is lost, and *Company1* data are purged from *Bank1*. Since *Server* keeps *C1* in the label, the flow from it to *Bank1* is also lost, but *Bank1* keeps *S* in its label and does not purge *Server* data. Then *C2* is added to the label of *Bank1*, giving *<{B1,S,C2}>*. This opens a data flow from *Company2* to *Bank1*. We still have the flow from *Company1* to *Bank2* through *Server*. Note that *S* has been kept in *Bank1'*s label to remember that *Bank1* might still have previously acquired *Server* data, which it might continue to need. This can be useful e.g. if in the future *Bank1* requests to access data sets in conflict with *Server,* or, if *Server* returns to its initial state by purging data acquired in the meantime, it will again be able to flow to *Bank1* without the need of authorizations.

(d) It is now decided to give *Server* in exclusive use to *Bank1*. To avoid the forbidden label combination *{C1,C2}*, *C1* is removed from the label of *Server*, and *Company1* data are purged from *Server*. *{B1,C2}* is then added to the label of *Server.* This makes *Bank1* and *Server* equivalent, they are now connected by a bidirectional flow and *Company2* data can flow to them.

If desired, now the two remaining unidirectional flows can be made bidirectional, by appropriate label changes. This would transform the system into two equivalence classes of entities, one containing *Bank1*, the *Server* and *Company2* with label *<{B1,S,C2}>* and the other containing *Bank2* and *Company1*, with label *<{B2,S,C1}>*. This having been done, all entities will be at their highest possible levels but other states can be reached by moving entities down the partial order, or by creating or removing entities.

Since the label transformations in this example have followed the partial order of Fig. 9, in upwards or downwards directions, the mentioned system's data security requirements have been respected in all network states.

We have assumed that all requirements and label sets are known in advance. In a real system, however, these may change. For example, at state (a) it is possible that the label set be limited to *<{}>,<{S}>,<{B1,S}>,<{B2,S}>*. At successive states, new categories may come in, possibly with new requirements. With the inclusion of category *C1*, for which there are no requirements with respect to previously included categories, the set of labels is increased with the following possibilities: *<{C1}>, <{S,C1}>, <{B1,S,C1}>, <{B2,S,C2}>*. The inclusion of category *C2* with its conflict requirements leads to the labels in Fig. 9. New requirements may create exceptions with respect to previous ones, e.g. we have mentioned in Ex. 4 that the combination *{B1,B2}* may be allowed in the presence of some new category, such as the *CentralBank*. This subject requires further study.

As we have seen in Sect. 6, many types of security requirements or policies can be expressed by using labels, and so this method is powerful, although to be practical it needs a useful administration model, which should be developed in terms of state transitions.

8. Algorithms and complexity

This section outlines some existing algorithms that can be used to support our method, and provides some rough estimates of the time that can be required for state changes, which would be down time for the whole network. We consider the ideal case where the network administrator maintains the $Channel_i$, CF_i and $\sqsubseteq_i$ matrices for the current state $\underline{i}$ (the first is indispensable, but the other two are useful to avoid recomputing them).

The useful algorithms, mentioned so far, are the following (in the complexity expressions below, n is the number of entities, m is the number of channels and O the order of complexity; only time complexity has been considered).

a) Transitive closure algorithm of a digraph, whose complexity is approximatively $O(n^3)$ (Aho, Ullman [2]).
b) Transitive reduction algorithm, which has the same time complexity, in fact it turns out to be essentially the same algorithm as a) (Aho, Ullman [2]).
c) Strongly connected component algorithms, such as the one of Tarjan [57]. These algorithms also yield the partial order of the components that they find. Their complexity is $O(n+m)$.
d) Digraph isomorphism algorithms. Research claims 'quasi-polynomial' efficiency with various O-formulae where $log(n)$ is in the exponent. A recent survey on this topic is in Grohe, Schweitzer [28].

These algorithms can be used respectively for:

a) Obtaining the *CF* relation from a *Channel* relation or *Lab* function.
b) Reducing a *CF* relation to yield a reduced number of channels.
c) Finding the equivalence classes and their partial order ($\sqsubseteq$ relation) from a *CF* relation.
d) Determining whether two partial order graphs are isomorphic. In our method, an application of this algorithm is for checking whether an implementation is correct with respect to a given partial order digraph, see Implementation methods at the end of Sect. 4.

Research continues in graph and partial order algorithms and better complexity bounds are being found. This summary discussion has the only goal of showing that, except for d), the computations needed can be done with efficient algorithms not exceeding polynomial, in fact cubic, complexity. So for operations a) to c), we can retain $O(n^3)$ as the worst-case order of complexity.

For each state change, the straightforward, unoptimized method is:

1) Compute the new *CF* relation.
2) Compute the new equivalence classes of entities, algorithm c).
3) Use one of the implementation methods of Sect. 4 to define the new Channel relation; we have seen that this may involve using algorithms b) and d).

The real computation time spent for the whole state change process is difficult to estimate, among others it is not possible to evaluate the time spent for step 3), where several solutions can be considered. But if implementation method 1) is used, and *CF* is simply reduced transitively, then we can take $O(n^3)$ to be the order of complexity of the whole process. So for a network of 10^5 entities, the order of complexity of the algorithms to be executed at each state change is $O(10^{15})$. If the routing is centralized, these are all internal calculations in one computer, the administrator's. Simulation runs for a closely related problem were given by Stambouli, Logrippo in [56]. We refer to this paper for details, and we mention here only that for 10,000 nodes or entities the calculation time was 97 seconds, raising to about 1,75 hours for 100,000 entities (however as mentioned exponential growth is excluded). Consider that networks will be often partitioned by application areas, and more efficient methods will certainly be found. Also, it may be possible to adapt for use in this area the extremely efficient graph processing programs that have been developed for use in the natural sciences and engineering.

We have mentioned that beyond this, one can devise implementation methods for optimizing according to some criteria the *Channel* relation. New physical channels may have to be opened or some existing ones may have to be closed. Some of the needed channels may be already available, perhaps by transitivity, others may have to be created with varying costs. Channels can be more or less efficient with respect to Quality of service requirements. So implementations will involve the consideration of various requirements, costs and weights. However in general, networks do not need to be reduced or optimized, on the contrary often redundancy is considered to be beneficial.

In practice, centralized methods may be unfeasible and then it will be necessary to devise decentralized and 'on the fly' methods. These are unlikely to produce optimized networks by any criteria, however, as just mentioned, this is rarely important.

9. Discussion

The basic reference for the problem of data flow control for security is still considered to be the lattice model of Denning [17] with its presentation by Sandhu [51], and variations proposed by others. This model considers networks of entities that are labeled by *security classes* (or levels) which must form lattices. By using a partial order model, our paper extends and generalizes the lattice model, namely:

1. We have noted that rarely networks are defined as lattices with labeling, in fact several other methods of defining networks are listed at the beginning of Sect. 4.
2. We have shown in Sect. 4 that *any* non-trivial network defines a partial order of equivalence classes of entities having different levels of secrecy and integrity.
3. We have shown in Sect. 6 and [56] that with the partial order model any capability list or access control matrix or *Channel* relation can be translated into a label-based (canonical) network and vice-versa, establishing the essential equivalence of all these network definition methods; whereas if labels must form lattices there are access control matrices that cannot be translated into label-based networks.
4. Our model does not need the defining property of lattices, which is the existence of *unique* joins and meets.
5. We have noted that transforming non-lattice networks into lattices requires the inclusion of arbitrary meets, joins and channels that can contradict requirements (end of Sect. 6); this transformation can be non-trivial (Bertet et al. [10]) and is unnecessary in our model.
6. We have also seen (Sect. 7.1) that, for the same reason, network transformations can be more easily defined on the basis of a partial order model, than on the basis of a lattice model.

The concept of levels of security has long been associated with Mandatory Access Control security models and considered to be too restrictive, and therefore of limited application: it is a fact that the support of these models in terms of commercial products is limited. On the contrary, levels of security exist in any network if the 'lattice' hypothesis is relaxed. This observation enables a more unified theory of access control and data flow control than it has been possible so far.

We have seen (Sect. 5) that for the placement of data, it is necessary to determine the level structure of the network, which can be done efficiently with strongly connected component algorithms, and place data accordingly. We have mentioned that network transformations may require policies to purge or declassify data.

Let us take a critical look at the two assumptions of reflexivity and transitivity, which are also adopted in most formulation of lattice-based theory. Reflexivity seems to be an easy assumption, as normally entities have access to their own data and if there is data flow partitioning within an entity, then the entity can be conceptually split. Transitivity is an assumption that is adopted by much research on data flow control. This assumption can be defended on two grounds. First, it is a pessimistic assumption that can lead to over-protected networks, often a useful property in security. Second, non-transitivity may be modeled in transitive networks by splitting some entities in two or more, some with outgoing edges and some without, and dividing the data accordingly. Non-transitive networks have been motivated and studied in several papers, among others in the mentioned work by Foley (references [20,21,24]) and also in Rushby [48]. Non-transitive flow assumptions are important in security, because in some cases it is necessary to assume that some data will not be passed on. This question is related to the question of transitivity of trust in social and other networks (Huang et al. [31], Adelmayer[1]).

Multi-layer security theory has been often concerned with hidden channels. As mentioned in Sect. 4, these can be seen as augmenting the *Channel* relation and so modifying a perceived partial order, which is a vulnerability of all access control and data flow control systems. Hidden channels outgoing the top-level entities, or ingoing the bottom-level ones are the most dangerous for secrecy or integrity respectively. Our framework does not correct these vulnerabilities but can model them and enables to see their effects. This seems to be a useful starting point for research on *threat analysis*.

In many organizations, data flow issues are dealt with by specializing data bases and rigidly regulating channels. For example, there may be separate data bases for financial data, for personnel data, and so on, with access control policies often following the RBAC model. In the method we propose, it should be possible to define global networks to express all such structures, where flows can be directly opened or closed by administrative decisions, for specific categories of data. In Ex. 11 we have also seen that constraints on labels can ensure that only allowed data flows are created. When the set of labels can be changed, other possibilities arise, e.g. in other work (Boulares et al. [15]), we have shown that it may be useful to dynamically create security levels corresponding to entity data contents, depending on the evolution of data flows.

The practical usefulness of our method can be questioned in cases where the *Channel* relation can change rapidly over time (Matousek et al. [38]) or where it is difficult to calculate, as can happen for example in telecommunications networks or ABAC. However at each state, short lived as it might be, there will be a partial order of equivalence classes with corresponding data flow relations. Hence it is necessary to determine what data flow relations must be kept invariant and plan transformations accordingly (Foley et al. [25]). For example, entities designated to contain the most secret data should remain at the top layer, while entities representing data gathering devices such as sensors should be kept at the lowest layer. This can be done by requirements on the allowed labels for some entities, e.g. it can be stipulated that entities of type ‘sensor’ can only have labels containing the names of the entities in their equivalence classes. Such requirements can be enforced through state changes.

In each partial order, data can flow upwards only, from their initial position towards the top. So for practical applications it will be necessary to have several coexisting partial orders at each network state. Logrippo and Stambouli [37] present an e-commerce example where two partial orders must coexist: one to carry orders from clients to providers, and a second one to carry billing data in the opposite direction. As well, in many organizations and in the military, field data flow from the field to the command, and directives flow from the command to the field. This can be achieved by extending our approach: data will have to be classified and labeled by type, and each type will have its own *Channel* relation. Further, it will be necessary to introduce the concept of *trusted entity,* which has a long history in data security (Bell, La Padula [8], Myers, Liskov [41]). Such entities can belong to several partial orders and can be trusted to keep separate the data of the different data types that can flow through them.

The problems of data security in organizational network and the IoT are very complex, since such systems have many needs, aspects and interactions, as well as specific architectures and data flows, often varying from application to application. The theory presented here, whose applicability to the IoT is mentioned in Ex. 1 and 11, is far from addressing all these complexities. Even within the realm of its potential usefulness, this work will have to be complemented by implementation methods for real-life networks, demonstrated on practical case studies, etc. However we have made the point that a theory based on partial orders can be feasible and more generally applicable as a conceptual framework than the still widely mentioned theory based on lattices.

Note that we have only dealt with *data flow*, and not with *information flow* in this paper. The latter can occur as the combined effect of data flows and inferences (Denning [18], Gollmann [27 Sect. 3.3]) and can create hidden channels. Inferences allow to create information, which then can be transferred as data, and may make it possible to transfer data that should not be transferred according to access control or data flow control policies (for example, a date of birth that should be a secret can perhaps be inferred from non-secret data and then broadcast). Mechanisms to control inferences are quite different from those used for data flow control, and they are the subject of a different literature. However it can be safely said that no inferences are possible on data that are not available, and in this sense data flow control is a more specific mechanism than information flow control.

10. Conclusions

We have shown in this paper that, although most research on data flow security adopts the lattice model, a partial order model is more general and simpler. Going back to the research questions proposed in Sect. 1, we have seen that in our theory the secrecy and integrity questions can be answered by computing the $\sqsubseteq$ relation. In Sect. 5, we have shown that we are able to classify entities by levels of secrecy and integrity. In the same section, we have seen how secrecy and integrity policies can be satisfied by appropriately placing entities in partial orders. In Sect. 7.3 we have shown that certain security requirements or policies can be maintained through state transitions by imposing the rule that only certain labels are allowed.

The examples of Section 6 have revisited, from this new point of view, several well-known concepts in data security theory and practice: conflicts (Brewer - Nash or Chinese wall), conglomeration, aggregation, composite labeling systems, etc. We have confirmed a result already presented in Stambouli, Logrippo [56] showing that, with our definitions, arbitrary access control lists can be translated into labeled systems where labels are simple sets of categories, and vice-versa; then data flow control rules or access control rules reduce to simple inclusion relations between such sets. Network transformations were defined as introduction or deletion of entities, or other changes in data flows, normally due to administrative or user action, or policies. We have also shown by examples (chiefly 1and 11) that our networks can be interpreted as Internet of things networks, an area where data secrecy, integrity and privacy are very important but few applicable principles are known.

Some questions for future research were mentioned in the paper. On the theoretical level, there is the question of reinterpreting established security concepts, for example the theory of RBAC administration can be reformulated in terms of state changes. On the practical level, there is the question of the implementation of our method. We have seen that efficient algorithms exist, which opens the door to software tools to support the related design and administration activities.

Acknowledgment. This research was funded in part by a grant of the Natural Sciences and Engineering Research Council of Canada. The author is grateful to his PhD student, Abdelouadoud Stambouli, for useful discussions, and to his colleague Jurek Czyzowicz, always ready to answer questions on partial orders and algorithms. The feedback from the referees led to several improvements.

References

1. F.M.Adelmayer, M. Walterbusch, P. Biermanski, F. Teuteberg. Trust transitivity and trust propagation in cloud computing ecosystems. Proc. 26th European Conf. on Inform. Systems (ECIS2018).
2. A.V. Aho, M.R. Garey, J.D. Ullman. The transitive reduction of a directed graph. SIAM J. Comput. 1(2), 1972, 131-137.
3. A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.
4. A. Al-Haj, B. Aziz. Enforcing multi-level security policies in data-base defined networks using row-level security. Proc. Intern. Conf. on Networked Systems (NetSys2019), 1-6.
5. P. Amthor, W. E. Kühnhauser, A.Pölck. WorSE: A workbench for model-based security engineering. Computers and Security, 24 (2014) 40-55.
6. J. Bacon, D. Evans, D.M.Eyers, M. Migliavacca, P.Pietzuch, B.Shand. Enforcing end-to-end application security in the Cloud. Proc. Middleware 2010, LNCS 6452, 293–312
7. J.Bang-Jensen, G.Z.Gutin. *Digraphs – Theory, algorithms and applications.* Springer, 2nd Edition, 2009.
8. D.E. Bell, L.J. La Padula. Secure computer systems: unified exposition and Multics interpretation. MTR-2997, Mitre Corp., Bedford, Mass., 1976.
9. S. Bera, S. Misra, A.V. Vasilakos. Software-defined networking for Internet of things: A survey. IEEE Internet of Things Journal 4(6), 2017, 1994-2008.
10. K. Bertet, M. Morvan, L. Nourine. Lazy Completion of a Partial Order to the Smallest Lattice. Proc. Intern. KRUSE Symposium: Knowledge Retrieval, Use and Storage for Efficiency, 1997, 72–81.
11. K.J. Biba, Integrity considerations for secure computer systems. TR-3153. Mitre Corp. Bedford, Mass., 1977.
12. G. Birkhoff. *Lattice Theory,* American Mathematical Society, 1967.
13. M. Bishop. *Computer security, Art and science*. 2nd edition. Addison-Wesley, 2019.
14. A. Botta, W. de Donato, V. Persico, A. Pescapé. On the integration of Cloud computing and Internet of things. Proc. Intern. conf. on Future Internet of things and Cloud (FiCloud 2014), IEEE Comp. Soc., 23-30.
15. S.Boulares, K.Adi, L.Logrippo. Information flow-based security levels assessment for access control systems. Proc. 6th Intern. MCETECH Conf. on eTechnologies, 2015. Springer LNBIP 209, 105-121.
16. F. Cuppens. A modal logic framework to solve aggregation problems. In: Database Security V: Status and Prospects, North-Holland,1992, 315-332.
17. D.E. Denning. A lattice model of secure information flow. Comm. ACM 19(5), 1976, 236-243.
18. D.E. Denning, P.J. Denning. Data Security. Computer Surveys, 11(3), 1979, 227-249.
19. D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli. *Role-based access control.* 2nd Ed. Artech House, 2007.

20. S.N. Foley. Lattices for security policies. Royal Signals and Radar Establishment, Malvern, Report No. 90005, 1990.
21. S.N. Foley. Unifying information flow policies. Royal Signals and Radar Establishment, Malvern, Report No. 90020, 1990.
22. S.N. Foley. A taxonomy for information flow policies and models. Proc. 1991 Symp. on Res. in security and privacy. IEEE, 98-108.
23. S.N. Foley. Separation of duties using High water mark. Proc. Comput. Security Foundations Worksh., IEEE, 1991.
24. S.N. Foley. Aggregation and separation as noninterference properties. Journal of Computer Security, 1(2),159–188, 1992.
25. S.N Foley, L. Gong, X. Qian. A security model of dynamic labeling providing a tiered approach to verification. Proc. 1996 IEEE Symp. On Security and Privacy, 142-153.
26. R. Fraïssé. *Theory of relations.* North-Holland, 1986.
27. D. Gollmann. *Computer Security.* 3rd Ed., Wiley, 2011.
28. M. Grohe, P. Schweitzer. The graph isomorphism problem. Comm. ACM 63(11), 2020, 128-134.
29. F. Harary, R.Z. Norman, D. Cartwright. *Structural models: an introduction to the theory of directed graphs.* Wiley, 1965.
30. V.C. Hu, D.F. Ferraiolo, R. Chandramouli, D.R. Kuhn. *Attribute-Based Access Control.* Artech House, 2018.
31. J. Huang, M.S. Fox, S, Mark. An ontology of trust: formal semantics and transitivity. Proc. 8th Intern. Conf. on Electronic Commerce (ICEC 2006), ACM Press, 259-270.
32. J. Jaskolka, R. Khedri, Q. Zhang. On the necessary conditions for covert channel existence: A state-of-the-art survey. Proc. 3rd Intern. Conf. on Ambient Systems, Networks and Technologies (ANT 2012). Elsevier Procedia Computer Science 10(2012), 458-465.
33. S. Khobragade, N. V. Narendra Kumar, R. K. Shyamasundar. Secure synthesis of IoT via readers-writers flow model. Proc. Intern. Conf. on Distrib. Computing and Internet Techn. (ICDCIT 2018), LNCS 10722, 86–104.
34. C. E. Landwehr. Privacy research directions. Comm. ACM 59(2) (2016) 29-31.
35. L. Logrippo. Logical method for reasoning about access control and data flow control models. Proc. 7th Intern. Symp. on foundations and practice of security (FPS 2014). LNCS 8930, 205-220.
36. L. Logrippo. Multi-level access control, directed graphs and partial orders in flow control for data secrecy and privacy. Proc. of the 10th Intern. Symp. on Foundations and Practice of Security (FPS 2017), LNCS 10723 (2018), 111-123.
37. L. Logrippo, A. Stambouli. Configuring data flows in the Internet of Things for security and privacy requirements. Proc. 11th International Symp. on Foundations and Practice of Security (FPS 2018). Springer LNCS Vol. 11358, 115-130.
38. P. Matousek, J. Rab, O. Rysavy, M. Sveda. A Formal Model for Network-Wide Security Analysis. Proc. 15th Ann. IEEE Intern. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS 2008), 171-181.
39. C. Meadows. Extending the Brewer-Nash model to a multilevel context. Proc. 1990 IEEE Symp. on security and privacy, IEEE Computer Society, 95-102.
40. L. Moreau. The foundations for provenance on the Web. Foundations and trends in web science. 2(2-3), 2010, 99-241

41. A. C. Myers, B. Liskov. Protecting privacy using the decentralized label model. ACM Trans. on Software Eng. and Methodology, 9(4), 2000, 410-442.
42. N.V. Narendra Kumar, R. Shyamasundar. A Complete generative label model for lattice-based access control models Proc. Software Engg and Formal Methods (SEFM 2017), Springer LNCS 10469, 35–53, 2017.
43. S. Osborn. Information flow analysis for RBAC system. Proc. 7th ACM Symp. on Access control models and technologies (SACMAT '02), 163-68.
44. J. Park, D. Nguyen, and R. Sandhu. A provenance-based access control model. Proc. Intern. Conf. on Privacy, Security and Trust, IEEE, 2012, 137-144.
45. T. Pasquier, J. Singh, D. Eyers, J. Bacon. CamFlow: Managed Data-Sharing for Cloud Services. IEEE Trans. on Cloud Computing, 5(3) 2017, 472-484.
46. T. Pasquier, J. Bacon, J. Singh, D. Eyers. 2016. Data-Centric Access Control for Cloud Computing. Proc. 21st ACM Symp. on Access Control Models and Technologies (SACMAT '16), 81-88.
47. T. Pasquier, D. Eyers, J. Bacon. Personal data and the internet of things. It is time to care about digital provenance. arXiv preprint arXiv:1904.00156, 2019.
48. J. Rushby. Noninterference, transitivity and channel-control security policies. Technical Report, SRI International, May 2005.
49. A. Sabelfeld, D. Sands. Dimensions and principles of declassification. Proc. 18th IEEE Computer Security Foundations Worksh. (CSFW'05), 255-271.
50. R.S. Sandhu. Lattice-based enforcement of Chinese Walls. Computers & Security Vol. 11(8), 1992, 753-763.
51. R.S. Sandhu. Lattice-based access control models. IEEE Computer 26(11), 1993, 9–19.
52. R.S. Sandhu, S. Jajodia. Data and database security and controls. Handbook of Information Security Management, Auerbach Publishers, 1993, 481-499.
53. R.S. Sandhu. On five definitions of data integrity. In Database Security VII: Status and Prospects, North-Holland, 1994, 257-267.
54. A. Sharifi, M.V. Tripunitara. Least-restrictive enforcement of the Chinese wall security policy. Proc. 18th ACM Symp. on access control methods and technologies (SACMAT 2013), ACM, 61-72.
55. J. Singh, T.Pasquier, J.Bacon, H.Ko, D.Eyers. Twenty security considerations for cloud-supported Internet of Things. IEEE Internet of Things Journal, 3(3) (2016), 269-284.
56. A. Stambouli, L. Logrippo. Data flow analysis from capability lists, with application to RBAC. Information Processing Letters, 141(2019), 30-40.
57. R. E. Tarjan. Depth-first search and linear graph algorithms, SIAM Journ. on Computing, 1(2) (1972), 146–160.
58. C. Weissman. Security controls in the ADEPT-50 time sharing system. Proc. 1969 AFIPS fall joint computer conf. AFIPS Press, 119-133.