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, hence 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 complex 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, conglomerates, aggregation, are introduced in examples. Then it is shown how entities can be added, removed or relocated in such partial orders, as a result of authorized user or administrative action, or of policies. Security requirements can be maintained 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, Internet of things.

1. Introduction

This paper presents some basic concepts that underlie many models for data security. We deal with 'entities' that can model subjects or users in organizational networks, or databases, or 'things' in the Internet of things (IoT) or in the Cloud, or 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* relation models the fact that data can reach certain entities in a network from certain others, directly or indirectly through channels. This relation is shown to define preorders of entities or partial orders of components (the latter representing classes of equivalent entities). On this basis, the relation can be used to define basic concepts of secrecy (also called confidentiality in the literature) and integrity. We consider also network state changes or transformations, caused by introduction of new entities, removal of entities, or relocation of entities. Data security methods based on labelling are presented as a special case. The efficiency of relevant algorithms is discussed. Several examples are introduced.

Note that we only deal with data flow, and not with 'information flow' in this paper. The latter can occur as the combined effect of data flows and inferences [15], and inferences are not considered here.

2. Previous work and contributions

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 textbook

Revised version of paper submitted for publication in March 2020

material [11 Sect. 5.3]. 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) implemented by Multi-level access control methods (MLS) based on 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 inflexible 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 that the applicability of MLS 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 two topics [52,12].

Bell and La Padula [7] developed a fairly complex theory of data flow security based on partial orders of subjects and objects, system states, and state transitions caused by transition rules triggered by data access requests. Denning [14] proposed 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 this second 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.

Simon Foley is the author who has most extensively written on data flow policies. Some of his papers on the subject are references [17 to 26], 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 $\leadsto$, and transitive ones, noted $\mapsto$. Foley also considered many types of specialized data security requirements, a few of which will be considered here.

Sandhu [48,47] presented the applicability of lattice-based models with labelling for information flow security, considering secrecy, integrity and conflict requirements. He demonstrated the use of his model for representing several data flow security policies. Again, his theory and examples were very influential for us. Our treatment of the Chinese wall policy is consistent with his observations, on this topic see also [51].

This valuable 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)[16] or Attribute based access control (ABAC) [29], which address access control and not data flow security directly. Data flow security is the main objective of this paper, because for real data security it is necessary to control where data can eventually end up [14].

Bacon and her team [5,43,44,52] 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 security and integrity requirements and includes methods for label upgrades and downgrades. The concepts presented here are consistent with their approach but constitute a more general theory of data flow security that they could use.

Paper [32] present a method for data flow security in the IoT based on the lattice model, and concepts of readers and writers. Our model is consistent with theirs, but simpler and more general. In fact, their examples are easily resolved in our formalism. Related research by the same group, presenting the use of labels for data flow control in the Cloud, is in [40].

The authors of [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 [34] uses graph theory to show that partial orders and multi-level models are necessary and sufficient models for designing data secrecy systems in networks that can be represented by directed graphs. Paper [53] shows that efficient algorithms exist to obtain partial order models for given access control matrices or Role based access control (RBAC) [16] permission lists; also it shows that, as a consequence, 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 [33]. Paper [35] 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 new paper simplifies and strengthens the above-mentioned conceptual models. Our Property 1, showing the relation between the partial order of entities in any network and the data flow direction, is independent of labeling, and holds in any reflexive, transitive network (Sect. 4). It is then shown 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 labeling is brought in in our model (Sect. 6), it is shown that secrecy and integrity requirements, together with other types of requirements such as conflicts, conglomerates, aggregates, can be represented by using composite labels that determine the place of entities in partial orders. This paper also shows how our model can be extended to express network transformations, expressing administrative, policy or user decisions that change the *CanFlow* relation; security requirements can be maintained through such transformations (Sect. 7), by allowing only certain labels. The available algorithms that can be used to support this method are reviewed and shown to be efficient in Section 8.

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, reflexivity will not be represented.

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 [10 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 $[x] \sqsubseteq [y]$ [10 Th. 3, 27 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 (we call them the *components*, representing equivalence classes), 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 [28 Chapt.3, 6 Sect. 1.5]. 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 usual terminology for MLS models [11] 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*.

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 [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* [11 Sect. 2.1]. The primitive relation *Channel(x,y)* expresses the fact that, in the network under consideration, there is a data transfer channel from *x* to *y*. In our *network graphs* (see Fig. 1(a), this relation is represented by directed arrows, and bidirectional arrows represent symmetrical *Channel* relations. Our concept of entity is generic and can be used to model many types of security entities such as subjects, users, roles, objects, data bases or IoT entities. Similarly, channels can be taken to be data transfer authorizations or communication links. The *Channel* relation can be used to model any data transfer possibility, e.g.:

1. Read and write authorizations expressed in Access control matrices [11 Chapter 2] or RBAC permission lists [16 Sect. 3.2.2, 41] can be represented as *Channel* relations. In fact, this should be possible for any access control system, even complex ones such as Attribute-based access control (ABAC) [29] because at any given state the system

must be able to decide whether access is authorized or not. If, in a given state, entity (subject) *x* can read or in any way receive data from entity (object) *y*, then *Channel(y,x)* is true in the network modeling that state, while *Channel(x,y)* is true if *x* can write on, or in any way send data, to *y.*

2. In particular, in access control systems of the Multi-level family (MLS) based on labelling, the sets of entities 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 [9] there are channels from entities at a certain level of integrity to entities at an equal or lower level of integrity.
3. Distributed networks have routing relations that can be defined as *Channel* relations in our sense. *Software defined networks (SDN)* [8,4] 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* [11 Sect. 17.1,31] 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. Directional 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 at any given network state, the *Channel* relation is well defined.

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 also 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 *CanFlow* relations could be identical. However in practice the first could be more easily determined from the data transfer methods 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.

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* iff *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 relation $\lesssim$ above, 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).

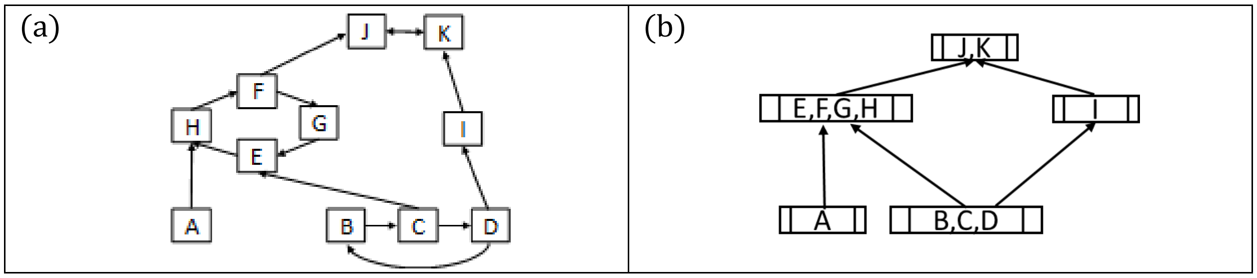


Figure 1: (a) A network of entities and (b) its partial order of equivalence classes

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 is isolated, but the last three work together, perhaps to complement each other's data. The nodes above might represent various types of processing nodes. We see here several security 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.

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 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 as $[x]\sqsubseteq[y]$.
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 existing network infrastructure. In practice, it may be necessary to take into consideration pre-existing channels, cost constraints etc. and this will be easier 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 checking that it is isomorphic to the initially given partial order digraph.

Canonical label-based networks, described in Sect. 6, are also implementations of partial orders with label-based access control rules.

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

Data security is often defined as having (at least) the two aspects of *secrecy* (also called *confidentiality*) and *integrity* [47]. 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 [34, 53]). 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 [50]. Our definition corresponds to what in [50] is called *information flow integrity*, for which the main historical reference is [9].

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

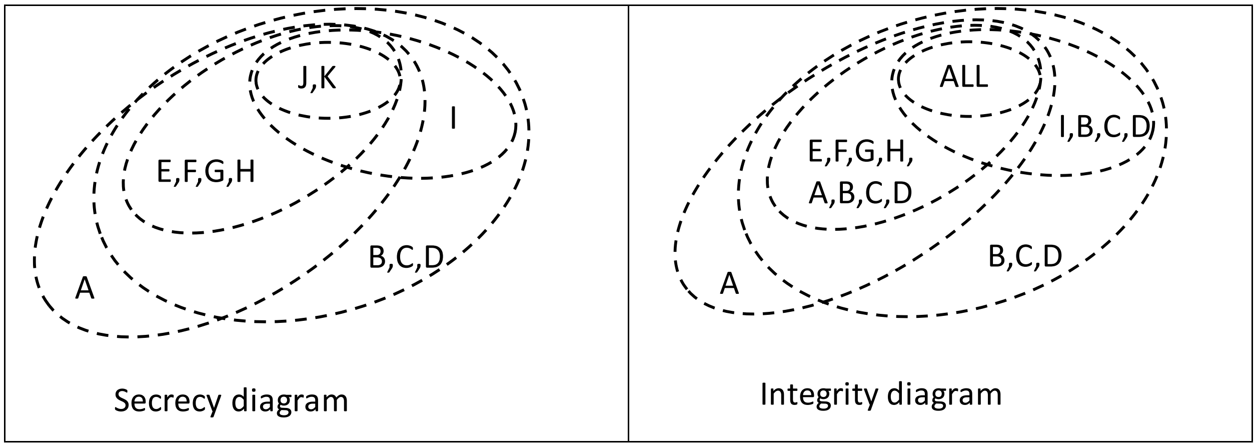


Figure 2: Secrecy and integrity diagrams for the network of Fig. 1.

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; *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 [34,35]; namely in [34] it is shown by example how, given dataflow requirements in an organization, it is possible to assign labels to subjects and objects to obtain a MLS that implements the requirements.

6. Label-based access control and requirements

The use of labels is a time-honored method for assigning entities to security levels. It well predates the Bell-La Padula model [7], 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 used in access control, where a distinction is made between subjects and objects, reading and writing. We reduce these four concepts to the two concepts of entities and channels between entities. We show in this section how several types of label-based requirements can be represented in our theory. 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 are elements of partially ordered *domains*. Domains can be any partially ordered sets. In the examples of this paper, we consider two types of domains:

- Elementary domains, such as security or integrity levels, with partial order relations understood in security theory

- Domain sets, for which the elements can be data *categories* and the partial order relation is set inclusion [11 Sect. 5.2.1].

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 said to be *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 each entity *x* in the network, the function *Lab(x)* is defined in the same uniform set of labels.

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. Entities assigned to the same labels are equivalent. So $[x] \sqsubseteq [y]$ *iff* $Lab(x) \leq Lab(y)$. By Property 1, $CF(x,y)$ *iff* $Lab(x) \leq Lab(y)$. If a label is a simple set, $Lab(x) \leq Lab(y)$ *iff* $Lab(x) \subseteq Lab(y)$, and so $CF(x,y)$ *iff* $Lab(x) \subseteq Lab(y)$. These relations are consistent with the conventional theory of data security, which are based on the *domination* relation.

Partially ordered label sets will be represented by DAGs with hexagonal nodes. Edges represent the $\leq$ relation between labels, and also the $\sqsubseteq$ relation between equivalence classes of entities with those labels.

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 simple domain sets, in fact simple sets of entity names, 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). They can be computed efficiently using strongly connected component algorithms, see Sect. 8.

Note that in such networks one can see, in each entity's label, the *provenance* of the data that can end up in the entity [38,42] (we use here a very basic concept of provenance). We can take an entity name such as *A* in a label as denoting the data category of data originating from the entity labeled *A*.

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}*. This labeling expresses explicitly the safety and integrity of each entity in the network.

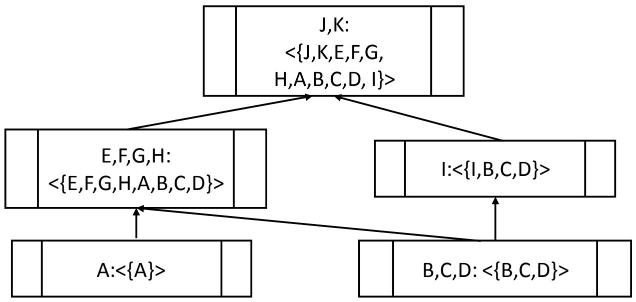

J,K: <{J,K,E,F,G, H,A,B,C,D, I}>
E,F,G,H: <{E,F,G,H,A,B,C,D}>
I:<{I,B,C,D}>
A:<{A}>
B,C,D: <{B,C,D}>

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

The following examples show how conflicts, aggregations, conglomerates, numerical requirements and other types of requirements can be represented with different types of labeling.

Example 4: Conflicts. The typical example of conflicts are policies specifying that no entity is allowed to contain data from organizations in conflict of interest [47,19,24] . Using labels that are simple category sets, ordered under the subset relation, we can say for example that no label is allowed to contain the subset *{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 Example 11, Sect. 7.2. Note that it can be specified that there is no conflict in the presence of other entities, 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 will 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 [22,24], in a different context. 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 Example 11 in Sect. 7.2.

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 [49,22,24,37,13,19]. The following example shows how aggregations can be specified with labels. Consider a network with five levels of security*: 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.

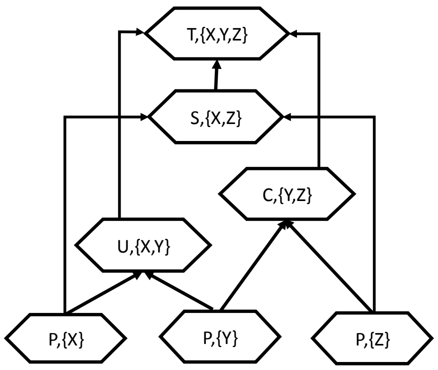


Figure 4: Partially ordered label set for aggregation example

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

Example 8: Simultaneous consideration of secrecy and integrity levels. As above, for secrecy we use two classifications: *Public, Secret*. Let us abbreviate them as *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.

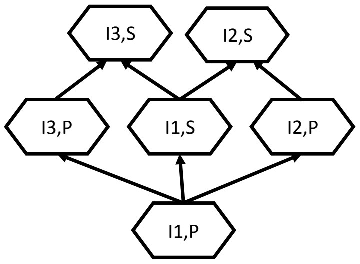
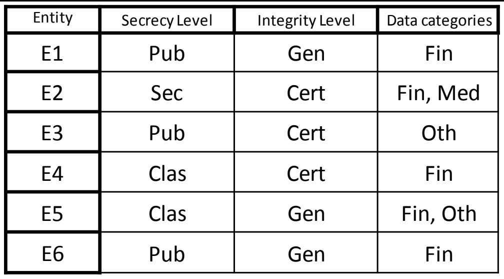


Figure 5: Partially ordered label set for combined secrecy and integrity example

Example 9: Simultaneous consideration of secrecy and integrity levels with data categories. In the following example, we consider not only secrecy and integrity requirements, but also requirements based on data categories. Fig. 6 shows six entities with their labels. We have three secrecy levels ordered *Pub<Clas<Sec* and two integrity levels ordered *Cert<Gen.* We also have three data categories, *Fin, Med, Oth* (*Financial, Medical, Other*). 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 *Other* are visible only to (are a secret of) entities *E3* and *E5*. However *E3* can only get *Public* or *Certified* data of this category, while *E5* can get *Public* or *Classified* data, also *Certified* or *Generic* data in this category.



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

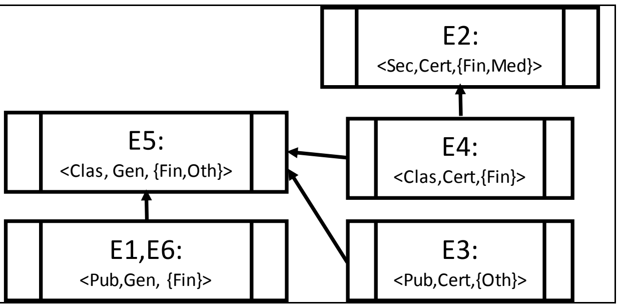


Figure 6: Labels and data flows for security, integrity and categories

Much more sophisticated options are possible, because of the many possibilities of label purposes and combinations. Each of Foley's papers cited above contains a number of examples of various security requirements realized through labeling. Going back to the example given for conflicts above, 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.

The product of different partial orders used above can be uniformed to only one partially ordered set by using the concept of category. In Example 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. So 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. This observation is further justified by the canonical labeling construction, that is general and where each label is a simple category set.

Note that, in all the examples above, the use of the lattice model would require adding unnecessary and even unwanted entities and labels, such as labels containing conflicting data categories. Label-based requirements can be enforced through network transformations by stipulating 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

State changes, i.e. state transitions or *transformations,* can occur in networks for a variety of reasons, the most common of which are user or administrative action, but also 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'. This is now textbook material [11 Sect. 5.3]. At this point, we will only say that for any security network, rigorous policies and auditing mechanisms must exist, establishing prerequisites and checks for safe transformations. Data purging will be mentioned in Sects. 7.2 and 7.3.

We consider an unbounded set of network *states* $N_0, N_1 \ldots$ Each N_i is a network with its entities $x_i, y_i \ldots$, $Channel_i$ and CF_i relations, and partial order $\sqsubseteq_i$. So entities keep their names through state changes but their indexes represent the current state. Transformations are defined on partial orders, where the data flow relations are more clearly represented. Three transformations are considered, as follows:

Definition 7: Network transformations.

We say that

1) *x is added* 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 $[y_{i+1}]$.

Because of the global nature of partial order relations, additions and removals of entities may cause relocations of other entities. Relocations can also be caused by the creation or removal of channels. Relocation of one entity causes relocation of others. Since the result of each transformation is a network, each non-identical transformation yields a new partial order of equivalence classes of entities, with new $\sqsubseteq$ and CF relations. By extension of the previous result, 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.

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.

We do not discuss in this paper who can ask for transformations or what policies might be used to allow or deny them.

Example 10. We go back to Fig. 1, and we take this as our N_0. We relocate entity A to the top and entity I to the bottom of the partial order, giving the partial order of equivalence classes shown in Fig. 7(c). Fig. 7(d) presents a possible implementation of this partial order. It was obtained by the *partial order implementation method 2* of Sect. 4, re-using 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 $i>0$. The secrecy of A has been increased to the maximum but its integrity has been decreased to the minimum. The converse has been done for I. (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.)

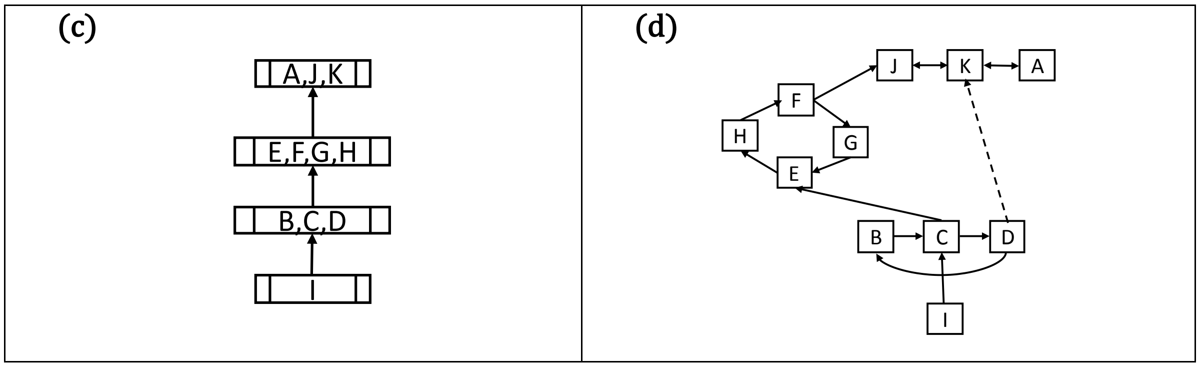


Figure 7: Network transformations through partial orders

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

We assume that the network administrator maintains the *Channel*$_i$, *CF*$_i$ and $\sqsubseteq_i$ matrices for the current state *i*.

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 added, nor about entities that have been removed: the former have no memory of previous flows and the latter cannot pass it on.

The interstate flow relation will be denoted by *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.

Essentially, 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. 4 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*).

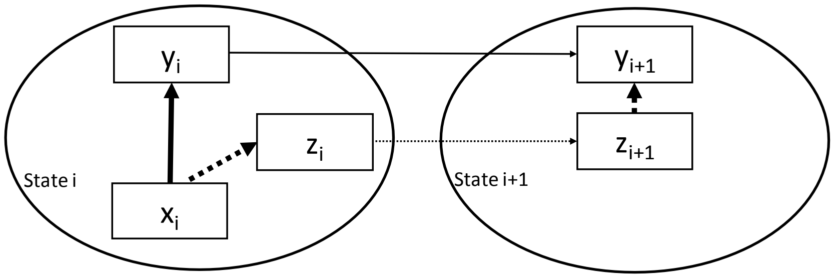


Figure 8: Interstate data flow

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* $[z_{i+1}] \sqsubseteq_{i+1} [y_{i+1}])$

The definition holds not only for distinct x,y,z but also for all combinations of: $x=y$ or $x=z$ or $y=z$.

This does not ensure data 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 entity may have to 'forget' some of the data that might have flown to it earlier. The desired security properties can 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 [38,42]. A survey on this problem is [46], an early discussion on it with theory and examples can be found in [24], paper [44] discusses it in Cloud and IoT contexts, while paper [39] provides interesting examples. We consider only purging here. 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. We will see this in Example 11.

It can be supposed that purging is done instantly in state transitions.

The above discussion is informal and tentative because we are not introducing in this paper a formalism to express policies or requirements of this type.

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, among which [47,19,22,23,26]. For consideration of the Cloud context, see [43]. 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 the result of changes in entity's labels. We have seen that in label-based security networks, the CF relations are determined by the partial orders of label sets. Let us write $Lab_i(x_i)$ to denote the label of x in state i. If $CF_i(x_i,y_i)$ is false but $CF_{i+1}(x_{i+1},y_{i+1})$ must be true, then $Lab_i(x_i) \leq Lab_i(y_i)$ is false and $Lab_{i+1}(x_{i+1}) \leq Lab_{i+1}(y_{i+1})$ must be made true. If labels are simple category sets, then $Lab_i(x_i) \subseteq Lab_i(y_i)$ is false and $Lab_{i+1}(x_{i+1}) \subseteq Lab_{i+1}(y_{i+1})$ must be made true. The reverse reasoning holds for removing flows. Purging policies may require that an entity be purged of all data of the removed categories. This is particularly clear with canonical labeling.

As noted in [23,24,46], on the basis of a defined partial order of labels, the 'high and low water mark' principle [55] is applied. Entities can be allowed to move up and down in the label partial order, which establishes their boundaries, i.e. high and low water marks. Data can flow according to the label partial order. The following example illustrates this.

Example 11. We will see how two types of requirements can be expressed simultaneously by using our method: conflicts and conglomerates, see Sect. 6. In this example, labels are simple category sets, which, as mentioned in Sect. 6, are general. They are partially ordered by the 'set inclusion' relation.

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

1) Separation of data (conflict) has to be observed between the two banks, the two companies, and *Bank2* with *Company 2*. With some obvious abbreviations, these requirements are expressed by forbidding labels containing the following subsets: *{B1,B2}, {C1,C2}, {B2,C2}*. Maintaining this requirement over state transitions will lead to maintaining a policy similar to Chinese wall, i.e. not identical to it as usually defined, but addressing the same concern of making it impossible for an entity to contain data originating from certain combinations of entities, see also Ex. 4 in Sect. 6.
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 reached.

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

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

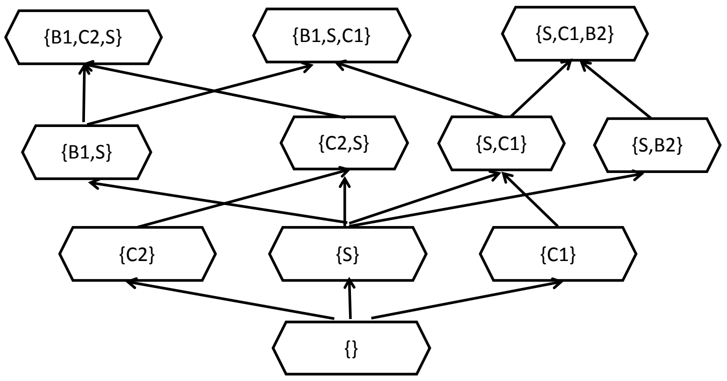


Figure 9. Partially ordered set of allowed labels for Example 11.

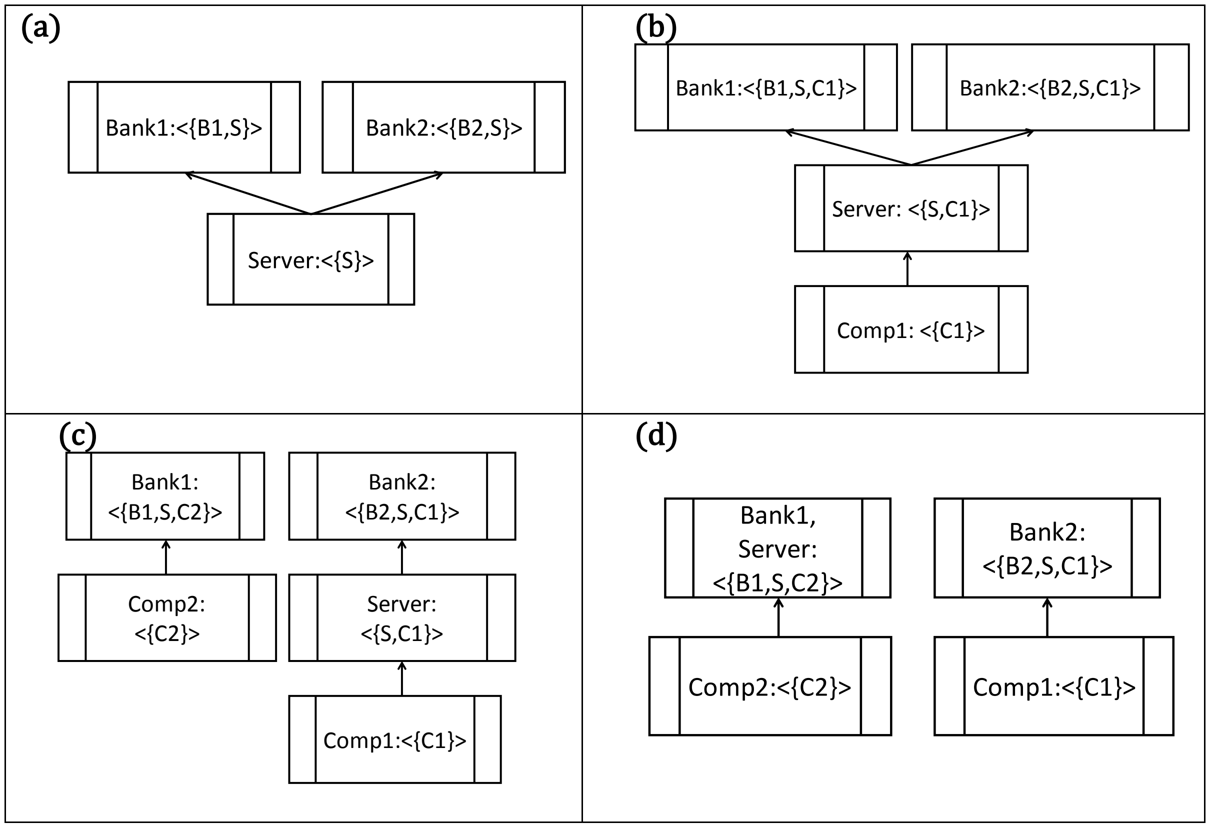


Figure 10: Transformations for Example 11

(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 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.*
(d) It is 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 it. *{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.

The two remaining unidirectional flows can now 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*, and the other containing *Bank2* and *Company2*. This having been done, all entities will be at their high-water mark. They can only evolve by descending.

Note that the label transformations in this example have followed the partial order of Fig. 9, in upwards or downwards directions, and so the system's data security requirements have been respected.

In the above example, 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*, that carries no requirements, 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 Example 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 already mentioned in Sect. 6, many types of requirements or policies can be expressed by using labels, and so this method is powerful, in fact probably more powerful than it has ever been used in practice.

8. Algorithms

As mentioned, we consider the ideal case where there is a central administrator who maintains the *Channel*, *CF* and $\sqsubseteq$ matrices. The useful algorithms, mentioned so far, are:

a) Transitive closure algorithm of a digraph, whose complexity is approximatively $O(n^3)$ [2].
b) Transitive reduction algorithm, which has the same time complexity, in fact it turns out to be essentially the same algorithm as a) [2].
c) Strongly connected component algorithms, such as [54]. These algorithms also yield the partial order of the components that they find. Their complexity is $O(n+m)$
d) Digraph isomorphism algorithms. Several papers exist claiming 'quasi-polynomial' efficiency with various O-formulae where $log(n)$ is in the exponent. The most current survey appears to be [56].

In the above: n is the number of entities, m the number of channels and O is the order of complexity; only time complexity has been considered.

These algorithms can be used respectively for:

a) Obtaining the *CF* relation from a *Channel* relation.
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 in the process of checking whether an implementation is correct with respect to a given partial order digraph, see Implementation methods in Sect. 4.

Research continues in graph algorithms and papers exist claiming complexity measures better than the ones mentioned above. 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. This is not reassuring, because it means that for a network of 10^3 entities, the order of complexity of the algorithm to be executed is $O(10^9)$. But these are all internal calculations in one computer, the administrator's. Simulation runs to solve a closely related problem were given in [53] and were shown to execute efficiently. 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 computational biology and other research areas.

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, maintaining global matrices as mentioned above 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

We have mentioned that, following [14], much of the research on data flow control for security uses the lattice model based on labeling. However, as mentioned at the beginning of Sect. 4, networks are rarely designed as lattices with labeling, also the essential properties of lattices, such as the existence of unique upper and lower bounds, are seldom used to reason about security properties. We have shown that, with the same reflexivity and transitivity assumptions of [14], any network can be seen as a partial order of equivalence classes of entities. The literature mentions that partial orders can be transformed into lattices by adding fictitious and even impossible entities and labels, however this is not necessary, since a data security theory and practical solutions with efficient algorithms can be built for the simpler partial order model, with or without labeling.

Let us take a critical look at our two basic assumptions. 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 in several. Concerning transitivity, much research in security networks makes the transitivity assumption, starting with [14]. 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 security networks have been motivated and studied in several papers, among others in the mentioned work of Foley and also in [45]. 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, see below. This question is related to the question of transitivity of trust in social and other networks [30, 1].

Multi-layer security theory is often concerned with hidden channels. As mentioned, these can be seen as augmenting the *Channel* relation and so modifying a perceived partial order, which is a well-known vulnerability of access control and data flow control systems. Our conceptual framework does not correct these vulnerabilities but can model them.

Long associated with Mandatory access control, label-based security systems and lattices, the theory of Multi-level security has been shown in this paper to be applicable to any reflexive and transitive data flow network, including organizational networks and IoT networks, with or without labelling. Property 1, in its 'lattice' definition [14] is postulated to be a necessary condition for data security, but in fact is true, in our 'partial order' definition, for all networks. The consequence is that secure entities, with respect to both secrecy and integrity, exist and can be efficiently found in any network that has a number of levels that is sufficient to implement the required partial order.

In many organizations, data flow issues are dealt with by specializing data bases and rigidly regulating channels between them. For example, there may be separate data bases for financial data, for personnel data, and so on. In the method we propose, flows can be open or closed according to administrative decisions, for specific categories of data. This enables more flexible organization-wide and time-dependent data flow policies.

We have in this paper developed a formalism to reason about data flows among entities that can contain data. We have not developed a formalism to reason about the data themselves, with their own security requirements and classifications. In fact, much of the existing data flow security and access control theory considers data only implicitly. We have seen that for the placement of data, it is necessary to determine the structure of the network, which can be done efficiently with strongly connected component algorithms, and place data accordingly. 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 security, and the upper the data that must have lower integrity and higher security. We have mentioned that network transformations may require policies to purge or declassify data, but we have not formalized these.

This theory would be inadequate if it allowed dataflows in one direction only. Several partial orders can be defined to coexist for a given set of entities at each state. Paper [35] presents an 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, probably by introducing the concept of *trusted entity,* which has a long history in data security [7,39]. For our purposes, trusted entities could be defined to purge, to hold or to transform certain data when their position in the network's partial orders changes. Certain data flows through such entities may have to be defined to be intransitive, contradicting for them the hypothesis mentioned above. Different policies will have to be used for different applications.

The practical usefulness of our method can also be questioned in cases where data flow relations can change rapidly over time [36], as can happen for example in Attribute-based access control (ABAC) and telecommunications networks. However it will be always true that at each state there will be a partial order of components that establishes certain data flow relations. Hence it is necessary to determine what data flow relations must be kept invariant and plan transformations accordingly [26]. For example, entities designated to contain the most secret data should remain at the top layer, while entities representing data gathering devices such as detectors or sensors should probably be kept at the lowest layer.

Research is necessary on these points.

10. Conclusions

As indicated in the literature review section, the bases for the theory discussed in this paper have been known for decades, however in this and previous papers we have presented the novel view that, under reasonable assumptions, all networks can be seen as intrinsically layered and thus layering for data security always exists and can be found. The early theory has therefore a reach that was unsuspected by its authors, since it acquires significance beyond lattices and labels. The main contribution of this paper is to present a simplified and coherent conceptualization for these ideas, with the concepts for its application in several areas, such as organizational data security systems and in the Internet of things or the Cloud. Few papers exist with results in the topic of data security in the IoT, and we hope to have shown that our approach generalizes and simplifies previous findings. Labeling methods are described and it is shown how, in the framework of this theory, they can model well-established data security paradigms, such as secrecy, integrity, conflicts, conglomerates, aggregates, as they could in the lattice-based theory. We have also shown how it is possible to describe network state transitions involving creation, removal and relocation of entities. Finally, it has been shown that efficient algorithms to implement this theory and methods exist, which opens the door for software tools to support several of the design and administration activities mentioned in this paper.

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, Abdel Stambouli, for useful discussions, and to his colleague Jurek Czyzowicz, always ready to answer questions on partial orders and algorithms.

References

1. 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. 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
6. J.Bang-Jensen, G.Z.Gutin. *Digraphs – Theory, algorithms and applications.* Springer, 2nd Edition, 2009.
7. D.E. Bell, L.J. La Padula. Secure computer systems: unified exposition and Multics interpretation. MTR-2997, Mitre Corp., Bedford, Mass., 1976.

8. 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.
9. K.J. Biba, Integrity considerations for secure computer systems. TR-3153. Mitre Corp. Bedford, Mass., 1977.
10. G. Birkhoff. *Lattice Theory,* American Mathematical Society, 1967.
11. M. Bishop. *Computer security, Art and science*. Addison-Wesley, 2003.
12. 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.
13. F. Cuppens. A modal logic framework to solve aggregation problems. In: Database Security V: Status and Prospects, North-Holland,1992, 315-332.
14. D.E. Denning. A lattice model of secure information flow. Comm. ACM 19(5), 1976, 236-243.
15. D.E. Denning, P.J. Denning. Data Security. Computer Surveys, 11(3), 1979, 227-249.
16. D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli. *Role-based access control.* 2nd Ed. Artech House, 2007.
17. S.N. Foley. A universal theory of information flow. Proc. 1987 IEEE Symp. on Security and Privacy, 1987, 116–121.
18. S.N. Foley. A model for secure information flow. Proc. Symp. on Security and Privacy, 1989. IEEE Comp. Soc. 248-258.
19. S.N.Foley. Secure information flow using security groups. Proc. Computer Security Foundations Workshop III, 1990, 62-72.
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. Reasoning about confidentiality requirements. Proc. Comput. Security Foundations Worksh. VII, IEEE, 1994, 150-160.
26. 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.
27. R. Fraïssé. *Theory of relations.* North-Holland, 1986.
28. F. Harary, R.Z. Norman, D. Cartwright. *Structural models: an introduction to the theory of directed graphs.* Wiley, 1965.
29. V.C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and considerations. NIST Special Publication 800-162, 2014.
30. 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.

31. 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.
32. 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.
33. 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.
34. 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.
35. 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.
36. 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.
37. C. Meadows. Extending the Brewer-Nash model to a multilevel context. Proc. 1990 IEEE Symp. on security and privacy, IEEE Computer Society, 95-102.
38. L. Moreau. The foundations for provenance on the Web. Foundations and trends in web science. 2(2-3), 2010, 99-241
39. A. C. Myers, B. Liskov. Protecting privacy using the decentralized label model. ACM Trans. on Software Eng. and Methodology, 9(4), 2000, 410-442.
40. N.V. Narendra Kumar, R. Shyamasundar. Realizing purpose-based privacy policies succinctly via Information-Flow Labels. Proc. Big Data and Cloud Computing (BDCloud'14), 753-760.
41. S. Osborn. Information flow analysis for RBAC system. Proc. 7th ACM Symp. on Access control models and technologies (SACMAT '02), 163-68.
42. 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.
43. 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.
44. 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.
45. J. Rushby. Noninterference, transitivity and channel-control security policies. Technical Report, SRI International, May 2005.
46. A. Sabelfeld, D. Sands. Dimensions and principles of declassification. Proc. 18th IEEE Computer Security Foundations Worksh. (CSFW'05), 255-271.
47. R.S. Sandhu. Lattice-based enforcement of Chinese Walls. Computers & Security Vol. 11(8), 1992, 753-763.
48. R.S. Sandhu. Lattice-based access control models. IEEE Computer 26(11), 1993, 9–19.
49. R.S. Sandhu, S. Jajodia. Data and database security and controls. Handbook of Information Security Management, Auerbach Publishers, 1993, 481-499.

50. R.S. Sandhu. On five definitions of data integrity. In Database Security VII: Status and Prospects, North-Holland, 1994, 257-267.
51. 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.
52. 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.
53. A. Stambouli, L. Logrippo. Data flow analysis from capability lists, with application to RBAC. Information Processing Letters, 141(2019), 30-40.
54. R. E. Tarjan. Depth-first search and linear graph algorithms, SIAM Journ. on Computing, 1(2) (1972), 146–160.
55. C. Weissman. Security controls in the ADEPT-50 time sharing system. Proc. 1969 AFIPS fall joint computer conf. AFIPS Press, 119-133.
56. Wikipedia. Graph isomorphism problem. Consulted December, 2019.