UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Sécurité des données dans les réseaux organisationnels et dans l'Internet of Things, utilisant un modèle d'ordre partiel

Par

Abdelouadoud STAMBOULI

Département d'informatique et d'ingénierie

Thèse présentée au
Département d'informatique et d'ingénierie
Pour l'obtention du diplôme de

Doctorat (Ph.D)

En sciences et technologie de l'information

Jury d'évaluation

Président du jury :	Prof. Omar Abdul Wahab, Ph. D. Département d'informatique et d'ingénierie, Université du Québec en Outaouais.
Examinateur interne :	Prof. Omer Landry Nguena Timo, Ph. D. Département d'informatique et d'ingénierie, Université du Québec en Outaouais.
Examinateur externe :	Prof. Nadia Tawbi, Ph.D. Université Laval, Québec.
Directeur de recherche :	Prof. Luigi Logrippo, Ph.D. Département d'informatique et d'ingénierie, Université du Québec en Outaouais.

UNIVERSITY OF QUÉBEC IN OUTAOUAIS

Data Security in Organizational networks and the Internet of Things using a partial order model

By

Abdelouadoud STAMBOULI

Department of Computer Science and Engineering

A thesis submitted to
Department of Computer Science and Engineering
for the degree of
Doctor of Philosophy (Ph.D)

In Science and Information Technology

Evaluation jury

President :	Prof. Omar Abdul Wahab, Ph. D. Department of Computer Science and Engineering, University of Québec in Outaouais.
Internal examiner:	Prof. Omer Landry Nguena Timo, Ph. D. Department of Computer Science and Engineering, University of Québec in Outaouais.
External examiner :	Prof. Nadia Tawbi, Ph.D. University of Laval, Québec.
Supervisor :	Prof. Luigi Logrippo, Ph.D. Department of Computer Science and Engineering, University of Québec in Outaouais

Acknowledgements

First of all, I praise and thank Allah (SWT) for his greatness and for giving me the strength and courage to complete this thesis.

I would like to express my deep gratitude to my supervisor Professor Luigi Logrippo for supporting me, committing and guiding me through my work and providing funding. I attribute my PhD to his encouragement and effort and without him this thesis would not have been completed

I also thank Professor Larbi Talbi for his help, his advice and for all he has done for me.

I am grateful to Professors Omar Abdul Wahab, Omer Landry Nguena Timo, and Nadia Tawbi for having accepted to be parrt of the examination committee and for having provided suggestions for improvements.

I fully thank all those who contributed from near or far to the completion of this research work, as well as all my work colleagues: Yassine Zouaoui, Khaled Kedjar, Temim Samah, Bekhouche Mohamed Amine, Vincent Fono, in addition to all professors and staff of the Computer Science and Engineering Department in particular: Mr. Abdelkrim Chebihi.

I would also like to sincerely thank all my friends whether they are here in Canada or in my home country.

I thank in a special way all the members of my family starting with my parents, my brother, my sisters, all of my large family members more precisely my aunt Zoulikha and her family for their constant support, their encouragement, their prayers and their contributions to the success of this research project.

Finally, I would like to thank everyone who contributed directly or indirectly to the realization of this project. Special mention for Dr. Imane B.

We thank the Natural Sciences and Engineering Research Council (NSERC) for partially funding this research.

Dedication

To my dear parents Athmane Stambouli and Samia Oukachbi.

You spared no effort in my education and were always there when I needed support. I owe all my moments of past and future success to you.

To my small and large family.

To all of my friends and acquaintances.

To my dear Professor Luigi Logrippo who was behind the culmination of this research project.

Abstract

Access control methods are used to provide data protection against unauthorized users.However, these models are only concerned with access decisions, which is not enough. To have complete protection, it is also necessary to control the transfer of data after the access. This is why we need data flow control.
So far, the data flow control problem found solutions in the traditional lattice model. However, this model is very restrictive since it requires a lattice structure. In this work, we propose an alternative solution that is efficiently feasible in any network.

Our solution takes into consideration the fact that multi-level systems are necessary for data secrecy. Furthermore, for every network, we can construct a multi-level system, which is a partial order of components that is proven necessary and sufficient for data secrecy, and unlike the lattice model, always exists and does not require adaptations. We back our claim with simulation results that prove the feasibility of our method for large networks.

We show that our method is applicable outside the organizational context, and into the large distributed systems of the Internet of Things. The partial order of components allows us to configure IoT networks to meet privacy and secrecy and integrity requirements.

Finally, we propose an implementation method of our method using Software-defined networks. An SDN controller is developed to enforce the data flow rules of our partial order model.

Résumé

Pour faire face aux défis de sécurité des données, les modèles de contrôle d'accès sont mis en place. Ces modèles offrent une protection des données contre la divulgation et la destruction. Cependant, ces modèles ne concernent que les décisions d'accès, et cela n'est pas suffisant. Pour avoir une protection complète, il est également nécessaire de contrôler le transfert de données après l'accès. C'est pourquoi nous avons besoin d'un contrôle du flux de données.

Jusqu'à présent, le problème de contrôle de flux de données a trouvé une solution dans le modèle traditionnel treillis. Cependant, nous pouvons voir que certaines propriétés de ce modèle ne sont pas pratiques et elles sont très resctrictives.

Dans ce travail, nous proposons une solution alternative réalisable dans n'importe quel réseau. Notre solution prend en compte le fait que les systèmes à plusieurs niveaux sont nécessaires au secret des données. De plus, pour chaque réseau, nous pouvons efficacement construire un système à plusieurs niveaux, qui est un ordre partiel de composants qui s'avère nécessaire et suffisant pour préserver la confidentialité des données. Contrairement au modèle treillis, notre modèle est plus pratique, existe toujours et ne nécessite aucune adaptation pour l'obtenir. Nous soutenons cela avec des résultats de simulation qui prouvent la faisabilité de notre méthode pour les grands réseaux.

Nous montrons que notre méthode est applicable en dehors du contexte organisationnel et dans le grand système distribué de l'Internet des objets. L'ordre partiel des composants nous permet de configurer les réseaux IoT dans le respect des exigences du secret de la confidentialité, et de l'intégrité des données.

Enfin, nous proposons une méthode d'implémentation de notre méthode à l'aide des Software-definedd networks .Un contrôleur SDN est développé pour appliquer les règles de flux de données de notre modèle d'ordre partiel.

Contents

List of Figures

List of Tables

Nomenclature

ABAC	Attribute Based Access Control
ACM	Access Control Matrix
API	Application Programming Interface
ARBAC	Administrative Role Based Access Control
BLP	Bell-LaPadula Moodel
CF	Can Flow
CH	Can Hold
DAC	Discretionary Access Control
DAG	Directed Acyclic Graph
IaaS	Infrastructure as a Service
IoT	Internet Of Things
IPAD	IP address
LBAC	Lattice Based Access Control
MAC	Mandatory Access Control
P2P	Peer-to-Peer
PaaS	Platform as a Service
RBAC	Role Based Access Control
RFID	Radio Frequency Identification
SaaS	Software as a Service
SDN	Software Defined Network
VM	Virtual Machine
XACML	eXtensible Access Comtrol Markup Language

Chapter 1

General introduction

1. Introduction

In the current context, the security of data and information faces many challenges. One of these challenges is for organizations to protect the data on which they rely against disclosures, alterations or destruction. These challenges have increased with the development of Internet technologies that make data more accessible in massive quantities. This last fact increases the risk of data leakage and makes controlling the security of data more difficult.

To face those challenges, we have the notion of access control. It plays a major role in the protection of the data. According to [55] access control "*... is concerned with determining the allowed activities of legitimate users, mediating every attempt by a user to access a resource in the system...*". This can be done by verifying if each subject (person, process, etc.) has the necessary rights to execute each requested action (write, read, delete, etc.) on objects (resources, files, databases, etc.). Access decisions are made according to sets of rules defined by security administrators; these sets of rules form what is called *security policies*. Access control is implemented through many models. However, some of these models do not totally fulfill the real and evolving needs of data protection, more specifically; they are not concerned with the propagation of data after subjects have accessed them. Hence, the importance of the concept of *data flow control*.

Data flow control is a concept that focuses on preventing the propagation and leakage of data from authenticated and legitimate users to other unauthorized subjects who should not have access to them [101]. This concept is implemented with the Mandatory access control models (MAC). However these multi-level models are often considered outdated, and modern access control models tend to be role and attribute-based. We intend to show in this research that multi-level systems are necessary for data secrecy and relate closely to practical needs.

Since most of the security models except MAC models lack data flow control mechanisms, we intend to propose, in this work, a suitable solution for this problem. Our solution will be a practical security model dedicated to control the data flow, i.e. to control where data can

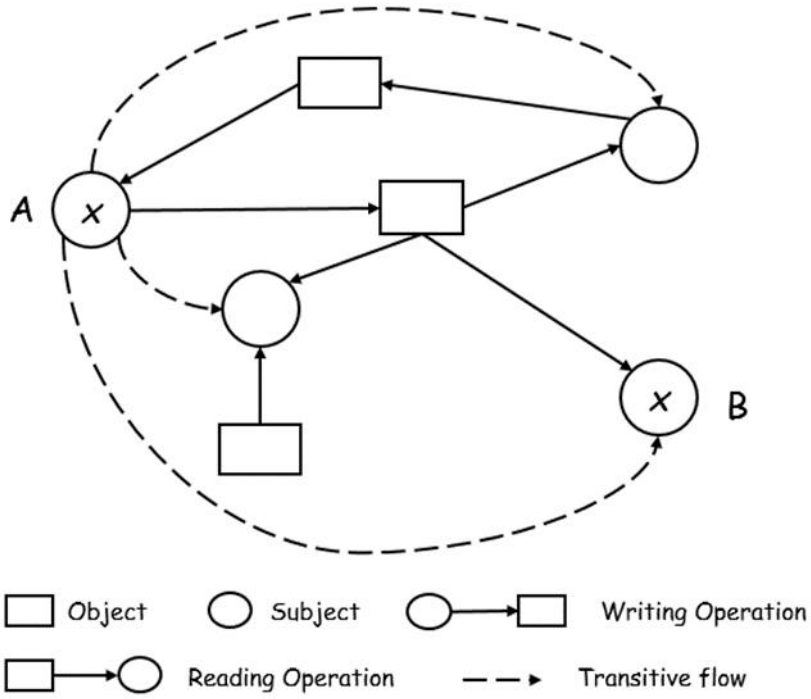


Figure 1: Illustration of a data flow in a system

end once accessed. This solution will be valid for organizational systems, as well as for new distributed systems such as the Internet of Things. Our model is a relational model just like the influential model of Denning [31]. It is, however, more practical and can be considered a generalization of the latter. It uses partial orders, which are necessary and sufficient for data flow secrecy and always exists in any data flow system without a need for any particular structure like the lattice model.

2. Motivations

Consider a system where subjects can write on objects and objects can be read by subjects. We say that there is a potential data flow between two subjects (objects) when a subject (object) can, by the effect of reading and writing operations, transmit data to other subjects (objects). For example, in figure 1, subject A can read data x from an object and other entities can receive x from A. Arrows represent possible data flows.

In this system, there are indirect data flows illustrated by dashed arrows. These indirect flows are the result of the exploitation of legal direct data flow illustrated by solid arrows via intermediary subjects and objects and may be legal or not. In practice, these data flows can compromise confidentiality; a typical example is one where insiders of a company can pass confidential data by taking advantage of intermediary subjects and objects.

Thus it is very important for confidentiality purposes to determine where the data of given objects can flow. Can a given subject know data coming from a specific object? Which data can flow to a particular subject? Dually, for integrity issues, can an object store data coming from another object? Can a given object happen to store data originating from a specific subject?

The data flow control problem found a solution in the lattice model. But, is this solution really practical? In many cases in practice, lattice properties such as the existence of upper and lower bounds are not satisfied in companies' network. Therefore, it can be useful to propose an alternative solution, which is feasible in any network.

This problem occurs not only in organizations, but also in dynamic systems such as the Internet of Things. The Internet of Things is a highly distributed system where data can flow among computational objects in complex data flow configurations. The Internet of Things is about connectivity, but it is just as much about data flow control. For example, in a healthcare system, a heart-rate sensor and a motion sensor may be separate things that communicate their data. Each thing's data flow is stored for a specific person, and must only be accessible by the treating staff, and isolated from other things and other people's data. Another example is an e-commerce network, with three clients, two stores and four suppliers. *Client 2* and *Client 3* are colleagues so they collaborate and share data and order only from *Store 2*. We have two stores in competition, *Store 2* works with *Supplier 4* and *Store 1* with the others, and four suppliers, *Supplier 1, Supplier 2* and *Supplier 3* are allies that collaborate and share client data. The example is illustrated in Figure 2. In this example, each client can order from a store, if the order's item stock is exhausted in the stores, the order is routed to the suppliers. An opposite data flow is present; it consists of the billing data that flows from the suppliers into the clients. So, for every client, data must only be accessible to the authorized stores and suppliers. We will come back to this example in our work.

In these cases, it is very important to enforce data flow constraints in order to be able to

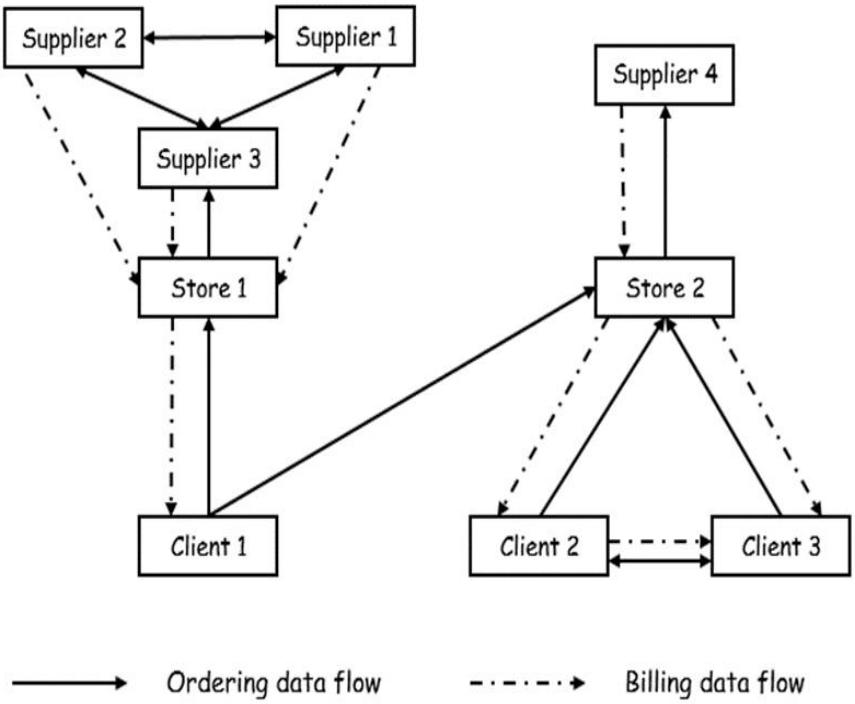


Figure 2: Illustration of an e-commerce system

answer questions such as how can we set up data communications channels between things so data originating in one device can or cannot reach another one? Given certain data flow relationship in an IoT system, and data originating in certain devices, which are the entities that will be privy of these data?, and finally, given an IoT network specifications, how can we configure an IoT architecture that is secure with respect to data?

Although there is a well-known model for data flow control which is the lattice model of Denning [31], we can see that this model presents some gaps that complicates its enforcement and implementation. We give a simple example that highlights some of the problems faced while using the lattice model. Let us consider a situation where we have a very simple system, consisting of only two companies *Co1* and *Co2* in conflict of interest. Neither company should receive data from the other. The following Figure 3 describes the modelling of secrecy requirements with the lattice model.

While using the lattice model, it is necessary to introduce upper and lower bounds for the

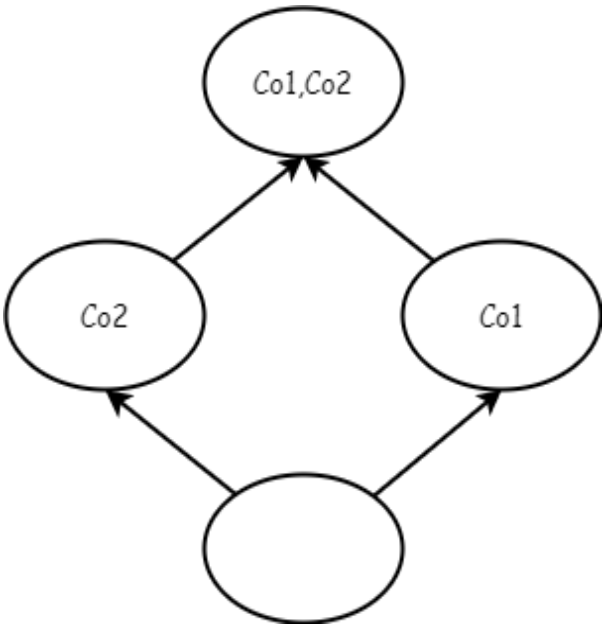


Figure 3: Lattice model for the given example.

two entities. These bounds don't exist in practice, in fact, they are contrary to requirements. We have an entity (the upper bound) that can receive data from both companies contrary to the conflict of interest specification. Implementing this model will require removing the elements that were just added. So, the lattice model is a theoretical model that cannot be used for implementation.

All those gaps motivated us to propose a new model for data flow control that is more general and more practical than the lattice model.

In our work, we develop a method for formally defining and optimally implementing data flow constraints in systems such as the ones described above. In addition, we prove that the method is more efficient and scalable than the existing ones.

3. Organisation of the thesis

In this chapter, we have presented a general introduction to the research topic, the motivations behind our work, and the intuitive hypotheses that we will use. The remaining of the thesis is organized as follows:

Chapter 2

In chapter 2, we present a set of concepts related to the field of computer security and access control. This chapter will also include definitions of concepts related to information flow control that will be used all along this work.

Chapter 3

In chapter 3, we will review research in the areas of access control and data flow control. We will start with the presentation of the classical models of access control, indicating their weaknesses regarding data flow control. We will also look in more detail at the work related to data flow control whether it is in organizations or other types of systems (mainly the Internet of Things).

Chapter 4

In chapter 4, we define some notions and algorithms used in our method and that constitutes the basis of our work. This includes graph and order theory notions and a formalism to reason about confidentiality and integrity.

Chapter 5

In chapter 5, we present our work on finding multi-level system inside any access control systems using graph and order theory. Simulations are described, proving that the method is feasible in real systems composed of many thousands of entities.

Chapter 6

In chapter 6, we leave the organizational aspect, and discuss the application of our method on the Internet of Things. We show that the method is applicable to such systems in order to meet privacy and secrecy requirements.

Chapter 7

In chapter 7, we propose a possible implementation of our model in the context of the Internet of Things using Software-defined networks (SDN). An SDN controller will be developed to enforce our partial order to control data flow in a given system. Simulations will also be done to prove the feasibility of the implementation.

Chapter 8

In chapter 8, we present a summary of the contributions of this work alongside some research perspective that can be done in future works.

Chapter 2

General Concepts

1. Introduction

In this chapter, we will briefly review the definitions of the three domains addressed in this work, namely *information security*, *access control* and *data flow control*. We will present the different concepts related to each domain, with special attention to concepts considered to be at the foundation of this work.

2. Information security

The field of information security has grown and evolved significantly over many years. This evolution aims to meet the requirements of security and privacy that have been the subject of much research after the internet revolution [121]. These requirements can be legal, sociological, political, and administrative factors. Technically, the main goal of information security is to maintain the confidentiality, integrity and availability of data [2,60] from those with malicious intentions. Confidentiality, integrity and availability are known as the *CIA Triad* of information security. This triad has evolved into what is commonly termed the *Parkerian hexad* [80], which includes confidentiality, integrity, availability possession (or control), authenticity, and utility [21].

Below we elaborate briefly on each of these concepts, plus some other related ones, which will be important for our work. Some of these concepts will be more precisely defined later in our work.

Data and information

The terms 'information' and 'data' are sometimes used interchangeably. For data, we have Merriam Webster's definition: *information in digital form that can be transmitted or processed.* And for information: *knowledge obtained from investigation, study, or instruction* [121]. Accordingly, data are bits and bytes interpreted as alphanumeric characters. The concept of information is

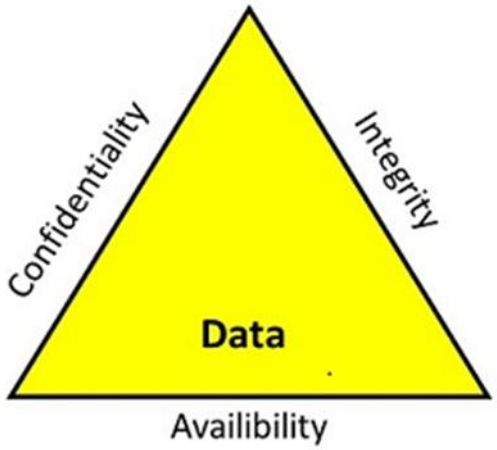


Figure 4: The CIA Triad

more general, it can include everything that can be inferred from data. For example, a date: 2016.05.10 is data. It is also information, but much additional information can be derived from it: for example, if it is interpreted as a birth date, that the person is five years old, that she or he was born in May, in spring, that is younger than us, etc. While data represent themselves, the representation of information is less obvious. Information can be encoded in data and data can be encoded in information in many different ways. This is the reason why in this work, we will avoid talking about information flow; information can flow not only by reading and writing operations, but also by inferences, which are outside our scope.

Confidentiality and secrecy

Confidentiality (also called *secrecy*) are implemented by granting access to data or resources only for authorized persons, entities or processes [10,92]. For example, a nurse cannot read information about a patient in another department, an emergency doctor can.

Integrity

Integrity means maintaining and assuring the accuracy and completeness of data over its entire lifecycle [21]. This means that data cannot be changed in any unauthorized manner. In our work, integrity will be taken as the property of a subject (object) to know (contain) only data of specific types or labels.

Availability

Availability refers to the ability to use (in our case, read or write) the desired data or resource at the desired moment by an authorized entity. Availability will not be discussed in this thesis.

Other related concepts such as *Authenticity* (proof of identity [28]), *Utility* (usefulness of data [92]), and *Possession* (Control or ownership of data [92]) have been introduced, but this work will not be concerned about them.

3. Access control

Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. The access control decision is enforced by mechanisms implementing regulations established by a security policy [22]. This security policy mainly aims at ensuring the phases of *access control* namely identification, authentication, authorization, access approval, and audit. All this, without harming the three information security proprieties (confidentiality, integrity, and availability).

Identification

Identification is the process of verifying with the system, the identity of a user (subject) through a unique information common to him. For example username, email address ...

Authentication

Authentication is the act of confirming the authenticity and the truth of the identity claimed true by a user during the identification phase [21]. In fact, once a user is identified within the system, the latter will verify the accuracy of the access request data with the data recorded in the system when creating the user account. Such data may include password, PIN , fingerprint...

Authorization

Authorization specifies the operations that subjects are allowed to execute within a system. Note that the fact that a user has been identified and authenticated does not imply the right to access data within the system [93] , this right is based on policies and access control models.

Access approval

Access approval is the function that grants or rejects access during operations. It compares the formal representation of the access request with the authorization policy to determine if the

access is granted or rejected [44].

Access audit

Access audit aims to assess the strength of the control environment and the adequacy of the related internal control framework in place over system access controls [48].

Now that we have presented the different phases of the access control process, we will define the concepts related to access control that can be found in this work.

Object

An *object* is a passive entity that contains data; it can represent a document, a database ... Access to an object implies access to the data that it contains.

Subject

A *subject* is an active entity that requests access to an object; it can represent a process, a person, or a device. Operation An operation is an action performed by a subject on an object. For example, read, write, delete, modify, append ... In this thesis, we will limit ourselves to reading and writing operations.

Permission

A *permission* is an authorisation given to a subject to perform an operation on an object. For example, a database administrator can get permissions to perform any action on any record in the database. In this work, we will only consider reading and writing permission, since others can be seen as derived permission: for example, deletion can be defined as replacing a data item with null data.

Least privilege

The principle of *least privilege* states that a user must be able to access the minimum of data and resources that are necessary for a legitimate purpose. This will prevent the abuse of unneeded permission [44].

Need to know

The principle of *Need to know* is an application of the least privilege principle. It states that a user can only access the information and resource that are necessary to complete their legitimate purpose.

Privacy

Privacy is the right of people to deny access to data about themselves that others might use to their disadvantage [119]. This process may be selective in which individuals keep some information secret and private while they choose to make other information public and not private [36].

Personal data

Personal data means data about an identifiable individual that is recorded in any form including, without restricting the generality of the foregoing: information relating to the race, national or ethnic origin, colour, religion, age or marital status of the individual, the address, fingerprints or blood type of the individual, etc. Theses should remain confidential in all cases [62].

Purpose

The *purpose* concept has an important function in protecting privacy [24]. It represents the reason for the availability and disclosure of information. For example, an Amazon's client discloses his address only for receiving his orders.

Separation of duties

Separation of duties is the concept of having more than one person required to complete a task in order to prevent fraud or error [22]. For example, in a company, a person in charge of opening envelopes that contain checks cannot be the one that records the checks in the system. This reduces the risk of fraud.

Classification

Data classification is the process of organizing data into levels. These levels indicate the sensitivity of the data and determine the security requirements necessary to access it. Every classified data requires a protection against unauthorized access, and the roughness of the protection depends on the classification level. For example, access to data classified in the secret level should be narrower than access to confidential information. [21]

Clearance

A security *clearance* is a status granted to users allowing them to access classified data. A user can access data classified less than or equal to his security clearance. A clearance by itself does not grant access; the organization should also determine that the cleared user needs to know the specific classified data [36].

Access control policy

An *access control policy* is a set of rules or principles used in an organization for security. This set determines the access decision for any request. In some cases, an access control policy is made of a single rule [22]. For example, a student cannot write on the grades file. But generally, an access control policy contains a whole set of rules.

Access control model

An *access control model* is an abstract system that can be used to formally implement an access control policy. The formalization should allow the proof of properties on security provided by the access control system being designed. [22].

4. Data flow control

In an organization, in order to perform operations on protected objects, subjects must have the authorization of an access control model. Even though the main models reflect this reality, most access control models do not take into account data flows implied by a given collection of access rights. However, some of these systems (that we will present in the next chapter) are concerned not only about accesses, but also about the propagation of data (data flow).

In the literature, data flow can be defined as the propagation of data from a source into a destination, but formally, there is no fixed definition of data flow, many definitions have been proposed according to the authors and depend on the models and the security objectives to achieve. These definitions will be presented in detail in the next chapter.

In general, and for this research, we say that there is a data flow from an object O_1 into another object O_2 whenever the data stored in O_1 has moved directly or indirectly into O_2 . This propagation is generally the result of a reading operation from O_1 followed by a sequence of writing and reading operations that ends with a writing operation on O_2.

Illegal data flow

Illegal or *unauthorized* data flow is an information leakage that occurs when data ends up revealed to unauthorized parties. In general, information leakage can occur through covert channels, Trojan programs, etc.

In a system, this data flow can start from a subject S_1 that executes an authorized reading operation over an object O_1 followed by a writing operation on an object O_2 for which a subject

S_2 has reading access. In this case, S_2 can read the data that flows from O_1 into O_2 even though he does not have that access right on O_1. In the cases of information leakage, several questions arise with the intention of subjects that have access right and initiate the leakage; this shows that the human factor is decisive and not very controllable in illegal flow situations. We will show that our approach can be adapted to solutions to control the data flow in different types of systems. Figure 5 shows an example of an illegal flow; where *Subject2* can know the data in *Object1* although it may not have given access to it.

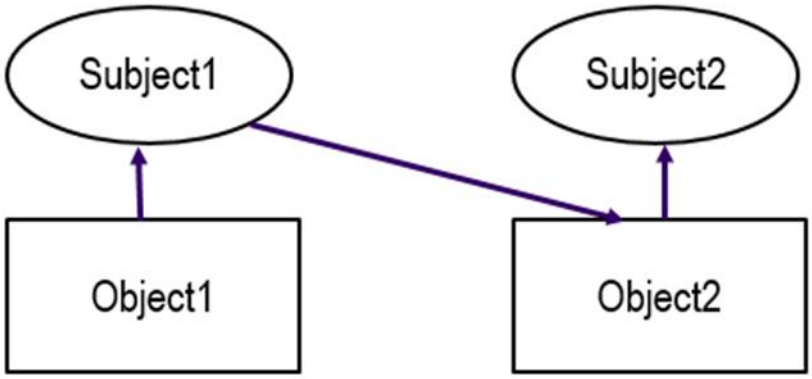


Figure 5: Illegal flow example

Data flow control

In a system or a network, *data flow control* allows to manage and monitor the data flowing in the system as well as their process propagation from one subject to another. Note that classified data can also flow outside the system and fall into the hands of unauthenticated subjects. In this case, data flow control is crucial. In fact, data flow control policies will regulate the propagation of data towards external entities in order to avoid data leakage.

Currently, almost all data flow control research is based on *Denning's* model [31] (see Chapter 3). Where the classification concept is used where information can flow from a *security class* A into a *security class* B if and only if B is greater than A or at least they are of the same level. For example, information can flow from class *Secret* into class *Top Secret*. This will prevent classified information to flow into unauthorized subjects and this shows that data flow control is an extension of access control. We will describe this in detail in the next chapter.

At the end of this chapter, it remains to emphasize that in this research, we will present of a new method of data flow control that maintains the confidentiality of the system, by using notions from partial order theory.

5. Conclusion

In this chapter, we have chosen to define concepts related to *computer security*, *access control*, and *data flow control*. These definitions will facilitate the general understanding of the subject. Some other definitions of concepts and ideas related to our work will be presented in chapter 4.

Chapter 3

Access Control and dataflow models

1. Introduction

As already mentioned, we are interested in this thesis in the problem of maintaining the secrecy and integrity of data in highly distributed systems, such at the Internet of Things. This problem is often called the data flow control problem. The problem is closely related to the problem of access control to data, because data flow control is usually implemented with mechanisms of access control. Therefore, we will review the literature on data flow control together with the literature on access control, although access control can be used to control access to objects that are not data. The following account should be read with our particular interest in mind, namely when we refer to 'objects', we intend to refer to 'data objects'.

In order to maintain secrecy and integrity, rules and laws are defined as security policies to ensure protection against unlawful access and misuse of information in the case of legitimate access. Access control models are defined to implement these policies. However, this is not enough; data flow models must come along in order to control data propagation after the access in the system, to detect any flows that violate the data flow policies.

In this chapter, we will examine the main models of access control and mention their limits for data flow control. After that, we will focus on the data flow models starting with the classical multi-leve.l models with particular concentration on the lattice model. We will then talk about the existing contributions on the problem of data flow control in the Internet of Things.

2. Access control matrix

The access control matrix is considered the basic framework for security protection analysis. It is defined by a triple *(S, O, A)*, where S is a set of subjects, O is a set of objects, A is an access matrix. The columns of this matrix correspond to particular data structure in the system and the rows correspond to the potential users of the systems. Each element in the matrix *A[s, o]* is a decision rule specifying a right or the action that a subject s is permitted to perform on the data structure (object) o, all this following the security policies [28].

Even though the access control matrix is very precise and can be used in a variety of systems, its size can be problematic. Indeed, with a large number of subjects and objects, that matrix will become very large and not very populated, which can cause practical problems in its management. To solve this problem, the access control matrix can be implemented in two ways .i.e. by access control lists, or by capability lists. Access control lists represent the access control matrix as a list of subjects' access rights for each object.

Capability lists represent the access control matrix as a list of objects' access rights for each subject.

3. Access control models

There are different types of security models whose main concern is to ensure confidentiality, integrity and availability [82]. These models can be categorized in different ways according to their common characteristics. The literature identifies four principal families of access control models:

- Discretionary access control models.
- Mandatory access control models.
- Role-based access control models.
- Attribute-based access control models.

There are other less known and less used access control models. We will give a brief presentation for one of them called Trust-based access control model.

3.1 Discretionary access control models

Discretionary access control models (DAC) are considered to be extensions of the UNIX file protection mechanism. They use access control matrices and are less restrictive than other models; in fact, in DAC the owner of a data object has the power to propagate and grant the rights it has on the object to other users, including rights to modify, and propagate them to third parties. On this idea, there have been several variations, such as the *Lampson* [69] and *Harrison Ruzzo Ullmann* [53] models. Although this model is used in various systems like operating systems, database management systems, and in some information system where security is left to users, it presents some fundamental problems; this model does not guarantee the control of

information flow due to the propagation of access rights by users [17], and once the user who propagates his rights can lose control over them. Moreover, this model is vulnerable to malicious programs such as Trojan horses, which can get access rights and so control the propagation of the information.

3.2 Mandatory access control models

Contrary to Discretionary access control models, Mandatory access control models do not allow subjects to intervene in the allocation of access rights, as access control is managed in a centralized way by the administrators to achieve its security objectives. Since this family of models is considered rigid and more secure, it allows not only access control but also data flow control. We will return in detail to those models in section 4.

3.3 Role based access control

Role-based access control is a security model to implement security policies in structured organizations; it is presented by *Ferraiolo* and *Kuhn* [41], and it became an international standard [109]. Role- based access control is widely used in practice and is intended to reduce the cost of security administration thanks to the notion of roles. It is designed to ensure that users have only appropriate access rights based on the *need to know* concept. In [108], Role-based access control is presented as a family of four models: *Core RBAC (RBAC0)*, *RBAC* with *role hierarchies (RBAC1)*, *RBAC* with constraints *(RBAC2)*, and the consolidated model *(RBAC3)* that includes all the previous ones.

Core RBAC presents five concepts: set of *users*, set of *roles*, and set of *permissions* composed of a set of *operations* applicable on a set of *objects* [40]. Roles derive from the architecture of organizations, and RBAC attributes permission on available objects (*PA: Permission assignment*). Users are assigned to roles that are appropriate for them according to their responsibilities in organizations (*UA: User Assignment*). Therefore, users will acquire permission associated to the roles to which they are assigned, so, RBAC does not assign permission to users individually, which provides flexibility and adaptation to changes that can happen in systems. In addition, in order to manage role activation for users, we have the concept of *sessions*, which is a correspondence between users and active roles, so, in a session, every user will have all or a portion

of his assigned roles as active roles, depending on the roles required to accomplish tasks. This corresponds to the least privilege principle. In our work, we have ignored this concept of session considering it not essential for our research, since each session can be considered as its own network with its own subjects, objects, and permission . Figure 6 presents the concepts of *Core RBAC*.

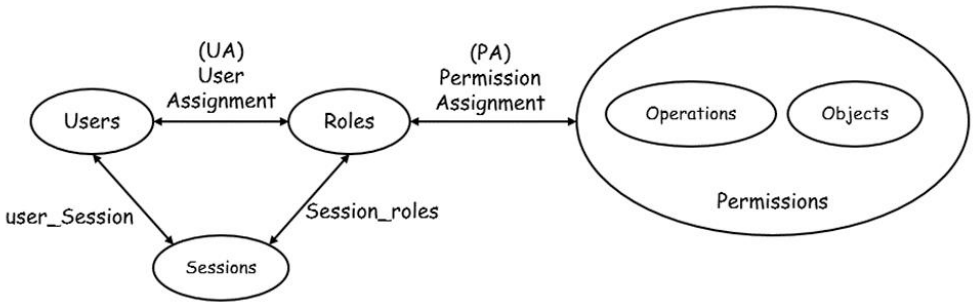


Figure 6: Concepts of *Core RBAC*

RBAC with *role hierarchies (RBAC1)* introduces the concept of role hierarchy, which is a partial order of roles, where the top-level roles inherit all the roles and therefore the permission of their juniors. In our use of *RBAC*, and to simplify, we choose to flatten the role hierarchy which means that the permission set of a role will directly include his permission and the permission of all his junior roles, without the need of role-role assignment.

RBAC with *constraints (RBAC2)* introduces the concept of constraints or *separation of duties*, which intends to prevent users from acquiring all the permission to lead to certain critical tasks. We distinguish two types of constraints: static constraints are imposed on the user assignment to role step, for example; no user can have two roles in conflict of interest. Dynamic constraints are added on role activation in sessions, for example; no user can have two roles that are in conflict of interest as active roles in a given session. We will not pursue further these concepts in this thesis.

The consolidated model (RBAC3) is considered as the full model, it contains all the previous principles. Figure 7 shows the concepts of *full RBAC*.

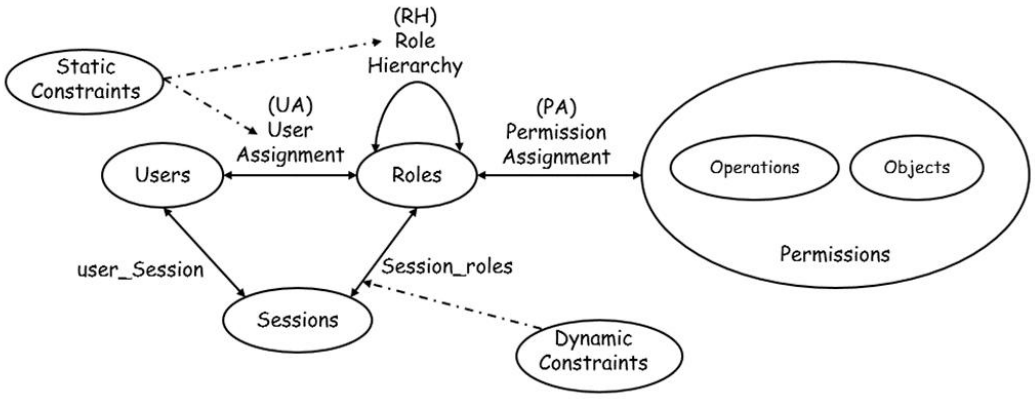


Figure 7: Concepts of *full RBAC*

Another important concept in RBAC is *role engineering*; it is the process of defining a role structure that optimally reflects all the tasks within a company deploying RBAC. Role engineering is difficult and considered an obstacle to RBAC deployment due to time consumption, error proneness and costs [132]. We will show later that the results obtained can be used in the process of *role engineering* in RBAC, for determining secrecy levels, eliminating or combining roles.

RBAC has a critical limitation: it does not have specific mechanisms for data flow control. Data flow can occur without any control between roles through indirect channels, or because of shared permission. An example will be the following: suppose that a subject $S1$ has a role $R1$ with the permission read on object $O1$ and writes on $O2$, and subject $S2$ has a role $R2$ with the permission read on $O2$. In this case, $S2$ can get a copy of the data in $O1$ through $S1$ even without permission. To solve this problem, research has shown how to configure RBAC in order to implement the mandatory access control policy. In the next sections, we will discuss some of this works and we will show that our method is applicable to RBAC to ensure data flow control.

3.3 Attribute-based access control

Attribute-based access control (ABAC) [56] is a flexible model where access decisions are made based on attributes assigned to categories. The four main categories are *subjects, objects, operations*, and *environmental conditions*. Each one of these categories has attributes that have predefined and pre-assigned characteristics. We can have subject attributes such as ID, roles,

age, and departments. Object attributes can be the object type (record, bank account...), localization, classification, etc. Operation attributes represent the actions that can be performed on given objects: read, write, delete physical access, etc. Finally, environmental attributes are the time and the date of the request, locations, etc.

In ABAC, an access request is a set of elements *(attribute (category) =value)* that represent the request parameters, for example: *Name (Subject) =Lisa* and *Role (Subject) = Nurse* and *Type (Action) =read* and *Type (Object) =Medical record*. In this example, nurse *Lisa* requests read permission from the *medical record*. Once the request is made, the access control system compares the values of attributes of each category of the request with the values of the attributes of the access control policies presented in the form of Boolean expressions that satisfy the authorization for a specific operation such as: Grant if *(Role (Subject) = Nurse)* and *(Type (Action) =read)* and *Type (Object) =Medical record*. In this case, the access is granted.

To provide this access decision, ABAC defines a very interesting architectural model described in Fig. 8 [56]. This figure shows the functional points of ABAC, and how these points work together to provide access control decisions following access control policies. We have four points: *Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Information Point (PIP)*, and *Policy Administration Point (PAP)*.

Policy Decision Point (PDP) is the functional unit that determines access control decisions by evaluating the applicable *DPs* (Decision policies) from the *DP store*.

Policy Enforcement Point (PEP) is the functional unit that forwards an access control request to the *PDP* and enforces the policy decision made by the *PDP*.

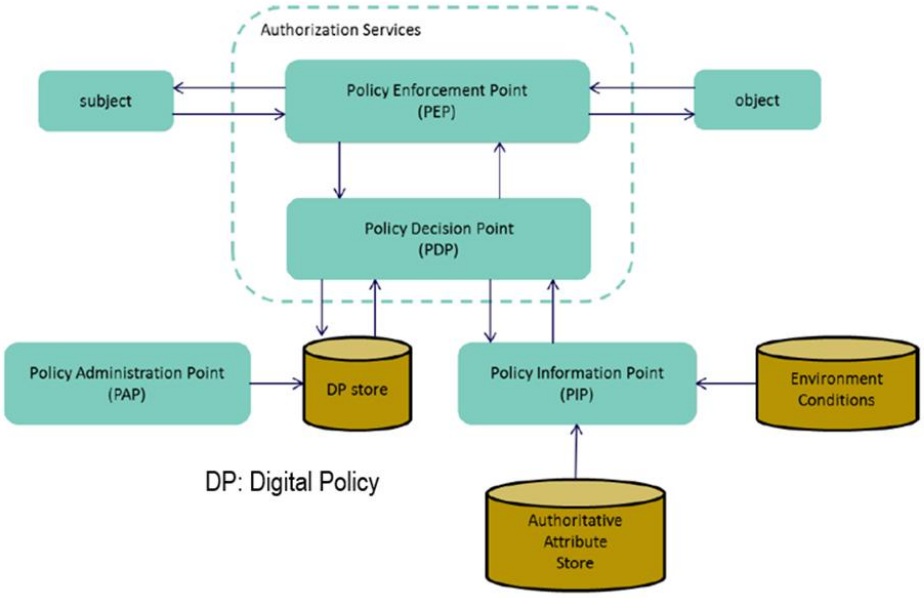


Figure 8: ABAC functional points [56]

Policy Information Point (PIP) provides from the Authoritative Attribute Store and the Environment conditions all the attributes and conditions needed by the *PDP* to make a decision.

Policy Administration Point (PAP) provides a user interface for creating, managing, testing access control policies and storing them in the *DP store*.

The standard XACML [56] presents a widely implemented version of ABAC.

Even thought ABAC is considered a flexible model with many positive aspects, for the same reasons as RBAC it does not have a data flow control mechanisms; we will see that our model can provide those mechanisms.

3.4 Trust-based access control

In contrast with the classical access control models that mainly are implemented in centralized and static environment, Trust-based access control is developed for dynamic collaborative environments, such as Grid, P2P and ad-hoc networks.

In Trust-based access control, subjects might be collaborative members or their processes, procedures, and tasks. Objects might be collaborative members or their resources. And just like other access control models, subjects will perform operations on objects, such as read, edit, browse execution, query, etc.

The essential parameter in this model is *trust*; trust means liability, and satisfaction that a trusted subject behaves according to expectations. Trust values are assigned in the interval [0, 1]. There are two basic approaches for obtaining a subject's trust: experience by interacting with the subject or recommendations from other trusted subjects [130]. Paper [72] presents a method for calculating trust values combing the trust information based on past experiences in interacting with the subject and the recommendations of others.

In a system, a subject's trust threshold is defined as the minimum trust value for the subject to obtain operation permission. When a subject's trust value is less than the subject trust threshold for an operation, the subject will be rejected to perform the operation. Moreover, an object's trust threshold is defined as the minimum trust value of the object that allows its resource to be accessed. When a subject's trust value is less than the object's trust threshold for an operation, the subjects will not perform the operation on the object. The trust refresh deadline is the useful life of a trust value. A subject's trust value should be refreshed at the trust refresh deadline. This deadline takes into consideration many factors such as the importance of objects' resource, the trust degree of the recommender, etc.

According to [72], a Trust-based access control model is a tuple as follows:

- E is the set of entities
- S$\subseteq$E is the set of subjects
- O$\subseteq$E is the set of objects
- OP is the set of operations on object's resources
- TV:E$\times$OP$\rightarrow$[0,1] (An entity's trust value for performing an operation)
- TT$-$S:O$\times$OP$\rightarrow$[0,1] (The subject's trust threshold for performing an operation on an object)
- TT$-$O:S$\times$OP$\rightarrow$[0,1] (The object's trust threshold which an object should have when a subject accesses it)
- F: S$\times$O$\times$OP$\rightarrow\{0,1\}$ (Access authorization rule) denotes a mapping of a subject's operation

permission on the object to set $\{0, 1\}$, where 1 denotes that the access is permitted and 0 denotes that the access is denied.

When a subject performs an operation on an object, the access control system decides to map the permission to 0 or 1 based on trust values of the subject and the object. This leads to the following Trust-based access control policy [72]:

$$\forall s \in S, o \in O, op \in OP, F(s, o, op) = TV(s, op) \geq TT - S(o, op) \bigwedge TV(o, p) \geq TT - O(s, op).$$

When an entity's trust value cannot be evaluated, the entity will be given a default trust value, which will be valid only on this occasion. Furthermore, an entity's trust value must be refreshed over a cycle of time for access control in the future.

Many frameworks based on the Trust-based access control model have been proposed in the literature. In [57], a Trust-based access control framework for P2P file-sharing system is proposed. This framework uses aspects of trust, recommendations and contributions, combined with discretionary access control schemes, and applies them to P2P file-sharing systems. The framework aims to preserve P2P file sharing system properties, essentially decentralization, peer classification, and file transfer. In [18], the Role-based access control model is used as a base for a Trust-based context-aware access control framework. A user is assigned to a role, based on the trust in the certified attributes the user presents. In this case, trust is of binary nature and is based on cryptographic controls. Context constraints, such as time and location, are further used to ensure fine-grained access to resources. Paper [78] argues that traditional access control models are not suitable access control solutions in the Internet of Things. A trust relationship between two devices will directly affect the interactions between them, and their willingness to share services and resources. An access control (FTBAC) model with a fuzzy trust value is proposed. The trust value between devices is calculated based on experience, knowledge, and recommendation, among other factors. The set of certificates and access requests are the access credentials. The FTBAC framework consists of three layers: 1) Device Layer: includes all IoT devices and their communications; 2) Requesting layer: mainly responsible for collecting EKR (Enterprise Knowledge Repository) information and calculating the fuzzy trust value; 3) Access Control layer: includes the decision-making and mapping process of the fuzzy trust value and

access based on the principle of least privilege. The simulation results show that the framework can ensure flexibility, scalability, and increased energy efficiency.

As most other access control methods, Trust-based access control does not deal with data flow control. For example, by using only the methods described above, in the account of her trust value, at some point Alice may be denied to read from object *O*, but allowed to read from object *O'*. Still, she might gain access to the contents of *O* through *Bob*, if *Bob's* trust value allows him to read from *O* and writes on *O'*. However, by using the partial order data flow control method proposed in this thesis, this would not be possible for the following reason: the permission for *Bob* and *Alice* with respect to *O* and *O'* implies that *O'*$\leq$ *Alice, Alice*$<$ *O,* *O*$\leq$ *Bob*, thus *O'*$<$ *Bob* and *Bob* cannot write on *O'*. The method we propose can be used in addition to Trust-based access control, in the IoT or in other contexts. How to do this can be the subject of future research.

4. Data flow control models

As mentioned in the previous section, Mandatory access control models (MAC) are data flow models. These models are restrictive and the access control is regulated in a centralized way by the security authorities of the system. The most common form of Mandatory access control models are *multi-level models (ML)* where security classes are assigned to subjects and objects, and the flow of data from an object x to an object y means existence of a data flow permission from the security class of x into the security class of y. Data flow models seeks to identify all the data flows that should be permissible in a system according to a defined data flow policy.

Many data flow models have been defined in the literature. Our focus will be on the lattice model and its derivatives.

4.1 Lattice-based model

One of the most widely known papers in this area is Denning's papers [31], in this paper a data flow model is defined by a 5-element tuple $<N, P, SC, \bigoplus, \rightarrow >$, where:

- N is the set of objects that may be files, segments, etc. Objects can be considered as data containers.

- P is the set of subjects or users. They are active entities that access objects and are responsible for data flow.

- SC is the set of *security classes*, these security classes are assigned to objects specifying the security classes of the data contained in objects. They are assigned to users to specify their security clearance.

- $\bigoplus$ Is a join operator on SC that specifies the security class of the objects that contain data coming from two different security classes. It can also be applied to any number of security classes; in that case, the result will be the security class of the least upper bound of the security classes included in the join.

- $\rightarrow$ Is a can flow relationship defined between pairs of security classes, so, if we have $A \rightarrow B$, that means that data can flow from *security class A* into *security class B*.

This last operator determines the status *secure* of the system, hence, a system is secure if and only if the relation "$\rightarrow$" holds for any data flow that can occur.

From the previous definition, Denning distinguishes a specific sub-category that forms a universally bounded lattice. Indeed the triple $<SC, \rightarrow, \bigoplus>$ forms an universally bounded lattice under the following assumptions :

1- $<SC, \rightarrow>$ is a partial order set.

2- SC is finite.

3- SC has a minimal with respect to $\rightarrow$.

4- $\bigoplus$ Is defined least upper bound operator.

A very simple example of a universally bounded lattice is a system with two security classes *Public (P)* and *Secret (S)* where data can flow from $P \rightarrow S$. The first assumption is verified, since $\rightarrow$ is a partial order on $\{S,P\}$ (Reflexivity : $S \rightarrow S$ and $P \rightarrow P$. Antisymmetry : $P \rightarrow S$, $P \neq S$ then *not* $S \rightarrow P$. Transitivity: we only have two security classes). The second axiom is verified; we have a finite set of security classes $\{S, P\}$. The third axiom is verified; there is a minimal of SC which is P. Finally, the fourth axiom is verified; the joint operator $\bigoplus$ is defined and has a least upper bound S.

This formalization is the basis of the theory of Lattice-based access control (LBAC) [106]; however, this model does not use the relationship and uses the dominance relationship defined

as follows [40]:

- A $\geq$ B (A dominates B) if and only if B$\rightarrow$A.
- A$>$B if and only if A $\geq$ B and A$\neq$B.
- If A$>$B then not A$\rightarrow$B but B$\rightarrow$A.

We note that the relation $\geq$ is a partial order on SC, with $\bigoplus$ defined and where the upper-bound operator is simply the maximum of the security classes with respect to the $\geq$ relation.

In addition to security classes and the partial order relation $\geq$ over them, in LBAC, data can be structured and *categories* can be created. We assign to users and objects categories, which are non-hierarchical in contrast with the security classes and can be project name, department names, etc. this leads to a *security label (security class, category)* that will be assigned to all the subjects and objects of a system. Moreover, a relation ***dom*** is defined as a partial order over the security labels set as follows:

A :(security class1, category1) dom B :(security class2, category2) if and only if Security class2$\leq$ Security class1 and Category2$\subseteq$ Category1

Table 1: Definition of the relation ***dom***

For example, we have a system with two security classes *Public, Private* where private $\geq$ public, and two categories *NUC* and *EUR*. Figure 9 shows the lattice of security classes (9.a), the lattice of the categories (9.b), and the global lattice of the system (9.c).

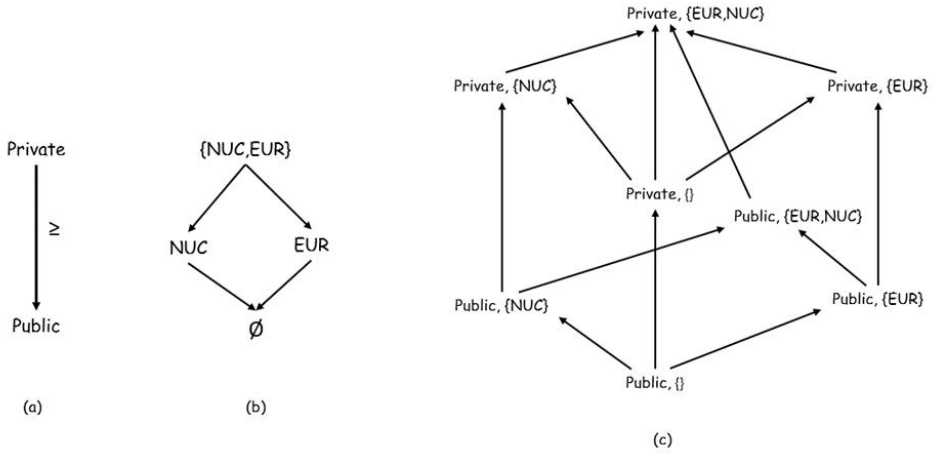


Figure 9: Lattice structures of the example

4.2 Bell-LaPadula model

The Bell–LaPadula model (BLP) [16] is a multi-level security model developed in order to enforce data secrecy in the military sector. Historically, it has been the first model devised to control data flow for security in organizations. This model has been presented in various different ways, and different variations for it have been proposed. We will present it here in a schematic fashion, to emphasize its relation to our own model. In this restrictive model, security labels on objects and clearances for subjects are defined, and a set of access control rules is designed to prevent data flow from a high confidentiality level into a low confidentiality level. In fact, this model is based on the two following properties illustrated in figure 10:

- ***The simple security property***: this property simply states, *"No read up"* meaning that a subject s cannot read an object o if the security label of the object o (Lo) is higher than the clearance of the subject s (Ls). In other words, s can read o if and only if $Ls \geq Lo$.

- ***The * (star) property***: this property simply states, *"No write down"* meaning that a subject s cannot write on an object o if the security label of the object o (Lo) is lower than the clearance of the subject s (Ls). In other words, s can write on o if and only if $Lo \geq Ls$.

In order to maintain confidentiality and data flow control, the *tranquility principle* [123] is applied. The principle states that the classification of a subject or object cannot be changed. This is called the strong tranquility principle. However, if necessary, the principle can be relaxed into a weak tranquility principle, where subjects and objects cannot change levels in ways that violate defined security policies. The motivation behind the weak tranquility principle is that real systems often want to observe the least privilege principle. In this case, every new process is started at an undefined level, even if the owner of this process is cleared to *Secret* level. If the owner accesses an "*Unclassified*" document, the process will automatically get the "*Unclassified*" level. In general, the process can accumulate higher clearance levels as actions require it but without exceeding the owner's maximum clearance [125].

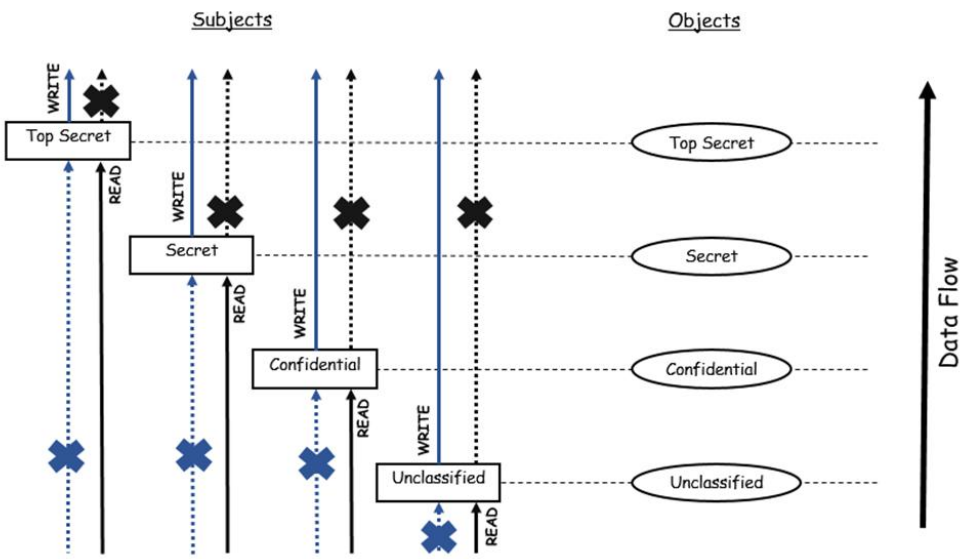


Figure 10: The Bell–LaPadula model (BLP)

The Bell–LaPadula model (BLP) has limitations: secret data can be leaked using covert channels [73], and the model does not take into consideration data integrity, where subjects with lower clearance can alter data for higher security clearance subjects. That is why, in certain applications such as multi-level databases, a ***strong star property*** is defined for integrity concerns. This property states that subjects can only write on objects having security levels equal to their clearance [107].

An important extension of this model is the introduction of data categories. This concept aims to limit the rights of access to certain types of data, and to implement the criterion of *need to know*.

In this case, in the set of data of an organization, one can have categories like *Asia, Europe, Nuclear*, etc. In addition to having levels of sensitivity and authorization, users and objects belong to these categories. This implies the need to define additional constraints such "*a subject can only access data in his own category*". So, for example: *Alice (Secret, {Asia})* can only access data from the category *{Asia}* in her security level and the levels below, and not all data classified "*secret*" or higher. Essentially, this corresponds to the model based on the ***dom*** relationship presented in Section 4.1.

4.3 Biba Integrity Model

The Biba integrity model [19] is, just as BLP, a multi-level security model. It was developed to address data integrity in systems. In fact, data integrity is very important, and, as we said in the previous section, high security data can be corrupted by low clearance subjects. In this model, each subject and object is assigned a level of integrity such as *High (H)*, *Medium (M)*, or *Low (L)*, and the access depends on these integrity levels. The Biba model is defined by two properties dual to BLP properties that prevent subjects with a certain integrity level to access objects with lower levels, *"No read down"* and to write on objects with higher levels *"No write up"*. The properties are defined as follows and illustrated in figure 11:

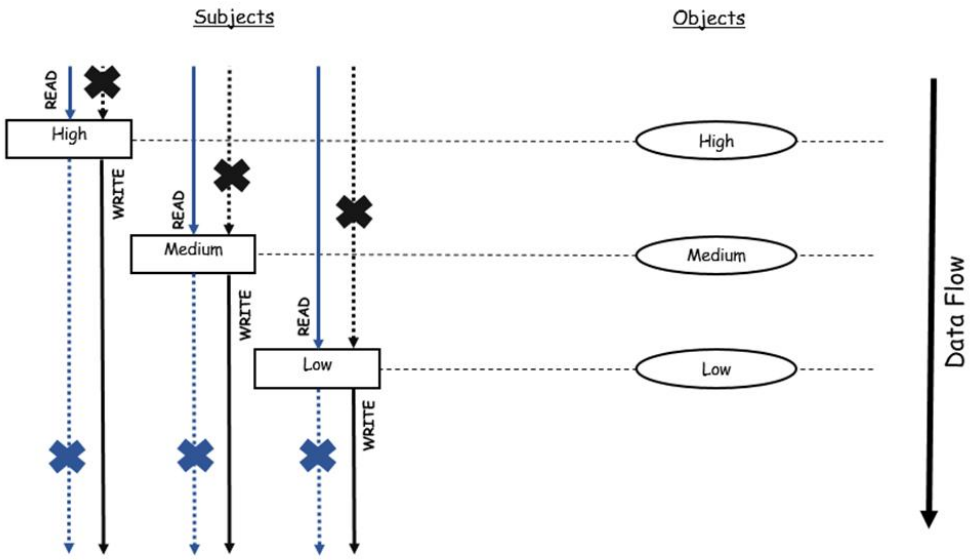


Figure 11: Biba integrity model

- ***The simple integrity property***: this property simply states, *"No read down"*; means that a subject s cannot read an object o if the integrity label of the object o *(Lo)* is lower than the integrity level of the subject s *(Ls)*. In other words, s can read o if and only if $Lo \geq Ls$.

- ***The * (star) integrity property:*** this property simply states, *"No write up"*; means that a subject s cannot write on an object o if the integrity label of the object o *(Lo)* is higher than the integrity level of the subject s *(Ls)*. In other words, s can write on o if and only if $Ls \geq Lo$.

4.4 Chinese wall model

The Chinese wall model [23], also called the *Brewer-Nash* model, was developed to block the transfer of confidential data between organizations in conflict of interests. Historically, its main application was to regulate data flow in consulting companies where different departments of a company would consult for competing organizations. An analysis of access histories is performed before granting any access. This model introduces new concepts such as organizations, and conflict-of-interest-classes.

- *Organizations*: they are organizations that access sets of data objects in their activities.
- *Conflict-of-interest-classes*: they are classes that regroup companies that are in conflict.

The idea of this model is that no data can flow between companies included in the same conflict-of-interest class. This can be prevented with the two following properties:

- ***The simple property***: states that a subject s can read an object o if and only if :
  - All objects that s has already read are in organizations that are in a conflict class different from the conflict class of the organization of o.
  - The subject s has already read an object in the same organization of o.
- ***The * (star) property***: states that a subject s can write on an object o if and only if :
  - The subject s can access o according to the simple property.
  - The subject s can only read objects that are in the same organization of o.

These two properties fulfill the need for confidentiality and integrity; in fact, a subject cannot receive data from organizations in the same conflict-of-interest class, and an object cannot receive data from organizations in the same conflict-of-interest class. However, confidential data can be leaked through covert channels, and this model is difficult to apply for physical users. [22]

5. Other data flow control approaches

The previous review has covered the classical approaches to access control, data flow control and data protection. The literature in this general area is vast and we will limit ourselves to the small subset that has influenced our work. We point out again that, although much research mentions *information* flow control, we limit ourselves to *data* flow control, because the former

implies control of inference mechanisms, which we don't claim in our approach. However at times we may mention the former, when the authors follow this terminology.

The area of data flow control for security has been extensively studied over the years, and the literature in this general area is also extensive, and concerns different contexts. Bedford et al. [14,15], and Desharnais et al. [32] deal with the context of language-based information security. Which means, using programing language techniques to provide information security assurance and enforce information flow control policies.

We are more interested in the context of data security, and most of the literature in this context is dedicated to the presentation of abstract models, or to the problem of avoidance of unwanted data flows. In this section, we present only the contributions that are closely related to our approach. We will not focus on papers that deal with other research contexts such as workflows, usage control, etc.

In particular, there are no papers that describe algorithms and simulations for data flow analysis for security, with quantitative results such as those that we describe in Chapter 5.

From the previous sections, we note that the majority of the classical access control models do not have mechanisms for data flow control; the latter is particular to the family of multi-level models MAC. Other research ideas have been proposed in the literature, which tends towards two approaches: either to configure the classical models to implement data flow control, or to propose new data flow control models for new technologies such as the Internet of Things and the cloud.

Since most Mandatory access control models deal with centralized systems, Myers et al. [83] have introduced the concept of enforcing data flow control in decentralized systems. This concept is based on a label template that can express complex security policies. A security label is first formed by a set of users, owners of information containers associated with this label. Each owner is then associated with a certain number of users, called readers, authorized to access containers associated with this security label. The security policy is interpreted as follows: for a user to be authorized to access the container associated with a security label, it must be designated as a reader of this security label by all owners of this security label. Owners if necessary can modify the security label and add more readers. This model will be the basis of several approaches aiming to enforce data flow control in the IoT and the cloud.

Since RBAC is the most widely used model in organizations, we see a particular interest in this model. In fact, to assure data flow control, some approaches try to enforce multi-level properties into RBAC systems. Among these, we find the paper by Crampton [30] where new mechanisms for permission inheritance within role hierarchies are defined. The author shows that the permission inheritance that exists in RBAC does not directly support a simulation of the *-property of BLP which requires that certain permission be inherited downwards. To achieve this goal, the author proposes some constraints that map permission inheritance into a partial function with hierarchies that can be used to simulate BLP behaviour. Furthermore, these permissions are assigned to roles that can be interpreted as security labels.

Some of the most interesting contributions on the subject of data flow control with RBAC are due to Osborn [91,90]. In [91], a configuration of RBAC to enforce mandatory and discretionary access control policies is proposed where it is shown that the two Bell-LaPadula rules can be expressed using RBAC constraints on role hierarchies and user assignment. In this case, permissions are assigned to roles on objects that have security labels (for example ru stand for read-unclassified, and ws stand for write-secret), and role hierarchy is mapped to a lattice with respect to the simple property and the * (star) property. After that, classified users are assigned to roles according to new constraints defined on user-assignment functions. An example is shown in figure 12.

Looking at the role hierarchy of Figure 12, we see that roles labelled *R1*, *R3* can be assigned to unclassified users, roles labelled *R2*, and *R6* can be assigned to secret users, and so on. However, although this technique guarantees data flow control at some levels of the hierarchy, it can create situations of permission conflicts and data flow situations that violates the Bell-LaPadula properties.

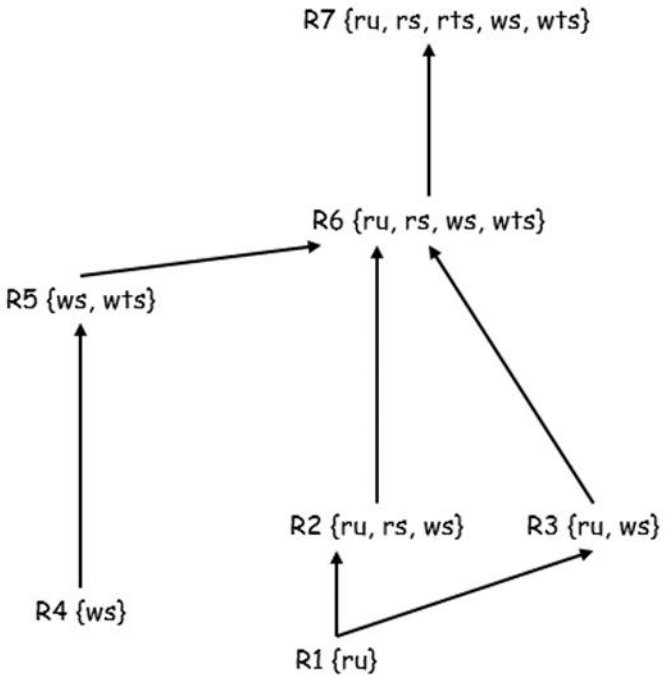


Figure 12: A role hierarchy with security labels (adapted from [86]).

For example, we have role $R7$ that present permission conflicts that can cause data flow (a user with this role can read-top secret and write-secret), so this role cannot be assigned to any user. This situation and many others are presented and solutions are proposed by Tuval et al. [124].

Nyanchama et al. [86] proposed a role graph model where nodes in an acyclic graph represent roles, and edges are constructed according to every role's set of permission and with respect to the inclusion relationship. Therefore, the role graph will be equivalent to the role hierarchy.

This model is used in the follow-up paper [90] in which it is shown how a role graph can be mapped on a data-flow graph. The resulting data flow graph represents a flow relationship between objects derived from the permission assignments in consideration of role relationships, and constraints on sessions between different roles.

The starting point of the mapping is a role graph, where roles are represented by nodes in an acyclic graph. Each role r is assigned a set *Effective(r)* that represents all privileges available to users that have the role r, including permission inherited from the junior roles of r. The edges

in the role graph represent the *is-junior* relationship: $r1 \rightarrow r2$ if and only if *Effective (r1)*$\subset$ *Effective (r2)*.

The construction of the mapping is equivalent to the construction of a *Can Flow* relationship as defined by us from objects to other objects. Such flow results from combinations of privileges in a role, a similar relationship will be used in our model but in a more general way. Two algorithms are proposed to construct the *Can Flow* relationship and the mapping; The first algorithm, *FlowStart* creates nodes labelled with the role name and the privilege, for example, a role r_1 with a privilege read on object o_1 (o_1, r) will result in a node (r_1,o_1,r). Once the nodes defined, edges are added from nodes labelled with a read privilege into nodes labelled with write privilege with respect to the fact that the two nodes at the two ends of the edge are labeled with the same role. Finally, edges are added between nodes containing the same object.

The second algorithm, *CanFlow* takes as input the output of *FlowStart* and aims to collapse cycles present in its input. To do that, all the nodes in the cycle are replaced by a single node whose label is the union of the labels on the cycle. The result is an acyclic graph that after removing references to reads and writes and the originating role shows a graph flow between objects of the system. Figure 13 shows an example where all the steps of the mapping are represented.

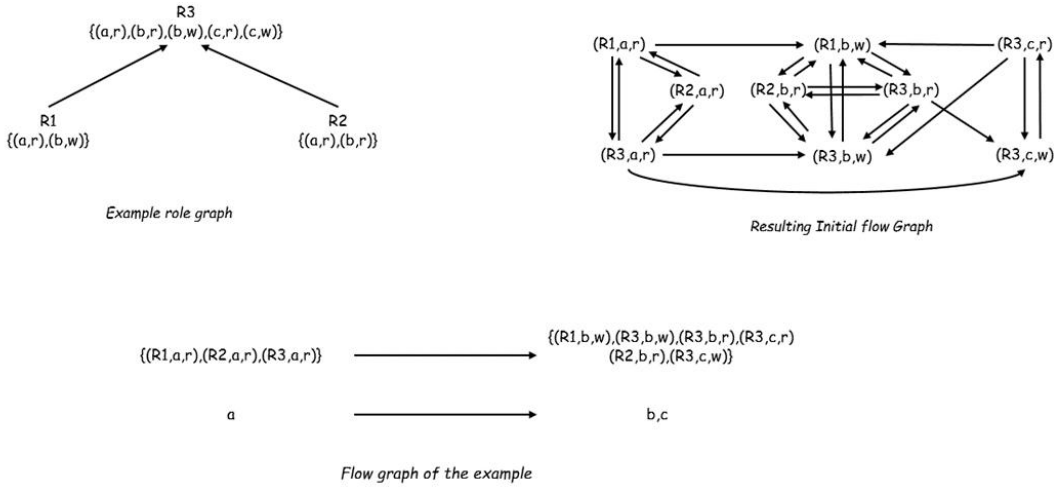


Figure 13: Example of a mapping (adapted from [90])

The final flow graph produced by the two algorithms can be mapped into an LBAC model

with a single security label. However, this may not always be possible. To determine if the result is a lattice, we must verify that for any given pair of nodes in the resulting graph, there is a unique least upper bound. If it is not the case, we can add it or merge all of the least upper bounds for a given pair of nodes into a single node. These two operations can be considered weaknesses of the lattice model, since there is no practical way to do these additions and mergers. We will discuss those weaknesses in the next chapter. Our method is similar to Osborn's, but using the concept of partial order instead of lattice we were able to simplify it. Further, Osborn's analysis is only applicable to RBAC systems in contrast to ours that is more general.

Gofman et al. [45] proposed an optimization of Osborn's algorithm using incremental methods; the resulting algorithm is proven to be less time and space consuming. Their algorithm also provides information such as the nature of the data flow, whether it is direct or if it is by transitivity, and to handle changes in RBAC relations. These points will be developed in our work. They also extend the incremental analysis to support parameterized RBAC.

A follow-up of the last paper is presented by Gofman et al [46], where the incremental algorithm is used to develop *RBAC-PAT*, a tool for analyzing RBAC and administrative Role-based access control ARBAC policies, which supports analysis of various properties including information flows.

The contribution that is closest to ours is Amthor et al. [9]. The authors present WorSE, a tool for security policy engineering. One of the applications of this tool is the data-flow analysis to detect covert information flows in large access control systems. The analysis consists of three steps: protection state extraction, model rewriting, and information flow analysis.

- Extracting the protection state: this step consists of the extraction of the system protection state, including subjects, objects and right assignments, this result in an access control matrix ACM. We use the example of this paper. Consider a company with two departments, *"Research & Development (R&D)"* and *"Sales"*. *Ann* is working in *R&D* as a project manager, *Bob* belongs to her project staff, and *Chris* is working in Sales. *Ann* has read and write access to *ProjectXFiles* and *ProjectXBoard*, resulting in information being able to flow from both files to *Ann* and vice versa. *Bob* has a read right on *ProjectXBoard*. Moreover, *Bob* has read and write permission on *SalesBoard*. *Chris* is working on *SalesFlyer* and has read and write access to this document. Additionally, he is allowed to read *SalesBoard*. The following figure illustrates the

protection state and the resulting ACM:

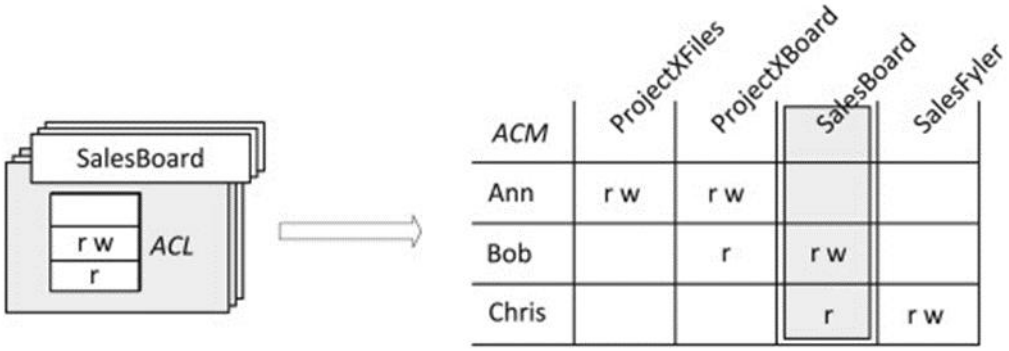


Figure 14: Access control matrix of the example [9]

- Rewriting the ACM : this step rewrites the ACM as a directed graph with subjects and objects represented as nodes and right assignment as edges. The following figure represents the directed graph obtained from the ACM of the previous example:

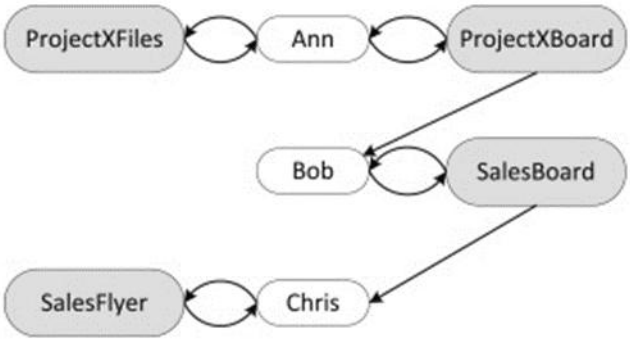


Figure 15: Information flow graph of the example [9]

- Information flow analysis: in this step an analysis of the flow graph is conducted according to specific analysis goals and questions that are derived from the access control policy.

In this paper, the authors present a powerful tool to maximize the efficiency and quality of security policy specification which includes data flow analysis as a function of the tool. However, the authors do not provide the detailed algorithm analysis and simulations that we provide in our work (see Chapter 5), hence they provide no information on the sizes up to which their tool could be practical. In addition, they do not introduce any conclusions about secrecy levels, partial

orders or multi-level system that can be gained from the analysis. They are only concerned about the analysis regarding covert information flow paths. Finally, the authors consider the general case where the permission can change but seem to miss the interesting conclusions that can be drawn when the permissions are fixed, as we do in our work (see chapter 5).

6 Other related contributions

Several other papers exist that are related to our subject but with different aims and conclusions than ours. We will review them more quickly than the previous ones. Some papers have dealt with methods for blocking illegal flows, especially in RBAC, or information flow control. Samarati et al. [104,105] proposed a model that overcomes the vulnerability of Discretionary access control by completing it with mechanisms to prevent illegal flows caused by Trojan horses. Our purpose is dual, i.e. to identify (in a current configuration) or permit (in a configuration to be established) all legal flows in an IoT system.

Izaki et al. [38] use RBAC to control information flow. They define safe roles where only legal flows can occur, then objects methods are classified and used to construct information flow graphs where illegal flows can be identified.

Nakamura et al. [84] developed a synchronization protocol to prevent illegal flows in RBAC systems, where operations are blocked if they illegally write or read over objects. The same direction is taken by Chon and al. in [25] but this time a role lock over objects is introduced, the objects are locked before any operation and an operation are aborted if the locking fails to prevent illegal flow. Zhang and Yang [134] clearly explain the problem of data leakage in RBAC-based models and propose the use of mandatory access control methods to produce label assignments to make leakage impossible.

Alongside with the research on enforcing data flow control in classical access control models, needs to enforce it in new technologies such as the Cloud or the Internet of Things have surfaced recently. Many papers address the issue of protecting data within IoT or cloud services. In all cases, data should not flow to unauthorized parties, including cloud insiders as well as cloud users [109,94].

Bacon et al. [13, 57] address the issue of end-to-end information flow control which is retarding the evolution of large-scale cloud computing. They propose a data tagging data flow

control scheme applicable in large-scale cloud and IoT. This dynamic data tagging replaces the classical static data tagging that is clearly not sufficient for large-scale distributed systems. Their approach provides an end-to-end security scheme that supports data isolation, isolation between users and services and isolation between some services. They achieve this by combining information flow control with RBAC policies to provide strong protection for data. This paper suggests that data tagging is necessary for configuring data flow security in the cloud and the IoT. We will discuss this further in the next chapters.

Singh et al. [114,115] identifies twelve security considerations for cloud supported IoT. The authors also investigate the use of Information Flow Control (IFC) to manage and audit data flows in cloud computing. They concluded that IFC has great potential in the broader IoT context. However, the sheer scale and the dynamic, federated nature of the IoT pose a number of significant research challenges that they identify in the paper.

Bacon et al. [12], and Sandhu et al. [108] describe the properties of cloud services and review a range of information flow control models and implementations to identify opportunities for using information flow control to manage and audit data flows in cloud computing context. They also discuss the potentiality of using it in the context of IoT. Finally, they identify some research challenges toward this issue.

Schütte et al. [110] is using the same kind of labelling that we will be using in our model. This paper presents LUCON a framework for distributed systems that enforce data flows across services by controlling how messages may be routed across services and how they are combined and processed. LUCON is adapted to message-based IoT system, where we find services that communicate with messages. A service accept a set of input messages and emits a set of output messages that may cross their trusted domains. The authors define the following concepts:

- Message labels: noted L classify a message according to data sources (example L = *{Personal data}*), or according to their secrecy level (example L= *{classification (secret)}*).

- Service Properties: noted P are used to describe the nature of services. For example, P= *{store}*.

- Message routes: represent sequences of statements, which can be called external services, and assign values to variables or control executions of the next statement (condition).

- Trust domains: they englobe services and routes. Routes definition and service control in

terms of propagating message labels are done only inside the trust domain.
LUCON uses the dynamic data flow control and the static model checking of data flows. Dynamic data flow control is used only to detect data leaks as they occur. The basic idea of their dynamic data flow control is to assign a set of labels to messages when they enter the system and to modify these labels as messages are processed by services. In addition, whenever a message is about to be sent to an external service, the policy is consulted to check whether a correspondingly labelled message may enter the service or not. Static model –checking is used to verify that a message route is free of data leaks and this can be done via a Prolog compilation. Finally, the authors present a runtime evaluation of policies under LUCON, and a compilation of routes into Prolog programs to achieve static model checking as mentioned above.

Blackstock et al. [20] 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 [85] use a formalism based on identifying separate subjects, objects, separate reading, and writing authorizations, rather than on a single CanFlow relationship as we will do in our model. They define 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.

The approach that is most similar to ours is Khobragade et al. [65]. In this paper, a distinction is made between subjects, objects, 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).

Many papers in this research area propose the use of authentication, encryption and access control methods in the IoT. Many other papers such as La Marra et al. [70, 71], and Zhang et al. [133] propose a usage control enforcement in the IoT. Although authentication, encryption and access control are mechanisms for realizing flow control, we will concentrate ourselves on methods for designing the overall flow control. These other techniques will undoubtedly be useful for implementing our method.

Although IoT networks are usually represented as directed graphs, we could not find a single paper that references or uses the results and methods of graph theory that are presented in our work. The use of these results, instead of the classical lattice model, is the salient distinguishing characteristic of our model. This makes our model simpler, more generic and more efficient to implement.

7. Conclusion

The purpose of information flow control is to control the diffusion of information in a system. An information flow policy defines authorized or prohibited information flows. A violation of the flow policy is detected when a user accesses all or part of information to which he should not have access according to this policy.

The most widely used information flow control policy model is the lattice model. In this model, the authorized information flows are described forming a lattice form. However, practically it is not always the case.

Other information flow control policy models have been proposed, but the majority of them are based on the lattice model either by modifying it, improving it or present a variant of it. Not a single work proposed a different approach. That what we will try to do in our work.

In the next chapter, we will introduce our data flow model based on graph theory and order theory results and we will discuss why it is more practical than the lattice model.

Chapter 4

Basic concepts of order theory and graph theory and partial order model

1. Introduction

In this chapter, we define some notions and algorithms used in our method and that constitutes the basis of our work. Recall that we are proposing a method to control the data flow in distributed systems to meet secrecy, integrity and privacy requirements. We have seen that the most widely known models in this research area are based on concepts from lattice theory. We have already mentioned that we use instead notions of order theory and graph theory, which are simpler and more general.

2. Order theory

Let R be a binary relation, we say that R is an order relation on a set E if and only if [113]:

- R is reflexive, meaning: $\forall\ x \in E,\ xRx$.
- R is transitive, meaning: $\forall\ x \in E, \forall\ y \in E, \forall\ z \in E,\ (\ xRy\ and\ yRz \rightarrow xRz)$.
- R is antisymmetric, meaning : $\forall\ x \in E, \forall\ y \in E,\ (\ xRy\ and\ yRx \rightarrow x{=}y)$.

Partial order relation

A partial order relation is an order relation (reflexive, transitive, and antisymmetric) that is not total. This means that the following connex *property* does not hold:

$$\forall x\ \forall y\ (x \in E \bigwedge\ y \in E) \rightarrow (xRy \bigvee\ yRx).$$

The connex property states that all the pairs (x,y) from the ordered set E must be in relation R, if it holds we call R a *total order.*[69]

3. Graph theory notions

A *finite graph* is a representation of relations between elements, called *nodes*, of a finite set. These relations can be oriented or not. We are only interested in oriented ones, representing

potential data flows.

Directed graph

A *directed graph* [52], or digraph is a graph that is made up of a set of vertices connected by directed edges. In formal terms, a directed graph is a pair $D = (V, A)$ where:

- V is a set of *vertices or nodes.*

- A is an ordered pairs of directed edges representing a relation between the nodes.

Reachability

Reachability refers to the ability to reach one node from another within a graph . We say that a node a is *reachable* from a node b if there exists a sequence of adjacent edges (a path) which starts with b and ends with a. *Reachability* can also refer to the ability to determine all the nodes reachable from a given node. Reachability can be computed in linear time using algorithms such as breadth first search or depth-first search.

Depth-first search algorithm

Is an algorithm for traversing or searching graphs. The algorithm starts at a designed node (*root node*) and explores as far as possible along each branch before backtracking. This algorithm has a complexity of $O(|V| + |A|)$ in the worst case, where $|V|$ is the cardinality of the set of nodes and $|A|$ is the cardinality of the set of edges.

Algorithm 1 Depth-first search algorithm [67]

Input: a graph D and a node a of G
Output: R a set of all nodes reachable from a marked as *Explored*

```
1: procedure DFS(D,A)
2:     mark a as Explored and add a to R
3:     for each edge(a,b) incident to a  do
4:         if b is not marked Explored then  then
5:             recursively invoke DFS(D,b)
6:         end if
7:     end for
8: end procedure
```

Cycles and strongly connected components

A cycle in a digraph is a subgraph of at least two nodes in which the nodes are mutually reachable. An acyclic digraph has no cycles, and is also called DAG , for directed acyclic graph. A diagraph D is strongly connected, if and only if every two nodes are mutually reachable.

A *strongly connected component* of a digraph D is a subgraph of D that is strongly connected, and is maximal with this property: no additional edges from D can be included in the subgraph without breaking its property of being strongly connected. [52]

Several algorithms are known to find the strongly connected components of diagraphs. These algorithms can do this in linear time such as: Kosaraju's algorithm, and Tarjan's strongly connected components algorithm .

Tarjan's strongly connected components algorithm

This algorithm [122] is based on the depth-first search algorithm and has a complexity of $O(|N|+|E|)$ in its worse case.

Condensation and partial orders

The condensation of a diagraph D is formed by merging the nodes of each strongly connected component into a single node. The resulting graph is acyclic (is a DAG) and is proved to have the same kind of connectedness as D [52]. In other words. if a node a is reachable from another node b in D, then the node in which a is condensed is reachable from the node in which b is condensed. Assuming reflexivity and transitivity in DAGs, the reachability relation in DAGs forms a partial order. This is because all the symmetric relationships are removed in condensation and thus a DAG can be taken to represent an antisymmetric, transitive, reflexive relationship, as defined above.

4. Multi-level access control, directed graphs and partial orders

The graph theory and partial order notions above are used in [75]. That paper shows that digraphs can be used to represent data flows networks of access control systems. Using the graph theory results seen in the previous section, it can be shown that the data flow networks are partial orders of maximal strongly connected components and the multi-level system they implement can be seen. This leads to the conclusion that every data flow diagraph can be understood as a partial order of components, and this can be shown using graph theory and partial order result. To better understand this, let us take an example. In Figure 16, we have the following digraph where the arrows can be interpreted to denote possible data flows among entities in a system (bidirectional arrows are used to represent two arrows in each direction).

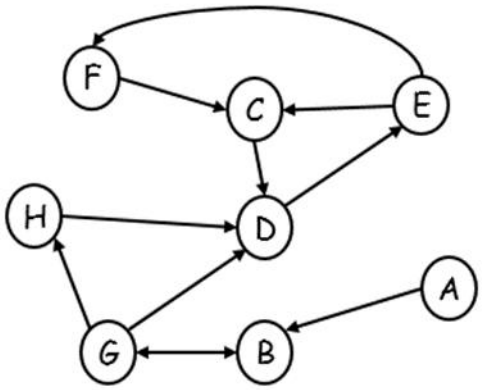


Figure 16: Digraph showing allowed data flows in a network

In the example, we see that entities B and G send and receive data from each other, meaning that these two entities can know or store the same data, so they can be considered one entity for data flow control. We speak of a maximal strongly connected component $\{B, G\}$. The entities forming the components can be condensed into a single entity. Proceeding in this way, we detect another strongly connected component $\{E, F, C, D\}$ and we condense it into a single entity. The component digraph in Figure 17 is obtained.

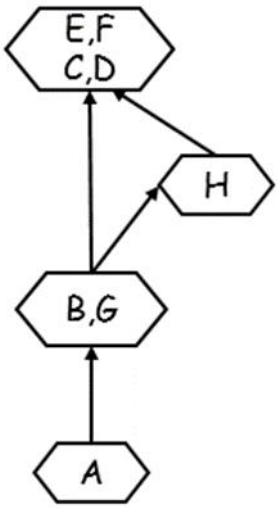


Figure 17: Data flow digraph of figure 16 as a partial order of components

For access and flow control, this result is very useful. In fact, each strongly connected component in Figure 17 represents an entity within which there can be complete data sharing without any secrecy since the entities in the component can know or store the same data, by passing them all along the edges. Then data can move upward into the next component in the partial order and cannot move down, this implements secrecy.

One other interesting conclusion is that from a data flow digraph such as the one in Figure 17, we can construct a multi-level access control system that implements it. We can do that by interpreting the nodes in Figure 17 as security levels and associating subjects and objects with them.

This multi-level access control system can be constructed in the following way:

- Data flow is permitted between elements of a component

- Data flow is permitted between elements of different components following the partial order relationship.

The paper in question proposes an example of the multi-level access control system construction.

5. Formalization of confidentiality and integrity concepts

In this section, we present the formalization that we use for reasoning about confidentiality and integrity presented in [74]. This paper introduced a new method for reasoning about properties of access control systems and provided a formalization of concepts of confidentiality and integrity based on some defined predicates. This method will be used and will be the base of our work. Starting from the predicates presented in this paper, we derive some other ones that are more appropriate to reason and prove data flow properties.

The method presented in the paper uses the following concepts:

- Three entities: *Subjects (S), Objects (O),* and *Data Variables (x).* Objects are considered containers of data variables.

- Two relationships that expresse access control rules between subjects and objects: *CanRead (CR), CanWrite (CW). CR(S_1,O_1)* means that subject S_1 has a read right on object O_1, and *CW(S_1,O_1)* means that subject S_1 has a write right over object O_1.

- Two relationships between subjects, objects, and data variables *CanKnow(CK), CanStore(CS).*

Subjects can know data variables by reading them from objects they can access, and objects can store data variables that are written on them by subjects authorized to do so. This can be done only once we assume that at some point, some subjects can know something or some objects can store something. This will be expressed with unconditional relationships, so, if $CS(O_1,x)$ is given unconditionally, we know that data variable x is in O_1. The following table summaries the rules of the *CK* and *CS* relationships:

• Unconditional relationships are expressed in the form: *CK(S,x)* or *CS(O,x)*.
• The inference rule for *CK* is : *(CS (O,x)* $\bigwedge$ *CR(S,O))* $\rightarrow$ *CK(S,x)*.
(If O can store x and S can read O, then S can know x)
• The inference rule for *CS* is : *(CK (S,x)*$\bigwedge$ *CW(S,O))* $\rightarrow$ *CS(O,x)*.
(If S can know x and S can write on O, then O can store x)
• All *CS* or *CK* relationships must be true either unconditionally or by the one of inference rules above

Table 2: Inference rules for *CK* and *CS* relationships

Auxiliary functions *CSS* and *CKS* which are equivalent to *CS* and *CK* can be used as follows:

- For any *S*, *CanKnowSet (S)* or *CKS (S)* is the set of data variable x for which *CK(S,x)* is true: *CKS(S) =def {x | CK(S,x) is true}.*

- For any *O*, *CanStoreSet (O)* or *CSS (O)* is the set of data variable x for which *CS(O,x)* is true: *CSS(S) =def {x | CS(S,x) is true}.*

The paper also shows the applicability of this formalization to a number of classical access control models such as: Bell-La Padula, Biba, Lattice-Based, RBAC, High-WaterMark, Chinese Wall... Some applications required the addition of some concepts such as variable labelling, which is necessary for the application to multi-level systems. The results show that the formalism is usable for those models, with their proprieties proven, and this formalism provides a framework for constructing systems that have confidentiality and integrity properties. We will not focus on that paper now, since we are only interested in the basic concepts of the method used in our work.

6. Conclusion

The presented method uses results of directed graph theory, showing that, under reasonable assumptions, it is possible to find a partial order of components in any data flow graph of any

access control system. Therefore, it is in principle possible to determinate which multi-level system it implements. In this chapter, we have presented the basic notions and results that are used in our work.

Chapter 5

Data flow analysis from capability lists, with application to RBAC and LaBAC

1. Introduction

In this chapter, we present our work, published in [120]. A synthesis of this paper is presented in the next section, while a slightly augmented version of it is attached at the end of the synthesis.

2. Paper synthesis

In research on data security and privacy, a question arises: assuming that certain data are available somewhere in a network; where else can they end up? Or more precisely, which subjects can be able to know, or which objects can be able to store, data originating from objects in the network? This problem is difficult to define because data can move in many different ways. However, an answer is especially important for objects containing secret information.

This problem has been extensively studied in other contexts such as workflows, provenance analysis, etc. However, there is no work in the literature like our showing by algorithm analysis and simulations, that data-flow analysis can be practical for systems up to tens of thousands of subjects and objects, while this is an important fact since it can motivate tool development .

The literature related to this problem often refers to the lattice model, even though this model is not very practical for various reasons:

- Lattices require joins and meets that rarely exist in organizations or networks and may force the inclusion of corresponding unwanted entities and dataflows.

- Lattices require the creation of upper bounds that in terms of data flow should be entities that can know everything and lower bounds that can know nothing. Corresponding entities rarely exist in systems in practice.

- Lattices do not tolerate symmetric relationships, which exist in practice.

In our work, we identify and use a partial order of components that is more realistic than the lattice model. In fact, the partial order model does not require any adaptations and always

exists inside any access control system including those based on roles and attributes.

To identify the partial orders in security networks we use two viewpoints: the inference-rule one, based on the already mentioned *CanRead (CR), CanWrite (CW), CanKnow (CK)*, and *CanStore (CS)* predicates with their inference rules. This gives rise to the *CHS-equivalence* and order. The second viewpoint is the graph-theoretical one based on efficient graph theory algorithms. We identify the algorithms to be used; we combine them to answer problems such as the single object data flow problem that concerns the subjects and objects that can know or store the data of given objects, or the global object data flow problem, to identify the boundaries of data reach for each object in the system. This viewpoint gives rise to the *CanFlow CF-equivalence* and order. Note that the two relationships *CHS-equivalence*, *CF- equivalence* are usually equivalent, except for special cases mentioned in the paper.

We pay more attention to the graph-theoretical viewpoint, and we explore the practical computability of finding *CF-equivalences* with their partial orders. To do this we give the complexity of the used algorithms and their combinations. It is shown that only polynomial-time algorithms used in our method.

To have more realistic estimates, simulations are performed using MATLAB in order to check real execution times. The results show the feasibility of our method for systems up to many thousands of subjects and objects (approximately 120,000), and since the complexity is polynomial, calculation times and system size will improve with more powerful computers. The obtained results are useful for data flow control and can be considered to be stepping stones to study tractability for complex access control systems, since these results can be used towards the development of efficient tools.

Furthermore, other results of interest are obtained:

- For RBAC, our method can be used to determine whether secrecy constraints are violated or not, and to determine in which objects certain data should be put to be secret of certain roles.

- For role engineering in RBAC, our results can be used to detect and eliminate roles that can know nothing, merge roles that can know the same data, and merge objects and permission related to them, if there are objects that can store the same data.

- Our method can be used to create Label-based access control systems based on partial

orders rather than lattices.

All this is described in the following paper.

3. The attached paper

Information Processing Letters 141 (2019) 30–40

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Data flow analysis from capability lists, with application to RBAC

Abdelouadoud Stambouli *, Luigi Logrippo

Université du Québec en Outaouais, Department of Computer Science and Engineering, Gatineau, Québec, J8Y 3G5, Canada

ARTICLE INFO

Article history:
Received 13 October 2017
Received in revised form 13 June 2018
Accepted 3 September 2018
Available online 20 September 2018
Communicated by Luca Viganò

Keywords:
Safety/security in digital systems
Access control
Data flow control
Role based access control
Label based access control

ABSTRACT

For the analysis of access control networks, where capability lists, access control matrices, or RBAC permissions are available, it can be very useful to be able to determine which subjects can be able to know, or which objects can be able to store, data originating from objects in the network. This information can be used in order to answer questions of secrecy, integrity and privacy, related to the data flow analysis problem. On the basis of a logical method, we present a graphical formalism capable to represent such networks and for which the data flow problems can be defined. We present algorithms to calculate answers to data flow questions. Complexity analysis and simulations show that these questions can be practically answered for networks of sizes up to several tens of thousands of subjects and objects, which is the size of many real-life organizations. We also show that the results obtained can be used in the process of role engineering in Role based access control, for determining secrecy levels, as well as for eliminating or combining roles or objects. Finally, a method is demonstrated to go from capability lists to Label-based access control systems.

1. Introduction and motivation

A fundamental question in research on information security and privacy in data networks is the following: assuming that certain data are available somewhere in a network, where else can they end up? To answer this question realistically, one must consider the information supports and the information paths available in the network. Obviously, in general this is an intractable problem, even difficult to define, because information can move, or can be blocked, in a network in many different ways, depending on access control, encryption, inference, channel characteristics, covert channels, protocols, errors, human intervention, etc.

We take a view based on access control methods according to established access control theory. First, we consider data only and not information (namely, we do not take into account data flows generated by inferring information from data and then encoding it in other data). Second, we assume that data can only be available in a network by being *known to subjects* or *stored in objects*. Third, we assume that data can move between subjects and objects over perfect channels according to policies specified by access control matrices, capability lists, permission in Role based access control, routing tables, or other methods directly translatable into any of these. If an object can store some data, and a subject can read from the object, then the subject can know the data. The subject can then write these data in other objects where they will be stored and from where other subjects can read them and so on. We show that, in a snapshot of the system where the capabilities do not change, the problem is very tractable and

* Corresponding author.
E-mail addresses: staa16@uqo.ca (A. Stambouli), luigi@uqo.ca (L. Logrippo).

https://doi.org/10.1016/j.ipl.2018.09.001
0020-0190/

allows solutions that can be stepping-stones towards other more complex problems.

Essentially, our approach is based on the following steps:

1. Using capability lists, we construct logical models, represented as graphs, of all possible data transfer paths in networks; the nodes in the graphs are subjects or objects, and the edges are reading and writing capabilities.
2. These models can then be explored in order to see where the data contained in certain data objects can possibly end up (the data flow problem).

As a byproduct of these steps, we are also able to determine secrecy areas and levels in the network. For a given object, we can determine that it is a *secret of* certain subjects and objects. For a network, we can determine what are its less or more secret subject and objects.

Our method is shown to be practically feasible up to a fairly large number of subjects and objects, enough to be able to apply it to many realistic systems. Although the goal is easily seen, practically important and not difficult to achieve, we are not aware of similar experiences having been reported in the literature, or of industrial tools having been produced towards it – two research tools are reported in [1] [10] (see below), but without time estimates. Therefore, we believe that this paper fills a void, and will inspire research towards applications, improvements and extensions.

Being able to practically solve our problem is important in order to answer several questions in security and privacy:

- The *secrecy* question: can secret data be known or stored where they shouldn't?
- The *integrity* question: can data coming from unreliable databases end up being stored in data bases that should be highly reliable?
- The *availability* question: can data reach users or data bases where it might be needed?

We briefly consider the application of our method to RBAC [7][21]. In spite of the practical success of RBAC, few publications address the problem of data flow analysis in RBAC, although this problem is easily understood. We present certain conclusions that can be reached by the analysis of such flows, which have consequences for the placement of secret data in RBAC systems and for role engineering. Concerning this second point, we show that certain roles can be considered unnecessary, either because they cannot receive any data, or because they can be merged with others.

The paper is structured as follows: Section 2 is our concise literature review. In section 3 we review some basic concepts and present our basic definitions. Section 4 presents a small example to illustrate our goals and methods. Section 5 presents the algorithms we use, with an evaluation of their theoretical complexity, leading to the conclusion that we need only polynomial algorithms. Section 6 presents our simulation method and simulation results. Section 7 discusses a specific technical point concerning the two equivalence relationships that we use in our work. Section 8 provides a larger example. Section 9 discusses some applications of our method to RBAC, especially for role engineering. In Section 10 a method is presented to go from capability lists to Label-based access control systems. Section 11 concludes the paper and mentions subjects for future research, which include methods for flow control in the Cloud and in the Internet of Things.

This paper is dedicated to the memory of Dr. Sylvia Louise Osborn, a pioneer in this research area and a source of encouragement for our work.

2. Literature review

While the literature on data flow control is extensive, there is almost no literature on algorithms and simulation results for data flow analysis in access control networks. We will mention only a few papers that we consider specifically related to our work.

Several of these papers are related to RBAC. The basic paper on data flow analysis in RBAC is the one of Osborn [21] in which it is shown how a role graph can be mapped on a data flow graph. The resulting data flow graph is proven to be the result of a flow relationship between objects, this relationship is derived from the permission assignments in consideration of role relationships, and constraints on sessions between different roles.

Gofman et al. [10] proposed an optimization of Osborn's algorithm using incremental methods, the resulting algorithm is proven to be less time and space consuming. Their algorithm provides also information such as the nature of the data flow, whether it is direct or if it is by transitivity. These points will be developed in our paper. A follow up of this article [11] presented a tool RBAC-PAT for the analysis of properties of RBAC and ARBAC policies including information flow.

Amthor et al. [1] present WorSE, a comprehensive workbench for model-based security engineering. They use both a state-based representation of security systems, and a relational one such as ours. Similar to us, they aim to detect indirect flows, which they call *covert flows*. They also use the concept of information flow graphs, which they reduce according to information equivalence relationships.

The last three papers will be further discussed in our Conclusions.

Several papers have dealt with methods for blocking illegal flows in RBAC, or information flow control. Samarati et al. [20] proposed a model that overcomes the vulnerability of discretionary access control by completing it with mechanisms to prevent illegal flows caused by Trojan horses. Izaki et al. [13] use RBAC to control information flow. They define safe roles where only legal flows occur, then objects methods are classified and used to construct an information flow graph in which illegal flows can be identified. Nakamura et al. [19] developed a synchronization protocol to prevent illegal flows in RBAC systems, where operations are blocked if they illegally write or read over objects. The same direction is taken by Chon et al. in [5] but this time a role lock over objects is introduced, the objects are locked before any operation and an operation is aborted if the locking fails to prevent illegal

flow. Zhang and Yang [23] clearly explain the problem of data leakage in RBAC-based models and propose the use of Mandatory access control methods to produce label assignments to make leakage impossible. Although some of these approaches are related to ours, the aim of our research is of determining efficiently what data flows are possible, given certain access control capabilities, rather than of controlling the flow of information, or of blocking certain flows.

In the abundant literature on RBAC role mining and engineering there is no mention of the possibility of merging or eliminating roles because of data flow characteristics, as it will be discussed in Section 8.

We have reviewed industrial documents describing RBAC audits, and we have found that these limit themselves to questions of appropriate role assignment, role abuse, role conflicts, etc.

In any case, our methods apply well beyond RBAC, to all systems for which reading or writing authorizations among subjects and objects can be established, see Section 3 and the Conclusions.

3. Basic concepts

We consider two types of *entities*: *subjects* and *objects*. The lower case letter s, with various primes or subscripts, will be used for variables for subjects, while the upper case letter S with various numerals will be used for constants over subjects. Similarly, the lower-case letter o will be used for variables over objects, while the upper case letter O will be used for constants over objects. The letter e will be used for variables of entities, when we will introduce concepts that hold whether an entity can be a subject or an object.

We start by reviewing and adapting to our needs some basic definitions first proposed by Lampson [15], and repeated many times in the literature, in many variations. In a system with i subject and j objects, an *access control matrix* is an $i \times j$ matrix M where the element $M_{m,n}$ is a set of capabilities, telling what are the capabilities of subject s_m over object o_n. A row m of the matrix is the list (or set) of capabilities of s_m. We consider only the capabilities of *reading* and *writing*. We choose to present capability lists as sets of relations. Following [16] we use the constant predicates *CR* and *CW* (for *CanRead* and *CanWrite* respectively):

- $CR(s, o)$ is true if subject s has read capability on o.
- $CW(s, o)$ is true if subjects s has write capability on o.

So capability lists or access control matrices can be represented as sets of *CR* and *CW* relationships. An example is given in Table 2. By using the predicates just presented, Table 2 can be described thus: $\{CW(S1, O3), CR(S2, O1), CR(S2, O2), CR(S2, O3), \ldots\}$. Intuitively, the existence of a *CR* (or *CW*) relationship between a subject and an object means that the subject can receive data from the object (or the subject can send data to the object). More abstractly, $CR(s, o)$ can be taken to mean that any data that o can store, s can know, and similarly for *CW*. In practice, this can correspond to several situations that we may want to consider: the traditional read and write permissions as in Unix; possession of encryption–decryption keys; the existence of channels, possibly hypothetical hidden ones; network routing relationships; data transfer paths in the Internet of Things. If we take roles for subjects, and we restrict RBAC permissions to reading and writing operations on resources, RBAC permissions can be directly translated into capability lists or access control matrices and vice versa [7, Section 3.3.2] [3].

Table 1
Boolean capability list for $S3$ in the example of Table 2 and Fig. 1.

	O1	O2	O3	O4
read	1	0	1	0
write	0	1	1	0

Our simulation programs will represent access control lists in terms of Boolean arrays, as in Table 1 (where $S3$ can read from $O1$ but cannot write in it, etc.).

We also use the two additional predicates *CanKnow and CanStore* [16], respectively abbreviated *CK* and *CS* which are related between themselves and with *CR*, *CW* by the following axiom and inference rules:

- *The axiom:* $CS(o, o)$ (in other words, any object can store itself).
- *The inference rule for CK is:* $(CR(s, o) \wedge CS(o, o')) \rightarrow CK(s, o')$ (i.e. if s can read o, then whatever o can store, s can know).
- *The inference rule for CS is:* $(CW(s, o) \wedge CK(s, o')) \rightarrow CS(o, o')$ (i.e. if s can write o, then whatever s can know, o can store).

We stipulate further that $CK(s, o)$ or $CS(o, o')$ can be true only by the inference rules or axiom.

We will also use the following auxiliary definitions:

- $CKS(s)$ for a subject s, is the set of all objects o that can be known by s, i.e. $CKS(s) = \{o : CK(s, o)\}$.
- $CSS(o)$ for an object o, is the set of all objects o' that can be stored by o, i.e. $CSS(o) = \{o' : CS(o, o')\}$.

For a subject s (or object o), $CKS(s)$ (or $CSS(o)$) is the set of all data objects that can be delivered to s (or stored in o) by sequences of read–write operations. Table 3 will show an example.

Clearly, for each s and o we can calculate these two sets from the axiom, the given *CR* and *CW* relationships, and the inference rules, and this paper introduces efficient algorithms to do this.

Our inference rules assume transitivity of the data transfer operation, i.e., that any or all data can always be passed on if a *CR* or *CW* capability exists. This is not always true in security, surely we can have entities that will pass on data according to some conditions or judgment. In our model, such situations can perhaps be represented by splitting subjects or objects according to their behavior in this respect. Transitivity is a pessimistic assumption that may lead to over-protected systems.

In [16] we used the additional concept of variable, using which the definitions are symmetric between subjects

and objects. We have opted for simpler, but asymmetric, definitions in this paper by which objects can contain objects, and subjects can know objects.

For a given network, the set of subjects and objects in the network with their *CR* and *CW* relationships can be shown graphically, yielding a *network directed graph* (*digraph*), by using the following conventions (see Fig. 1):

- *The nodes in the digraph are the entities in the network; we use ovals for subjects and rectangles for objects.*
- *There exists a directed edge from object o to subject s iff* $CR(s, o)$,
- *There exists a directed edge from subject s to object o iff* $CW(s, o)$.

In choosing the notation, we have tried to strike a compromise between the usual verbal and graphical notations, which are not mutually consistent. The verbal notation puts the subject first, while the graphical notation shows the data flow. So, we represent $CR(s, o)$ with an arrow from o to s, while we represent $CW(s, o)$ as an arrow from s to o. Distinguishing between these two relationships is useful in access control theory, where a distinction is made between subjects and objects. But if we eliminate this distinction, and use the concept of *entity* to denote both subjects or objects, then we can define a relation *CanFlow* or $CF(e, e')$, corresponding to a directed path from e to e'.

$CF(e, e')$ is true iff one of the following is true:

- $e = e'$
- e' *is a subject and e is an object and* $CR(e', e)$
- e *is a subject and* e' *is an object and* $CW(e, e')$
- *there exists* e'' *such that* $CF(e, e'')$ *and* $CF(e'', e')$.

By these definitions, *CF* is a transitive, reflexive relationship, or a *preorder*. We take the network digraph described above to represent this preorder, except for reflexive relationships that are implicit.

We also generalize the functions *CK* and *CS* by introducing the function *CanHold* or *CH*, as follows:

If e is a subject s, then $CH(e) = CK(s)$. If e is an object o, then $CH(o) = CS(o)$.

Finally, the functions *CKS* and *CSS* can be generalized into a function *CanHoldSet* or *CHS*: if e is a subject s, then $CHS(e) = CKS(s)$; if e is an object o, then $CHS(e) = CSS(o)$.

A *partial order* is a relationship that is transitive, reflexive and anti-symmetric. If we partition a preorder into equivalence classes, the result is a partial order of equivalence classes: this is because the resulting relationship is transitive and reflexive, while no symmetric relationships could remain among the equivalence classes [8, Section 2.2]. This result has its correspondent in digraph theory, as follows. A *component* in a transitive, reflexive digraph is a set of nodes that are mutually reachable, and which is maximal in the sense that it is not included in a larger set of nodes with the same property. If each component is *condensed* into a single node, then the resulting digraph represents a partial order (all the symmetric paths having been encapsulated in components). Further, this resulting digraph has the same connectivity as the original one, in the sense that if node x is reachable from node y in the original digraph, then the node that represents the component of x is reachable from the node that represent the component of y [12, Chapter 3]. For more discussion and examples on this, see [17].

Table 2
Capability lists for the network of Fig. 1.

S1	CR	none
	CW	O3
S2	CR	O1, O2, O3
	CW	O2
S3	CR	O1, O3
	CW	O2, O3
S4	CR	O2, O4
	CW	O2, O4
S5	CR	O4
	CW	O4

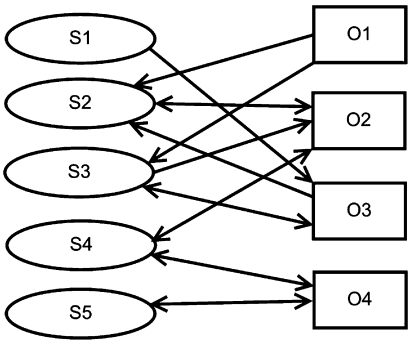


Fig. 1. Network example.

We say that two entities are:

- *CF-equivalent* if they belong to the same component in their network or digraph;
- *CHS-equivalent* if they have the same CHS.

Clearly, CF-equivalence implies CHS-equivalence, but the opposite is false in very specific cases, see Section 9. This property enables us to use CF-equivalence in order to find CHS-equivalence.

4. An example

The example in this section will show the important concepts and results of our method.

Consider a network with five subjects $S1$ to $S5$ and four objects $O1$ to $O4$. The capability lists, expressed as mentioned as *CR*, *CW* relationships in tabular form, are shown in Table 2, where the capabilities for $S1$ are that it can only write on $O3$, and so on. Table 1 shows the same capabilities for $S3$ in the form of a Boolean matrix.

Fig. 1 gives a graphical representation of this network, by the conventions defined in Section 3.

A first question we can ask for such a network is: for an object o, what are the subjects and objects that can know or store it? We call this the *single object flow* problem. This question can be important for objects that can contain secret data. Answering this question leads to partitioning the network in two *areas*, one where the object can be known

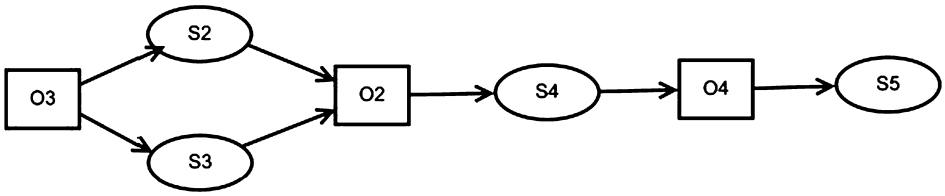


Fig. 2. The flow of object $O3$ in the network, the area of $O3$.

Table 3
CanKnow and *CanStore* sets for the network.

CKS(S1) = { }	CSS(O1) = {O1}
CKS(S2) = {O1, O2, O3, O4}	CSS(O2) = {O1, O2, O3, O4}
CKS(S3) = {O1, O3}	CSS(O3) = {O1, O3}
CKS(S4) = {O1, O2, O3, O4}	CSS(O4) = {O1, O2, O3, O4}
CKS(S5) = {O1, O2, O3, O4}	

or stored (the sets of subjects and objects of which o is a *secret*) and another where it cannot. Fig. 2, to be read left-to-right, shows the *area of* $O3$ in the network of Fig. 1.

Intuitively this is clear, but formally it can be justified in terms of the inference rules in Section 3, since by Table 2 we know:

$$CR(S2, O3) \wedge CR(S3, O3) \wedge CW(S2, O2) \wedge CW(S3, O2)$$
$$\wedge CR(S4, O2) \wedge CW(S4, O4) \wedge CR(S5, O4)$$

Using the axiom $CS(O3, O3)$, the inference rules allow us to progressively derive $CK(S2, O3)$, $CK(S3, O3)$, $CS(O2, O3)$ (twice), $CK(S4, O3), CS(O4, O3), CK(S5, O3)$. By inspection, one can see that no other such derivations are possible for any other subjects or objects in the network.

This information is useful for a system administrator wishing to audit, add or remove certain flows. As a further step, suppose that in a network there is a set of objects whose association can lead to the discovery, or inference, of some critical information. A practical example of this case could be the rule: only executives can know both tables: *Role-by-employee* and *Salary-by-role*. It is then important to determine what exactly are the entities that can know or store all such data together. If these tables are in two different objects, then to answer this question it is sufficient to repeat the above reasoning twice, once for each object.

A global answer to all questions of this type can also be given, we call this the *global object flow* problem. To do this, we calculate the *CKS* and *CSS* functions for all subjects and objects, see Table 3. E.g. for $S3$, we have seen above that $O3 \in CKS(S3)$. By using the inference rules for $O1$, we will also find that $O1 \in CKS(S3)$, and no other possibilities exist.

Looking at Table 3, we can see that:

- Several subjects (objects) can know (store) the same objects, and so they are CHS-equivalent, see for example $S3, O3$.
- There is a partial ordering of these equivalence classes, determined by the inclusion of the *CHS*, e.g. in this sense the equivalence class containing $S3, O3$ is included in the equivalence class containing $S2, S4, S5, O2, O4$.

The CHS equivalences can be computed by using directly the axiom and inference rules of Section 3, as we have done above. However this would require the use of languages appropriate to program inference rules directly; such languages are interpretive and not the best in terms of computational efficiency. But since CF equivalence implies CHS equivalence, the former can be used in order to find the latter by using efficient digraph algorithms. In Section 9 we will see that there are only very limited cases where CHS equivalence does not imply CF equivalence. By looking for CF equivalences, we obtain the simplified digraph of Fig. 3, where the subjects (objects) that can know (store) an object have been grouped in the *area of* the object.

Note that Fig. 3 shows all and only the data flows of Fig. 1.

- The boxes in Fig. 3 contain entities that constitute components in the original network. They are not only CF-equivalent, but also CHS-equivalent. In other words, it can be checked that $S3, O3$ and $S2, S4, S5, O2, O4$ constitute components in the graph of Fig. 1 and so they must all have the same *CHS*.
- There are arrows between boxes to denote the partial order between the components. The arrows denote also *CF* relationships between the entities in the components.

This partial order allows us to arrange the network by increasing *levels of secrecy*, according to the concepts presented in [17]. On one hand, the contents of $O1$ are the least secret, since they can propagate to all other subjects and objects (except for $S1$). On the other hand, it is clear that the most secret data should be stored in $O2$ or $O4$: there, they can be shared with $S2, S4, S5$ but they cannot propagate further.

We can now construct several networks that are equivalent to the initial one (from the dataflow point of view) by rearranging subjects and objects according to the following rules:

- Within each component there can be any *CR, CW* relationships that generate symmetric *CF* relationships among all pairs of entities.
- Between components, there can be any edges that are consistent with the partial order: e.g. in Fig. 3 we have $CW(S3, O2)$ but other possibilities would be just as good, such as some or all of: $CW(S3, O4)$, $CR(S2, O3)$, $CR(S4, O3)$, $CR(S5, O3)$, see Fig. 4.

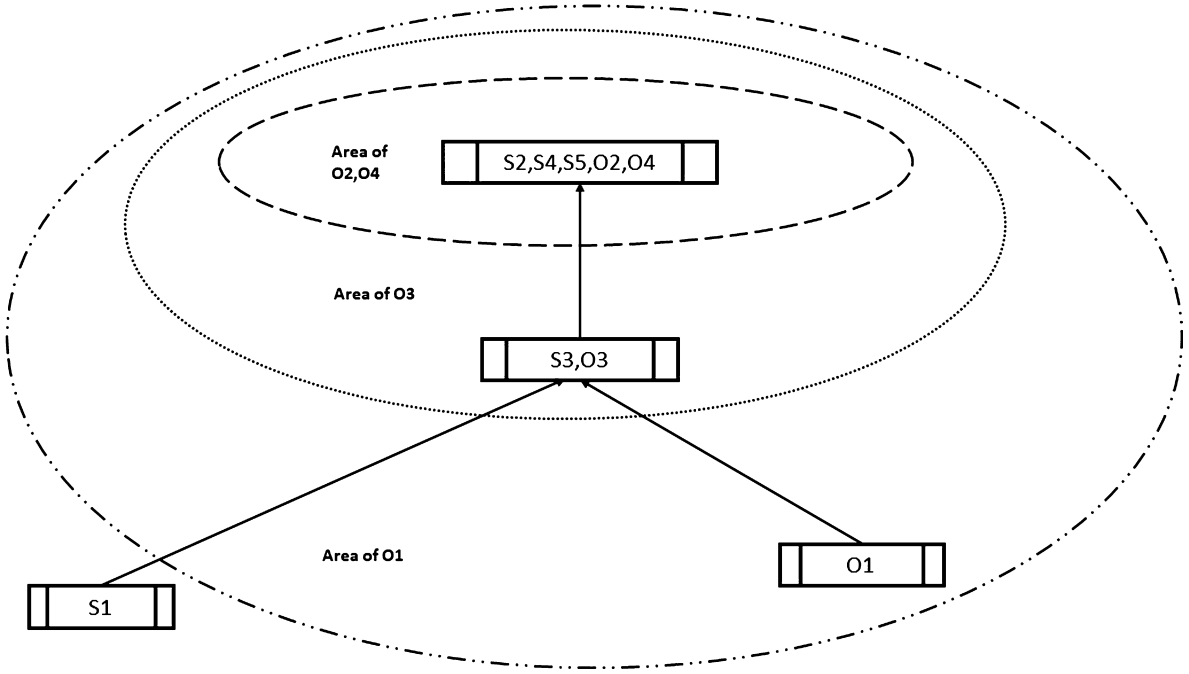


Fig. 3. Network showing the partial order of equivalent subjects and objects.

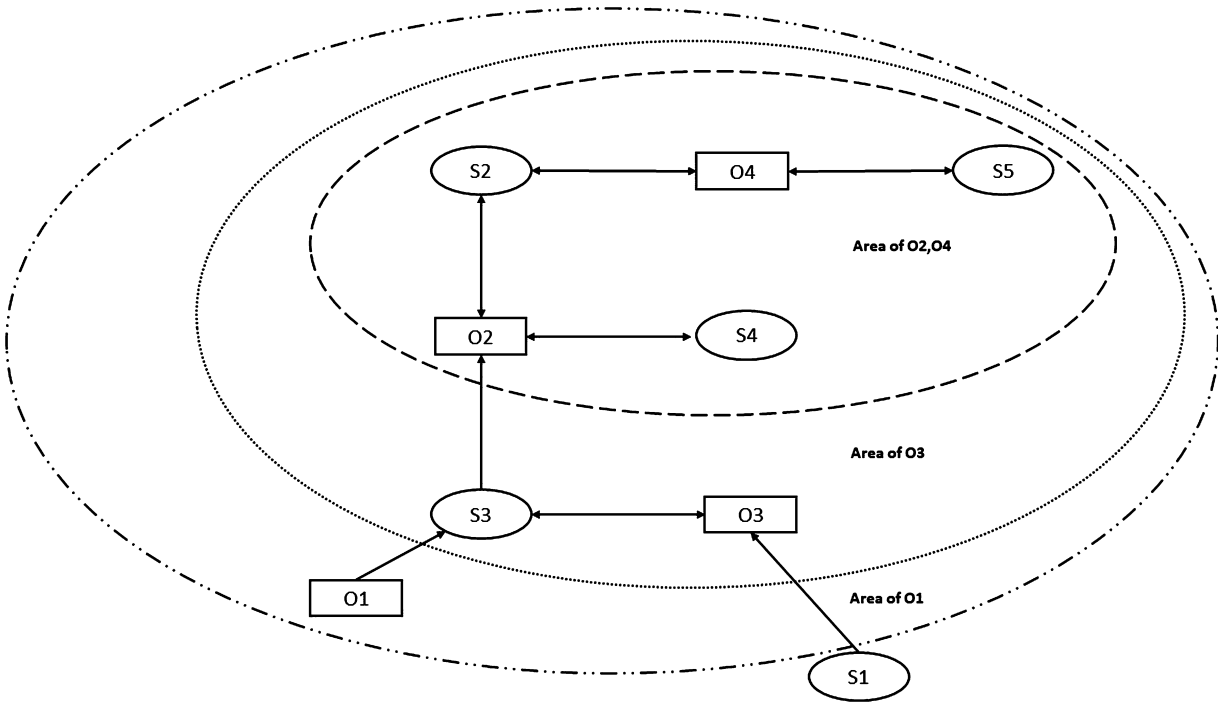


Fig. 4. A network equivalent to the initial one.

The decision of performing such restructurations or simplifications normally should be left to the network administrator, because good reasons might exist for not restructuring at all, or for choosing one restructuration rather than another. For example, a network administrator may get a more efficient network by adding all the capabilities that are implied by transitivity. In general, if $CW(s, o)$ is false but by transitivity there is a flow between s and o, should the administrator make $CW(s, o)$ true? This is a non-obvious decision, since in practice indirect writing may depend on read and write actions (and decisions) of intermediary subjects. On the other hand, one may wish to reduce the capabilities as much as possible in order to limit the security checks.

Our algorithms can draw these diagrams but they become impractical beyond small sizes. But the algorithms can provide all the information that an administrator needs in order to do simplifications and reorganizations.

Beside the *CKS*s and *CSS*s and order relationships they can provide the tables of empty entities and equivalent entities.

5. Algorithms and complexity

We now explore the practical computability of finding CF equivalences with their partial orders. To do this, we can take advantage of the efficient implementations of graph algorithms available in several tools. We use combinations of the following algorithms:

1. Constructing the internal representation of the digraph for the relationships *CR* and *CW*, such as the one of Fig. 1. This is an immediate translation of the capability lists. The time complexity is $\boldsymbol{O}(|S| \times |O|)$, i.e. the product of the number of subjects by the number of objects.
2. Calculating the single object flow, see Fig. 2. This can be done by using depth-first search. The complexity of this algorithm is determined by the sum of the number of nodes plus the number of edges. For our application, this translates in time complexity $\boldsymbol{O}(|S| + |O| + (|S| \times |O|))$.
3. Finding the strongly connected components of the digraph obtained in Step 1, i.e. the CF-equivalent sets, together with their partial order, according to the principles of Sect. 3. Several efficient algorithms are known to do this. The one that we chose is Tarjan's algorithm [22]. The complexity of this algorithm is linear on the number of edges plus the number of nodes, which in our case translates into $\boldsymbol{O}(|S| \times |O| + |S| + |O|)$.
4. To answer the global flow problem Step 2 is repeated for each object, so the complexity of this step is $\boldsymbol{O}(|O| \times (|S| \times |O|))$, or $\boldsymbol{O}(|O|^2 \times |S|)$.
5. Transitive reduction. This is not indispensable but it leads to digraphs that are easier to read. The complexity of this algorithm in the case of acyclic digraphs is, once again, $\boldsymbol{O}(|S| + |O| \times |S| \times |O|)$.

In complexity evaluation, it is customary to consider only the dominant component, and so for each of the algorithms 1, 2, 3 and 5 the time complexity can be taken to be $\boldsymbol{O}(|S| \times |O|)$, which is polynomial, in fact quadratic if $|S| = |O|$. For the single object flow problem we use algorithms 1 and 2, leading to a total complexity of $\boldsymbol{O}(2 \times (|S| \times |O|))$. For the global object flow problem, we use algorithms 1, 3, 4, 5, leading to a total complexity of $\boldsymbol{O}((3 \times (|S| \times |O|)) + (|O|^2 \times |S|))$. Applied to the example of Section 4, this sequence of algorithms leads to the digraph of Fig. 3. These complexity orders could be refined, for example we have not considered the fact that Step 3 can reduce significantly the number of subjects and objects, compare Fig. 1 with Fig. 3.

For the purpose of this paper, the important conclusion is that only polynomial-time algorithms are needed for our method: in complexity theory, such algorithms are considered to be *efficient*.

6. Simulation results

The complexity analysis that we have done so far describes upper bounds on the growth of functions depending on the number of entities but tells little about real execution times. Further, upper bounds are very seldom reached and good implementations apply many optimizations. It is then useful to perform simulations to have realistic estimates. We ran a number of simulations in MATLAB [9] to check real execution times. We used an Intel Xeon Processor E5-2603 v4 with 6 cores, 1.7 GHz and 64 GB of RAM, in a DELL Precision Tower 7910 customized with hardware and software for heterogeneous processing. This is a fairly powerful system for research applications.

For each size considered, a network was generated with a random Boolean capability list for each subject. The ratio of numbers of protected objects per subject can vary greatly among types of systems, but normally there are many more objects than subjects. Supposing application to RBAC, and following the advice of an expert [2] we considered networks where there are 24 objects for every subject, which means that, for an organization of size 100, we have 4 subjects and 96 objects. However, the simulation times do not change much if we change these ratios, e.g. they are similar if we assume that there are equal numbers of subjects and objects.

The execution times are shown in Fig. 5. Fig. 5.a) shows the times for Steps 1 and 2 in Section 5, Fig. 5.b) shows the times for Steps 1 and 3, and Fig. 5.c) shows the times for Steps 1, 3, 4, 5, leading to representations such as the one of Fig. 3. We see that the curves confirm our complexity analysis of polynomial times. The number of nodes is the number of entities.

For Fig. 5.c), our program hit a memory limit at approximately 120,000 nodes after about 196 minutes and for comparison we terminated the simulation at the same number for the other two simulations. Since the algorithms are polynomial, calculation times will improve with more powerful computers.

Several types of context-dependent optimizations can be envisaged. Different objects can be treated as one if they have the same access control lists, which can be the case in many organizations where we have object categories such as financial, personnel, etc., with the same access control list for all objects in each category, see the concept of *view* in OrBAC [14]. Or also, the techniques we propose could be performed level-by-level within hierarchies of object categories. If this is done, the set of objects could be the (probably much smaller) set of object categories instead of the set of all objects in the organization. Likewise, the number of subjects can be significantly reduced by considering roles instead of subjects. Yet other optimizations may be possible if, as it often happens in practice, a large network can be partitioned in several partially independent networks. Fig. 5 shows how much better execution times are if the number of entities is reduced by a factor of 10. This is a subject of further study, namely for applications to social networks.

Nodes	Time (s)
100	0.049
500	0.085
1000	0.244
2000	0.782
5000	4.905
8000	11.871
10 000	18.150
20 000	77.430
50 000	132.357
100 000	460.459
120 000	687.326

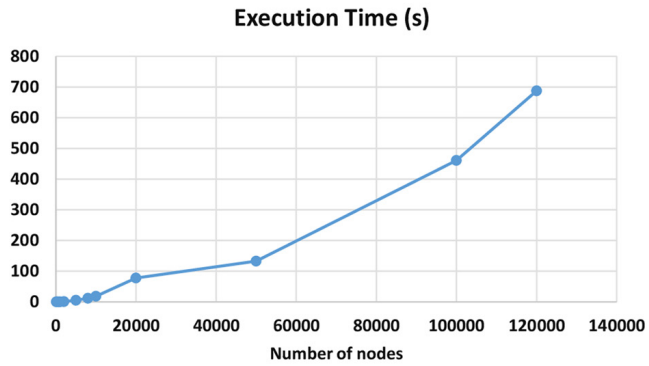


Fig.6 .a) Execution time for the single-object data flow algorithm

Nodes	Time (s)
100	0.02
500	0.167
1000	0.296
2000	0.839
5000	4.637
8000	16.469
10 000	17.911
20 000	52.944
50 000	369.145
100 000	718.027
120 000	1230.480

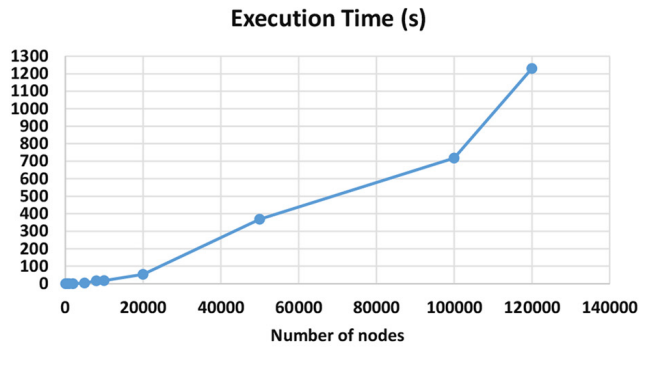


Fig.6 .b) Execution time for the equivalence classes of subjects and objects construction algorithm

Nodes	Time (s)
100	0.230
500	0.620
1000	1.267
2000	2.988
5000	15.438
8000	51.182
10 000	97.616
20 000	599.937
50 000	2836.405
100 000	6328.552
120 000	11797.636

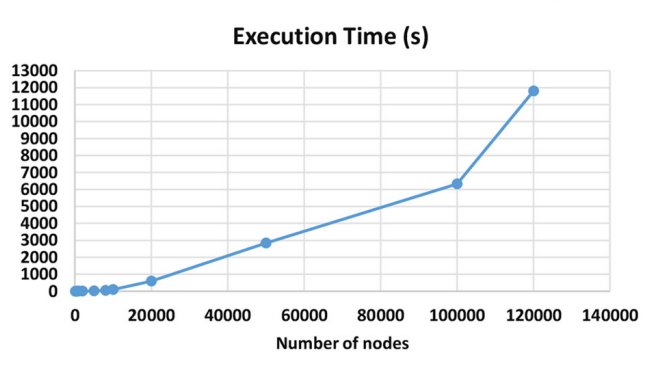


Fig.6 .c) Execution time for the global object flow algorithm

Fig. 5. Simulation times with MATLAB.

7. Relation between CF and CHS-equivalence

It would be interesting to be able to conclude that the two equivalences we have worked with are identical, this would mean that by looking for CF-equivalences one can find *all* CHS-equivalences and vice versa. Unfortunately this is not true, but only because of very special cases. To see this, consider a very simple network with one object O and two subjects $S1$ and $S2$, and:

$CR(S1, O), CR(S2, O)$

We have: $CHS(O) = CHS(S1) = CHS(S2) = \{O\}$: the three entities are not CF-equivalent and so our method cannot find that they are CHS-equivalent. More generally, a useful result would be: $CF(e, e')$ *iff* $CHS(e) \subseteq CHS(e')$ but the example shows that this is false in the reverse direction.

Note that this difficulty is limited to one level only, since if $CW(S1, O1)$, then $CHS(O1) = \{O, O1\}$ which breaks the *CHS* equivalence.

This difficulty can disappear if the difference between subjects and objects is eliminated, if some conditions are imposed, or if the model is changed. We excluded the first possibility because we are addressing access control systems. A full discussion on this topic would lead to complexities that are extraneous to this paper, whose focus is on algorithms.

For the purpose of this paper, the conclusion to be retained from this section is that our method may fail to

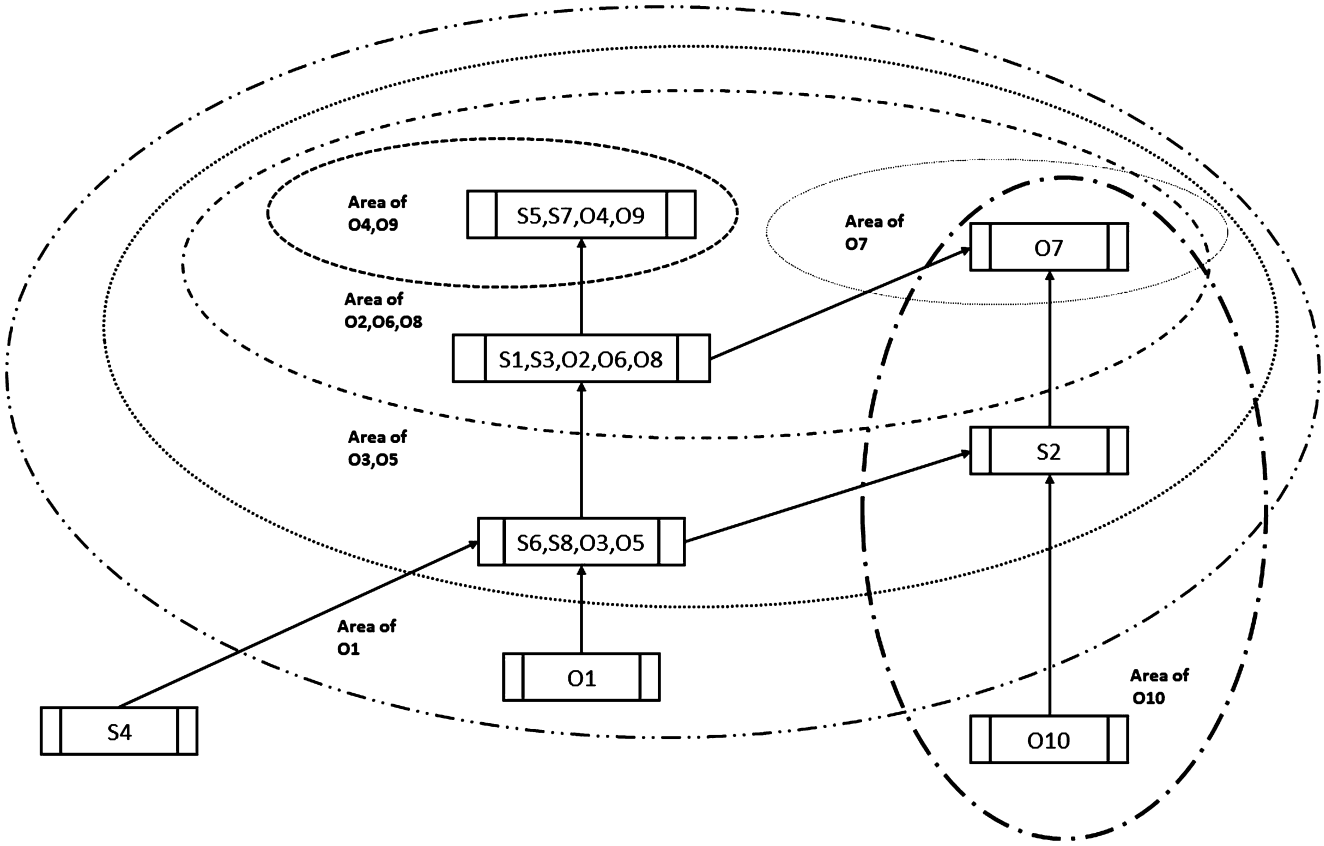


Fig. 6. Equivalence classes, partial order and areas for the example of Table 4.

Table 4
Capability lists for a larger example.

S1	CR	O2, O8	S5	CR	O4
	CW	O2, O4, O6		CW	O9
S2	CR	O5, O10	S6	CR	O1, O3
	CW	O7		CW	O5
S3	CR	O5, O6, O8	S7	CR	O9
	CW	O7, O8		CW	O4, O9
S4	CR	None	S8	CR	O5
	CW	O3		CW	O3

detect very few, if any, CHS equivalence sets and this may only lead to partial orders still valid, but slightly more complex than they need to be.

8. A larger example

The example of Section 4 was very small, to be easily understood. In this section, we give a larger example, but still small enough that it can be checked manually. The capability lists are given in Table 4 and the partial order, the equivalence classes and the areas in Fig. 6. As mentioned in Section 4, Fig. 6 defines a class of possible implementations of the flows defined in Table 4. As earlier, we have oriented the graph to show increasing secrecy levels.

9. Extension to RBAC and role engineering

A full discussion of RBAC is beyond the aims of this paper. We give here a summary of how some of the methods we have presented could be applied to some aspects of RBAC.

RBAC has capabilities that go well beyond access control to data, but has no specific mechanisms for data flow control. The unwanted data transfers can be controlled in RBAC indirectly by carefully designing roles, permissions and by static or dynamic constraints. However to do so, it is very useful to be able to answer our initial question, which is, where can data end up in a given RBAC configuration, in order to decide what changes need to be applied.

We limit ourselves to *core RBAC* without role hierarchies or constraints, and we consider only a single session, hence users and subjects can be identified. Only read and write permissions on data objects will be considered. So our previous *subjects* can be seen as *roles*. Capability lists for roles can be derived with these assumptions, thus the analyses that we have discussed so far can be applied to RBAC networks. As an exercise, the reader may wish to apply our method to the RBAC example of [21]. If we assume 10 users per role, then there are considerable efficiency improvements as we can see from Fig. 5. If, in addition, we assume that policies group objects with identical permissions, then there may be other significant improvements.

This leads to two applications of our method in RBAC, which can both be derived from observations presented in Section 4. The first is the following. If secrecy constraints exist, such as one by which the contents of certain objects can only be known to certain roles, then the proposed algorithms can be used to determine whether these constraints are violated. They can also be used to determine

in which objects certain data should be put in order to be secrets of certain roles.

A second application is to role engineering. Role engineering for RBAC is the process of defining roles, user-role assignments, and permission-role assignments for a given organization [7]. The following facts can be important in role engineering:

- That certain roles can know no data, and so perhaps they can be eliminated: such would be a role corresponding to $S1$ in Section 4 or $S4$ in Fig. 6.
- That certain roles can know the same data as others, and so perhaps they can be merged, such is the case for roles corresponding to $S2, S4, S5$ in the example of Fig. 3.
- That certain objects can store the same data, and so the objects and the permissions related to them can perhaps be merged, such is the case for objects $O2, O6, O8$ in the example of Fig. 6.

The proposed algorithms provide such information. But the results of our method can only suggest changes to the RBAC model: the decision of implementing these changes must rest with the role engineer or the administrator, since reasons can exist not to do so.

As far as we know, we are the first to propose such methods for role engineering.

Several issues remain for further study. If sessions are introduced, then we must make assumptions on whether users can retain data from one session to another. Further, the feasibility of our technique in the presence of constraints should be studied. Papers [1][10] present other ideas that are well worth exploring from the point of view of available algorithms and simulation.

10. From capability lists to Label-based access control

In its simplest form, a Label-based access control (LaBAC) model [4] can be defined by associating labels to subjects and objects, and using access control rules based on labels. We show how the concepts developed so far lead to LaBAC models, which define data flow controls that are identical to the ones defined by the capability lists from which they were generated. We write $s{:}lab(s)$ or $o{:}lab(o)$ to mean that subject s (object o) has label $lab(s)$ ($lab(o)$). Given networks such as the ones of Fig. 3 or Fig. 6, the following method is sufficient:

1) Assign to each subject and object a label designating the set of the areas in which it finds itself.
2) The access control rules are:
   - $CR(s{:}lab(s), o{:}lab(o))$ *iff* $lab(o) \subseteq lab(s)$
   - $CW(s{:}lab(s), o{:}lab(o))$ *iff* $lab(s) \subseteq lab(o)$

For example, in Fig. 6, we have: $O10 : \{O10\}$ and $S2 : \{O1, O3, O5, O10\}$, hence $CR(S2, O10)$. Clearly, the LaBAC model for this example defines a flow control that is identical to the flow control defined by the capability list of Table 4. The construction is generic and so for any capability list, an equivalent LaBAC model can be generated efficiently.

One could create a correspondence between the labels above and more conventional security labels, e.g. $\{O1, O2, O3, O4, O5, O6, O8, O9\}$ could be called *"TopSecret1"* and so on as desired.

11. Conclusions and future work

We have identified some data flow analysis problems in access control networks, and we have shown that they are practically solvable with well-known and efficient graph algorithms, using as input the capability lists of the subjects, which will be permissions lists in the case of RBAC. This finding has been validated by both algorithmic analysis and simulation. Simulation has shown that our method is usable for networks of several tens of thousands of subjects and objects; in fact, if we consider ongoing progress in the area of graph processing [18], possibly for any practically conceivable organization sizes. The method can be used to answer several important questions such as: if some secret data are stored in some database, who can be able to read them, directly or indirectly? For each subject or object, exactly what data can it have available? If certain data should be secret of certain subjects or objects, where should the data be stored? What subjects and objects are equivalent for the data they can be able to hold? What reorganizations are possible, without changing the data flows? Our concept of *area* of an object in a network seems to be new and useful, and so is the possibility of identifying secrecy levels in arbitrary (possibly RBAC) networks.

This paper can also be seen as an experimental confirmation of the principles presented in [17]. We have shown that the partial order of components in a data flow network can be seen as a hierarchy of secrecy levels. This generalizes the well-known concept of lattice flow model presented in [6]. It generalizes it for two reasons: partial orders are a less restrictive structure than lattices, and partial orders can be always found efficiently, by using our method, for any network of entities that is described in terms of reading and writing capabilities. Based on these concepts, one can imagine graphic interfaces that would make it possible to design systems with secrecy requirements by manipulating on the screen graphic representations such as the one proposed here. For scalability however, abstraction mechanisms such as encapsulation will have to be devised.

With respect to previous work, the contribution that is closest to ours is [1]. These authors describe powerful methods and tools that go much beyond what we have done, but don't provide the detailed algorithm analysis and simulations that we provide here, hence they provide no information on the sizes up to which their tools could be practical. In addition, although they introduce the concept of 'reduction by information equivalence classes', they do not introduce the generic secrecy level concepts that we have mentioned, based on the order-theoretical concepts of Section 3. Similar observations apply to [10] [11]. The authors of these three papers consider the general case where the permissions can change, but seem to miss the interesting conclusions that can be drawn when the permissions are fixed, as we do in this paper. The conceptual bases they use are also fairly different from ours. Reading

of these papers raises many interesting questions at the intersection of these partially complementary approaches. These papers also give an idea in what kinds of tools our algorithms can be used.

Other topics for future work present themselves, for example, in connection with RBAC: extension to RBAC with static constraints, and using our method for RBAC auditing. Extension to Boolean conditions, dynamic constraints and changes in capabilities (such as by administrative action) are further beyond. The applicability of these methods in Internet of Things and Cloud networks is also on our research agenda.

Acknowledgements

This research was funded in part by the Natural Sciences and Engineering Research Council of Canada. We are grateful to Jurek Czyzowicz for discussion on graph algorithms, to Ahmed Lakhssassi for having allowed us to use his powerful computer, to Sofiene Boulares for discussion on the theory and practice of access control methods, and to Mustapha Benmahbous of Xpertics for having shared with us some views based on his long experience in role engineering. The anonymous referees have contributed to the clarity and completeness of this paper.

References

[1] P. Amthor, W.F. Künhauser, A. Pölk, WorSE: a workbench for model-based security engineering, Comput. Secur. 42 (2014) 40–55.
[2] M. Benmahbous, Personal communication, 2017.
[3] J. Barkley, Comparing simple role based access control models and access control lists, in: Proc. of the 2nd ACM Workshop on RBAC, 1997, pp. 127–132.
[4] P. Biswas, R. Sandhu, R. Krishnan, Label-based access control: an ABAC model with enumerated authorization policy, in: Proc. 2016 ACM Intern. Workshop on Attribute Based Access Control (ABAC 2016), pp. 1–12.
[5] R. Chon, T. Enokido, V. Wietrzsk, M. Takizawa, Role locks to prevent illegal information flow among objects, in: Proc. of IEEE 18th Intern. Conf. on Advanced Information Networking and Applications (AINA-2004), vol. 1, pp. 196–201.
[6] D.E. Denning, A lattice model of secure information flow, Commun. ACM 19 (5) (1976) 236–243.
[7] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, Role-Based Access Control, 2nd ed., Artech House, 2007.
[8] R. Fraïssé, Theory of Relations, North-Holland, 1986.
[9] A. Gilat, MATLAB: An Introduction with Applications, 2nd ed., John Wiley & Sons, 2004.
[10] M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller, Incremental information flow analysis of role based access control, in: International Conference on Security and Management, 2009, pp. 397–403.
[11] M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller, RBAC-PAT: a policy analysis tool for role based access control, in: Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), in: LNCS, vol. 5505, pp. 46–49.
[12] F. Harary, R.Z. Norman, D. Cartwright, Structural Models: An Introduction to the Theory of Directed Graphs, Wiley, 1965.
[13] K. Izaki, K. Tanaka, M. Takizawa, Information flow control in role-based model for distributed objects, in: Proc. 8th Intern. Conf. on Parallel and Distributed Systems (ICPADS 2001), pp. 363–370.
[14] A.A.E. Kalam, R.E.I. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, G. Trouessin, Organization based access control, in: Policies for Distributed Systems and Networks, Proceedings. POLICY 2003. IEEE 4th International Workshop on, pp. 120–131.
[15] B.W. Lampson, Protection, in: Proc. 5th Princeton Conf. on Information Sciences and Systems, 1971, pp. 437–443.
[16] L. Logrippo, Logical method for reasoning about access control and data flow control models, in: Proc. of the 7th Intern. Symp. on Foundations and Practice of Security (FPS 2014), in: LNCS, vol. 8930, 2015, pp. 205–220.
[17] L. Logrippo, Multi-level access control, directed graphs and partial orders in flow control for data secrecy and privacy, in: Proc. of the 10th Intern. Symp. on Foundations and Practice of Security (FPS 2017), in: LNCS, vol. 10723, 2018, pp. 111–123.
[18] MIT News, Device allows a personal computer to process huge graphs, http://news.mit.edu/2018/device-allows-personal-computer-process-huge-graphs-0531. (Accessed 4 May 2018).
[19] S. Nakamura, D. Duolikun, A. Aikebaier, T. Enokido, M. Takizawa, Synchronization protocols to prevent illegal information flow in role-based access control systems, in: Proc. of International Conference on Complex Intelligent and Software Intensive Systems (CISIS-2014), pp. 279–286.
[20] P. Samarati, E. Bertino, A. Ciampichetti, S. Jajodia, Information flow control in object-oriented systems, IEEE Trans. Knowl. Data Eng. 9 (14) (1997) 524–538.
[21] S.L. Osborn, Information flow analysis of an RBAC system, in: Proc. of the Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002), pp. 163–168.
[22] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[23] C.N. Zhang, C. Yang, Information flow analysis on role-based access control model, Inf. Manag. Comput. Secur. 10 (5) (2002) 225–236.

Chapter 6

Configuring data flows in the Internet of things for security and privacy requirements

1. Introduction

In this chapter, we present our work, published in [77]. A synthesis of this paper is presented in the next section, while the paper in question is attached at the end of the synthesis.

2. Paper synthesis

The concept of 'Internet of Things' expresses the networked integration of sets of physical and virtual devices in potentially complex systems that carry and process data and control information from sensors or terminals to end users (which can be human users or other machines). It is a highly distributed system that promises integration of the real world with the information world, eventually creating important economic, social and workplace benefits. Among the many known applications are hospital systems, e-commerce systems, smart homes, transportation and energy systems, and many others.

Many sensitive data will travel in the IoT, and these data may flow among 'things' in very complex data flow configurations. We are interested in realizing data security and privacy in such systems. For that, it is important that only certain data flows be allowed and in all cases data should not flow to unauthorized parties; otherwise, sensitive data may be destroyed, altered, stolen, and even held for ransom. Examples of secrecy requirements in IoT systems can be seen in healthcare systems, for example, heart-rate sensor and a motion sensor may be separate things that communicate their data. Each sensor's data stream is stored for the person being monitored, and must only be accessible by the treating staff, and isolated from other 'things' and other people's data. Another example can be an e-commerce system, where ordering information of a client must not reach other clients. This gives rise to several questions: how can we set up data communications channels between 'things' so data originating in one device can or cannot reach another one? Given certain data flow relationship in an IoT system,

and data originating in certain devices, which are the entities that will be privy of these data? And finally, which are the most secret or least secret entities, in the sense that data that are in them cannot or can propagate to others?

Many solutions have been implemented for these problems in particular systems, those solutions often refer to the lattice model of Denning by proposing applications, variants and enhancements of it. We offer a generic solution that can be implemented in principle in any system. This solution is based on the fact that any directed graph, representing a data flow, can be seen as a partial order of equivalence classes. Each equivalence class is a set of nodes that can share the data, and the partial order establishes the direction of the data flow that must exist in any such system. For each type of data, these principles enable us to partition any IoT network in areas that can know the data, and other areas that cannot know them. Efficient algorithms exist to do this partitioning in general.

In our method, entities can be added according to needs, these entities must come with labels stating what data they can contain. This gives rise to a *CanHold (CH)* relathionships. Then communication channels or flow channels are added according to inclusion relationships between the *CH* relationship, so we can derive *CanFlow(X,Y) iff CanHold(X)* $\subseteq$ *CanHold(Y).*

This can be dynamically done every time a new entity is defined and added to the network. After adding communications channels, an efficient partial order detection algorithm is run to clean the graph and leave only the necessary channels.

We present two examples in our paper, a hospital example of network creation, and an e-commerce example of separate data flows. These examples, especially the second one, show that several data flows can coexist in a given network, and that our method can handle them, by tagging data according to the data flows to which they belong.

The final sections of the paper present an overview of a policy language that can be created and developed for IoT secrecy requirements. The language must provide primitives to define entities with their labels alongside with operators to add and remove entities and attributes. Finally, implementation issues are presented, which will be further discussed in our last chapter.

3. The attached paper

Configuring Data Flows in the Internet of Things for Security and Privacy Requirements

Luigi Logrippo(✉) and Abdelouadoud Stambouli

Department of Computer Science and Engineering Gatineau,
Université du Québec en Outaouais, Gatineau, Québec, Canada
{luigi, staal6}@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?
- *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

N. Zincir-Heywood et al. (Eds.): FPS 2018, LNCS 11358, pp. 115–130, 2019.
https://doi.org/10.1007/978-3-030-18419-3_8

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 [9, 11, 12].

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 [6], 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, 12, 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 Sect. 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 HomeComputer (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)* $\subseteq$ *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 [10]. 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 [10].

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 Property 1, this implies *CF(**S, S′**)*, where $\boldsymbol{S}$ is the set of all such patients and $\boldsymbol{S'}$ 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 Fig. 1(a) for an example. According to the properties presented in the previous section, a streamlined digraph can be obtained in the following way:

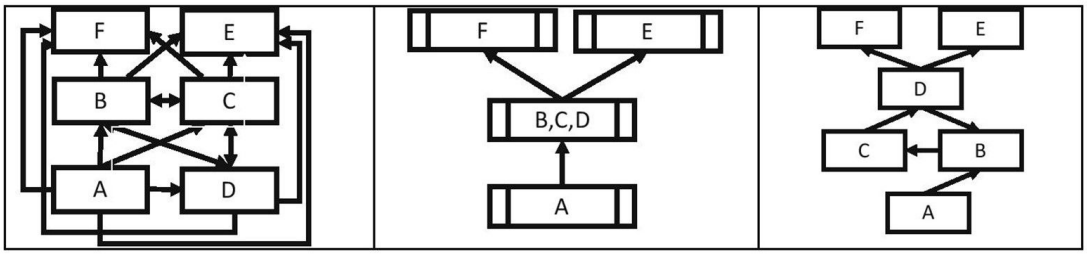


Fig. 1. (a): A data flow graph; (b) its partial order of components; (c) an equivlent data flow graph

(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 Fig. 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 $\boldsymbol{S}$ and $\boldsymbol{S'}$ such that $S \subseteq S'$, connect one or more element (s) in $\boldsymbol{S}$ to one or more element(s) in $\boldsymbol{S'}$ (Fig. 1(c). By transitivity, $CF(\boldsymbol{S}, \boldsymbol{S'})$.

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 Fig. 1(a) are indirect by transitivity in Fig. 1(c). 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 Fig. 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 Fig. 2, however we will show how it can be built step by step.

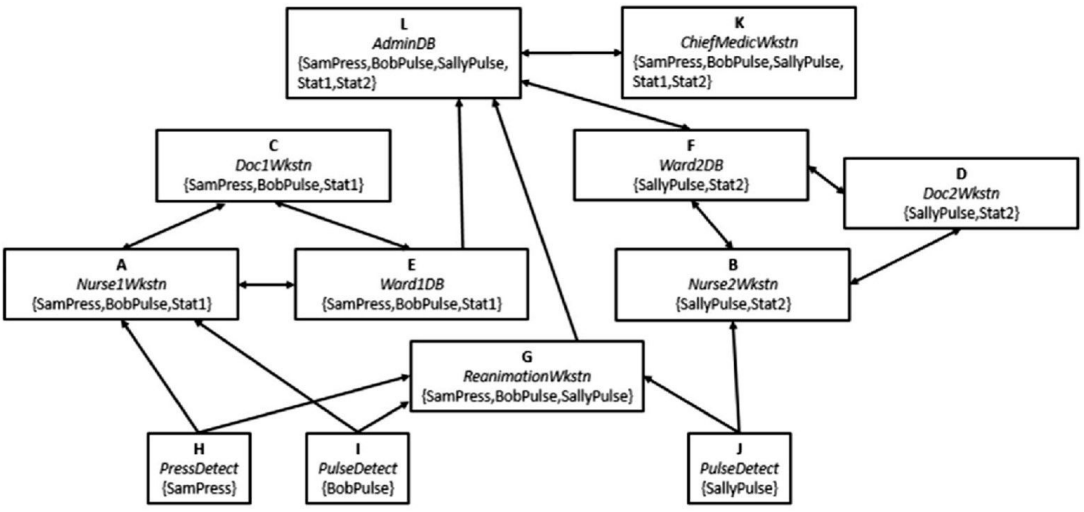


Fig. 2. A hospital example

The types of the entities are: *PressDetect, PulseDetect, NurseWkstn, DocWkstn, ReanimationWkstn, WardDB, ChiefMedicWkstn, AdminDB* (in Fig. 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, Sally-Press* 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 Sect. 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.

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 Fig. 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 Fig. 3.

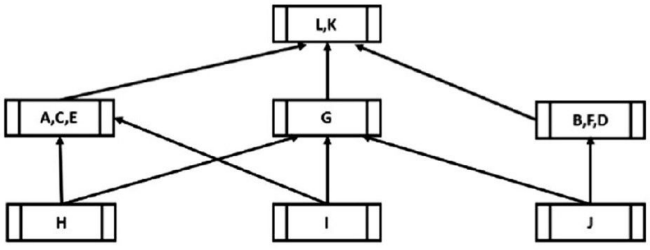


Fig. 3. Partial order of components for the hospital example

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 mis-configuration 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.

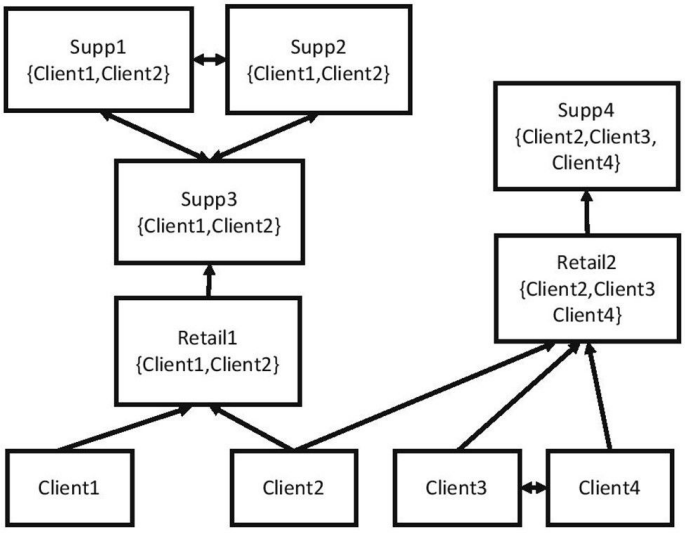


Fig. 4. An e-commerce example

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

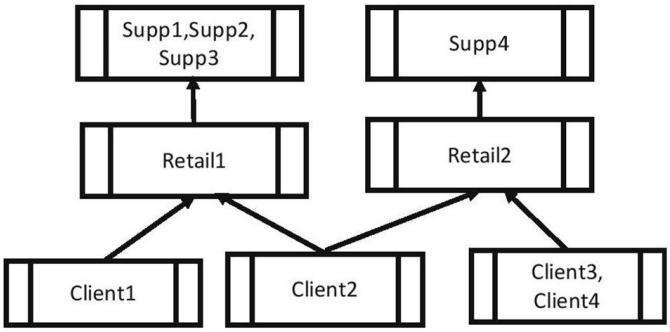


Fig. 5. Partial order of components for the e-commerce example

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.

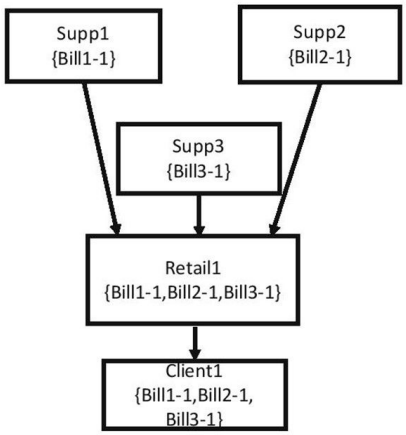


Fig. 6. Partial billing flow in the e-commerce example

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. 1, 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 [12, 18]. 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 Patient (Sam).
New NurseWkstn(EmergWkstn)        Assign (Sam,Emerg)
New Nurse (Alice)                 New PressDetect(PRD0001)
Assign (Alice,Emerg).             Assign (PRD0001, Sam)
Assign (EmergWkstn,Emerg)         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 Sect. 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 methods, 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 N.V. Narendra Kumar for having carefully reviewed the paper.

References

1. Bacon, J., et al.: Enforcing end-to-end application security in the cloud. In: Gupta, I., Mascolo, C. (eds.) Middleware 2010. LNCS, vol. 6452, pp. 293–312. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16955-7_15
2. Botta, A., de Donato, W., Persico, V., Pescapé, A.: Integration of cloud computing and internet of things: a survey. Future Gener. Comput. Syst. **56**, 684–700 (2016)
3. Blackstock, M., Lea, R.: Towards a distributed data flow paradigm for the Web of Things. In: Proceedings 5th ACM International Workshop on the Web of Things (WoT 2014), pp. 34–39 (2014)
4. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
5. Etalle, S., Hinrichs, T.L., Lee, A.J., Trivellato, D., Zannone, N.: Policy administration in tag-based authorization. In: Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Miri, A., Tawbi, N. (eds.) FPS 2012. LNCS, vol. 7743, pp. 162–179. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37119-6_11
6. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: Role-Based Access Control, 2nd edn. Artech House, Boston (2007)
7. Fraïssé, R.: Theory of Relations. North-Holland, Amsterdam (1986)
8. Gilat, A.: MATLAB: An Introduction with Applications, 2nd edn. Wiley, Hoboken (2004)
9. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
10. Harary, F., Norman, R.Z., Cartwright, D.: Structural MODELs: An Introduction to the Theory of Directed Graphs. Wiley, New York (1965)
11. Jiang, L., Xu, L.D., Cai, H., Jiang, Z., Bu, F., Xu, B.: An IoT-oriented data storage framework in cloud computing platform. IEEE Trans. Ind. Inf. **10**(2), 1443–1451 (2014)
12. Khobragade, S., Narendra Kumar, N.V., Shyamasundar, R.K.: Secure synthesis of IoT via readers-writers flow model. In: Negi, A., Bhatnagar, R., Parida, L. (eds.) ICDCIT 2018. LNCS, vol. 10722, pp. 86–104. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72344-0_5
13. Landwehr, C.E.: Privacy research directions. Commun. ACM **59**(2), 29–31 (2016)
14. Logrippo, L.: Multi-level access control, directed graphs and partial orders in flow control for data secrecy and privacy. In: Imine, A., Fernandez, José M., Marion, J.-Y., Logrippo, L., Garcia-Alfaro, J. (eds.) FPS 2017. LNCS, vol. 10723, pp. 111–123. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75650-9_8
15. Narendra Kumar, N.V., Shyamasundar, R.: Realizing purpose-based privacy policies succinctly via information-flow labels. In: Big Data and Cloud Computing (BDCloud 2014), pp. 753–760 (2014)

16. Ouaddah, A., Mousannif, H., Abou Elkalam, A., Ait Ouahman, A.: Access control in the internet of things: big challenges and new opportunities. Comput. Netw. **112**, 237–262 (2017)
17. Park, J., Nguyen, D., Sandhu, R.: A provenance-based access control model. In: 2012 10th Annual International Conference on Privacy, Security and Trust, pp. 137–144 (2012)
18. Pasquier, T., Bacon, J., Singh, J., Eyers, D.: 2016. Data-centric access control for cloud computing. In: Proceedings of 21st ACM Symposium on Access Control Models and Technologies (SACMAT 2016), pp. 81–88 (2016)
19. Samarati, P., Bertino, E., Ciampichetti, A., Jajodia, S.: Information flow control in object-oriented systems. IEEE Trans. Knowl. Data Eng. **9**(14), 524–538 (1997)
20. Sandhu, R.S.: Lattice-based enforcement of Chinese Walls. Comput. Secur. **11**(8), 753–763 (1992)
21. Schütte, J., Brost, G.S.: LUCON: data flow control for message-based IoT systems. arXiv preprint arXiv:1805.05887, 2018 - arxiv.org
22. Singh, J., Pasquier, T., Bacon, J., Ko, H., Eyers, D.: Twenty security considerations for cloud-supported Internet of Things. IEEE Internet Things J. **3**(3), 269–284 (2016)
23. Singh, J., Pasquier, T., Bacon, J., Powles, J., Diaconu, R., Eyres, D.: Big ideas paper: policy-driven middleware for a legally-compliant Internet of Things. In: Proceeding Middleware 2016 Proceedings of the 17th International Middleware Conference, Art. No. 13 (2016)
24. Stambouli, A., Logrippo, L.: Data flow analysis from capability lists, with application to RBAC. Inf. Process. Lett. **141**, 30–40 (2019)
25. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
26. Winter, T., Thubert, P. (eds.): RPL: IPv6 routing protocol for low-power and lossy networks. Internet Engineering Task Force IETF RFC 6550, March 2012

Chapter 7

Implementation of a multi-level model using SDN in an IoT environment

In this chapter, we will focus on the implementation of our method in the context of the Internet of Things. We will use labels and Software-defined network in order to enforce the policies of our model for secrecy and privacy requirements.

1. Introduction

The work presented in the previous chapters, as well as the study of the related research, has led us to the conclusion that the use of labels is very important in data flow control for security. As mentioned in chapter 6, we will draw inspiration from the work presented in Etalle et al. [39] where a policy administration method for distributed systems in presented. In this model, a function Tag is defined, this function maps subjects or objects to the set of tags assigned to them. Furthermore, a security administrator can write logic-based authorization policies that define access rights in terms of these tags.

Another interesting work is presented in Singh et al. [116].where entities and data are labelled with two labels, one label for *secrecy (S)* and another one for *integrity (I)*. For secrecy requirements, security policies are defined in such a way, that data stream from the entities can only flow into other entities labelled to receive such data.

This type of tagging can be used in the intended implementation of our method. However, in our case we plan to tag data to determine what flow the data belongs to, since as we showed in chapter 6, several flows can coexist in a system.

Once the tagging is done, we can propose our semantic for the Authorization Policies according to the labels and CF relationships .

The literature describes five main IoT security requirements [97] : 1) RFID tag information security; 2) Wireless communication security; 3) network security; 4) privacy and secrecy; 5) information processing security.

In our research, we have a special attention for secrecy and privacy requirements. It is very important that in a network, only authorized entities can access the network's data (secrecy). It is also very important that certain data can be created or modified by authorized entities only (integrity). With the spread of IoT's utilization and the increasing numbers of connected objects, huge amounts of data are generated. With the growing usage of IoT applications, privacy concerns are raised, and the protection of those data becomes one of the most important challenges.

Recent advances in computer networking have introduced a new technology for future communications, the Software Defined Network (SDN). In this type of network, the control plane and the data plane are separated, and the management of the network is centralized and done by an SDN controller. This centralization allows a fast reaction to security threats, traffic filtering and facilitates security policy deployment, as we shall see.

There is research in the literature that proposes secured SDN-based frameworks for the IoT in order to improve security, we will cite those papers in section 3. However, the main concerns of this research are the management and deployment of security policies, identity management, and in some cases detection or prevention of intrusions. Little has been done in data privacy, data flow control and enforcement of data flow policies as we intend to do in this work.

In this work, we propose a method to enforce and implement data flow policies using SDN for the IoT environments. We chose to work on a centralized IoT architecture where all generated data will be transferred and stored in cloud platforms and then accessed by users' applications. This access will be determined by our partial order model, where every network that intends to implement secrecy and privacy is seen as a partial order of equivalence classes of entities that shows how data flows in the network.

2. Software defined networks

SDN - Software-Defined Networking - has been very debated in the networking world recently. It is an evolution of the classic network model into a network defined by application. So, SDN is perceived today as a design that opens up the organization to applications. It has been said that SDN is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for today's applications [43] . This architecture separates the network

control (control plane) and forwarding functions (data plane) enabling the network control to become directly programmable via applications. This programming is done via SDN controllers instead of classical networking protocols. The network also become centrally managed. This centralization allows the controller to maintain a global view of the network and controls it through standards such as Open Flow, which is a protocol defined by the Open Networking Foundation to transfer forwarding rules from the controllers into the network equipment. So, all the forwarding rules and flow control rules are injected from the controller into the equipment via Open Flow.

When we talk about SDN, we mainly talk about the solutions that allow network programming. There are several ways to do such programming. One way is to individually program each piece of equipment on the network. Each application interacts with the equipment via APIs. A second way is via network virtualization. But the most common way and the one that we use in our work is programming via a controller [81] . In this case, an application gives an abstract and global order to a controller, which in turn translates this request into a series of orders to the network equipment concerned. Many controllers exist for such a task, we will be back to this point is our simulation results section.

An SDN-based network architecture divides the processes (configuration, resource allocation, traffic prioritization, and traffic redirection in the underlying hardware) into three basic layers: the application layer, the control layer (control plane), and the infrastructure layer (data plane). Each of the layers has a well-defined boundary, and a specific role, and the use of the APIs [81] is done to ensure communications between all these layers. The standard architecture of SDN is illustrated in the following figure:

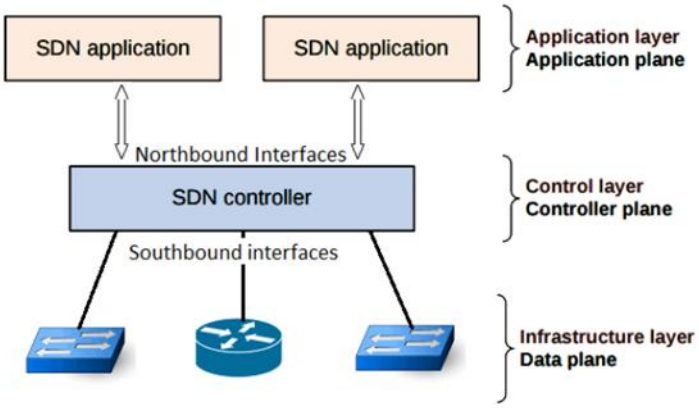


Figure 18: Standard architecture of an SDN [5]

- The infrastructure layer or the data plane: a data plane is defined by a set of network components, such as switches, routers, virtual devices, firewalls, etc. The purpose of a data plane is to transmit network traffic, efficiently, based on a certain set of transmission rules ordered by the control plane. Communication between the data plane and the control plane is provided by southern interfaces called the Southbound API.

- Control layer or the control plane: the control plane is responsible for making decisions about traffic through the network. The central component of a control plane is the SDN controller. An SDN controller translates application requirements and business goals such as the need to prioritize traffic (quality of service), access control (privilege), bandwidth management, and more. Then, this information is communicated to the components of the data plane.

- Application layer or the application plane: the application plane contains network specific programs and commercial software. An abstract view of the underlying network is presented to applications via Northbound APIs. - South interfaces (Southbound API): the South interfaces constitute a protocol between the SDN controller and the data plane such as OpenFlow [89] . They control transfer operations, event notifications, statistical reports and also advertise network capabilities. Essentially, they allow a controller to define the behaviour of hardware on the network.

- The Northbound APIs: the North interfaces represent an abstraction of network functions with a programmable interface. In other words, they allow applications to consume network

services, and dynamically change their behaviour. The architecture and North interfaces of SDN controllers vary by vendors. Some vendors incorporate SDN controllers into their applications, while others define North interfaces to facilitate the communication protocol between the controllers and their own SDN services at the application layer.

In terms of security, SDN proposes some solutions to some security concerns especially for the IoT. SDN can help control the network and leverage segmentation to mitigate potential network intrusions. This means that by taking a cloud-based approach, enterprises can use SDN to optimize, route, and automate IoT security and control data flow as we do in our approach.

3. Related work

Several papers mention access control in the IoT, for example Smriti et al. [118], and Xie et al. [129]. However, with the recent emergence of SDN as a substitute for traditional networking methods based on hardware [37] [127], many researchers lean to use this new concept to propose new architectures that can suit better IoT networks. This shift is due to the fact that software defined networks (SDN) respond better to the needs and challenges of these networks. Through SDN, all the network elements will be programmable using a user interface, and all network operations and elements will be implemented completely in software, which facilitates the control over the network. All those properties make SDN an ideal answer to future networks challenges such as the handling of enormous number of connected devices and the quick recovery from different types of failures.

Many papers tend to propose new IoT architectures based on SDN for better handling of the evolution of such networks. Mamdouh et al. [79] presents a new architecture for IoT infrastructure based on network virtualization including SDN. Their SDN paradigm for the IoT consists of three different planes. The data plane regroups all the IoT network elements as simply forwarding devices. The control plane residing in the SDN controller and the management plane are complementary planes to the latter and they are jointly responsible for the management and control of network operations. This architecture provides better network sharing and can handle big data input from IoT devices, as well it simplifies management tasks.

Yassein et al. [131] propose some solutions that combine SDN and IoT networks in order to respond to the latter challenges. Hakiri et al. [49], Wu et al. [128], and Qin et al. [96] propose

solutions based on SDN to handle and to manage large numbers of devices and to schedule the flow of the data generated by those devices.

Other papers propose specific solutions that use SDNs to secure IoT networks. Flauzac et al. [42], and Olivier et al. [88] first introduced the notion of multi-domain SDN. In [88] The network is divided into multiple SDNs where for each SDN, we have an SDN controller as a cluster head. Then the securization of domains will be the task of the controller that authenticates the network devices and then pushes the appropriate flow to the switch software. Finally, to ensure security of the whole network, security policies are shared among other domain controllers using the concept of security grid. However the authors do not provide any implementation or performance analysis for their architecture.

These papers were followed by the work of [47], where Gonzalez et al. suggested DISFIRE, a Smart Firewall to provide a safe structure for SDN networks. The network is divided into clusters with an SDN controller in each cluster. These clusters execute a safety strategy. For this objective, they used a protocol named OpFlex as an alternative to OpenFlow. The SDN controller can then execute a firewall that can cancel any unauthorized devices.

Aggarwal et al. [7] proposes a solution to secure IoT devices against external attacks instead of the data flow security that we implement. The method uses the implementation SDN & Edge Computing and the security of the devices depends on the way they are connected to the Internet.

Karmakar et al. [63] propose a security architecture for IoT networks using SDN features. This solution is divided in two phases. First, the devices are authenticated to the SDN controller using a lightweight protocol based on Elliptic Curve Cryptosystem (ECC), and then using a policy based Security Application (PbSA). Security policies are enforced by the SDN controller. To enforce such security policies, each device of the network is assigned a number of attributes. Then, predefined security expressions will use those attributes to determine the behaviour of the switches of the network.

Prabhakar et al. [95] presented an SDN framework for securing IoT networks against external attacks principally against anti–distributed denial-of-service attack (DDoS). This paper presents some design that incorporates the cloud and fog to prove the capabilities of SDN in monitoring dynamic policy enforcement, access control at run time. Finally, they simulate DDoS attacks

to show the capability of their solution to detect and mitigate such attacks.

Sahoo et al. [102] propose a safe architecture for IoT network using multiple domain SDN. The authors identify five mainly safety features to consider when building a strong security model. These features are: authentication, particularity, impartiality, availability and non-denial.

Papers [35,50] propose secured solution for cloud-based IoT. Djouani et al. [35] They use the same domains architecture presented in other works [42, 88] , but with the addition of the encryption of the data by the devices before they send them through the network. Han et al.[50] the authors developed a three-layer framework (perception layer, software-defined network layer, and cloud-based application layer) that integrate SDN and Cloud-IoT. The developed framework consists of 23 indicators for security features, those indicators are scattered in each layer meaning that each layer has its own indicators. Each one of those indicators was given a weight based on online interviews with researchers alongside with three weighting methods. Finally, those indicators are mapped into CloudIoT platforms such as Google Brillo and Microsoft Azure IoT to get an overall end-to-end security framework.

Flauzac et al. [42] presents a secured SDN framework for the IoT. In this paper, an IoT device is seen as a combination of legacy interfaces and SDN controllers. They do not use a centralized SDN controller, but a few devices will have SDN capabilities and will act like SDN controllers. In addition, those controllers alongside border controllers will distribute routing and security rules. The main critique of such solutions is the use of IoT devices to play the role of controllers, given the limited resources that the devices have. Such additional chargers of treatment and control can be an issue.

As we have seen, the vast majority of these papers tackle security factors such as exchange and deployment of security policies within the network in the case of SDN domains, intrusion detection, security against external attacks, and especially certification. Some of the proposed security solutions use cryptographic algorithms that may require non-trivial computational resources. Considering that most IoT devices are associated with low energy and limited computing resources, such solutions may be impossible to implement with the application of traditional cryptographic mechanisms.

It should be clear that none of these papers presents a data flow control method comparable

to the one presented here.

The closest work to ours, since it deals with data flow control and privacy concerns, is Al-Haj et al. [8]. This paper presents a solution to enforce security policies to control the routing configuration in database-defined networks. To achieve this, they use row-level security checks and the lattice model alongside the RAVEL architecture [126]. The following figure shows the general architecture of the solution.

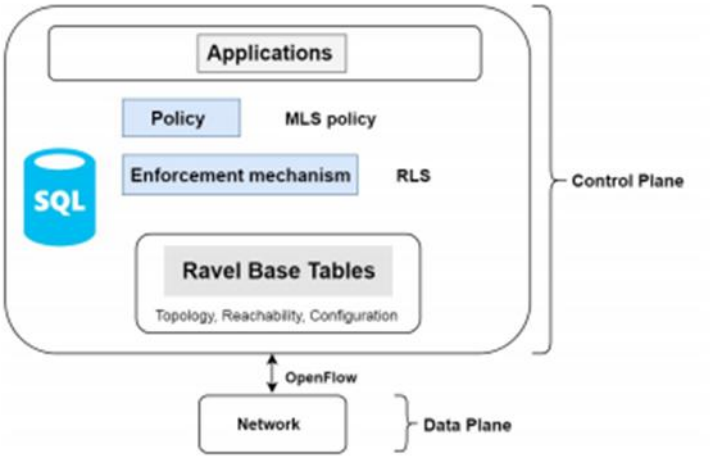


Figure 19: The solution proposed in [126]

Their solution consists of constructing routing tables by using the lattice-based model [31], encoding the tables in the database-defined network architecture RAVEL [126] and enforcing multi-level security policies using row-level security as an enforcement mechanism. The authors deal separately with secrecy and integrity. To enforce upward flow of data, the authors propose to define the flow path in the CF (Can Flow) table. This path consists of sequences of nodes that data can flow into. Once a path is defined, each node in this path starting from the first one will be given a security label. Finally, a security policy is defined in respect to a multi-level model, which states that data can only flow upward from a security level into a higher one. The enforcement of downward data flow for integrity is dual.

Our work very considerably generalizes the work done in this paper. Instead of using the lattice model, we use the much more general partial order model. Further, we treat secrecy and integrity aspects with a single mechanism. Important aspects that we handle but that

they do not consider at all are the idea of different data flows in a single network and network transformation. One useful idea of theirs that we leave to further research is the use of a database approach to represent data flow policies.

4. Our Implementation method

4.1 Architecture

As we said in the introduction, we choose to work on a centralized IoT architecture. Many papers in the literature propose centralized architectures such as Christos et al. [27], Hany et al. [51], and Roy et al. [98]. Centralization might seem to be inconsistent with the decentralized nature of the IoT. However, our efficient centralized algorithms can reconfigure networks dynamically as necessary. They don't seem to be excessively constraining in closed systems such as hospitals, industrial plants, smart homes, and the like. It can be questioned at this point whether a decentralized data security architecture is at all possible, and we conjecture that any that could be found wouldn't be able to implement exactly our partial order model.

With the rapid development of processing and storage technologies and the success of the Internet, computing resources have become cheaper, more powerful and more available than ever before. This technological trend has enabled the realization of a new computing model called cloud computing, in which resources are provided as general utilities that can be leased and released by users through the Internet in an on-demand fashion [11]. In other words, cloud computing is a computing platform that resides in a large data centre and is able to dynamically provide servers with the ability to address a wide range of needs, from scientific research to e-commerce, etc.

The cloud offers many services to the users such as SaaS, PaaS and IaaS [34]. In our implementation method, we only call on the storage service using cloud servers. The storage service is considered to be an IaaS service since it requires an infrastructure that has the capability to provide the consumer storage resources.

There are many benefits in the use of the cloud storing service such as

- For data security, the cloud offers many advanced features to ensure the secure handling and the storage of data. Cloud storage providers provide baseline protection for their platforms

and the data they process, such as authentication, access control and encryption. In addition, most user organizations complement these protection mechanisms with their own added security measures to enhance cloud data protection.

• Usability, accessibility, and availability: most cloud services come with easy-to-use downloadable interfaces. These are usually simple and can be easily uploaded without expert knowledge.

• Storing away organizations or user information in a cloud server can secure it against unintentional loss (e.g. due to fires, floods, or seismic tremors). Data backups are automated and usually take place daily. So, users can retrieve their data in case of an emergency.

• Cloud storage can also be scaled to meet the company's changing priorities and ongoing growth. As a business grows, users can scale their cloud service plans to include more storage and even get their own private storage servers. An important aspect of cloud storing is that is collaboration oriented. Users with permissions and connected devices can access the same data stored in the cloud, like in our case, users in the same class can access the same data. With these benefits, come some drawbacks to using cloud storage, such as

- Internet dependency: without the internet, data in cloud storage become inaccessible . If there is an internet failure, it might corrupt the data which you were downloading.

- Costs : most of the best cloud services are expensive. If you go for a less expensive plan, you might have to compromise with some of the features.

In the cloud, a data container and a server can be two distinct entities interconnected via the network. For simplification, we choose to represent them as a single entity. This approach can significantly automate and facilitate network management especially for security, where the management of network security is done by trusted centralized entities.

In centralized IoT systems, all devices are connected through cloud servers and communication between different devices must be achieved through centralized servers. The centralized IoT architecture consists of three main layers: Sensing layer, networking layer and application layer.

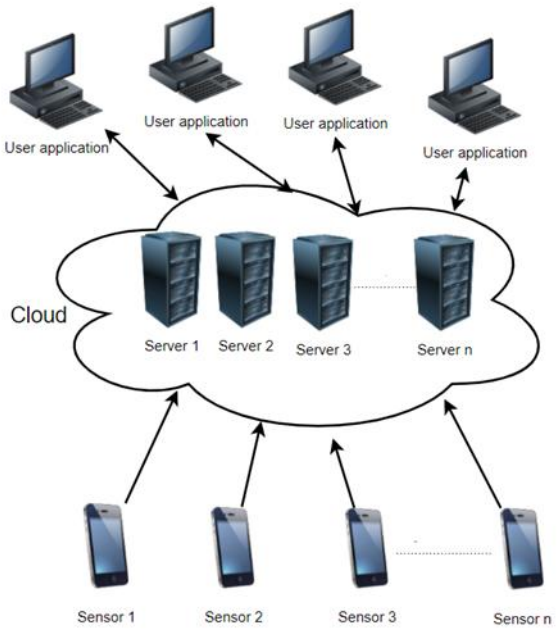


Figure 20: Centralized IoT arrchitecture

The sensing layer consists of different types of sensors, RFIDs and other collecting devices. This layer collects all data about the environment and sends them to the cloud servers via centralized gateways. Figure 20 shows this architecture in general terms, Fig. 21 shows it in the way that it will be used in this thesis, and Fig. 22 shows the architecture of Fig. 21 using the graphic notation to be used later in this chapter.

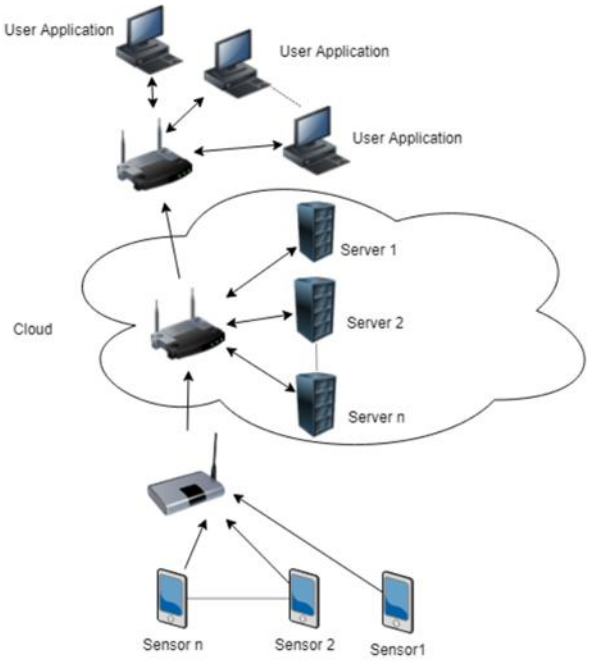


Figure 21: Implementation architecture of the case study

The networking layer is used to connect all the IoT objects to the Internet, it also contains all the servers used to store the data collected from the sensing layer. Several communication technologies and protocols are used in this layer such as 3G/4G/5G, Zigbee, Bluetooth, Wi-Fi to transport data from the sensing layer to the application layer on one hand, and inside the networking layer between the servers, on the other hand. In our case, we chose to work with Wi-Fi since with this technology every entity or object in the system will have an IP address that identifies it.

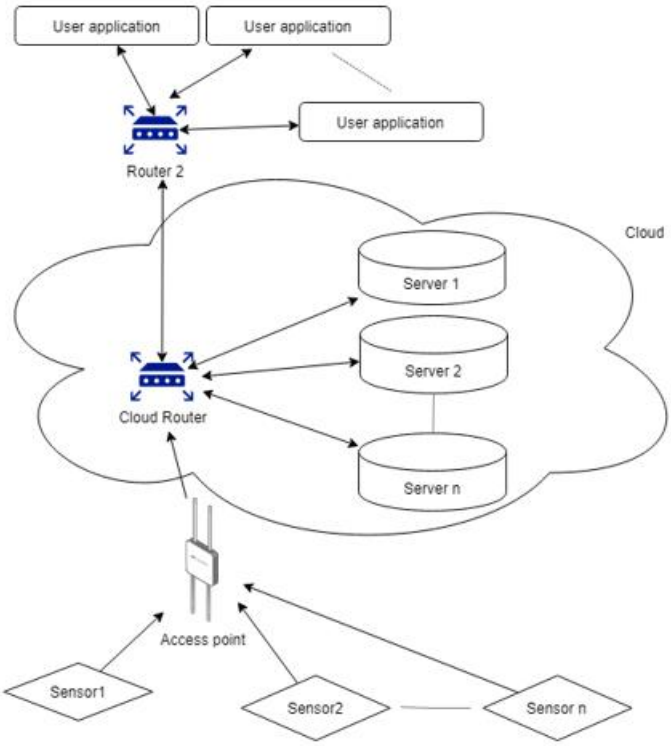


Figure 22: Implementation architecture used in this chapter.

The application layer involves various IoT applications that can use the data collected by sensors such as in healthcare system, smart cities, etc.

We assume that there are bidirectional communication channels between all types of connected objects (sensors, servers, workstations... and between networking and application layers, inside the networking layer). These channels will have to be used in a particular way to implement our method. Some of the concepts used in our architecture are

• Connecting servers in the cloud using cloud routers allows us to create communication channels between different servers in the cloud. Routers are centralized entities that all the servers are connected to. We can find this architecture in [6].

• Server to server communication: since servers are connected using the cloud router, data can flow from a server into another server. We can do this using Websocket connection between servers so either side can send data to the other.

• Server to client communication: normally in a network, we have a client to server communication, where the client sends a request to the server and the latter grants the request. However, in our case, we can have a server to client communication. This can be done just like the server-to-server communication using socket.io which is a library that enables bidirectional communication between web clients and servers.

Once our architecture is done, we introduce the concept of SDN, which we will show capable of implementing our partial order model. By separating the control plane from the data plane or the forwarding plane, SDN allows central management of the network. It becomes easy to define security policies for networks with the software-controlled nature of SDN networks. In our implementation, we will use the classical Openflow protocol for the connection between the data plane and the control plane through the southern API. The controller will have two routers to take care of, the first one is the cloud router, which interconnects the servers of the cloud, and the second is the router that connects the cloud layer or the networking layer with the application layer. We will not program the device to connect the sensing layer with the cloud layer (we can use a simple access point), since this device is mainly charged with forwarding the collected data to the cloud router without control since the latter will be doing control.

We chose to work with a single controller for our architecture, this controller will be in charge of all the forwarding planes (routers) in our architecture. Many works in the literature used a single controller for wide area SDN. In El-Garoui et al. [38] and Dias et al. [33], authors use the same controller as us (Ryu controller) to control multiple routers in their wide area SDN. For this to be done, we must assume that we work with a more classical cloud architecture, where we have only one data centre that stores data in one site. In the case of distributed cloud, other types of controllers must be used such as hierarchical of flat controllers. However, this is out of our scope in this work. The following figure present our final architecture.

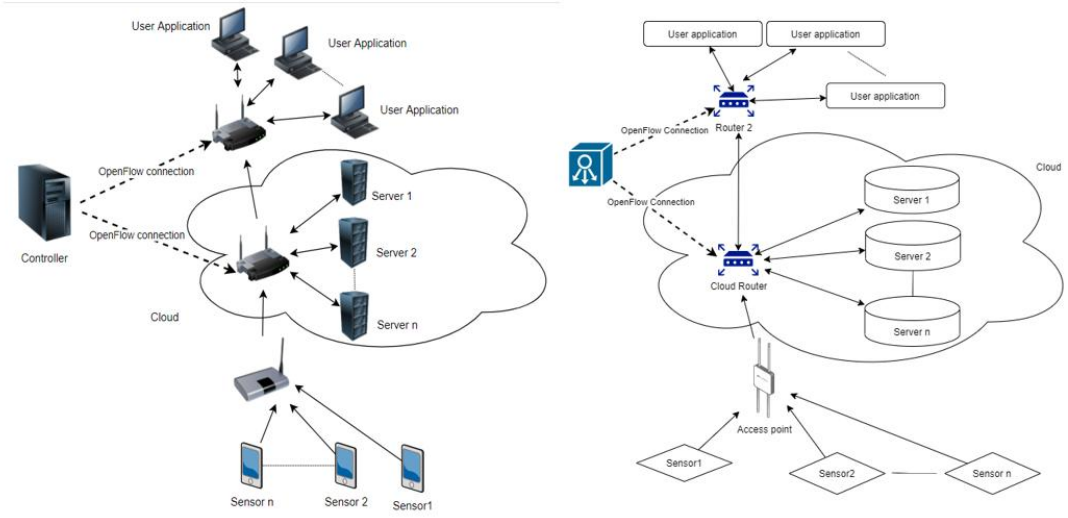


Figure 23: Final architecture of our implementation.

Our method starts by following the principles stated in Chapters 4,5, and 6 and is summarized in Figure 24, see also the following case study. We start with a network representing an application layer architecture of directly connected application entities. The first step is to determine the partial order of equivalence classes of entities in this network and to assign labels to the entities. Essentially, this step identifies all the sets of application entities that are connected with each other in such a way that data can flow from any of them to any other and collapses each such set into an equivalence class. What is left is a partial order of equivalence classes with sink and source elements. A straightforward algorithm assigns to each application entity a label which is a set of categories and which defines the position of the entity in the partial order, while at the same time defining the provenance of the data that can end up in the entity. The initial network will not be in the form of the centralized cloud-based architecture of Fig. 20, since application entities will be shown as communicating directly and not through the cloud. So the next step is to create the Cloud infrastructure. This will be done by assigning at least one server or storage entity as they may be called henceforth, to each equivalence class that doesn't already have one. The label of these new entities will be the same as the other entities in its equivalence class, hence the partial order of equivalence classes will be the same, with storage entities added to each equivalence class. The collection of these storage entities

form the cloud and implement the Networking Layer of the IoT. The storage entities are logically separated in various equivalence classes, but physically they can be anywhere, e.g. close to the site of the applications in their equivalence class, or centralized together in a computing centre, etc. So although we often mention in this chapter that our networks are 'centralized', this means centralized control and doesn't imply physical centralization. The inclusion of the storage entities and the use of a cloud architecture is new with respect to the method described in Stambouli and Logrippo [120,77].

Once the partial order obtained, we use it to create what we call a labeling table. This table shows what are the labels of the various entities and it is a representation of the partial order in a data structure that will be implemented in the controller. This will be shown in the next section. Alongside the labeling table, we will implement our data flow control policies. The controller will use them both to provide the routers with the adequate forwarding table used to monitor traffic on the network. This forwarding table will implement our secrecy and privacy requirements.

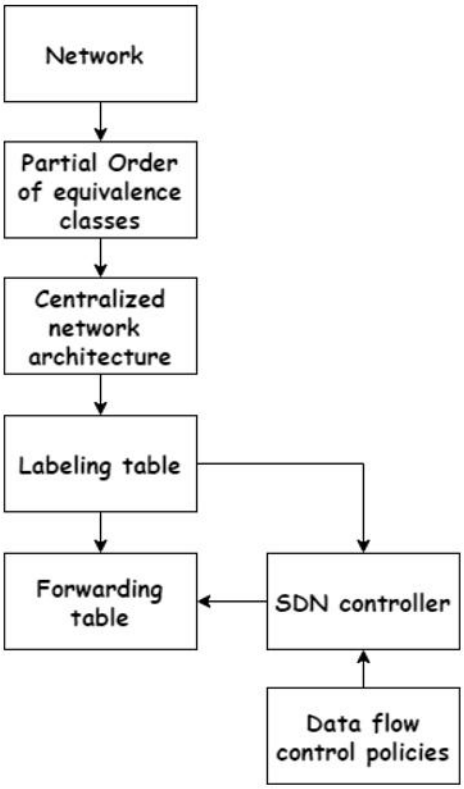


Figure 24: Summary of our method

4.2 The labeling table

In order for routers to control data flows, they must have forwarding tables. These tables will be provided by the SDN controllers. In our method, to obtain forwarding tables we must implement a *Labeling table* into the controller. These tables are directly derived from the labels of the entities, described in Chapter 6.A labeling table is a two-column table that indicates for each entity of the network, depending on the labels, which are the entities that the latter can receive data from. A labeling table can be seen as a representation of the partial order of the system. Thus, if we have an entity at a certain level of a partial order that flows into an entity at a higher level, the label of the first entity is included in the label of the higher-level entity. The first entity will be present in the column Holds of the second entity in the labeling table.

Thus, the labeling table can be constructed by the following rule:

$$\textit{If label (A)} \subseteq \textit{label (B) then A} \in \textit{LabelingTable.Holds (B)}$$

Where *LabelingTable.Holds(B)* is a list that represents the set *label(B)*.

In order to identify the different entities of a network, we will use their IP addresses (IPAD): since we choose to use Wi-Fi communications, every entity has a unique IP address. So for example, if an entity B can hold an entity A, we will have the following labeling table:

Entity	Holds
IPAD (B)	*IPAD (A)*

Table 3: Labeling table structure

The labeling table will be different from a router to another, for each router we will implement a labeling table that only incorporates the entities that are connected to this router. If a packet arrives to a router and the entity cannot be found connected to this router, the router will automatically forward this packet to next router in the architecture. This will prevent an overload of routers and will provide a better processing time.

4.3 Data flow control policy enforcement

Once the labeling tables implemented, we will implement our data flow policy that states the following:

$$\textit{CF (A, B) if and only IPAD (A)} \in \textit{Labeling table.Holds (IPAD (B))}$$

Which translates the rule of Chapter 6 *CF (A, B) iff Label(A)* $\subseteq$ *label (B).*
Now using this policy and the labeling table, the router can decide if a received packet should be forwarded or dropped. An IPv4 and IPv6 packets have source and destination headers, with these two headers a mapping with the labeling table can be done to test if the policy is satisfied and this will determine if the packet should be dropped or forwarded.

The labeling tables will be used by the controller to construct the forwarding table and deploy it to the routers. It is this forwarding table that will be used to control the flow in the network. The forwarding tables consist of flow entries. Of the several columns they may have, we take into consideration only the columns Match Rules and Action. Using the labeling

table, the controller will determine the match rules and the action for each flow entry in the forwarding table. In this case, for example, if in the holding table IPAD(A) holds IPAD(B) then the controller will create a flow entry using the IPAD source (IP src) and IPAD destination (IP dst) as match rules and define the forward action for such a flow since it is an authorized flow. When a packet arrives to the router, the latter will compare the IPAD destination and IPAD source in the packet headers with the match rules of the entries in the forwarding table. If there is a correspondence with any entry, the router will perform the matching action to this entry. Otherwise, the packet will be dropped. In our case, since we use the labeling tables to create the forwarding tables, all the flow entries will have the action *forward*. So, if there is any correspondence between the packet headers and the match rules of the entry, the packet will be forwarded to the destination. Otherwise, the packet will be dropped.

To simplify our presentation, we consider that we deal with small to medium size systems that can be implemented by using a single router in each layer. However, if the size of the network gets bigger and exceeds the connectivity capacity of one router, we can add other routers and divide the communication in a layer between those routers. In the network community, it is said that average routers in use today can have a maximum of 250 users connected to them [2]. However, we can increase this number by connecting routers between them sequentially. The only condition to maintain in this case is that we need to connect the routers between in the two directions. In this case we can cascade the routers. Many modern routers adapt automatically if a port is connected to another router. In our case we only show the two endpoints routers, the cloud router and router 2. There can be a number of routers between them that will transport the data flow using the internet. To simplify the description our architecture and implementation, we only took in consideration the two important routers in the series. Each router will have its proper labeling table depending on the entities that it connects.

We summarize our implementation method as follows. The sensing layer will collect data and send them through access points to the cloud router. Once data reaches the cloud router, this will decide if the packet should be dropped or forwarded. If the destination IPAD present in the packet header is for an entity that is connected to the cloud router but the flow is not connected to the cloud router, the latter will automatically send the packet to the next router that will do the same processing. This method will assure that only authorized entities will

have access to the data. The partial order of equivalence classes, which is essential for data privacy and secrecy, will then be implemented. Controllers formulate the policies for the routers in device-specific languages. In our simulations, the labeling tables were coded in Python.

5. Case study

As a case study we consider the following system, a hospital system whose configuration is shown in Figure 6. A similar system was proposed in [77], but it is refined here.

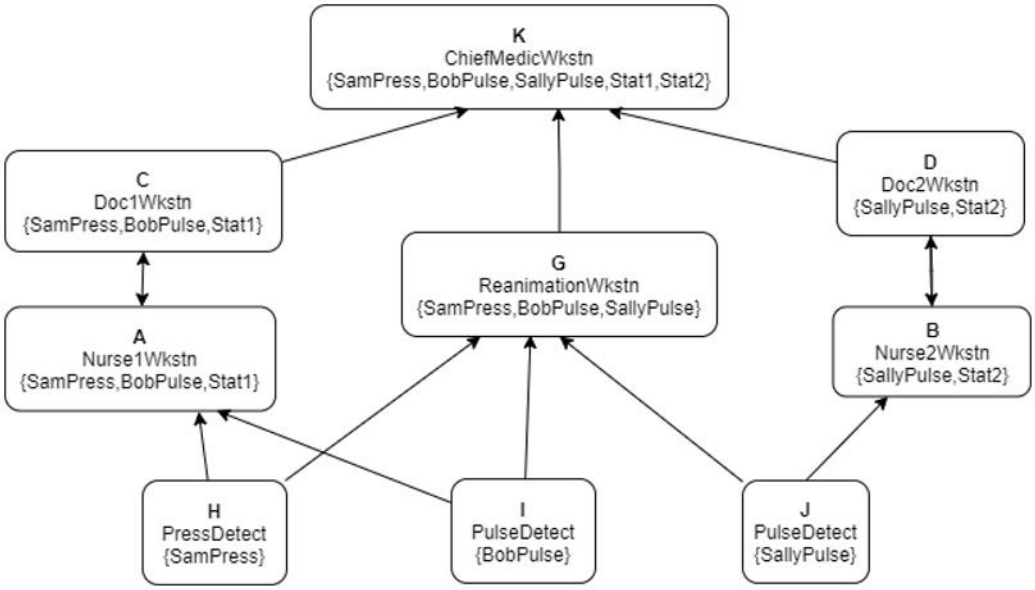


Figure 25: Hospital example

There are three sensors, one for the blood pressure of Sam, one for the pulse of *Bob*, and one for the pulse of *Sally*. They all feed data to the *Reanimation workstation*. There are two *wards* in the hospital, each with a *nurse* and a *doctor*, with their *workstations*. *Sam* and *Bob* belong to the first ward, while *Sally* belongs to the second. The sensors of each patient feed into the workstation of doctors and nurses in their wards. In the end, all data feed into the workstation of the *Chief of Medicine,*

The security policies to be implemented are

• The sensors should have highest integrity but also low secrecy, since their data are needed by all other entities.

• The Chief of Medicine Department should have the lowest integrity, since it uses data collected from all other entities, but also the highest secrecy, since it contains highly sensitive data.

• Wards and the Reanimation workstation take data from the sensors, process them and forward the results to the Chief of Medicine, thus should have intermediate levels of integrity and secrecy.

• Conflicts: a) Patient data should be known only in the patients' treating team, but also in the Reanimation and the Chief of Medicine departments that can have knowledge of all such data. In addition,

b) Each treating team keeps its own statistics that should be known only to it and to the Chief of Medicine.

There are nine application entities in this system. Each entity has a label, specifying the data categories it can know. There is an arrow between an entity and another iff the label of the first entity is included in the one of the second. It can be checked that the policies above are implemented by the choice of label sets. For example, *Nurse1* can know only the blood pressure of *Sam*, the pulse of *Bob*, and its team statistics. This configuration implements a partial order of equivalence classes. Note the equivalence classes *{A,C}* and *{B,D}*, since the entities in these equivalence classes have symmetric channels and thus can know the same data. Using double-sided rectangles for equivalence classes, the partial order of equivalence classes for the system of Fig. 25 is shown in Fig. 26. Note that in order to simplify the diagrams we show them transitively reduced. For example, a direct flow from H to K is allowed, and it will be in our SDN implementation.

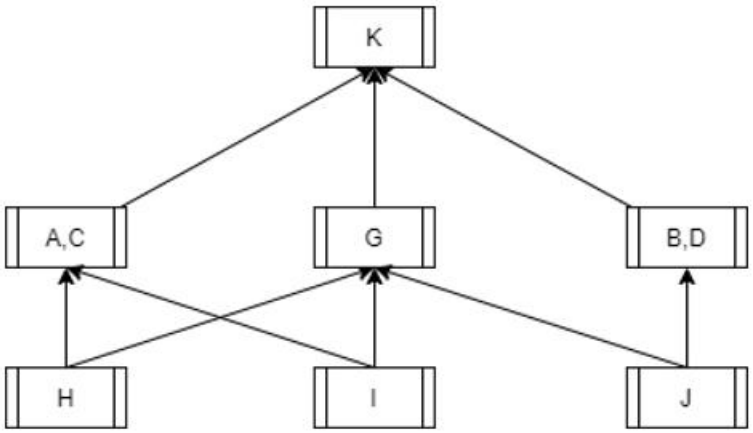


Figure 26: Partial order of equivalence classes for the example of Fig. 25

This configuration highlights the data flow relationships between the basic entities of the system. The sets associated with entities are their labels, which denote their possible contents, and the set inclusion relationships are the data flow relationships. This configuration will implement a partial order of equivalence classes that is essential for privacy and secrecy. Note that we have not shown in this configuration any servers or databases.

We will use our implementation method for this very simple, but realistic example. We implement the general architecture shown in Fig. 22 by using databases, access points and routers. First of all, we introduce the databases located in the cloud to store the data generated by the sensors. The databases are grouped in the cloud with a cloud router that interconnects them. The sensors are connected to access points that transfer their data to first-level cloud routers. These cloud routers forward the data to the databases. Finally, second-level routers are configured to connect the user endpoints to the first level of cloud routers. These last routers allow users to access the data stored in the cloud databases. Our final implementation architecture is shown in Figure 29.

In order to implement our case study in this context, some changes have to be done compared to the initial configuration shown in Figure 25. Some entities are added since the centralized aspects of our method forbid any flows between entities without passing through the central

layer of the cloud. Storage entities (ex: databases, servers...) will be added to the cloud, for each user entity, an equivalent storage entity will be associated with it. This storage entity will have the same label as its equivalent user entities to allow data flow between them. So, we have added for example an entity *G'* that allows the *ReanimationWkstn* to access the data from the sensors as is in the initial configuration. As we deal with a centralized system, it is essential that every entity have a bidirectional connection with a storage entity that is present in the cloud. However, we do not need to add a personal storage entity for every entity in the system. Some entities may share a single storage entity. We only need a database for each equivalence class. Also an entity is allowed to access only the database or databases in its equivalence class. The steps of adding the different databases can be found in the algorithm presented in section 5 of this chapter.

Also, in our implementation, we delete any entity to entity connections that are not transiting by a database. However, this does not change the connectivity of the system, since the required data flows can still be obtained by transitivity. The centralized architecture is presented in figure 27.

We can observe that figure 27 is almost the same as figure 25, the difference being that some storage entities have been added and some flows have been removed in order to eliminate any communications that do not respect the centralized aspect of our system. The partial order of Figure 27 is shown in Figure 28.

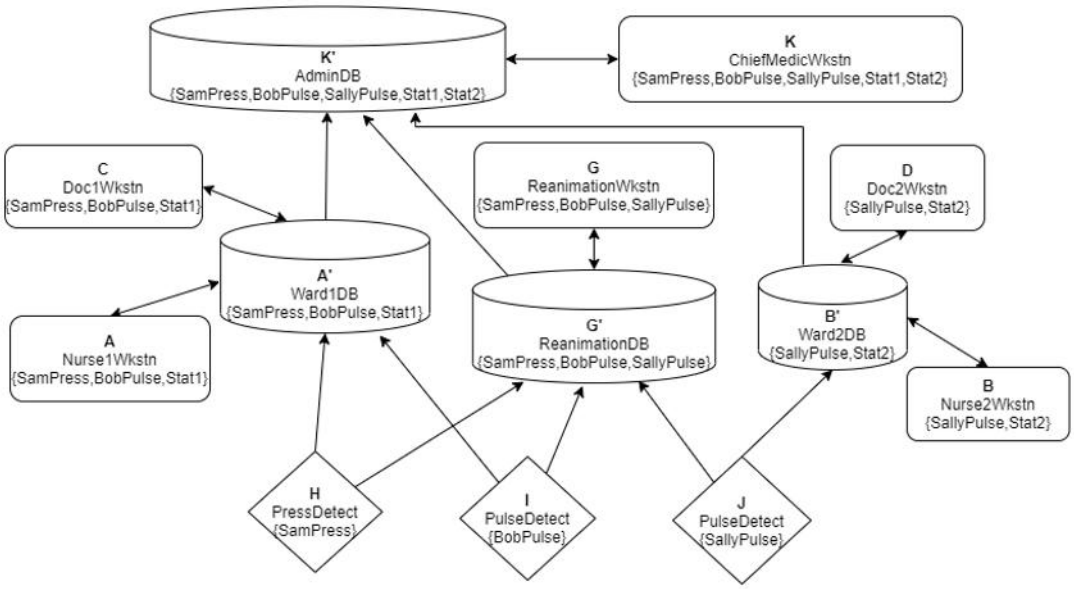


Figure 27: IoT configuration for our case study

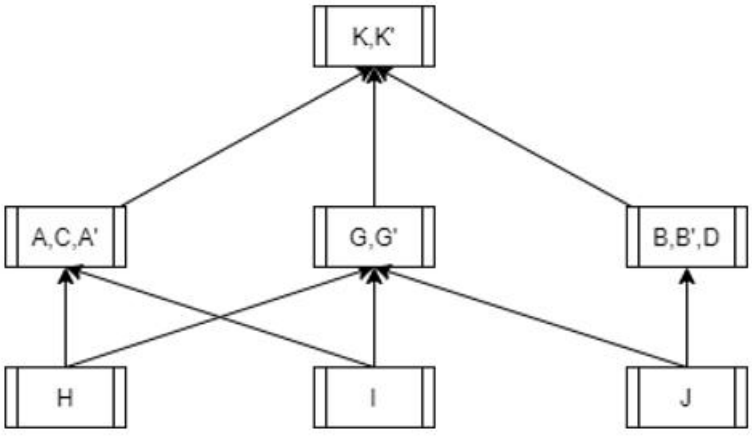


Figure 28: Partial order for the IoT configuration

By adding the required routers, we obtain the configuration shown in Fig. 29. Note that all the storage entities are connected to the cloud router, the application entities are connected to Router 2, while the sensors are connected to an access point, which in its own turn is connected

to the cloud router, just as in Fig. 22.

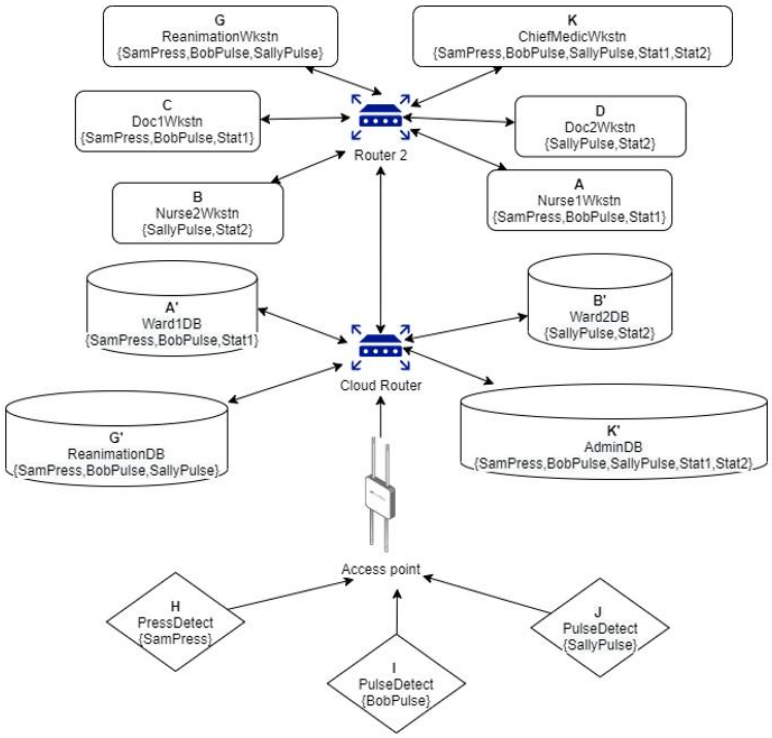


Figure 29: The physical topology for the case study

As we said in the previous section, to ensure the required flows and only them, we must configure our routers; we do this by adding labeling tables to the routers. In our example, the labeling table of the cloud router will have entries only for the entities connected to it and will be as follows:

Entity	Holds
IPAD (B')	IPAD (J), IPAD (B), IPAD (D), IPAD (B')
IPAD (A')	IPAD (H), IPAD (I), IPAD (A), IPAD (C), IPAD (A')
IPAD (G')	IPAD (I), IPAD (J), IPAD (H), IPAD (G), IPAD (G')
IPAD (K')	All

Table 4: Labeling table of the cloud router

So, for example, for data sent from the sensor that detects *SallyPulse (J)*, this data will arrive at the cloud router through the access point. The router will then check the labeling

Entity	Holds
IPAD (H)	IPAD (H)
IPAD (I)	IPAD (I)
IPAD (J)	IPAD (J)

Table 6: Labeling table of the sensors.

table and will verify which row satisfies the data flow policy. In this case, entities *G,B* are the ones that are allowed to receive this data so the router will forward data to these entities and drop the data for the other entities.

If the destination is not found in any row of the first router, the latter will automatically send the data to the second router, in our case the second router will have the following labeling table:

Entity	Holds
IPAD (D)	IPAD (J), IPAD (B), IPAD (D), IPAD (B')
IPAD (B)	IPAD (J), IPAD (B), IPAD (D), IPAD (B')
IPAD (G)	IPAD (I), IPAD (J), IPAD (H), IPAD (G), IPAD (G')
IPAD (C)	IPAD (H), IPAD (I), IPAD (A), IPAD (C), IPAD (A')
IPAD (A)	IPAD (H), IPAD (I), IPAD (A), IPAD (C), IPAD (A')
IPAD (K)	All

Table 5: Labeling table of the router 2

For the sensors, each sensor can only have itself as label. The labeling table of the sensors is shown in Table 6.

In this table, for each entity, we have restricted their labeling to their equivalent databases. This is because in our system no direct device to device communication is allowed, all the data transfer must pass though the central layer. However, we allow database-to-database communication in the same layer in order to permit data flows into higher levels.

Once the data reaches the second router, the same treatment is done, we check which row verifies the policy, we process data to that entity and we drop the rest.

Once connections are made, we can see that our implementation method produces the same configuration as the initial one, which means that the same partial order of equivalence classes has been obtained in the implementation.

At first sight, the final configuration of Fig. 29 seems to have no relation with the partial order of equivalence classes we started from, the only similarity being in the fact that the

detectors are at the bottom layer of both. However, by the contents of the labeling tables, the data flows between entities are the same. This means that the initially given policies of secrecy and integrity, as well as conflicts, are properly implemented.

Note that, although simple, this case study can be scaled up by introducing many more sensors, many more wards, many more workstations, etc. In order to make this possible, the entities would have to be parameterized or indexed, such as *PulseDetect1, PulseDetect2, etc.* The method of data flow control and its SDN implementation will remain the same, but the diagrams would be too large to fit in the pages.

6. Implementation algorithm

To summarize our implementation method, we propose Algorithm 2 that highlights how we go from the initial system architecture that can be considered as a logical topology into an actual physical one that can be configured and implemented in a real context.

This algorithm takes as input an established logical architecture in the form of a partial order of equivalence classes. However, we can transform this algorithm into an incremental algorithm where we start from an empty network and build our physical architecture, obtaining a partial order from it afterwards.

Then we will transform Algorithm 2 into another algorithm that will allow to add entities as the system evolves.. For each added entity, we must verify its type (sensor, database or workstation). This verification will allow us to determine to which router we should connect the entity. This will be Algorithm 3.

7. Logical architecture and physical architecture

When we mention the *logical architecture*, we always refer to the partial order of equivalent classes. In this architecture the level of an entity is very important. Entities at the lowest level are considered to be the least secret ones, and every time the level of an entity climbs its secrecy-level increases until we reach the top level considered the most secret one in the system. However, this concept does not apply to the physical architecture. The layers of the *physical architecture* are not related to secrecy levels in any way. An entity in the application layer that

Algorithm 2 Implementation algorithm

Input: Logical topology (graph that describes the data flow among entities, see Fig. 25)
Output: Physical topology (topology that shows how the system can be configured in a real context, see Fig. 28)

```
1: for all the workstations of the network do
2:     Create an equivalent storage entity
3: end for
4: for all entities E in the Input do
5:     for  all E that are not sensors do
6:         if E does not have a bidirectional connection with any storage entity then
7:             add a bidirectional connection between E and it equivalent stroge entity
8:         end if
9:     end for
10:    for  all sensors do
11:        remove all connections between the sensors and entity E
12:        add a connection from the sensor into storage entity equivalent to E
13:    end for
14:    save the new topology as Input
15:    for all entities X in the Input do
16:        add the same entity X in the Output
17:    end for
18:    for all the entities X in the Output do
19:        if entity X is a sensor then
20:            Place X in the sensing layer
21:        else
22:            if entity X is a storage device then
23:                Place X in the cloud
24:            end if
25:            Place X in the application layer
26:        end if
27:    end for
28: end for
29: create a cloud router R1 and a router R2 and an access point AP
30: add a connection from the access point AP into the cloud router R1
31: add a bidirectional connection between all the remaining routers (R1 and R2)
32: for all the entities C in the cloud do
33:    add a bidirectional connection between C and a cloud router R1
34: end for
35: for all the entities A in the application layer do
36:    add a bidirectional connection from A to a router R2
37: end for
38: for all the entities B in the sensing layer do
39:    add a connection from B to the access point AP
40: end for
41: Return Output
```

Algorithm 3 Incremental Implementation algorithm

Input: none

Output: Physical topology (topology that shows how the system can be configured in incremental context)

```
create a cloud router R1 and a router R2 and an access point AP
add a connection from the access point AP into the cloud router R1
add a bidirectional connection between all the remaining routers (R1 and R2)
for each added entity E do
    if E is a sensor then
        add a connection from E to AP
    else
        if E is a storage entity (database) then
            add a bidirectional connection between E and R1
        else
            add a bidirectional connection between E and R2
        end if
    end if
end for
Return Output
```

is in the top layer cannot be considered to be the most secret one. However there is a relation in the sensing layer. In fact, entities in the sensing layer in the physical architecture will always be present at the bottom level of the logical architecture; hence they will be the least secret ones in the system. To illustrate this, let us take our previous hospital example, we denote our three physical layers 1, 2 and 3 respectively. The secrecy levels of logical architecture will be denoted *S1, S2* and *S3* with *S3* considered to be the most secret one. Table 7 gives a mapping between the positions of the entities in both the physical and logical architectures.

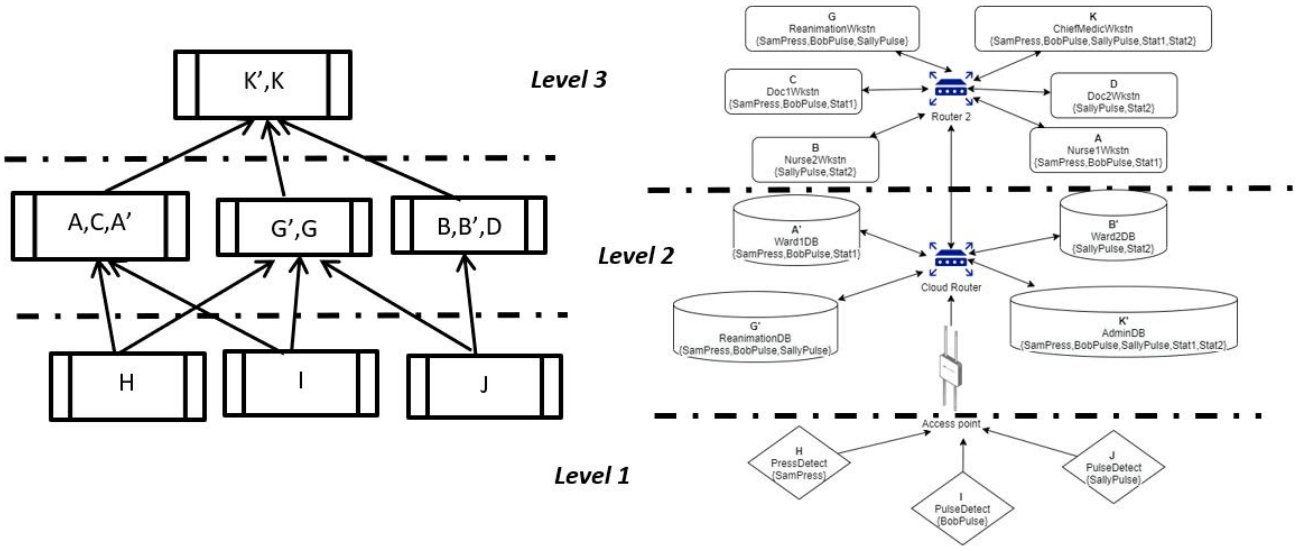


Figure 30: Levels on both physical and logical topologies

As we can see in the following table, except for the sensing layer, there is no correspondence between the physical layer and the secrecy level of the entities.

Entity	Secrecy level	Physical layer
A	S2	3
B	S2	3
C	S2	3
D	S2	3
G	S2	3
K	S3	3
I	S1	1
J	S1	1
H	S1	1
A'	S2	2
B'	S2	2
G'	S2	2
K'	S3	2

Table 7: Position of entities in both physical and logical topologies

8. Creating, removing and moving entities

In real contexts, networks are not static but dynamic. Transformations in networks can occur for many reasons, in many cases due to the intervention of the system administrator. Other

changes can occur by effect of policies. Particularly important are policies described in terms of time, for example in many systems there are policies that say that before or after a certain time, certain entities change their authorizations, or disappear altogether. We consider three types of transformations, for each type, we will show what are the changes that can occur in the implementation of the method. Meaning that we only concentrate on the changes that will occur on the physical topology. The changes in the logical topology and the partial order will be discussed briefly:

- Removing an entity:This will change the partial order, hence the labeling table, and also changes the physical configuration, since the removed entity will not be included in the new physical configuration. If the removed entity is associated with a storage entity and they form a component by themselves, the equivalent storage entity will also be removed from the physical configuration.

- Adding an entity:adding entities may lead to local or global changes in both the order relationships and the physical architecture. It can lead to adding databases alongside with adding new connections.

- Changing the level of an entity:this is equivalent to removing the entity and then adding it to new locations. This can lead to major changes in the order relationship; however, since there is no real connection between secrecy levels and positions in the physical architecture, the changes are barely noticeable on the physical architecture. Such changes are manly done on the labeling tables of the routers. The entity will still remain in its place the physical architecture.

In principle, any such operation may have global consequences and so after each transformation of the network, we would recalculate its new partial order, by using one of the mentioned efficient general-purpose algorithms for finding the partial order of components [76] . This will lead to recalculation of the labeling tables and the forward-ing tables. If appropriate, the size of the tables may be reduced by the application of algorithms such as transitive reduction of the network. Our discussion of transformations could end here. However, the following sections will help understand several cases that may not always require global recalculations. It should be considered that each network will have different update needs. For example, some networks may have very frequent label changes, but much less frequent additions or removal: in this case, the algorithms and data structures will have to be optimized for performing quick label changes,

and it may not matter if they perform less well for the other operations. Adding backward links in the label tables will help speed up certain searches, but will also increase the amount of memory required for the tables, so such decisions should be left to the de-signers of specific systems. There are in the literature many data structures and algorithms for optimizing the basic methods we propose below, and we leave this to further research.

8.1 Adding new entities

We can say that the importance of changes that the system will undergo depends on the type of the entities that we add to it. Some added entities will barely affect the system. We can differentiate three types of entities that can be added; sensors, workstation, and storage entities.

8.1.1 Adding a sensor If it is decided to add a new sensor to the system, this will lead to no changes in the physical configuration besides the appearance of this new entity. A line must be added in the labeling table for the new entity. Further, the label of the new entity must be compared with the labels of the entities already in the network, to add its IPAD to the Hold columns of the entities whose labels include the label of the new entity. The forwarding tables must be updated accordingly.

8.1.2 Adding a storage entity (Database): If it is decided to add a new storage entity into the cloud layer, this new entity will have to belong to one of the already existing equivalence classes, which already must have at least one storage entity (otherwise the new storage entity will be disconnected from all application entities). It is sufficient in this case to add a bidirectional channel between the new entity and any storage entity of its class, although in practice other connections may be called for.

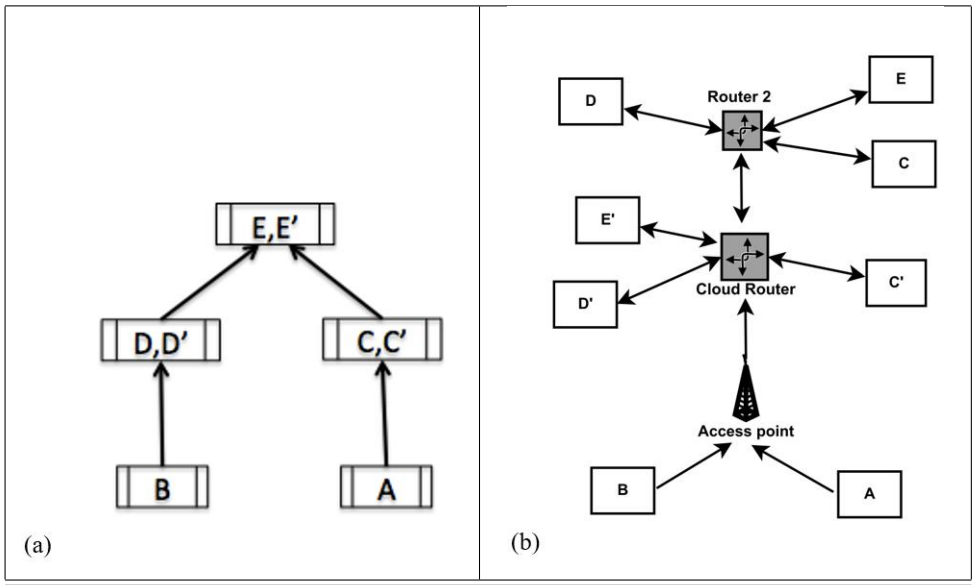
8.1.3 Adding a workstation: Adding a new application entity such as a workstation will be more challenging. With the centralized aspect of our system, all data must go through a storage entity before being acceded by a workstation. In this case, two main scenarios arise, according to whether the new entity will belong to an existing equivalence class or whether instead it will be in a new equivalence class.

A) The first case is easily treated. The new entity will access the same data entities as the other entities of its class. An example of this will be the following, let us suppose that, in our example, we want to add a new entity *SpecialistDocWkstn* named *L*, this specialist can be consulted by *doctor 2* in the case of an emergency, meaning that the entity *SpecialistDocWkstn* will access the data of *doc2Wkstn* through the existing storage entity named *Ward2DB*. The changes that both topologies will endure are the following: the new entity will be added to the application layer of the physical configuration, and it is sufficient to add a bidirectional channel between the new entity and any storage entity of its class, although in practice other connections may be called for.

B) The second case is the case of the addition of an application entity that creates a new equivalence class which is not isolated in the network. In this scenario, as we know, we must add a corresponding necessary storage entity with this new entity. The new application entity will be added to the application layer, while its corresponding storage entity will be added to the cloud layer. Both entities will have the same label, a new one. Their label must be compared with the labels of the existing entities. Data flows must be created:

- From the entities whose labels are included in the label of the new entities to the new entities
- From the new entities to entities whose labels include the labels of the new entities. The exact configuration of the channels to create these new flows can be determined either automatically or by intervention of an administrator. Corresponding updates must be done in the labeling and routing tables.

To illustrate this second scenario, we consider the following example. Let us say that we have the partial order presented in figure 30 (a), its physical topology is shown in figure 30 (b) and the labeling tables contained in the routers are shown in figure 30 (c). Note that entities *A* and *B* are considered as sensors, entities *D', C'* and *E'* are storage entities (databases) and the rest are workstations.



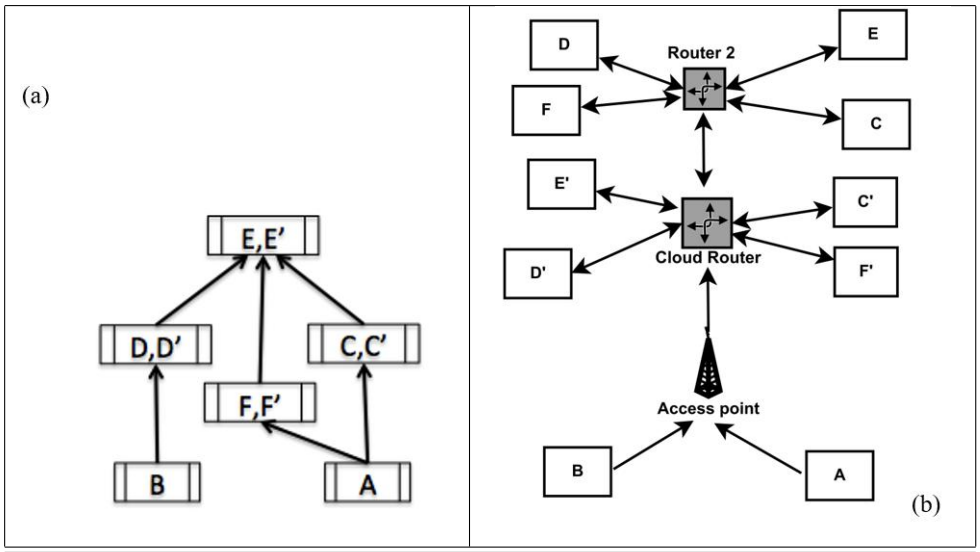
(c)

Sensor's Labeling table	
Entity	**Holds**
IPAD (A)	IPAD (A)
IPAD (B)	IPAD (B)
Labeling table of the cloud router	
Entity	**Holds**
IPAD (C')	IPAD (A), IPAD (C), IPAD (C')
IPAD (D')	IPAD (B), IPAD (D), IPAD (D')
IPAD (E')	All
Labeling table of the router 2	
Entity	**Holds**
IPAD (C)	IPAD (A), IPAD (C), IPAD (C')
IPAD (D)	IPAD (B), IPAD (D), IPAD (D')
IPAD (E)	All

Figure 31: (a). Partial order of the example. (b). Its corresponding physical architecture. (c). The labeling table of the two routers

Now, we add a new independent entity F into the application layer. As we said above, this will lead to the addition of a database associated with the new entity, we call this F' . This new entity will receive data form the sensor A. We can see in figure 31 , that this addition will also affect the labeling tables, some lines are added to the tables in order to include the new entities and the new connections.

On the partial order side, we see that the main change is the addition of a new equivalence class that contains the new entity and its corresponding database, however, no other changes have occurred in the rest of the existing equivalence classes. All the changes are highlighted in the figure 31.



(c)

Sensor's Labeling table	
Entity	**Holds**
IPAD (A)	IPAD (A)
IPAD (B)	IPAD (B)
Labeling table of the cloud router	
Entity	**Holds**
IPAD (C')	IPAD (A), IPAD (C), IPAD (C')
IPAD (D')	IPAD (B), IPAD (D), IPAD (D')
IPAD (F')	IPAD (A), IPAD (F), IPAD (F')
IPAD (E')	All
Labeling table of the router 2	
Entity	**Holds**
IPAD (C)	IPAD (A), IPAD (C), IPAD (C')
IPAD (D)	IPAD (B), IPAD (D), IPAD (D')
IPAD (F)	IPAD (A), IPAD (F), IPAD (F')
IPAD (E)	All

Figure 32: (a). The new partial order of the example. (b). its new corresponding physical architecture. (c). The new labeling table of the two routers

In order to summarize the entity addition updates, we present Algorithm 4:

Algorithm 4 Entity addition

Input: Initial physical topology
Output: Updated physical topology

1: **if** the entity E to add is a sensor **then** (1)
2: add entity E
3: add connection between entity E and the Access point
4: add $IPAD(E)$ into the labeling tables of all the routers for the entities that E can flow to according to labels
5: **else**
6: **if** E is a storage entity (database) **then** (2)
7: add entity E
8: add connection between entity E and the cloud router
9: add a line in the cloud routers labeling table that contains the new entity E and the entities that it can hold according to labels
10: add $IPAD(E)$ into the labeling table of router R2 for the entities that E can flow to according to labels
11: **else**
12: **if** the entity E to add is a workstation **then** (3)
13: **if** the entity E has an equivalent database **then**
14: add entity E
15: add connection between entity E and router 2
16: add a line in the router 2 labeling table that contain the new entity E and the entities that it can holds according to labels
17: add IP address of the new entity E into the labeling table of cloud router for the authorized entities lines only
18: **else**
19: add entity E
20: Repeat step (2) and create a new equivalent database
21: Repeat step (3)
22: **end if**
23: **end if**
24: **end if**
25: **end if**
26: Return Output

Our solution is to keep the labeling tables sorted by label length and to search them by using a binary search by length. The initial labeling table can be constructed bottom up, leading to the desired sorting. To find if there is an already existent equivalence class where the new entity should be added, we need to compare the label of the new entity with the elements in the table. To do so, we can use a binary search algorithm since the table is sorted. Binary search runs in logarithmic time in the worst case giving it a O (log n) time complexity, where n is the number of elements in the table. If an exact match is found, the new entity will be added to the already

existent equivalence class. Otherwise, a new class is created that will contain the new application entity and a new storage entity that will also be created. Creation of an entity requires adding the new entity to the labeling table. To keep the table sorted, the added entity must be added in a specific position in the interval of elements that have the same label length. This can be done by shifting one position down all the elements present after the intended insertion position. In general, if we have n elements, we need to shift all the n elements. This gives a worst-case time complexity of O(n), where n is the number of elements to shift in the table. To conclude, the overall time complexity of the addition of an application entity will be O (log n) +O(n)= O(n) which give us a linear time complexity. So, these well-known efficient algorithms can be used in order to implement our entity addition algorithm.

8.2 Entity removal

Another update that can happen in our system is removal an entity. This will cause changes in the physical configuration. The changes will depend on the type of the entity that we intend to remove, as in the case of addition of entities scenario. We have three potential cases: removing a sensor, removing a storage entity, and removing a workstation.

8.2.1 Removing a sensor If a sensor needs to be removed, the only change that needs to occur is the removal of all the entries of the removed sensors in the labeling tables. The resulting network configuration will be the same as the initial one except for the absence of the removed sensor.

8.2.2 Removing a storage entity If we want to remove a database or a storage entity, things may be different. Since we deal with a centralized system where each storage entity has an equivalent entity in the application layer, we cannot just remove a storage entity regardless of the centralized character of our system. If we want to remove a storage entity, we must be sure that the equivalent entity has another equivalent database that shares the same information with it. That means that in the partial order, the equivalence class that contains the storage entity that we intend to remove and its corresponding application layer entity, must contain at least another storage entity. If it is the case, we can remove the storage entity, redirect the

workstation into the second storage entity of the equivalence class, and remove all the entries of the removed entities from the labeling table. If it is not the case, and we only have one storage entity in the equivalence class, we cannot remove the storage entity since this will alter the centralized aspect of our system.

8.2.3 Removing a workstation The last scenario, is removing an entity from the application layer. In this scenario, we also need to check the equivalence classes of the partial order. We have two cases, if the equivalence class that contains the entity to remove has another application layer entity in it, we only remove the intended entity and we leave the corresponding storage entity to be used by other entities. If it is not the case, we remove the intended entity and the corresponding storage entity since there is no need for it. On the physical layer, we can have either one removed entity or two removed entities depending on the equivalence classes' configuration. For any removed entities, we always remove their entries from all the labeling tables.

In order to summarize the entity removal updates, we present Algorithm 5:

Algorithm 5 Entity removal

Input: Initial physical topology
Output: Updated physical topology

```
1: if the entity E to add is a sensor then
2:     remove E
3:     remove all the entries of the entity E from the labeling tables
4: else
5:     if the entity E to remove is a database (storage entity) then
6:         remove entity E
7:         remove all the entries of the entity E from the labeling tables
8:     else
9:         Entity impossible to remove
10:    end if
11:    if the entity E to remove is a workstation then
12:        if the equivalence class of the entity E contain at least on other application layer
    entity then
13:            remove entity E
14:            remove all the entries of the entity E from the labeling tables
15:        else
16:            remove entity E
17:            remove corresponding storage entity
18:            remove all the entries of both entities from the labeling tables
19:        end if
20:    end if
21: end if
22: Return Output
```

Deleting an entity means that we need to remove this entry and all the occurrences of the label name from the labeling table . We have to search for these occurrences element by element and this gives us a time complexity of O(n) in the worst case, where n is the number of elements in the table. Once the entity and its entries deleted, we need to shift back the table to keep it sorted. This time the shift will be up. This can be done in an O(n) time complexity in the worst case. To conclude, the overall time complexity of the removal of an entity will be O (n) +O(n)= O(n) which is also the same linear time complexity, and these well-known efficient algorithms can be used here as well.

8.3 Label changes

Another transformation that must be taken into consideration is the change of an entity's label. This may be done by an administrator or by the effect of a policy in order to create or remove data transfer channels or increase or decrease the secrecy or integrity of an entity, this change

leads to the change of the position of the entity in the partial order. Usually, the change is from a certain secrecy level into a higher secrecy level. We do not consider the opposite change from a given secrecy level into a lower one, because this can lead to a secrecy breach, since this entity may have already accessed information classified higher than its new secrecy level.

This transformation can be considered equivalent to removal followed by an addition, so the previous considerations apply. As in the case of removal, it has to be checked whether the storage entity or entities connected to it are still needed. As in the case of addition, it has to be checked whether the new entity will make an equivalence class by itself or whether it will join an existing equivalence class.

When we talk about updating an entity, we refer principally to application layer entities (workstations). However, there can be cases where we must take into consideration storage entities as well. For example, if we want to update a workstation from a certain equivalence class into a higher-level class, we must check if this higher level already contains a storage entity. In this case, we do not need to update the equivalent storage entity to the workstation; we will only add the workstation into the higher-level class.

The entries of the updated entity will be added into the labeling tables on the lines of the entities included in the new higher-level class, which guarantees message forwarding between all the components of the class. In our model, storage entities are used simply to contain the data of their equivalence classes and appear or disappear according to this need. The cloud has mechanisms to do this. However for practical reasons it may be necessary to move a storage entity to a higher-level class. This can be done by a label change. However, it must be assured that at least one storage entity remains in the previous equivalence class.
In order to summarize the entity updates, we present the following algorithm:

Algorithm 6 Entity update

Input: Initial physical topology
Output: Updated physical topology

if the entity E to update is an application layer entity (workstation) from class A1 into A2 **then**
    if the new equivalence class A2 of E has a storage entity **then**
        add the entries of E into the labeling tables in the lines of all the entities of its new equivalence class A2
    else
        add the entries of E and its corresponding storage entity into the labeling tables in the lines of all the entities of its new equivalence class A2.
    end if
else
    if the entity E to update is a storage entity from class A1 into A2 **then**
        if there is a remaining storage entity in the previous equivalence class A1 **then**
            add the entries of E into the labeling tables in the lines of all the entities of its new equivalence class A2.
        else
            create a new storage entity F
            add the entries of F into the labeling tables in the lines of all the entities of the class A1
            add the entries of E into the labeling tables in the lines of all the entities of its new equivalence class A2
        end if
    end if
end if
Return Output

9. Multiple flows scenario

In the previous study case and implementation, we only considered the existence of a single data flow in the network. This data flow can be used to construct a partial order model to achieve secrecy and privacy requirements. However, in some cases, separate data flows can be present in a network. Each one of these flows will have different secrecy requirements and will need to be controlled separately, hence it will have its own partial order model. In Chapter 6 we have shown an e-commerce example where we have two data flows, one to carry orders and another in the opposite direction to carry billing data. Taking in consideration this context,we add in our case study a second downward flow that we call *Diagnostic*. Figure 33 shows the configuration of this new flow. Note that in this figure, the direction of the data flow changes and becomes from top to bottom.Hence, the secrecy levels of entities will also change, for example entity *ChiefMedicWkstn* will become the least secret entity in the system.

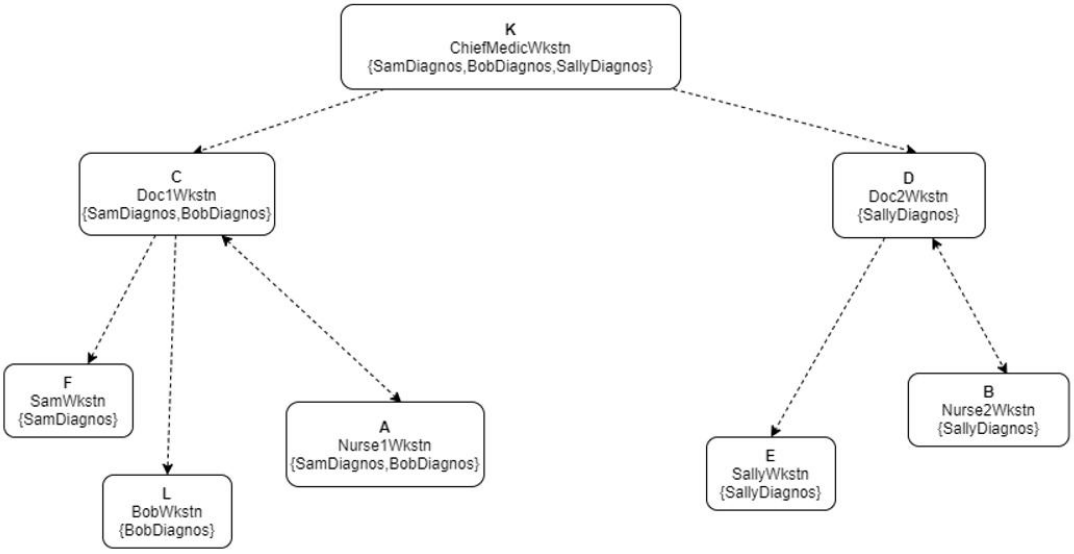


Figure 33: The configuration of the *Diagnostic* flow

Just like the previous example, we deal with a centralized architecture. Hence, storage entities must be added to the configuration. This will result in configuration of Fig. 34.

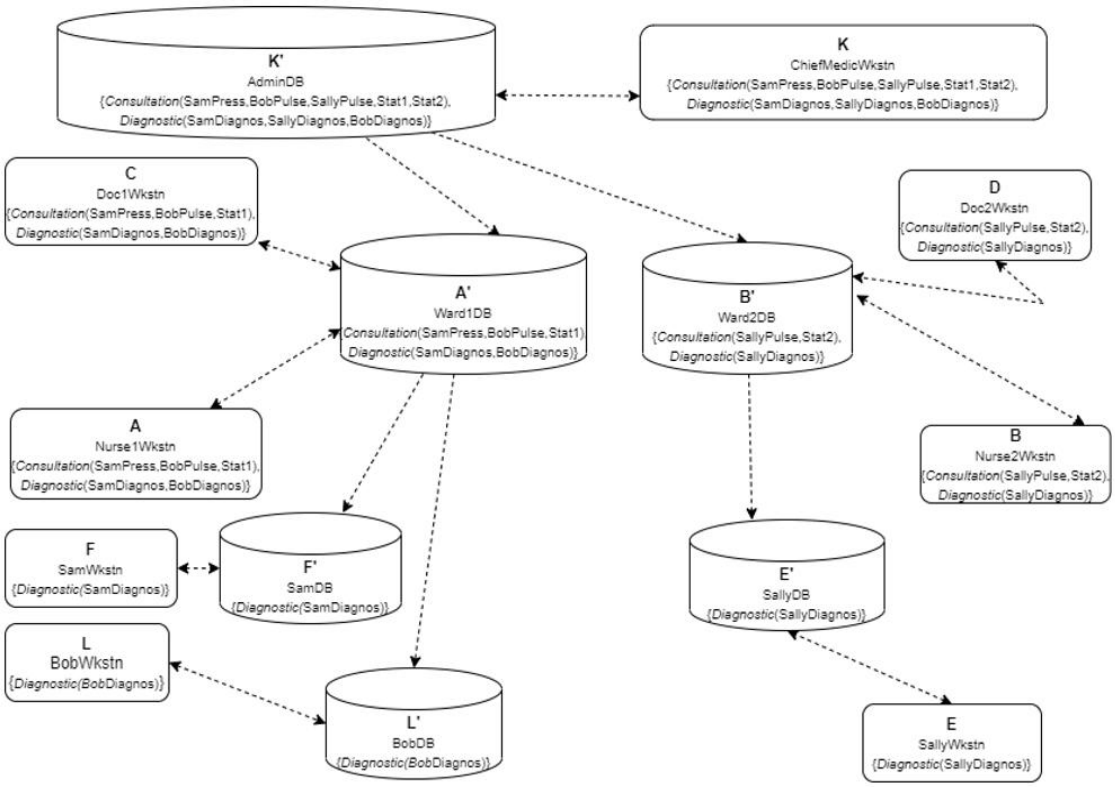


Figure 34: The final configuration of the *Diagnostic* flow

We go back to our cases study and we add this *Diagnostic* flow alongside the previously defined one that we call *Consultation* . We say that the previous case study deals with consultation information where data flow from patients towards the medical staff as we have seen. We add to this Diagnostic information that travels in the opposite direction and has its own requirements in terms of secrecy, which leads to a different partial order.

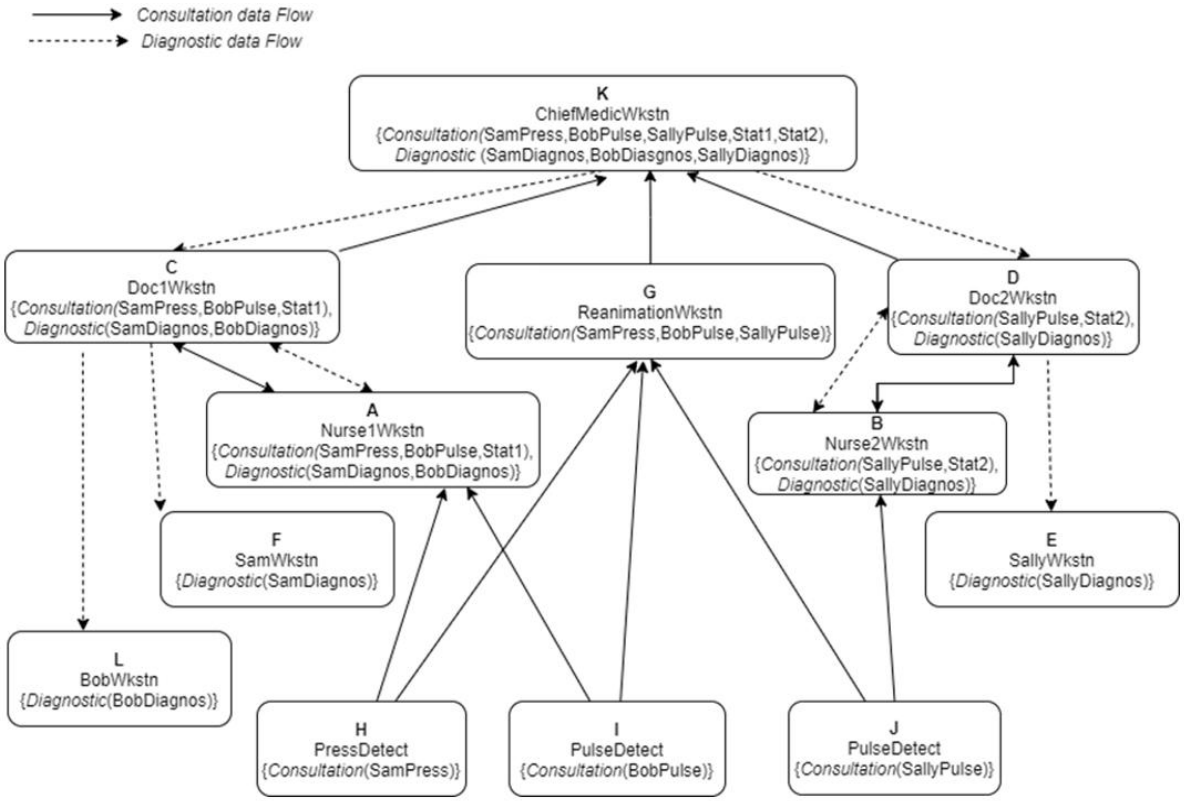


Figure 35: Two-flow network for the hospital example

If we compare this scenario to the first one, we see that there are some new added entites *BobWkstn, SamWkstn,* and *SallyWkstn* respectively *L, F*, and *E* which represent the patient applications that will allow them to consult the diagnostic data flow. So, for secrecy or privacy requirements, each patient must only consult their own disagnostic data that they will get from their treating medical staff.

To keep the two flows separate, we should identify the label sets that are relevant for each flow, so many entities will have two labels. For example, the label of *ChiefMedicWkstn* is as follows: *{Consulta-tion(SamPress,BobPulse,SallyPulse,Stat1,Stat2),Diagnostic(SamDiagnos,SallyDiagnos,BobDiagnos)}*. This means that *ChiefMedicWkstn* participates in two flows, one for Consultaion and one for Diagnostic, and that for each flow, *ChiefMedicWkstn* has access to data of the corresponding labels.

This example also shows the necessity of having trusted entities that can access different

data but are trusted to deliver the right data to the rightful subjects only. This can be the case for example with *Doc1Wkstn* that in the Diagnostic flow can access both *Bob* and *Sam* data labeled respectively *BobDisagnos* and *SamDiagnos*. However, this entity is trusted to deliver *Bob*'s data to *BobWkstn* and *Sam*'s data to *SamWkstn*. This can be the case also for the entities *ChiefMedicWkstn, Doc2Wkstn, Nurse1Wkstn*, and *Nurse2Wkstn*.

In order to implement this model, we need again to create a centralized system where all the data is saved in the cloud before being accessed by the users. For this purpose, we add storage entities to the newly created entities for the patients. The storage entities for the patients can represent for example small storage spaces allocated through the patient's account created during the registration on the hospital servers.

Figure 36 represents the logical topology for this network. In this figure, we have two sets of labels, one set for each flow. Then the labels to be used to construct labeling tables and forwarding tables after that depend on the type of flow.

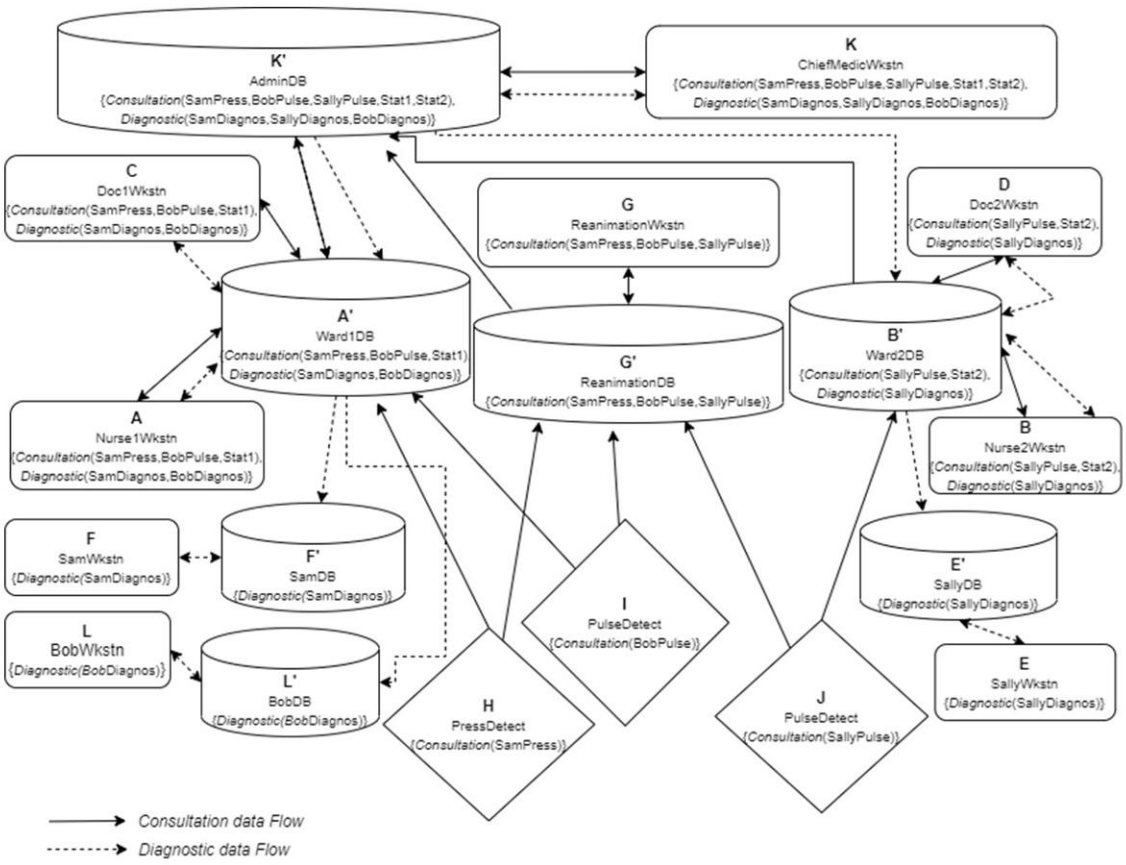


Figure 36: Logical topology for the second network

For the secrecy requirements of the *Diagnostic* flow, we obtain the following partial order.

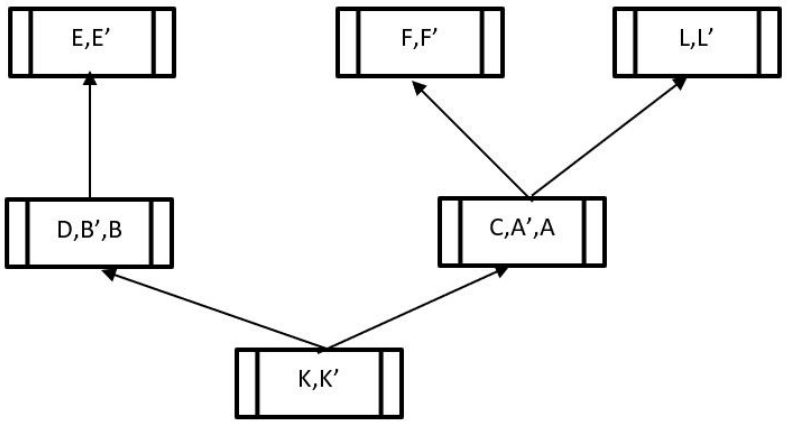


Figure 37: Partial order for the *Diagnostic* data flow

To implement this scenario of two separate flows in one network, we need to modify the physical topology that will be used for the implementation. The new topology is presented in figure 37.

In this topology, as the one seen earlier, we have two routers that interconnect our network entities, one to interconnect our storage entities and one to connect our workstation. The new added entities for this network (patients' workstations and their respective storage entities) will be connected to these two routers.

The main difference, in terms of labeling tables, is that each router will have two labeling tables, one for each flow. Depending on the chosen type of data flow, the controller will use one of the tables to create a different forwarding tables for different flows. Hence, the controller will have a forwarding table for each flow.

In the case of *Consultation* data flow, the labelling table for the two routers will be the same as the one for the earlier network. However, for the *Diagnostic* flow the tables will be as follows:

Entity	Holds
IPAD (K')	IPAD (K'), IPAD (K)
IPAD (A')	IPAD (K), IPAD (A), IPAD (C),IPAD (A')
IPAD (B')	IPAD (K), IPAD (B), IPAD (D),IPAD (B')
IPAD (E')	IPAD (K), IPAD (D), IPAD (B), IPAD (E),IPAD (E')
IPAD (F')	IPAD (K), IPAD (C), IPAD (A), IPAD (F).IPAD (F')
IPAD (L')	IPAD (K), IPAD (C), IPAD (A), IPAD (L),IPAD (L')

Table 8: Labeling table for the cloud router in the case of *Diagnostic* flow

Entity	Holds
IPAD (K)	IPAD (K'),IPAD (K)
IPAD(C)	IPAD (K), IPAD (A),IPAD(C)
IPAD (A)	IPAD (K), IPAD (C), IPAD (A'),IPAD(A)
IPAD (D)	IPAD (K), IPAD (B), IPAD (B'), IPAD(D)
IPAD(B)	IPAD (K), IPAD (D), IPAD (B'), IPAD(B)
IPAD (E)	IPAD (K), IPAD (D), IPAD (B), IPAD (E'), IPAD(E)
IPAD (F)	IPAD (K), IPAD (C), IPAD (A), IPAD (F'), IPAD(F)
IPAD (L)	IPAD (K), IPAD (C), IPAD (A), IPAD (L'), IPAD(L)

Table 9: Labeling table for router 2 in the case of *Diagnostic* flow

During the implementation and development of our controller, we then need to tag the data-flows that we have in the network. As we said above, for each flow, we will have a set of labels associated with it. Depending on the type of flow determined by the policies, the controller will be producing a forwarding table and this table will be used by the routers to monitor the flow in the network in order to meet our secrecy requirements and our partial order model.

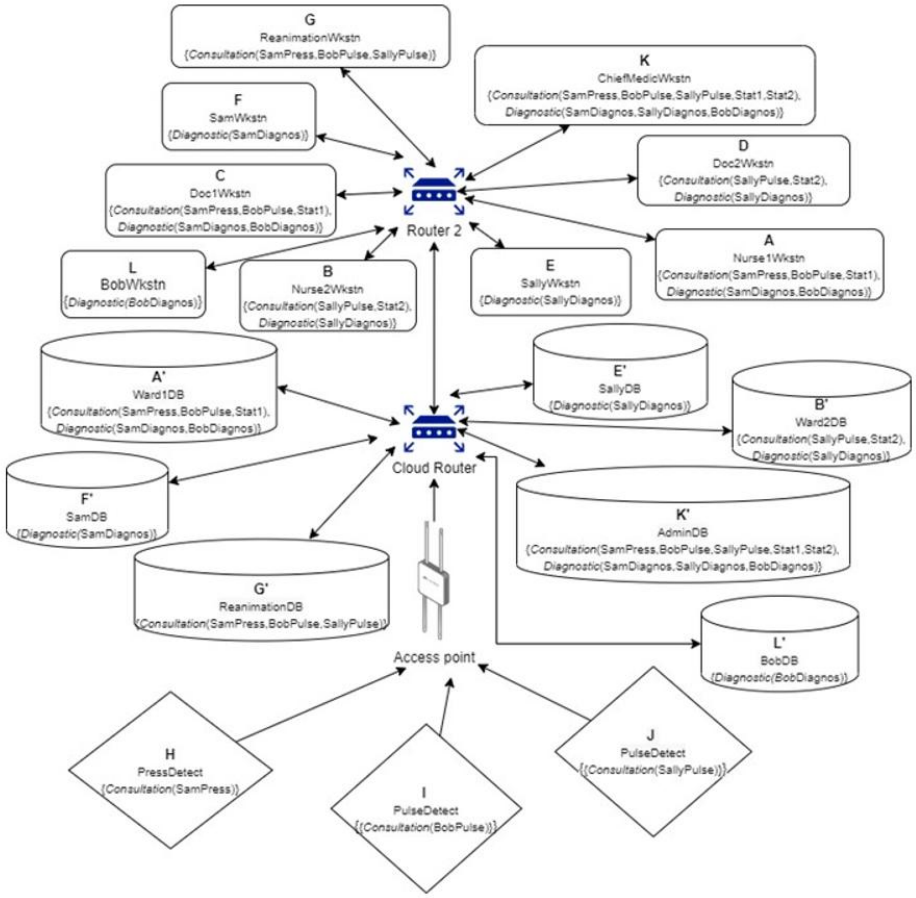


Figure 38: Physical topology for the second scenario

10. Simulation and implementation of the controller

In this section, we present our development of an SDN controller that implements the partial order model using the physical architecture described in the previous section. The simulations that will be done in this section aim only to test the privacy and secrecy requirements, meaning that we will only test if data flow will arrive to authorized entities. We do not take in consideration any other parameters. Parameters such as performances of the controller, scalability... etc. are not specific to our research and have been tested in other research.

10.1 Simulation tools and environment

In order to simulate our SDN network and before the creation of the SDN controller, we need to define our simulation tools. A variety of simulation tools are available for SDN. The most used are Mininet [3], Estinet [26,1] and ns3 [99,4]. The first two simulators were developed especially for SDN use. The last one is a general network simulator that was adapted to simulate SDN networks.

Since we are using OpenFlow as the most significant part of our implementation,we decided to use Mininet as simulation tool for our work. OpenFlow is the protocol that will be in charge of communication between our controller and the forwarding devices that we have in our network (cloud router and Router 2).

Mininet is a network emulator that runs a collection of end hosts, switches, routers, and links on a single Linux kernel. It uses virtualization to make the system look like a complete network, running the same kernel, system, and user code. Mininet hosts behave just like real machines that can run arbitrary programs. The programs can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like real Ethernet switches or routers as in our case. Mininet's virtual component (hosts, switches, links, and controllers) are created using software rather than hardware, and for the most part their behaviour is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

A Mininet network consists of the following components:

1- Isolated Hosts: An emulated host in Mininet is a group of user-level processes. Hosts provide the network with exclusive ownership of interfaces, ports, and routing tables. For example, two web servers in the network can coexist, both listening to private eth0 interfaces on port 80.

2- Emulated Links: The data rate of each link is enforced by Linux Traffic Control (tc), which has a number of packet schedulers to shape traffic to a configured rate. Each emulated host has its own virtual Ethernet interface(s) (created and installed with ip link add/set). A virtual Ethernet (or veth) pair, acts like a wire connecting two virtual interfaces, or virtual

switch ports; packets sent through one interface are delivered to the other, and each interface appears as a fully functional Ethernet port to all system and application software.

3- Emulated Switches: Mininet typically uses the default Linux Bridge or Open vSwitch running in kernel mode to switch packets across interfaces. Switches and routers can run in the kernel (for speed) or in user space (so we can modify them easily).

The reason of our choice of Mininet is that this simulator is specially designed to run OpenFlow networks. All packages and tools needed for such simulation are available. Also this simulator has many other advantages that made it suitable for our use:

1- Mininet is fast and easy to use : starting up a network can be done easily using Python scripts. This means that your run-edit-debug loop can be very quick.

2- You can create custom topologies: a single switch, larger Internet-like topologies, IoT networks.

3- You can run real programs: anything that runs on Linux is available for you to run, from web servers to TCP window monitoring tools to Wireshark.

4- You can customize packet forwarding: Mininet's switches are programmable using the OpenFlow protocol by setting a controller that will be responsible for policy enforcement.

5- You can run Mininet on your laptop, on a server, in a VM, on a native Linux box. There is no specific hardware needed to run it. And you can share and replicate results: anyone with a computer can run your code once you've packaged it up.

6- Mininet is an open-source project so it is available for use at any given time.

7- Mininet is under active development. Behind it there is a developer commu-nity that can try to explain it, fix it, or help user fix it.

Finally, Keti et al. [64] and Rogerio et al. [100] conducted tests to study Mininet limitations related to the simulation environment, resource capabilities, and to evaluate the scalability of Mininet in terms of creating topologies with varying number of nodes and different environment scenarios. The studies concluded that Mininet can be utilized as one of the powerful tools in emulating the SDN and virtual networks.

Once we choose and set up our simulation environment, we move to the second phase of our implementation which consists of the development of a controller that will be in charge of enforcing our partial order flow policy.

In the context of SDN, many open-source controllers are available to use. The most common ones are POX, Ryu, FloodLight and OpenDaylight. Each of these controllers has its own features. SDN controllers features may include [87] :

1- Programming language: SDN Controllers can be developed using programming language such as C/C++, Python, and Java. The language used to develop the controllers have an impact on the controller's performance. By performance we mean the capacity to run cross-platforms, allowing fast memory access and good memory management, and finally the ease of learning the programming language.

2- OpenFlow support: since OpenFlow is the key communication protocol in the southbound API (between data plane and control plane), the controller must be able to implement perfectly OpenFlow and especially the newer version of it.

3- Network programmability: the controller support of network programmability relies on its degree of integration of a wide number of northbound interfaces, a good user interface and a command-line interface.

4- Efficiency and performance: these criteria are used to refer to different parameters such as performance, scalability, reliability, and security.

5- Southbound and northbound API: describe the versions of both bound APIs supported by the controllers. For an optimal performance, the controller must support the newest versions of protocols in both bounds.

6- Partnership: this parameter highlights the participation of reputable organizations in the development of controllers. Organizations such as : Cisco , IBM ... are active contributors in the SDN controllers market. An SDN controller with a good partnership will have chances to be maintained for a long time.

Figure 39 shows a comparison between different available controllers according to the features described above and done by Ola et al.[87].

	Program-Ming Language	GUI	Docum-Entation	Modularity	Distributed/ Centralized	Platform Support	Productivity	Southbound Apis	Northbound Apis	Partener	Multithreading Support	Opemstack Support
ONOS	Java	Web Based	Good	High	D	Linux,MAC OS, And Windows	Fair	OF1.0, 1.3, NETCONF	REST API	ON.LAB, At&T, Ciena,Cisco, Ericsson,Fujitsu, Huawei,Intel, Nec,Nsf.Ntt Comunnication, Sk Telecom	Y	N
Open-Day-Light	Java	Web Based	Very Good	High	D	Linux,MAC OS, And Windows	Fair	OF1.0, 1.3, 1.4, NET-CONF/ YANG, OVSDB, PCEP, BGP/LS, LISP, SNMP	REST API	Linux Foundation With Memberships Covering Over 40 Companies, Such As Cisco, IBM, NEC	Y	Y
NOX	C++	Python + QT4	Poor	Low	C	Most Supported On Linux	Fair	OF 1.0	REST API	Nicira	NOX_MT	N
POX	Python	Python + QT4	Poor	Low	C	Linux,MAC OS, And Windows	High	OF 1.0	REST API	Nicira	N	N
RYU	Python	Yes	Fair	Fair	C	Most Supported On Linux	High	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OF-CONFIG	REST For South bound	Nippo Telegraph And Telephone Corporation	Y	Y
Beacon	Java	Web Based	Fair	Fair	C	Linux,MAC OS, And Windows	Fair	OF 1.0	REST API	Standford University	Y	N
Maestro	Java	-	Poor	Fair	C	Linux,MAC OS, And Windows	Fair	OF 1.0	REST API	RICE, NSF	Y	N
Flood-Light	Java	Web/ Java Based	Good	Fair	C	Linux,MAC OS, And Windows	Fair	OF 1.0 , 1.3	REST API	Big Switch Networks	Y	N
Iris	Java	Web Based	Fair	Fair	C	Linux,MAC OS, And Windows	Fair	OF 1.0, 1.3, OVSDB	REST API	ETRI	Y	N
MUL	C	Web Based	Fair	Fair	C	Most Supported On Linux	Fair	OF 1.4, 1.3, 1.0, OVSDB, OF-CONFIG	REST API	Kulcloud	Y	Y
Runos	C++	Web Based	Fair	Fair	D	Most Supported On Linux	Fair	OF 1.3	REST API	ARCCN	Y	N
Lib-Fluid	C++	-	Fair	Fair	-	Most Supported On Linux	Fair	OF 1.0, 1.3	-	ONF	Y	N

Figure 39: Comparison between SDN controllers [87]

Khondoker et al. [66], Shalimov et al. [112], and Shah et al. [111] compared SDN controllers according to different criteria, however, we believe that the study presented above is more general and can be used to determine the appropriate controller to use.

We have then chosen to develop a Ryu controller [68]. A Ryu Controller is an open, python-based Software-Defined Networking (SDN) controller. It is designed to increase the agility of the network by making it easy to manage and adapt how traffic is handled. Ryu provides software components with well-defined APIs that make it easy for developers to create new network management and control applications. Our choice is motivated by the fact that Ryu is the most suitable controller for use in Mininet environment since it supports OpenFlow 1.0, 1.2, 1.3, 1.4. Another reason is that,because of the fact that is is Python-based, it is easier in Ryu to develop new network management and control applications in comparison with other controllers. Finally, Saleh et al. [103] and Islam et al. [58] have reported on testing the performance of the Ryu controller in many simulation scenarios and have concluded that the controller is very suitable for prototyping, experimentation, research and demonstrations.

Fig. 39 represents the Ryu controller architecture.

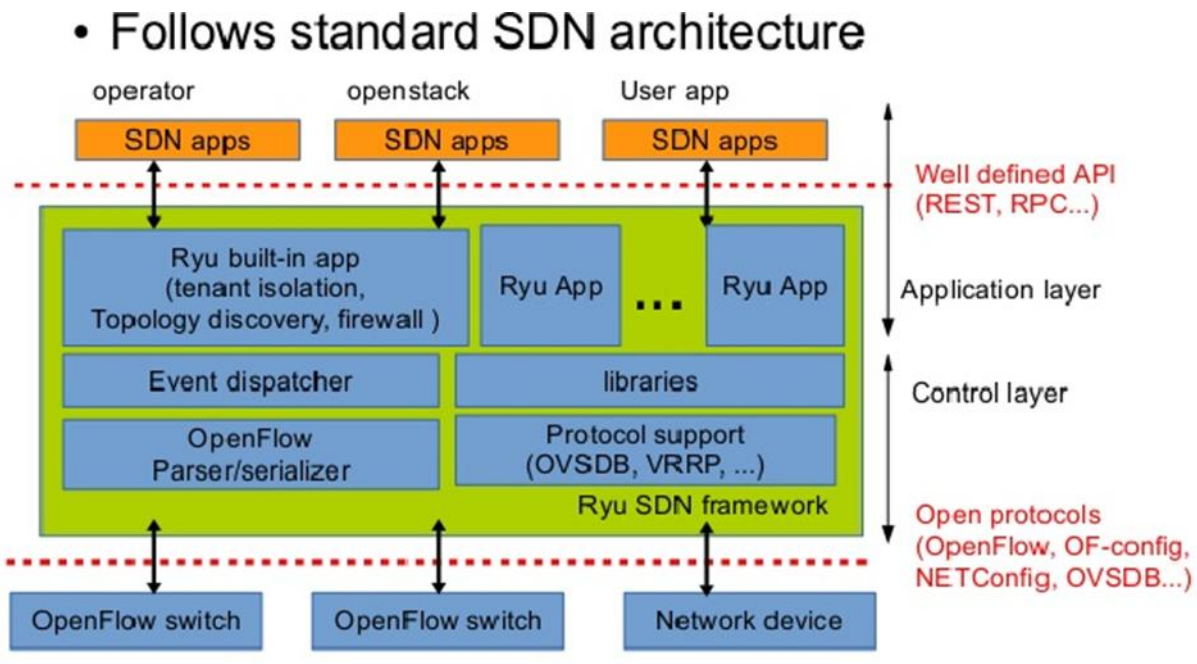


Figure 40: Ryu controller architecture [111]

10.2 Simulation of our case study

We have done the simulation of our case study by using the simulation tools presented in the previous section. We have implemented the architecture presented in Fig. 27 in a Mininet simulator, then we have developed a Ryu controller with Python to control the data flow in the network for secrecy and privacy requirements according to our partial order model.

To create our topology, we have used the Python API to write a topology Python script. First, we had to create an empty network and add nodes or entities into it. To create this empty network, we manually create a default controller called *Controller c0*. This default controller will be replaced later on with our Ryu controller. The creation of the empty network and the controller c0 can be done as follows:

```
def myNetwork():
net = Mininet( topo=None, build=False, ipBase='10.0.0.0/8')
info( '*** Adding controller\n' )
c0=net.addController(name='c0', controller=RemoteController, ip='127.0.0.1',
    protocol='tcp', port=6653)
```

Once the empty network is created, we add our switches and hosts using *net.addSwitch()* and *net.addHost()* classes. For each added entity, we need to specify the host information such as: host name, ip address and default route.

```
s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
A = net.addHost('A', cls=Host, ip='10.0.0.1', defaultRoute=None)
```

In our architecture, we use routers. However, in SDN context using OpenFlow protocol we do not care about routers and switches we only refer to OpenFlow switches. The reason is that Openflow works by updating entries to the forwarding table in the router or switch. Therefore, it is not a routing or switching protocol at all. OpenFlow is about forwarding. The forwarding table is defined by the controller through flow rules that can be defined by the user and is what all routers and switches use to dispatch frames and packets to their output ports.

Once all the switches and hosts had been created, we only had to create the links between them. This was done with the class *net.addLink()*. Our network and controller could then be

started .

```
info( '*** Starting network\n') net.build() info( '*** Starting controllers\n')
for controller in net.controllers: controller.start()
info( '*** Starting switches\n')
net.get('s1').start([c0])
net.get('s2').start([c0])
info( '*** Post configure switches and hosts\n')
CLI(net) net.stop()
if __name__ == '__main__': setLogLevel( 'info') myNetwork()
```

The following figure generated by Mininet shows our topology created into the Mininet environment, it can be seen that it corresponds to Fig 28.

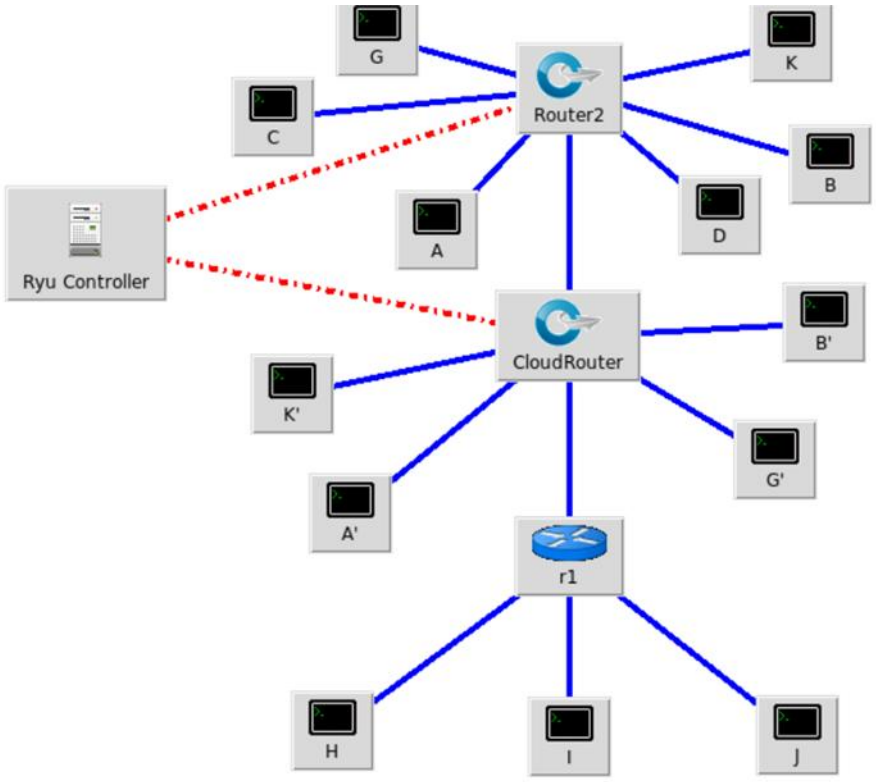


Figure 41: Simulated architecture generated by Mininet

The following table recapitulates the meaning of the hostnames presented in the topology.

Host ID	Definition
A	Nurse1wkstn
B	Nurse2Wkstn
C	Doc1Wkstn
D	Doc2Wkstn
J	PulseDetect (Sally)
H	PressDetect (Sam)
I	PulseDetect (Bob)
G	ReanimationWkstn
K	ChiefMedicWkstn
A'	Ward1DB
B'	Ward2DB
G'	ReanimationDB
K'	AdminDB

Table 10: Entities correpondance in the simulation

Once the topology established, we move to the second phase, the development of our Ryu controller. For that, a second Python script is written. This new script will implement into the controller the labeling table we described in section 4 alongside the data flow rule. The Ryu controller will then generate a forwarding table that will enforce the partial order model of Fig. 27. Only the authorized flows in the partial order will be allowed.

Once the implementation done, we will simulate the flow in the network, we are only interested in the data flow and reachability. We will use the *ping* command to see if the result matches the secrecy requirements or our partial order. Fig.42 shows the results that we obtained during the simulation.

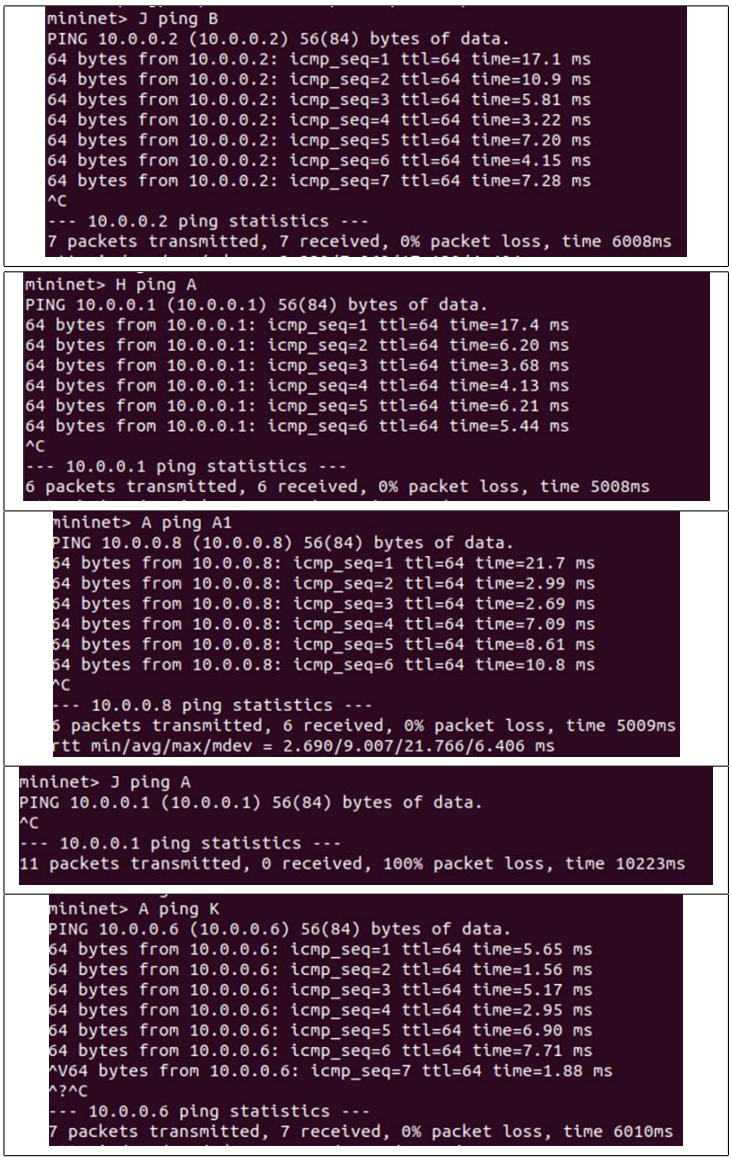


Figure 42: Some simulation results of our implementation

In Fig.42, we can see some results for different scenarios. We have some connection established between entities in the case of authorized data flow (J to B, H to A, A to A', A to K), and some unauthorized flows where there is no connection (J to A). All the results are in concordance with the partial order and flow rules.

In the case of changes in the network (addition of entities, removal of entities, label update, see Sect. 7), their implementation will be done by following the methods described in the

different algorithm of section 7. We need to add the labels to the labeling tables in the case of entity addition. In the case of removal, we only remove the entity from the topology, and remove all it entries on the label table. This will generate a new forwarding table that will meet the new secrecy requirements of the new partial order.

10.3 Simulation of multiple flows case study

For the multiple flows case study, we deal with two flows *Diagnostic* and *Consultation*, meaning that we need to have two different labeling tables, one for each flow. The simulated architecture is shown in figure 37. Before the simulation, we must choose which flow we want to simulate. If we choose the Consultation flow, the results will be the same as those we have seen earlier. However, in the case of Diagnostic flow we will be dealing with a different partial order and different secrecy requirements. Following are the results of the simulation of the Diagnostic flow. Once more, we are only interested in simulating the data flow of information and reachability. Fig.43 shows some results..

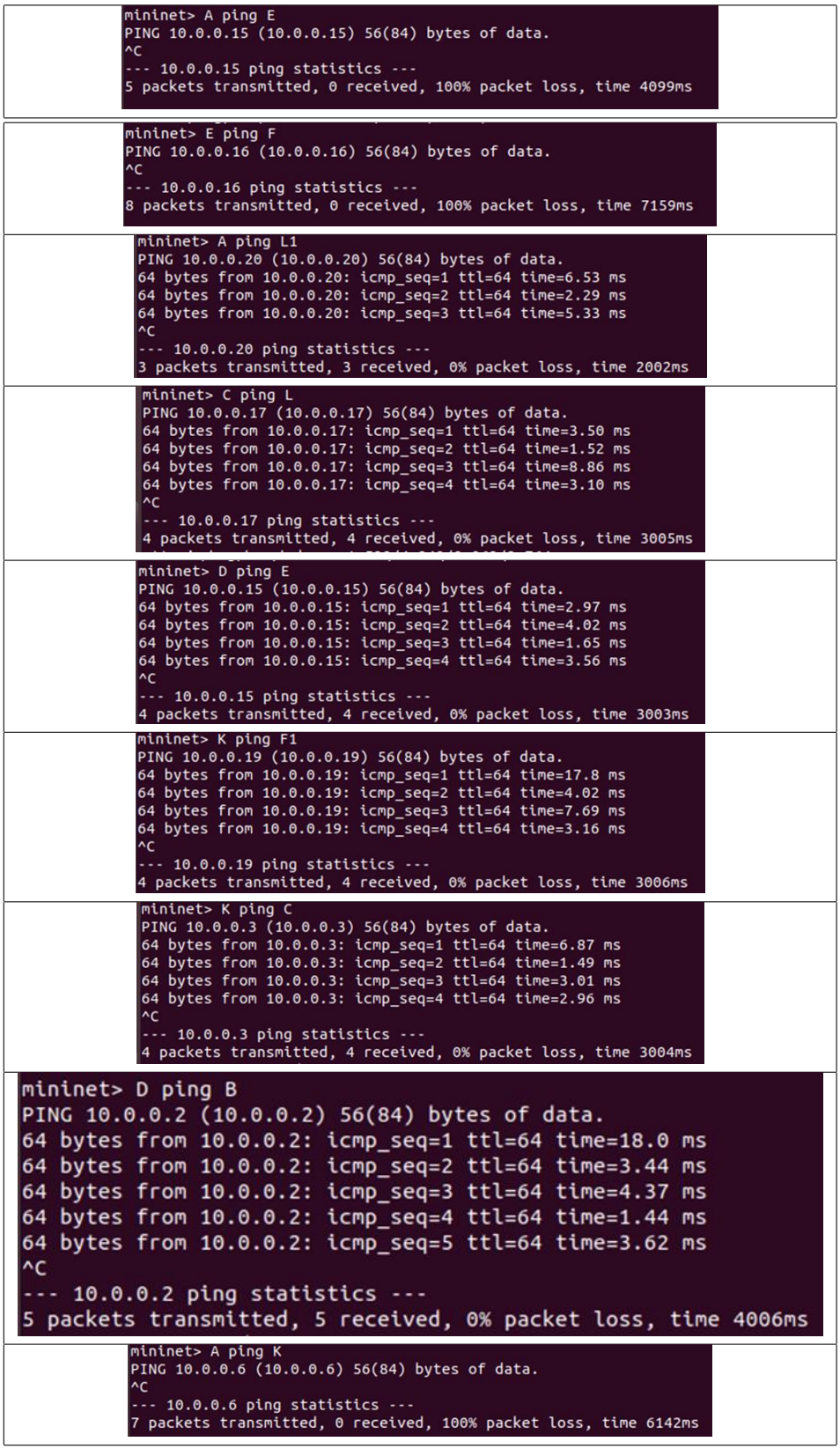


Figure 43: Some simulation results of the two flows scenario

The results show a respect of the partial orders and flow rules for all flows. We have some authorized flows such as : *A* to *L'*, *C* to *L*, *D* to *E*, *K* to *F'*, *K* to *C*, *D* to *B*. We can also see the cases of unauthorized flows such as flows : *A* to *E*, *E* to *F*, and *A* to *K* . The reason for the test refusal in the case of unauthorized flow is due to the fact that these flows do not satisfy partial order relations.Which means that the concerned entities do not respect the label inclusion relation.

In the tests that we have done in both cases, we have tried to cover different situations that can occur in our system such as:

- Data flow between entities in the same equivalence class

- Data flow between two entites in different equivalence classes where flow between them is authorized according to partial order model policies.

- Data flow between two entites in different equivalence classes where flow between them is frobidden according to partial order model policies.

We realize that the testing that we have performed is far from constituting a systematic testing procedure, and we leave this as a topic for future research.

11. Conclusion

In this chapter, we present an implementation method to our partial order model in the context of the Internet of things. The implementation uses Software-Defined Networks; an SDN controller is developed in order to control data flow in the network using the flow rules of our model. In order to control the flows, the controller will use the labeling tables that we implement into it and will construct forwarding tables that will be transferred into the routers through the protocol OpenFlow. This can be done in the two cases of single or multiple networks that we can distinguish by using labels.

Simulation results of reachability show a perfect match between the controller decisions in allowing or forbidding flows and the partial order of the given flow.

Chapter 8

Conclusion and future works

1. Accomplished work

We addressed the problem of data flow control in organizations and in distributed systems such as the Internet of Things. We started by showing that the existing access control models, with the exception of MAC, are far from guaranteeing a satisfying data flow control. Most models, including recent and more used ones such as RBAC and ABAC, are concerned about access to information, but much less concerned about where the information will end.

The area of data flow has been extensively studied in different contexts. However, in our context, all the existing security data flow models are based on the lattice model. This model defines security properties in terms of itself, which makes it necessary to include inexistent or impossible entities in order to obtain the lattice structure. For this reason, this model is not much used in practice.

Essentially, in our research we have generalized the lattice model. A lattice is a special case of a partial order. The method we have developed does not force the creation of a lattice structure. It is based on the concept of quasi order (or semi-order) that can be transformed into a partial order of components. A partial order is necessary and sufficient for data secrecy and privacy, and always exists. Thus, our model is more general, more realistic and more applicable than the traditional lattice model. The development, evaluation and the implementation of this model is the main contribution of our work.

We have shown in our thesis that our model can be implemented in both organizational networks and in new generation networks such as the Internet of Things.

Our method applies also to other access control models once the access control matrix determined by them has been calculated. In fact, once the layered model that is inside an access control system has been found, we can determine which are the objects suitable to contain the most secret data. Those objects are those that cannot be read by other entities in the system. Similarly, we can determine which are the objects suitable to contain the data that should have the highest integrity, these are the objects that cannot be written by other entities. These

concepts are new and useful to identify secrecy and integrity levels in arbitrary access control systems.

To support our method, efficient polynomial time algorithms are available in the area of graph theory. By using these algorithms, we can find the layered system that is determined by an access control system, we can construct the partial order of components for the system, and finally we do the data flow analysis that determines the area of each data. Our simulation results show that this can be done in practice for up to tens of thousands of subjects and objects. These results are presented in chapter 5.

In the case of RBAC, we have a meaningful contribution. In fact, we have shown that, if the permissions are limited to reading and writing operations, there is an immediate translation from RBAC permission to capability lists and finally, into multi-level systems.. The results obtained can be used in the role engineering phase of RBAC implementation, by eliminating unused roles, or combining roles and objects that are in the same components. Further, by using our approach, we can construct a Label-based access control system from the initial access control system. This is developed in chapter 5.

Our approach is applicable beyond the organizational context. It also contributes in the context of distributed systems such as the Internet of Things. We have proposed a solution for data flow control in the IoT using our partial order model. Entities of the IoT networks are labeled according to the type of data they can hold, which makes it possible to configure data transfer channels such that all and only logically allowed flows are possible. This takes care of both secrecy and integrity requirements.

In practical systems, data flows can vary in time because of environmental conditions, and entities can be added or removed, which influences the data flows relations of the network. Therefore, in general, locally adding or removing entities or permission can have ripple effects that can involve other entities and permission, possibly in the whole network. We can address this either by recalculating the partial order of the system after each update. or by using implementation algorithms proposed in this thesis that depend on the type of modification done.

In order to implement our method in real system, we chose to work with SDN. This new network technology allows us to control all aspects of the network centrally throught an SDN

controller. We have proposed a transformation of a logical data flow control architecture into a physical one using a centralized IoT. This centralization will allows us a better control of the security in the system. Afterward, we have developed our own Ryu controller that will be responsible of the management of the network. Since we are dealing with data flow, the main purpose of our controller will be then to control data flow in the netwrok according to rules of the partial order model.

Finally, in each partial order, data can flow only upward. So for practical applications it will be necessary to have several coexisting partial orders at each network state. This can be achieved by extending our approach. Data will be labeled by flow type, and each flow will have its own partial order. Furthermore, it will be necessary to have trusted entities that belong to several partial orders and can be trusted to keep separate the different type of flows that can pass through them.

For the convenience of the readers, we will now summarize our contributions in point form:

1- An analysis of the main families of access control and flow control models, in order to show some of the limits of these models in terms of data flow control (Chapter 3).

2- A critical analysis of the main data flow control model used in the literature, namely the lattice model due to Denning (Chapters 3,4)

3- The proposal of our model based on partial orders, with simulations to prove its feasibility in systems up to many thousands of entities. (Chapter 5)

4- Proving the feasibility of our method beyond the organizational context, and presenting an application of it in the context of the IoT (Chapter 6).

5- Proposing an implementation of our method in the context of IoT, yielding a method to construct IoT configurations that meet privacy and secrecy requirements. Our proposition considers network reconfigurations and multiple data flows (Chapter 7).

6- The development of an SDN controller that aims to enforce partial order model's policies in order to control data flow in the network to meet privacy and secrecy requirements (Chapter 7).

2. Limits

In the conception of our model, we only dealt with data flow, and not information flow. The latter can be the consequence of a combination of data flows and inferences and can create hidden channels. Inferences can result in the creation of information that can be transferred as data, and can lead to the transfer of new data that should not be transferred according to access control and data flow policies (for example, secret data can be inferred from non-secret data and then transferred).

Mechanisms to control inferences are different from those used in data flow control, and they are the subject of a different literature and methods. However, it can be safely said that no inferences are possible on data that are not available, and in this sense data flow control is more specific than information flow control.

3.Future work

Research possibilities beyond the contributions of this thesis include the study of more distributed network architectures than those that have been considered, in two directions:

- In the implementation of our method, we only took in consideration a centralized IoT architecture, meaning that all collected data must pass through a central entity, in this case the cloud. An extension of our method into a more distributed IoT architecture where all the entities are interconnected without a need for a centralized entity can be a subject of future research.

- In our implementation architecture, we have considered a classical cloud vision, where the cloud service is controlled from in a single location. This allowed us to develop a control method suited for our centralized architecture. However, with the development of new technologies, we are moving toward more distributed cloud architectures where cloud services are controlled from multiple different locations. Such architectures require distributed SDN controllers (flat architecture controllers and hierarchical architecture controllers). The use of such controllers can be studied in future contributions.

- Finally, our simulations have shown that our approach can be efficiently implemented. An industrial tool could regroup all that has been done in this thesis. The tool will allow security

administrators to create data secure networks using the predicates seen in Chapter 6. Once the network is created, the data-flow analysis and implementation possibilities will be automatically generated with the tool. However, the implementation of such a commercial tool will require an industrial effort. We have already issued a proposal for an industrial tool for data security.

References

[1] https://www.estinet.com/ns/ (Consulted 10th march,2021).

[2] https://www.lifewire.com/how-many-devices-can-share-a-wifi-network-818298. (Consulted on April 28, 2021)

[3]] http://mininet.org/ (Consulted 14th december,2020)

[4] https://www.nsnam.org/ (Consulted 13th february,2021)

[5] https://nutanshinde.wordpress.com/2016/01/31/building-blocks-of-sdn-network/ (Consulted 23th february,2021)

[6] M. Ahmad Khan. A survey of security issues for cloud computing. Journal of Network and Computer Applications.71 (2016), 11-29.

[7] C. Aggarwal and K. Srivastava. Securing IOT devices using SDN and edge computing. 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun (2016), 877-882.

[8] A. Al-Haj, B. Aziz. Enforcing Multilevel Security Policies in Database-Defined Networks using Row-Level Security. International Conference on Networked Systems (NetSys), Munich, Germany (2009), 1-6.

[9] P. Amthor, W. Künhauser, A. Pölk. WorSE: A workbench for model-based security engineering. Computers &Security 42 (2014), 40-55.

[10] J. Andress. The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice. Syngress, 2014.

[11] M. Avram. Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective,. Procedia Technology, 12(2014), 529-534.

[12] J. Bacon, D. Eyers, T. Pasquier, J. Singh, I. Papagiannis, P. Pietzuch. Information Flow Control for Secure Cloud Computing. IEEE Transactions on Network and Service Management 11(1) (2014), 76-89.

[13] J. Bacon, D. Evans, D.Eyers, M. Migliavacca, P.Pietzuch, B.Shand. Enforcing end-to-end application security in the Cloud. Proc. Middleware 2010, LNCS 6452, 293–312.

[14] A. Bedford, J. Desharnais, T. G. Godonou, N. Tawbi. Enforcing information flow by combining static and dynamic analysis. Foundations and Practice of Security – 6th International Symposium (FPS 2013), 83-101,

[15] A. Bedford, S. Chong, J. Desharnais, E. Kozyri, N. Tawbi. A progress-sensitive flow-sensitive inlined information-flow control monitor (extended version). Comput. Secur. 71(2017), 114-131.

[16] D.Bell, L. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation.1976.

[17] M. Benantar. Access control systems: security, identity management and trust models. First edition. Springer, 2006.

[18] R. Bhatti, E. Bertino, A. Ghafoor. A Trust-Based Context-Aware Access Control Model for Web-Services. Distributed and Parallel Databases archive 18(1) (2005), 83–105.

[19] K. Biba. Integrity Considerations for Secure Computer Systems. Technical report ESD-TR-76-372, 1977.

[20] 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.

[21] J. Boritz. IS Practitioners' Views on Core Concepts of Information Integrity. International Journal of Accounting Information Systems. Elsevier. 2011.

[22] R. Botha, J. Eloff. Separation of Duties for Access Control Enforcement in Workflow Environments. IBM Systems Journal, 40(3) (2001).

[23] D. Brewer, M. Nash. The Chinese wall security policy. IEEE Symposium on security and privacy (1989), 206-214.

[24] J. Byun, N. Li. Purpose based access control for privacy protection in relational database systems, VLDB J, 17(4) (2008), 603-618.

[25] R. Chon, T. Enokido,V. Wietrzsk, M. Takizawa. Role locks to prevent illegal information flow among objects. Proc. of IEEE 18th Intern. Conf. on Advanced Information Networking and Applications, 1 (2004), 196-201.

[26] W. Chou, C. Yang, C. Ming. EstiNet openflow network simulator and emulator. Communications Magazine, IEEE. 51 (2013), 110-117.

[27] S.Christos, P. Kostas, K. B.Gyu, B. Gupta . Secure Integration of Internet-of-Things and Cloud Computing. Future Generation Computer Systems (2013).

[28] L. Clemmer. Information Security Concepts: Authenticity Available from: http://www.brighthub.com/computing/smb-security/articles/31234.aspx. [consulted February 26th, 2018].

[29] R. Conway, W. Maxwell, H. Morgan. On the implementation of security measures in information systems. Communication of the ACM, 15(4) (1972), 211-220.

[30] J. Crampton. On permission, inheritance and role hierarchies. 10th ACM Conference on computer and communication security (2003).

[31] D. Denning. A lattice model of secure information flow. Commun. ACM 1(5) (1976), 236-243.

[32] J. Desharnais, E. Kanyabwero, N. Tawbi. Enforcing information flow policies by a three-valued analysis. LNCS 7531 (2012), 114–129.

[33] M. Dias de Assunção, R. Carpa, L. Lefèvre, et al. Designing and building SDN testbeds for energy-aware traffic engineering services. Photon Netw Commun 34 (2017), .396–410

[34] T. Dillon, C. Wu, E. Chang. Cloud Computing: Issues and Challenges. 24th IEEE International Conference on Advanced Information Networking and Applications (2010), 27-33.

[35] R. Djouani, K. Djouani, F. Boutekkouk, R. Sahbi .A Security Proposal for IoT integrated with SDN and Cloud. 6th International Conference on Wireless Networks and Mobile Communications (WINCOM) (2018), 1-5.

[36] DoD Manuel. DoD Information Security Program: Protection of Classified Information. US department of defense, 2013.

[37] Q. Duan, N. Ansari, M. Toy. Software-defined network virtualization: an architectural framework for integrating SDN and NFV for service provisioning. in future networks IEEE Netw, 30 (5) (2016), 10-16.

[38] L. EL-Garoui, S. Pierre, S. Chamberland. A New SDN-Based Routing Protocol for Improving Delay in Smart City Environments. Smart Cities. 3(3) (2020), 1004-1021.

[39] 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. Springer LNCS, vol 7743.

[40] D. Ferraiolo, D.Kuhn, R. Chandramouli. Role-Based Access Control 2nd edition. Artech House, 2007.

[41] D. Ferraiolo, J. Cugini, D. Kuhn. Role-Based Access Control (RBAC): Features and Motivations. 11th Annual computer security application conference (1995). 241-248.

[42] O. Flauzac, C. Gonzalez, A. Hachani, F. Nolot .SDN Based Architecture for loT and Improvement of the Security. IEEE 29th Int'l. Conf. Advanced Information Networking and Applications Workshops (WAINA) (2015), pp.688-693.

[43] Foundation, O.N. Software defined networking (SDN). Open networking foundation: https://www.opennetworking.org/sdn-definition/ . (Consulted october, 11th 2020).

[44] P. Galvin, G. Gagne, A.Silberschatz. Operating system concepts seventh edition. John Wiley&Sons, 2013.

[45] M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller. Incremental information flow analysis of role based access control. In: International Conference on Security and Management, 2009, pp.397–403.

[46] M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller. Rbac-pat: A policy analysis tool for role based access control. Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), LNCS 5505, 46-49.

[47] C.Gonzalez, S.Charfadine, O.Flauzac, F.Nolot. SDN-based security framework for the IoT in distributed grid. International Multidisciplinary Conference on Computer and Energy Science (SpliTech), 2016.

[48] Government of Canada. Internal Audit and Accountability Branch. Citizenship and Immigration Canada Final Report. May 2012.

[49] A. Hakiri, P. Berthou, A. Gokhale, S. Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. IEEE Communications Mag 53 (9) (2015), 48-54.

[50] Z. Han, X. Li, K. Huang, Z. Feng .A Software Defined Network-Based Security Assessment Framework for Cloud IoT. IEEE Internet of Things Journal, 5 (3) (2018), 1424-1434.

[51] A.Hany, W. Gary.Intersections between IoT and distributed ledger. Advances in Computers. Role of Blockchain Technology in IoT Applications (3) (2019).

[52] F. Harary, R.Z. Norman, D. Cartwright. Structural models: an introduction to the theory of directed graphs. Wiley, 1965.

[53] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. Communication of the ACM, 19(1976), 461-471.

[54] J. Hintzbergen, K. Hintzbergen, H. Baars, A. Smulders. Foundations of Information Security Based on Iso27001 and Iso27002. Best Practice. Van Haren Publishing, 2010.

[55] V. Hu, D. Ferraiolo, D. Kuhn. Assessment of Access Control Systems. Institute of Standards and Technology, 2006.

[56] V.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 (2014).

[57] T. Huu, M. Hitchens, V. Varadharajan, P. Watters. A Trust based Access Control Framework for P2P File-Sharing Systems. Proceedings of the 38th Annual Hawaii International Conference on System Sciences (2005).

[58] M.T., Islam, M. Refat. Node to Node Performance Evaluation through RYU SDN Controller. Wireless Pers Commun 112 (2020), 555–570.

[59] ISO/IEC 13335-1 Information technology. Security techniques: Management of information and communications technology security -- Part 1: Concepts and models for information and communications technology security management. 2004.

[60] ISO/IEC 27001. Techniques de sécurité – Systéme de management de la sécurité de l'information – Exigences. 2015.

[61] K. Izaki, K. Tanaka, M. Takizawa. Information flow control in role-based model for distributed objects. Proc. 8th Intern. Conf. on Parallel and Distributed Systems (ICPADS 2001), 363-370.

[62] Justice Laws Website. Privacy Act R.S.C., 1985, c. P-21, Available from: http://laws-lois.justice.gc.ca/eng/acts/P-21/page-1.html. [consulted February 26th, 2018].

[63] K. Karmakar, V. Varadharajan, S. Nepal, U. Tupakula. SDN Enabled Secure IoT Architecture. IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA (2019), 581-585.

[64] F. Keti, S. Askar. Emulation of Software Defined Networks Using Mininet in Different Simulation Environments. 6th International Conference on Intelligent Systems, Modelling and Simulation, Kuala Lumpur, Malaysia, 2015, 205-210.

[65] 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.

[66] R. Khondoker, A. Zaalouk, R. Marx, K. Bayarou. Feature-based comparison and selection of Software Defined Networking (SDN) controllers. Computer Applications and Information Systems (WCCAIS) (2014), 1-7.

[67] J. Kleinberg, E. Tardos . Algorithm design. Pearson, 2005.

[68] R. Kubo, T. Fujita, Y. Agawa, H. Suzuki . Ryu SDN Framework— Open-source SDN Platform Software. NTT Technical Review,12 (8) (2014) .

[69] B. Lampson. Protection. ACM Operating Systems Review, (1974), 18-24.

[70] A. La Marra, F. Martinelli, P. Mori, A. Saracino. Implementing Usage Control in Internet of Things: A Smart Home Use Case. IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, 2017, pp. 1056-1063.

[71] A. La Marra,F. Martinelli P. Mori, A.Rizos A. Saracino. Introducing Usage Control in MQTT. International Workshop on Security and Privacy Requirements Engineering. 2017, 35-43.

[72] Y. Liu. Trust-Based Access Control for Collaborative System. ISECS International Colloquium on Computing, Communication, Control, and Management (2008), 444-448.

[73] G. Liu, G. Zhang, R. Zhang, J. Cui, Q. Wang, S. Ji. An improved blp model with response blind area eliminated. 2017 International Symposium on Networks, Computers and Communications (ISNCC), Marrakech, 2017, 1-6.

[74] L. Logrippo. Logical Method for Reasoning about Access Control and Data Flow Control Models. Proc. of the 7th International Symposium on Foundations and Practice of Security (FPS 2014). Springer LNCS 8930, 205–220.

[75] L. Logrippo. Multi-level access control, directed graphs and partial orders in flow control for data secrecy and privacy. Foundations and Practice of Security. FPS 2017. Springer LNCS 10723 (2018), 111-123.

[76] L. Logrippo . Multi-level models for data security in networks and in the Internet of things. Journal of Information Security and Applications 58 (2021).

[77] L. Logrippo, A.Stambouli. Configuring data flows in the Internet of Things for security and privacy requirements. Presented at the 11th International Symposium on Foundations and Practice of Security .Montreal, 2018. Springer LNCS Vol. 11358.

[78] P. Mahalle, P. Thakre, N. Prasad, R. Prasa .A fuzzy approach to trust based access control in internet of things. 3rd IEEE International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE) (2013), 1–5.

[79] A. Mamdouh, K. Almustafa, K. Amjad Meerja. Cloud based SDN and NFV architectures for IoT infrastructure. Egyptian Informatics Journal 20 (2019), 1-10.

[80] A. Marcon, L. Olivo, A. Santin, M. Stihler, J. Bachtold. A UCONabc Resilient Authorization Evaluation for Cloud Computing,. IEEE Transactions on Parallel and Distributed Systems, 25(2) (2014), 457–467.

[81] M. Meng, S. Steinhardt, A. Shubert.Application Programming Interface Documentation. Journal of technical writing and communication, Vol.48(3) (2017), 295-330.

[82] T. Mikko, H. Kukkonen. A review of information security issues and respective research contributions. SIGMIS Database, 38(1) (2007), 60-80.

[83] A.Myers, B. Liskov. A decentralized model for information flow control. SIGOPS Oper. Syst. Rev. 31, 5 (1997), 129-142.

[84] S. Nakamura, D. Duolikun, A. Aikebaier, T. Enokido, M. Takizawa. Synchronization Protocols to Prevent Illegal Information Flow in Role-based Access Control Systems. Proc. of International Conference on Complex Intelligent and Software Intensive Systems (CISIS-2014), 279-286.

[85] 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.

[86] M. Nyanchama, S. Osborn. The role graph model and conflict of interest. ACM Transactions on Information and System Security 2 (1) (1999), 3-33.

[87] S. Ola, E. Imad, K. Ayman, C.Ali. SDN controllers: A comparative study. 18th Mediterranean Electrotechnical Conference (MELECON) (2016), 1-6.

[88] F. Olivier, C. Gonzalez, F. Nolot. New Security Architecture for IoT Network. Procedia Computer Science. 52 (2015), 1028-1033.

[89] Open Network Foundation, "Openflow switch specification", Protocol version 0x06, 26 Mars 2015. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf,

[90] S. Osborn. Information flow analysis of an RBAC system. Proc. of the seventh ACM symposium on Access control models and technologies, (SACMAT 2002), 163-168.

[91] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Transactions on Information and System Security, 3 (2) (2000), 85-106.

[92] D. Parker. Fighting Computer Crime: A New Framework for Protecting Information. John Wiley & Sons, Inc, 1998.

[93] D. Parker. Our Excessively Simplistic Information Security Model and How to Fix It. Available from: http://www.bluetoad.com/publication/index.php?i=41813&m=0&l=&p=1&pre=&ver=swf. [accessed February 26, 2018].

[94] T. Pasquier, J. Bacon, J. Singh, D. Eyers. Data-Centric Access Control for Cloud Computing. Proc. 21st ACM Symp. On Access Control Models and Technologies (SACMAT '16), 81-88.

[95] K. Prabhakar, N. Jisha, A. Krishnashree. SDN Framework for Securing IoT Networks. Ubiquitous Communications and Network Computing (2017), 116-129.

[96] Z. Qin, G. Denker, C. Giannelli, P. Bellavista and N. Venkatasubramanian. A Software Defined Networking architecture for the Internet-of-Things. IEEE Network Operations and Management Symposium (NOMS), 2014.

[97] C. Qiang, G. Quan, B.Yu, L. Yang. Research on security issues of the Internet of Things. International Journal of Future Communication and Networking, 6 (6) (2013), 1-10.

[98] W.Roy, S. Bill, J. Scott. Enabling the Internet of Things. Computer (48) (2014), 28-35.

[99] G. Riley, T. Henderson. The ns-3 Network Simulator. In: Wehrle K., Güneş M., Gross J. (eds) Modeling and Tools for Network Simulation. Springer, Berlin, Heidelberg (2010), 15-34.

[100] O.Rogerio, S.Christiane, S. Ailton, P.Ligia. Using Mininet for emulation and prototyping Software-Defined Networks. IEEE Colombian Conference on Communications and Computing, 2014,1-6.

[101] A. Russo, A. Sabelfeld. Dynamic vs. Static Flow-Sensitive Security Analysis. 23rd IEEE Computer Security Foundations Symposium, Edinburgh, 2010, 186-199.

[102] S. Sahoo, B. Sahoo, A. Panda. A secured SDN framework for IoT. International Conference on Man and Machine Interfacing (MAMI) (2015).

[103] A. Saleh, G. Bhargavi, S. Mohammed. Ryu controller's scalability experiment on software defined networks, IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), 2018, 1-5.

[104] 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.

[105] P. Samarati, S. De Capitani di Vimercati. Access Control: Policies, Models and Mechanisms. LNCS 2171 (2000).

[106] R. Sandhu. Lattice-based access control models. IEEE Computer, 26 (11) (1993), 9-19.

[107] R. Sandhu. Relational Database Access Controls. Handbook of Information Security Management. Auerbach Publishers, (1994), 145–160.

[108] R. Sandhu, E. Coyne, H. Feinstein, C. Youman. Role-Based Access Control Models. IEEE Computer, 29(2) (1996), 38-46.

[109] R. Sandhu, D. Ferraiolo, D. Kuhn. The NIST model for Role Based Access Control. 5th ACM Workshop on Role Based Access Control (2000).

[110] J. Schütte, G.S. Brost. LUCON: Data flow control for message-based IoT systems. arXiv preprint arXiv:1805.05887, 2018 - arxiv.org.

[111] S. Shah, J. Faiz, M. Farooq, A. Shafi, S. Mehdi. An architectural evaluation of SDN controllers. IEEE International Conference on Communications (ICC). 2013, 3504-3508.

[112] A. Shalimov D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky. Advanced study of SDN/OpenFlow controllers. Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, ACM, 2013.

[113] D. Simovici, D. Chabane. Partially Ordered Sets. Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics. Springer, 2008.

[114] J. Singh, T. Pasquier, J. Bacon, J. Powles, R. Diaconu, D. Eyers. Big ideas paper: Policy-driven middleware for a legally-compliant Internet of Things. In Proceedings of the 17th International Middleware Conference .ACM. 2016.

[115] 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.

[116] J. Singh, T. Pasquier, and J. Bacon. Securing Tags to Control Information Flows within the Internet of Things. in International Conference on Recent Advances in Internet of Things (RIoT'15), 2015.

[117] M. Siponen, H. Oinas-Kukkonen. A review of information security issues and respective research contributions. SIGMIS Database 38 (1) (2007), 60-80.

[118] B. Smriti, R. Sandhu. ABAC-CC: Attribute-Based Access Control and Communication Control for Internet of Things. ACM Symposium on Access Control Models and Technologies (SACMAT) (2020), 203-212.

[119] D. Solove. Understanding Privacy. Cambridge, Mass.: Harvard University Press, 2008.

[120] A. Stambouli, L. Logrippo. Data flow analysis from capability lists, with application to RBAC. Information Processing Letters (Elsevier) 141 (2019), 30–40.

[121] M. Stramp. Information security; Principle and practise 2nd edition .Wiley, 2011.

[122] R. Tarjan,. Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (2) (1972), 146–160.

[123] D. Turner. Digital Authentication: The Basics. Cryptomathic, 2016.

[124] N. Tuval, E. Gudes. Resolving Information Flow Conflicts in RBAC Systems. Data and Applications Security, LNCS 4127 (2006), 148-162.

[125] J. Vacca. Computer and Information Security Management Handbook, 2 ed. Morgan Kaufmann Series, 2009.

[126] A. Wang, X. Mei, J. Croft, M. Caesar, B. Godfrey. Ravel: A database-defined network, Proceedings of the Symposium on SDN Research (2016), 1-7.

[127] F. Wibowo, M. Gregory, K. Ahmed, K. Gomez. Multi-domain software defined networking: research status and challenges. J Netw Comput Appl 2017.

[128] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, J. A. Mccann.UbiFlow: Mobility management in urban-scale software defined loT. IEEE Conference on Computer Communications (INFOCOM), 2015.

[129] R. Xie, H. Li, G.Shi ,Y.Guo B.Niu, M.Su .Provenance-based data flow control mechanism for Internet of things. Transactions on Emerging Telecommunications Technologies (2020).

[130] L. Xiong, L. Liu. A reputation-based trust model for collaborative e-commerce communities. in IEEE Int. Conf. E-Commerce (2003), 275-285.

[131] M. Yassein, S. Aljawarneh, M. Al-Rousan, W. Mardini ,W. Al-Rashdan, Combined software-defined network (SDN) and Internet of Things (IoT), International Conference on Electrical and Computing Technologies and Applications (ICECTA)(2017), 1-6.

[132] D. Zhang, K. Ramamohanarao, R. Zhang, S. Versteeg. Efficient Graph Based Approach to Large Scale Role Engineering. Transactions on data privacy, 7 (2014), 1-26.

[133] G. Zhang, G. Wentao. The Research of Access Control Based on UCON in the Internet of Things. Journal of Software, 6 (4) (2011), 724-731

[134] C. Zhang, C. Yang. Information flow analysis on Role-based access control model. Information Management &Computer Security, 10 (5) (2002), 225-236