Configuring data flows in the Internet of Things for security and privacy requirements[1]

Luigi Logrippo, Abdelouadoud Stambouli

Université du Québec en Outaouais
Department of Computer Science and Engineering
Gatineau, Québec, Canada
luigi@uqo.ca, staa16@uqo.ca

Abstract. The Internet of Things is a highly distributed, highly dynamic environment where data can flow among entities (the 'things') in complex data flow configurations. For data secrecy, it is important that only certain data flows be allowed. Research in this area is often based on the use of the well-known lattice model. However, as shown in previous papers, by using a basic result of directed graph theory (or of order theory) it is possible to use a less constrained model based on partial orders, for which a formal notion of secrecy can be defined. We define a notion of 'allowed contents' for each 'thing' and then the data flows follow by inclusion relationships. By taking advantage of transitivity of data flows and of strongly connected component algorithms, these data flow relationships can then be simplified. It is shown that several data flow relationships can coexist in a network. Two small examples are presented, one on hospital applications and another on e-commerce. Implementation issues are discussed.

Keywords: Internet of things, data secrecy, data confidentiality, privacy, data flow control, partial orders.

1 Introduction and motivation

Given that we have a network of *entities* representing an abstract view of an Internet of things (IoT) network, how can we set up the data communications channels between entities so that data originating in one entity can or cannot reach another entity? Being able to answer this question is important to answer questions of:

- *Secrecy* (also called *confidentiality*): can data stored in an entity reach another entity? This question has clear consequences for the question of *privacy*, which will be implied henceforth.
- *Integrity*: can data originating from an entity at some level of integrity reach an entity at higher level of integrity, thus potentially polluting it?

[1] Presented at the 11th International Symposium on Foundations and Practice of Security (FPS 2018). Montreal, Nov. 13-15, 2018. Springer LNCS Vol. 11358, 115-130.

- *Availability*: can an entity always access the data it needs?

It is of course a basic requirement for the IoT that "data should be able to flow as needed" with as little confinement as possible, but also "data should not flow to unauthorized parties" [22]. With respect to privacy, we agree that "the fundamental nature of a privacy violation is an improper information flow" [13]. Paper [23] takes a broad view of the importance of information flow control to achieve legal obligations in the IoT. Many IoT diagrams in the literature show bidirectional channels among entities, however clearly for secrecy and privacy some channels may have to be unidirectional or absent altogether. These problems are common between the IoT and Cloud, since IoT is often implemented on Cloud platforms [10] [11] [22].

We propose in this paper a new method to design networks with data flow topologies (i.e. configurations of entities and channels) that can satisfy secrecy requirements as specified in terms of logic expressions. We will also mention why we believe that integrity requirements are also addressed by the same method. Related important questions that we discuss in this paper are the following: given certain data flow relationships in an IoT system, and the fact that we know that certain data originate in certain entities, which are the entities that will be privy to these data? Which are the most secret or least secret entities, in the sense that data that are in them cannot or can propagate to others?

As already discussed in [14] and [24], it turns out that a simple result of directed graph theory, associated with well-known efficient algorithms, can be used to provide solutions for this problem, which are generic, i.e. independent of the application, or of the devices used to implement the entities, or of the physical network. These papers did not explicitly consider the IoT, which will be the focus of this paper.

2 Literature review

Although the concept of Internet of Things is not much older than our century, the literature on the general subject of 'security in the IoT' is already extensive, and several survey papers exist. However most of this literature is about attacks, vulnerabilities, access control, and is not particularly related to the problem of data flow control for security. We are interested in globally controlling all the possible 'things' where the data of certain other 'things' can end through sequences of data transfers, while access control controls data transfers between pairs of 'things'. In this brief literature review, we cite only papers that are closely related to our problem and proposed solution.

An extremely influential pioneering paper on the general subject of data flow in programs and networks is the one by Denning [4]. It showed that by using principles of lattice structuring, data flow security properties can be guaranteed in programs or networks. Almost all papers cited here refer to the lattice model by proposing applications, variants and enhancements of it. Our model has in common with the lattice model the fact that it is relational, rather than state-transition based. It generalizes Denning's model because it uses partial orders instead of lattices. In [14] it is shown that partial orders are necessary and

sufficient for data flow secrecy and that they always exist. In [24] it is shown that they can be efficiently found.

It should be noted that the subjects of flow control in the IoT and in the Cloud are closely intertwined and on the way to integration [2]. Many papers in this general research area propose the use of authentication, encryption and access control methods, including variations of RBAC, in the IoT. Many of these papers are reviewed in [16]. Although authentication, encryption and access control are mechanism for realizing flow control, we will limit our consideration here to methods for designing the overall flow control.

Much classical literature, such as the paper of Samarati et al. [19] deals with the problem of preventing or blocking illegal flows. Our purpose is dual, i.e. to identify (in a current configuration) or permit (in a configuration to be established) all legal flows.

Blackstock and Lea [3] address the need for IoT data flow platforms to create "systems suitable for executing on a range of real-time environments, toward supporting distributed IoT programs that can be partitioned between servers, gateways and devices". They describe their experiences with two existing data flow platforms towards designing their own.

Narendra Kumar and Shyamasundar [15] use a formalism based on identifying separate *subjects* and *objects* (rather than *entities* only) and separate reading and writing authorizations, rather than on a single *CanFlow* relationship as we do. They define a Readers and Writers Flow Model based on Denning's lattice model. Each entity is provided with a label defining the entities that can read from it and the entities that can write on it. This paper is in the context of Cloud computing. In a very recent follow-up paper [12], Khobragade and the two authors just cited extend their method to the IoT. We will come back to these papers in the Conclusions.

Bacon et al. [1][22][18] have developed data flow control methods and software for the Cloud and the IoT using data tagging. We agree that data tagging seems to be necessary for configuring data flows. as we will see later in our paper.

Schütte and Brost [21] present a policy language, LUCON, designed to control the routing of messages across services. Message routes can be model-checked to see whether they violate policies. The method uses message labels to which policies refer in order to decide what happens to the messages during routing. Again, this kind of labelling will play a role in our model.

These papers agree on the fact that generic solutions for IoT data flow control exist, and should be used before application-specific ones. We share this opinion.

Although IoT networks are usually represented as directed graphs (digraphs), we could not find a single paper that references or uses the basic result of digraph theory that is presented in the following section, and which is the basis of our method. The use of this result, instead of the classical lattice model, is the salient distinguishing characteristic of our approach.

3 Basic concepts

This section is mainly an adaptation to the IoT context of results presented in [14][24]. We consider sets of abstract *entities* that can communicate among themselves by unidirectional abstract *channels* (bidirectional arrows in our diagrams mean that there are two channels in opposite directions). Entities represent 'things' or objects such as sensors, databases, etc. Each entity has computing and storage power, can also have sensing capabilities. We call *network topology* a given set of entities with channels between them, which is fixed at any network *state*. We use the letter *e* with primes and subscripts to denote variables for entities. We write $CF_1(e,e')$ (*can flow*) to say that there is a channel that can carry data from *e* to *e'*. In practice, CF_1 can be implemented in several ways, this will be discussed later. We write *CF(e,e')* if there is a communication path, consisting of one or several channels, from *e* to *e'*. If the data of *e* is encrypted so that *e'* cannot decrypt it, then the relation is false.

Definition 1: *CF(e,e')* is true if:

a) $CF_1(e,e')$ or
b) there is a *e''* such that $CF_1(e,e'')$ and *CF(e'',e')*.

So *CF* is a *transitive* relationship. This is a pessimistic view, which can make networks over-protected; it ignores the fact that some entities may decide to block some data. Our (perhaps simplistic) view is that if Alice talks to Bob and Carl talks to Alice, Carl can expect whatever he says to end up with Bob. We also assume that *CF* is *reflexive*, since we can assume the existence of a channel from any entity to itself, although for simplicity such channels will be left implicit. It is important to note that, by its transitivity and reflexivity, *CF* is a *quasi-order* [7]. The relation *CF* will be shown in the form of directed graphs (or *digraphs*). To enhance this generic view, in Section 6 we shall informally introduce the notion of several separate data flows.

We say that an entity *e can hold* data *x*, written *CH(e,x)*, iff data item *x* can be present in *e*. *CH(e,x)* can be a fact known a-priori, an axiom. For example, a sensor in a refrigerator can hold a temperature reading. In other cases, *CH(e,x)* can be a derived fact, if there exists a *e'* such that *CH(e',x)* and *CF(e',e).* So, if there is a channel from the sensor to a Home-Computer (HC), then the HC can also hold the temperature reading.

It would be possible to continue in the same way, considering the level of granularity of single data items; however in this paper we won't need to reason at this fine level of granularity. Henceforth we use the notation *CH(e,e')* to say that entity *e* can hold the data in *e'*. In the example above, if there is a channel that can carry data from a sensor in the refrigerator to the HC, we say that both the sensor and the HC *can hold* all data in the sensor. In our examples below we will construct networks by using a reverse principle, i.e. starting from the fact that HC can hold all data in the sensor, we conclude that there is a flow, or a channel, from the sensor to the HC.

Formally, we write:

Definition 2:

a) *CH(e,e)* is our axiom
b) *CH(e,e') iff CF(e',e)* is our inference rule

The following definition will allow us to refer to all the data that can be contained in an entity, given a data flow configuration:

Definition 3:

CHS(e) = {e' such that CH(e,e')}

CHS stands for *CanHoldSet*. For example, if there is a path from a fridge to a HC, and from a thermostat to the same HC, then the entity HC can hold temperatures from both the fridge and the thermostat.

Clearly, we have:

Property 1:

a) *CF(e,e') iff CHS(e)⊆CHS(e')*

b) *CF(e,e') and CF(e',e) iff CHS(e) = CHS(e')*

These definitions could appear to be counterintuitive at first because it might be thought that two different entities could be able to acquire the same data directly and independently, consider for example two independent sensors that sense the same conditions. When this is possible, it is still safe to assume, from the security point of view, that there is a bidirectional channel between the two entities.

We now describe the basic result of digraph theory that is at the basis of our method. This result also appears, in simpler form, in the theory of relations and in the theory of orders [7], but in our research area the graph-theoretical view may be the most useful, and so we adapt here the theory presented in [9]. We define *components* of our digraphs to be sets of entities such that for any two entities *e* and *e'* in the set, *CF(e,e')* and *CF(e',e)*. By Property 1, *CHS(e)=CHS(e')*. We are interested only in components that are *maximal*, i.e. they are not contained in larger components, so henceforth the adjective maximal will be implicit when we will mention components. Let us *condense* each component in a single node in the digraph. Since the original *CF* digraph represented a quasi-ordering, the resulting condensed digraph represents a *partial order* [7]: this is because all symmetric relationships have disappeared, having been encapsulated in nodes. Let us call *[e]* the node corresponding to the component containing entity *e* (clearly,the mapping $e \rightarrow [e]$ is a function). It is easily seen that there is a path (a *CF* relationship) from *e* to *e'* in the original digraph iff there is a path from *[e]* to *[e']* in the condensed digraph [9].

There are well-known and efficient (linear-time) algorithms to find condensed digraphs, and their use will be demonstrated in this paper. It is important to note that such condensed digraph will be acyclic. We will call such algorithms *strongly connected components algorithms*. We use Tarjan's algorithm [25] as implemented in MATLAB [8].

As a generalization of the above notation, we allow specifying flow relationships between sets of entities. For $\boldsymbol{S}$ and $\boldsymbol{S'}$ finite sets of entities, *CF(**S**,**S'**)* means that *CF(e,e')* holds between *each* $e \in \boldsymbol{S}$ and *all* $e' \in \boldsymbol{S'}$. The sets can be specified either by enumeration, or by set-theoretical expressions, based on the attributes of entities as we will see. For example, if $\boldsymbol{S}$ is a set of patients having a specific illness and $\boldsymbol{S'}$ is a set of doctors that specialize in that illness, then *CF(**S**,**S'**)* means that data can flow from each patient in $\boldsymbol{S}$ to all doctors in $\boldsymbol{S'}$.

So entities *e* have named attributes, in variable numbers and according to application needs. We assume that each entity has attributes according to application needs. Hence the constraint above can be specified as follows:

CH*(e,e') **if** patient(e) **and** illness e=stroke **and** doctor(e') **and** specialty(e')=stroke*

By Propery 1, this implies *CF(**S**,**S'**)*, where ***S*** is the set of all such patients and ***S***' is the set of all such doctors.

We refine this view by allowing *CF* relationships to be specialized by the type of data involved. We shall see later an e-commerce example where there are two different *CF* relationships, one for ordering and one for billing. Many different *CF* relationships can coexist in a network. In real life, Alice might talk to Bob on work matters, but not on private matters. Knowing this, Carl can talk to Alice on private matters, assuming that this will not end up with Bob.

Note finally that we use the term *component* in the described graph theoretical meaning. This is quite different from the term's use in some IoT literature, where it can denote hardware entities with specific physical characteristics. In our sense, an entity that belongs to a singleton equivalence class is also a component. On the other hand, entities that are physically identical can belong to different components in our sense. We use the term *device* to refer to components in this other sense.

4 The method

Our method is based on the idea that *if an entity* e *can hold all the data that an entity* e' *can hold (plus possibly other data) then* CF(e',e) *should be true*. So each entity will have attributes and will be associated with a logical expression, based on the available attributes, defining the set of data that it can hold. The channels will be placed by calculating the inclusion relationships between these sets. This can be done dynamically, in the sense that every time a new entity is defined, the channels it should have can be calculated, by checking the data set inclusion relationships between the new entity and the existing ones (this step is not trivial from the point of view of computational complexity, and will be discussed in future papers).

The method described so far may be impractical, since it might generate 'too many channels'. In terms of digraphs, its result can be visualized as a transitively closed digraph, see Figure 1a) for an example. According to the properties presented in the previous section, a streamlined digraph can be obtained in the following way:

1) Using a strongly connected components algorithm, calculate the compressed digraph for the *CF* relation (recall that it will be acyclic).
2) Calculate the transitive reduction of the compressed digraph, see Figure 1 b) where each box represents a component (often a single algorithm will do both 1) and 2).
3) For each component in this last digraph, connect the entities (if they are more than one) in any way that maintains the mutual reachability relation;
4) For any two components ***S*** and ***S'*** such that ***S*⊆*S'***, connect one or more element(s) in ***S*** to one or more element(s) in ***S'*** (Figure 1c). By transitivity, *CF(**S**,**S'**)*.

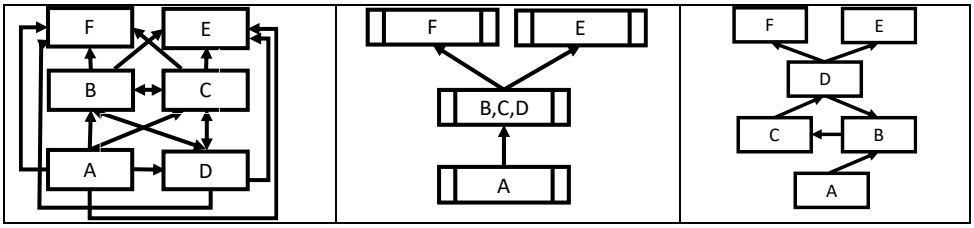

Figure 1a): A data flow graph; b) Its partial order of components; c) An equivalent data flow graph

In practice data flow graphs may be large and complex, with component nesting difficult to unravel, but each of the algorithms involved is (at most) polynomial, thus 'efficient' in complexity theory terms.Therefore, it might be conceivable to re-execute the procedure every time there is a change in a network, or periodically, whenever a network reconfiguration is desired.

Note that several communication paths that are direct in Figure 1a) are indirect by transitivity in Figure 1c).Our method may produce communication patterns inappropriate for the intended application, but it won't produce any that violate secrecy constraints. Step 3) can be done in different ways: in Figure 1 we implemented the goal of reducing the number of edges by using transitivity, but other goals can be implemented. Therefore, we propose the use of this method as a basic method only, that may be adapted to the needs of specific networks.

The use of this method will be assumed in the rest of the paper. The theory above will be used implicitly.

5 Network creation: A hospital example

We use here small examples to demonstrate our method, but as mentioned this is scalable because of the existence of efficient algorithms.

As a first example, we consider a toy hospital system. In its final configuration, the system will be as in Figure 2, however we will show how it can be built step by step.

The types of the entities are: *PressDetect, PulseDetect, NurseWkstn, DocWkstn, ReanimationWkstn,WardDB, ChiefMedicWkstn, AdminDB* (in Figure 2, numerals are added to the type names in order to distinguish different instances). We have three patients, *Sam*, *Bob* and *Sally*, which however do not appear as entities but as parts of the labels of the entities. In other words, we have labels: *SamPress, BobPulse, SallyPress* etc. We also have some statistical data that are created in some entities. When an entity of one of the mentioned types is created, it is associated with a label, which indicates the data that it can hold. We use a command *New* to create a new entity with a label. A channel, which is a *CF*

relationship, is automatically created between the new entity and all the previously created entities such as the label of the new entity is included in the label of the previously existing entity. Then the method of Section 4 can be executed to reduce the edges if possible. For example,

New(A) = Nurse1Wkstn{SamPress, BobPulse, Stats1}.
New(B) = Nurse2Wkstn{SallyPulse,Stats2}.
New(C) = Doc1Wkstn{SamPress, BobPulse, Stats1}.
New(D) = Doc2Wkstn{SallyPulse,Stats2}.
New(E) = Ward1DB{SamPress, BobPulse, Stats1}.
New(F) = Ward2DB{SallyPulse,Stats2}.
New(G) = ReanimationWkstn{SamPress, BobPulse, SallyPulse}.
Etc.

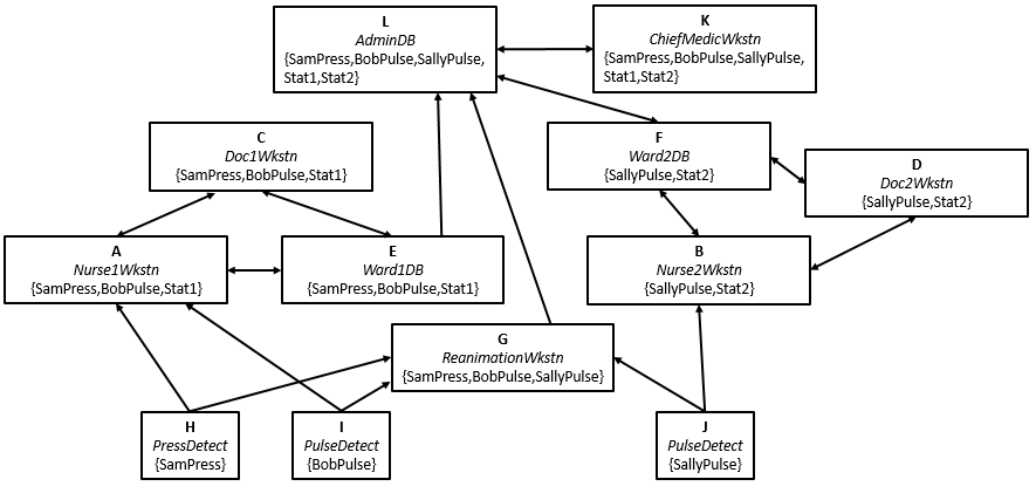

Figure 2: A hospital example

After *C* is created, we have *CF(A,C)* and *CF(C,A)* since *CH(A)=CH(C).* Similarly for *D* and *B.* After *H* is created, we have *CF(H,A), CF(H,C)* etc. So channels are created between entities as the entities are created. In the graph of Figure 2, we have placed the entities in ascending order of inclusion, starting from those that have the smallest *CHSs* at the bottom. Certain things of interest can be seen: for example, we say that *BobPulse* is a *secret of* the set of entities *{I,G,A,E,C,L,K}* which are the only entities that *CanHold* this data (in terms of [14] [24] this is the *Area of BobPulse*). It can also be said that entities that appear towards the bottom of the partial order are the least secretive, because they allow their data to flow

up to other entities. On the contrary, entities appearing at the top *{L,K}* are the most secretive, because their data cannot flow further. They are also the entities that can hold the most data, in fact they can hold all data available in the network in this example.

Note that there are some non-singleton components in this digraph, they are *{A,C,E},{B,F,D},{L,K}*. In each such component, the entities are mutually reachable and so they can all hold the same data. The partial order of equivalence classes for this digraph is shown in Figure 3.

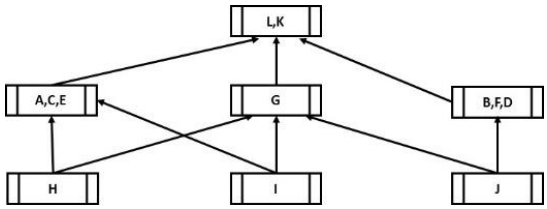

Figure 3. Partial order of components for the hospital example

Note also the following important point. Suppose that for some reason, the entities are created in the following order: first *H*, then *L*, then *C*. In practice this might lead to a misconfiguration and there might be application-specific protocols to prevent this from happening. Without this, our method will produce the following results. After *L* is created, data can flow from *H* to *L*, and after *C* is created, data can flow from *H* to *C*, also from *C* to *L*. At this point, transitive reduction will eliminate the direct flow from *H* to *L*. Secrecy constraints will never be violated and whatever order is followed in the creation of entities, we will always end up with the network depicted in Fig. 2.

6 Separate data flows: An e-commerce example

We introduce now a second example, where for readability we have done some simplifications of notation with respect to the previous one (for example, we don't explicitly show that each entity contains its own data). This is an e-commerce network with four clients, two retailers and four suppliers. Client 3 and 4 collaborate and so they share data. We have two retailers. Finally, we have four suppliers, of which the first three collaborate and so share client data. After having created all the entities, the network is as shown in Fig. 4.

.

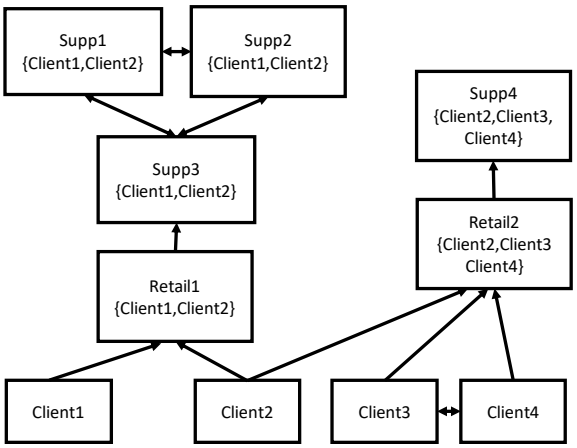

Figure 4. An e-commerce example

The partial order of equivalence classes for this example is shown in Figure *5*.

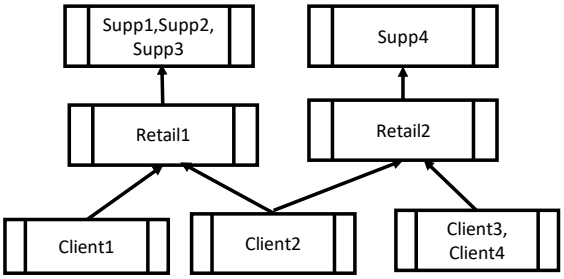

Figure 5. Partial order of components for the e-commerce example

Again, this example can be analyzed to see what data are secret of which entities.

This example is useful to show the (usual) necessity of having several coexistent but separate data flows in the same network. The previous diagrams dealt with ordering information. Billing information travels in the opposite direction, and has different secrecy requirements. Since each client should get only its own bills (except perhaps for Clients 3 and 4 who share bills), then this requires defining as many separate data flows as there are clients. Figure 6 shows the data flow for the bills of *Client 1* (in this case we show a downward flow for consistency with the previous figure). To keep the two flows separate, we can identify the label sets that are relevant for each flow. For example, the labels for *Supp1* could be as follows: *Supp1:Order{Client1,Client2};Bill1{Bill1-1};Bill2{Bill1-2}*. This

means that *Supp1* participates in three data flows, one for ordering and two for billing, for each of its two possible clients. This starts to be complex, but seems to be necessary for the secrecy of bills.

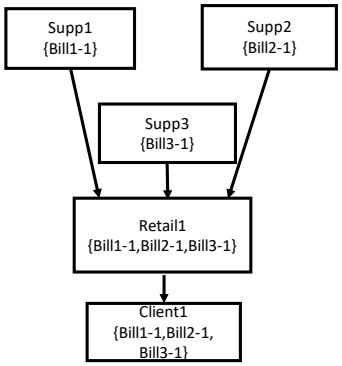

Figure 6. Partial billing flow in the e-commerce example

7 Network re-configurations

We must allow for data flow changes that can be requested by end users or administrators, entities which we see as external to our networks. These changes must be in some way approved by authorities in charge of protecting security and privacy in the system. Often one such change will motivate others in order to maintain useful dataflows. In Fig. 4, suppose that there is a request by which *Client1* should be allowed to subscribe to *Retail 2*, and the request is granted. Then *Retail2{Client2,Client3,Client4}* becomes *Retail2{Client1,Client2,Client3,Client4}*. *Retail2* can no longer flow to *Supp4*. So another authorization seems to be necessary for *Supp4{Client2,Client3,Client4}* to become *Supp4{Client1,Client2,Client3,Client4}* and *Supp4* or some conflict of interests among clients may refuse this second change. Therefore, the requesting and granting of authorizations cannot be purely local, it must be done with a global plan to do all the other changes that may be necessary to return to a desired global flow.A method to avoid such problems would be to deny authorizations unless it is possible to grant at the same time all others that are necessary to maintain the required flow structure. All such authorizations would have to be granted at the same time. The mechanisms for these label changes will vary according to the nature of the system. In almost every system there will be incompatibilities that cannot be violated, e.g. if two clients are in conflict of interest, it must be impossible for each of them to hold secret data from the other, or even for a third party to jointly hold their secret data. Label changes that lead to such combinations must be refused.

From the point of view of access control theory, it is interesting to note that such transformations are essentially the same that are implicit in classical mechanisms such as ‘High

water mark' and 'Chinese wall'. The former is the attribution of new authorizations and the latter is the preclusion of certain combinations of authorizations [20].

We leave entity disappappearance, removal of authorizations, declassification of data, etc. to further research [18][12]. In some cases, local repairs may be possible, and at worst a global reorganization according to our method might be necessary.

8 Towards a language for IoT secrecy requirements

Clearly, it is necessary to have a language for defining abstract entity types and allow the creation of different topologies of instantiated entities, with different allowed contents. We will in this section give an idea of how such a language could be constructed, although it has the potential of becoming quite complex. For ease of use, this language might have to provide for the definition of entities that are not devices or 'things' but can be instrumental in defining attributes of 'things', such as wards, nurses, doctors, patients and clients, etc
Essentially, the language must provide:

- primitives to define entity *types* with attributes, such as entity Ward, entity Supplier etc. We will distinguish two types: types for real 'things' in the network, and they will prefixed by capital T, and types for logical concepts used to define the attributes of the 'things'. These will be prefixed by a capital L.
- an operator to define *New* entities with given attributes, and one to *Dismiss* them
- operators to *Add* or *Remove* attributes from entities
- primitives to define constraints for data flows; below we have simply a *CH* relationship as we will see.

In our hospital example, we could have the following types:

LType Patient(PatientId)	(to define logical type Patient)
TType PressDetect(DetectId)	(to define a device PressDetect with a DetectId)
TType PulseDetect(DetectId)	(to define a PulseDetect)
LType Ward(WardId).	
LType Nurse(NurseId)	
TType NurseWkstn(WkstnId)	

etc., and the following operators:

Assign (DetectId,Patientid)	(to assign a detector to a patient)
Assign (PatientId, WardId)	(to assign a patient to a ward)
Assign (NurseId,WardId)	(to assign a nurse to a ward)
Assign (WkstnId,WardId)	(to assign a workstation to a ward)
Etc.	

We need also a number of *CanHold* definitions, which generalize the previously introduced labels, such as the following one:

CH*(WkstnId,DetectId)* ***if*** *Assign(PatientId,WardId(WkstnId))* ***and*** *Assign(DetectId,PatientId)*

The network construction can start as follows:

New Ward (Emerg).
New NurseWkstn(EmergWkstn)
New Nurse (Alice)
Assign (Alice,Emerg).
Assign (EmergWkstn,Emerg)
New Patient (Sam).
Assign (Sam,Emerg)
New PressDetect(PRD0001)
Assign (PRD0001, Sam)
Etc.

Now, by the *CH* definition we know that *EmergWkstn* can hold the data in Sam's pulse detector *PRD0001* and so data can flow from the latter to the former. This establishes a *CF* relationship, hence a channel. So we have created a network with two physical devices and a channel, and the procedure presented in Section 4 can be executed, although of course it won't find anything to improve.

In some systems there can be many more *CanFlow* requirements than *NoFlow* requirements, or many more *CanNotHold* requirements than *CanHold*. in fact the negative requirements could be more obvious for the designer than the positive. One could allow the designer to specify the negative requirements, and then the positive ones could be found by complementing the negatives. Another possibility would be to allow the designer to specify both positive and negative requirements, but this would require a system to detect and correct inconsistencies. We leave these issues for future research.

9 Implementation issues

The conventional way to enforce data flows is by enforcing access controls on the individual channels. The literature on techniques available for this is extensive, and one recent comprehensive paper with a good literature review is [16].

Tags allow deciding whether certain data can cross certain channels; they must follow the data as they move in the network. Data tagging for access control and flow control has been studied in the literature [1] [5] [18], as well as provenance tagging [17] but they are not part of widely implemented access control method, because these consider tags only for subjects and data objects (such as databases). In order to implement our method, data must be tagged in two ways: to determine what flow the data belongs to (ordering, billing) and to determine the data's provenance (Pressure detector, Client1 …).

It is likely that implementations of our method will require a combination of routing and encryption. Routing will be based on the tags and then the question that comes up is how to combine our method with IoT routing methods.

The Routing Protocol for Low-Power and Lossy Networks (RPL) is a protocol defined by the Internet Engineering Task Force (IETF). It is one of the best-known protocols for routing in the IoT [26], and it supports ad hoc configuration. RPL uses for routing DODAGs (Destination-Oriented Directed Acyclic Graphs). DODAGs describe efficient routes between the sink and other nodes for both collecting and distributing data traffic. DODAGs are usually constructed on the basis of criteria of transmission efficiency called OF (Objective Functions). New entities will autonomously find their place in the network by using

OFs. The ideas presented in this paper may lead to research on methods for combining our own acyclic data flow digraphs with the DODAGs, thus including data flow constraints in RPL routing, hence possibly in Objective Functions.

Encryption appears to be necessary to establish channels that go through nodes that should not be able to read the data.

Clearly, implementation issues require further research.

10 Discussion

Although one of the basic requirements IoT is that the system topology should be very dynamic and varied, at present and for the foreseeable future, the IoT is a vast collection of customized subsystems, each with its own set of users, data sets and functionalities, as well as data security requirements: e.g. hospital networks for hospitals, home networks for homes, warehouse systems for companies, fleets for truck companies, etc. Each subsystem will have its own specific organization and data flows. In specific networks, entities are organized according to these structures, and new entities that enter a network must join pre-extisting structures. These structures of course can be changed, but each change must be prepared by the re-evaluation of several aspects, in our case of the data security aspects.

We have limited ourselves to a high-level view, based on entities that have a functional meaning for the end-user. In reality, the IoT includes types of entities that we have not considered, such as routers, gateways, etc. Our view could be extended to such other entities: routers and gateways are also limited by the kinds of data that they can hold. However if encrypted data is transmitted through an entity that cannot decrypt it, then we cannot say that data cant flow to this second entity by effect of this transmission. This transmission belongs to a lower logical layer.

Although we have concentrated ourselves on secrecy, we argue that our method takes care simultaneously of the main aspects of the two data security properties of secrecy and integrity. This is because each of these properties specifies what should be the origin of the data each entity can hold, and this is what our labels specify. Concerning availability, our method can only allow to conclude that certain data ‘can be available’ to certain entities, but for them to be actually available the ‘possible’ data transfers must be executed. In other words, our model does not guarantee that a system will actually function, it can only guarantee conformity to data security requirements, essentially that certain data can or cannot reach certain ‘things’.

11 Conclusions and future work

We have presented a method for configuring IoT networks in such a way as to comply to logically specified security data flow constraints. The method is exact, in the sense that it

allows all and only specified data flows. It is also scalable, since it uses efficient algorithms. It could be seen as a generalization of well-known Mandatory Access Control methods.We have also proposed a language for specifying the constraints.

With respect to previous work, the approach that is most similar to ours is the one of [12]. As mentioned, in this paper a distinction is made between subjects and objects and labels are assigned to subjects and objects to define which objects subjects can read or write. However in the IoT it may be impossible to distinguish between subjects and objects, or between reading and writing (these distinctions are common in access control, less common in the IoT). In addition, the labels in our method determine directly what the data contents of each entity can be. Perhaps the approaches of our two methods can be mutually transformed, but ours uses a more direct notation, based on the possible data contents of the 'things', as specified by our logical expressions.

Surely, the solutions we have given for our two examples are not different from those that could have been obtained by intuition, without using our method. We take this fact as a confirmation that our method finds acceptable solutions, and would continue to find them for examples of thousands of entities, possibly generated by requirement languages such as the one we have sketched.

Based on our concepts, one can imagine graphic interfaces that would make it possible to design IoT systems with secrecy requirements by manipulating on the screen graphic representations such as the ones we have used. For scalability however, abstraction mechanisms such as encapsulation will have to be devised. It is interesting to consider that the problem of removing unwanted communication paths in existing networks is much more difficult than the problem of allowing only certain paths at the design stage, in fact we have not been able to find any solution for the first problem. This is because unwanted paths can be part of other paths that are wanted.

Acknowledgment. This research was funded in part by the Natural Sciences and Engineering Research Council of Canada. We are grateful to Dr. N.V. Narendra Kumar for having carefully reviewed the paper.

References

1. 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.
2. A. Botta, W. de Donato, V. Persico, A. Pescapé. Integration of Cloud computing and Internet of Things: A survey. Future Generation Computer Systems 56 (2016) 684-700.
3. M. Blackstock, R. Lea. Towards a distributed data flow paradigm for the Web of Things. Proc. 5th ACM Intern. Workshop on the Web of Things (WoT 2014), 34-39.
4. D.E. Denning. A lattice model of secure information flow. Comm. ACM 19(5), 1976, 236-243.
5. S. Etalle, T.L. Hinrichs, A.J. Lee, D. Trivellato, N. Zannone. Policy Administration in Tag-Based Authorization. In: Proc. 9th Intern, Symp. on Foundations and Practice of Security. FPS 2012. LNCS, vol 7743. Springer.

6. D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli. Role-based access control. 2nd Ed. Artech House, 2007.
7. R. Fraïssé. *Theory of relations*. North-Holland, 1986.
8. A. Gilat. *MATLAB: An Introduction with applications.* 2nd Ed. John Wiley & Sons, 2004.
9. F. Harary, R.Z. Norman, D. Cartwright. *Structural models: an introduction to the theory of directed graphs.* Wiley, 1965.
10. J.Gubbi, R.Buyya, S.Marusic, M.Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems 29(7), (2013), 1645-1660.
11. L.Jiang ; L.D. Xu ; H. Cai ; Z. Jiang ; F.Bu ; B. Xu. An IoT-oriented data storage framework in Cloud computing platform. IEEE Trans. on Industrial Informatics, 10(2) (2014), 1443 – 1451.
12. 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.
13. C. E. Landwehr. Privacy research directions. Comm. ACM 59(2) (2016) 29-31.
14. 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.
15. N.V. Narendra Kumar, R. Shyamasundar. Realizing purpose-based privacy policies succinctly via Information-Flow Labels. Big Data and Cloud Computing (BDCloud'14), 753-760.
16. A. Ouaddah, H.Mousannif, A. Abou Elkalam, A. Ait Ouahman. Access control in the internet of things: big challenges and new opportunities. Computer Networks, 112 (2017), 237-262.
17. J. Park, D. Nguyen, R. Sandhu. A provenance-based access control model, 2012 10th Annual Intern. Conf. on Privacy, Security and Trust (2012), 137-144.
18. 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.
19. P. Samarati, E. Bertino, A. Ciampichetti, S. Jajodia. Information flow control in object-oriented systems. IEEE Trans. On Knowledge and Data Eng., 9(14), 1997, 524-538.
20. R.S. Sandhu. Lattice-based enforcement of Chinese Walls. Computers & Security Vol. 11(8), 1992, 753-763
21. J. Schütte, G.S. Brost. LUCON: Data flow control for message-based IoT systems. arXiv preprint arXiv:1805.05887, 2018 - arxiv.org
22. 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.
23. J. Singh, T. Pasquier, J. Bacon, J. Powles, R. Diaconu, D. Eyres. Big ideas paper: policy-driven middleware for a legally-compliant Internet of Things. Proceeding Middleware '16 Proceedings of the 17th International Middleware Conference. Art. No. 13.
24. A. Stambouli, L. Logrippo. Data flow analysis from capability lists, with application to RBAC. Information Processing Letters, 141(2019), 30-40.
25. R. E. Tarjan. Depth-first search and linear graph algorithms, SIAM Journal on Computing, 1(2) (1972), 146–160.
26. T. Winter, P. Thubert (eds.). RPL: IPv6 routing protocol for low-power and lossy networks. Internet Engineering Task Force IETF RFC 6550, March 2012.