Smart Object-Oriented Access Control

Distributed access control for the Internet of Things

Thomas Cattermole

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Philosophy

of

University College London.

Department of Computer Science

University College London

May 8, 2023

I, Thomas Cattermole, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Ensuring that data and devices are secure is of critical importance to information technology. While access control has held a key role in traditional computer security, its role in the evolving Internet of Things is less clear. In particular, the access control literature has suggested that new challenges, such as multi-user controls, fine-grained controls, and dynamic controls, prompt a foundational re-thinking of access control. We analyse these challenges, finding instead that the main foundational challenge posed by the Internet of Things involves decentralization: accurately describing access control in Internet of Things environments (e.g., the Smart Home) requires a new model of multiple, independent access control systems. To address this challenge, we propose a meta-model (i.e., a model of models): Smart Object-Oriented Access Control (SOOAC). This model is an extension of the XACML framework, built from principles relating to modularity adapted from object-oriented programming and design.

SOOAC draws attention to a new class of problem involving the resolution of policy conflicts that emerge from the interaction of smart devices in the home. Contrary to traditional (local) policy conflicts, these global policy conflicts emerge when contradictory policies exist across multiple access control systems. We give a running example of a global policy conflict involving transitive access. To automatically avoid global policy conflicts before they arise, we extend SOOAC with a recursive algorithm through which devices communicate access requests before allowing or denying access themselves. This algorithm ensures that both individual devices and the collective smart home are secure. We implement SOOAC within a prototype smart home and assess its validity in terms of effectiveness and effi-

ciency. Our analysis shows that SOOAC is successful at avoiding policy conflicts before they emerge, in real time. Finally, we explore improvements that can be made to SOOAC and suggest directions for future work.

Impact Statement

This thesis has the potential to impact work carried out inside and outside academia. In particular, contributions include:

- to the theory of access control, by providing
  - a re-analysis of the field from its origins,
  - a new paradigm for decentralized access control,
  - a policy model to contrast with existing models;
- to Internet of Things (IoT) security research, by providing
  - a new access control model tailor-made for the IoT and the Smart Home,
  - a solution to existing threats in the current Smart Home;
- to industry, through
  - the development of software for resolving policy conflicts in IoT access control;
- and to security practitioners, by providing
  - tools for analyzing and resolving real-world problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Life was simple before World War II. After that, we had systems.

Grace Hopper

In this chapter:

- we identify the main problems in the thesis;
- we motivate the need for solutions to these problems;
- we provide a running example showing the main problem and motivating its solution;
- we identify the original contributions of the thesis;
- we define the success criteria for our proposed solution;
- we describe the thesis structure.

1.1 Motivations

Ensuring that data and devices are secure is of critical importance to information technology. Access control — that is, the restricting of operations that can be performed by subjects (e.g., users, processes) on objects (e.g., data, devices) — has historically held a preeminent role in security, being *"the traditional center of gravity of computer security"* [16].

Despite this strong tradition, researchers have been led to question to what extent access control models developed nearer the dawn of computer security are still relevant today [125, 135, 119, 138, 120]. In particular, the Internet of Things, *"which brings internet connectivity, data processing and analytics to the world of physical objects,"* [72] has been considered to pose significant challenges to access control. For example, Internet of Things environments typically include many different types of devices, with varying operational capabilities; are dynamic, liable to change quickly; and must satisfy the differing needs of many users.

Such challenges have prompted some to call for a re-understanding of access control from its foundations. As Calo et al [39] put it, *"we need to envision a new type of access control paradigm [as traditional models] may turn out to be inadequate to deal with the Internet of Things."*

Our initial motivation is to assess the validity of this claim, which we summarise in the following research question.

> **RQ 1**: What new challenges for access control are posed by the Internet of Things?

Addressing this question requires both an assessment of what traditional access control has been (including the challenges it has faced in the past) and what is required of it in the context of the Internet of Things. We therefore separate out **RQ 1** into the following component parts.

> **RQ 1.1**: What historical access control models are there and what challenges led to their introduction?
>
> **RQ 1.2**: What are the challenges posed by the Internet of Things and do they prompt a re-understanding of access control?

Our main findings in response to these questions are that (i) traditional access control models are characterized by centralization: they assume that a single entity is responsible for restricting access on behalf of all subjects and objects within a given system, and (ii) the Internet of Things, as it currently exists, and in the direction it appears to be heading, is incompatible with such centralization.

Rather than being centralized, we find that access control in the Internet of Things is currently *decentralized*: multiple entities are responsible for controlling access in the Internet of Things. This is seen particularly clearly in the Smart Home, in which devices form distinct ecosystems of devices, each with their own associated access control, within one broader system of the home environment.

While there is a rich history of modelling centralized access control, models for decentralized access control are less developed. Our second research questions seeks to address this.

RQ2: Can a model for decentralized access control be constructed?

To answer this question, we propose extending the XACML architecture [152] to form the basis for a new access control meta-model (i.e. a model of access control models [22]), which we term *Smart Object-Oriented Access Control (SOOAC)*. We construct this meta-model according to modular principles of object-oriented programming and design. This meta-model allows us to capture a general notion of access control, using the concept of the *Access Control Unit*, which allows us to distinguish centralized from decentralized access control.

Using this meta-model, we observe that a significant challenge for decentralized access control is the emergence of policy conflicts across ecosystems of devices (see Section 1.2). We identify this new problem, which we term *global* policy conflict resolution. We recognize that resolving global policy conflicts requires that the entities responsible for access control collaborate. Borrowing from Roman et al [137], we term this type of decentralized access control *distributed access control*. This leads to our third research question.

RQ 3: Can a model for distributed access control be constructed?

To address **RQ 3**, we equip devices with a systematic procedure for communicating appropriate information with each other in order to avoid global policy conflicts. We require that when a device is requested access, it must first send an appropriately modified access request to any devices that it has access to before returning a response. This process is recursive in the sense that each device in an access chain carries out the same procedure. In this way, though individual devices are secured separately, they attain a body of knowledge that allows them to secure the system as a whole.

In order to assess SOOAC's practicability in the real world, we implement it in a prototype smart home with a number of users and devices. In this way, we address the following research question.

> **RQ 4**: Can a model for distributed access control be implemented in a real-world smart home?

We separate out this question according two metrics, which define the success criteria of the approach: efficiency and effectiveness. These form the final research questions.

> **RQ 4.1**: Can this model run efficiently, in a timely manner without without impacting the user experience in the Smart Home?
> **RQ 4.2**: Can this model run effectively, avoiding policy conflicts in a smart home with multiple users and devices?

To get a sense of the type of problem we are attempting to resolve with SOOAC, we give the following running example, which we will refer to throughout.

1.2 The Running Example — Resolving Global Policy Conflicts

Consider a simple smart home with an admin (e.g., the home owner), a guest, a smart speaker, and a smart (door) lock. The admin controls access to the speaker

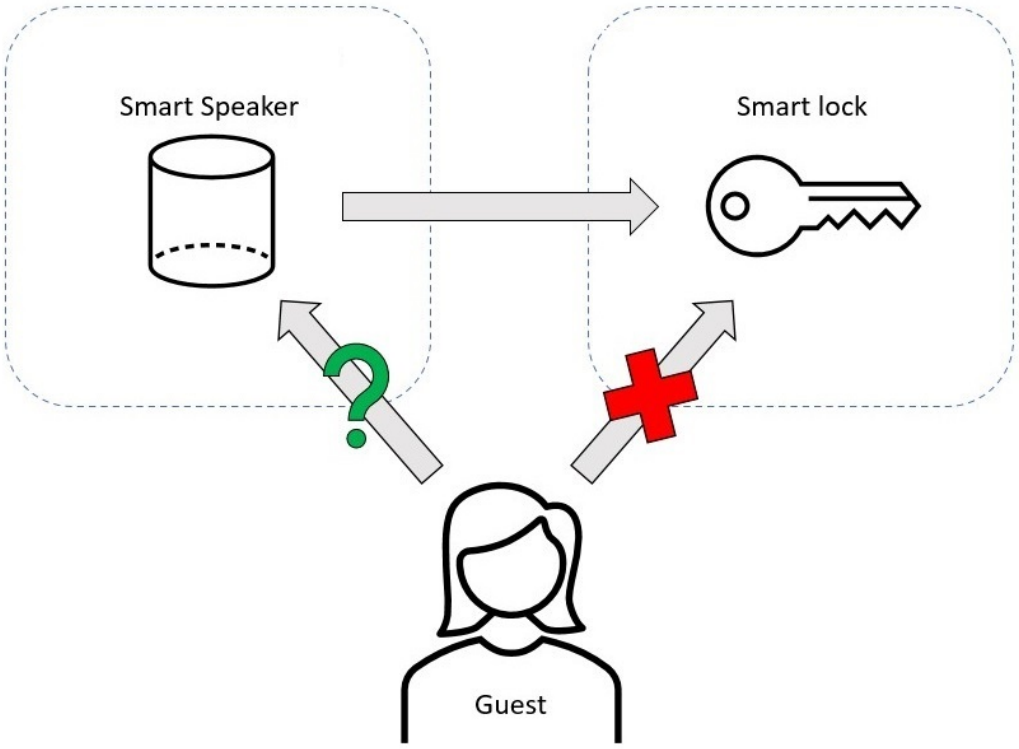

Figure 1.1: The main problem of the thesis: resolving policy conflicts for decentralized access control. Setting policies for devices in separate ecosystems can lead to conflicts in the broader system of the Smart Home. How can these conflicts be resolved in a systematic way?

and the lock on separate apps. In other words, policies are set separately for the two devices. This situation is depicted in Figure 1.1 and is described as follows.

The admin would like the smart lock to be accessible from the smart speaker (Policy 1, set at the smart lock) in order to lock and unlock the front door by voice command. They would also like the guest to have access to the smart speaker (Policy 2, set at the smart speaker), so that the guest can play music when they come to visit. However, the admin would not like the guest to be able to lock and unlock the front door, so access between the guest and the lock should be denied (Policy 3, set at the smart lock).

The admin is concerned, though, that these three policies could undermine the security of the smart home because of transitive access: the guest can use their access to the speaker (Policy 2) to piggyback on the speaker's access to the lock

(Policy 1) to issue a command to lock or unlock the door (violating Policy 3). From the local perspective of both access control apps (Policies 1 and 3 at the smart lock, and Policy 2 at the smart speaker), there will be no policy conflict, but from the global perspective of the smart home (Policies 1–3), there *will* be a conflict.

To resolve this conflict manually, the admin can simply choose to not give the guest access to the smart speaker. But this is not a *general* solution; there may be many devices involved in a conflict, creating complex access chains, and so ideally the conflicts would be resolved automatically. The main contribution of this thesis is providing such a solution in which devices avoid global conflicts themselves.

1.3 Thesis Routemap

In the next chapter, we (i) give a working definition of access control, (ii) give an account of access control's historical role in security, and (iii) delineate some of the different terms closest related to access control in the literature. We use (i)–(iii) as the conceptual groundwork to describe historical (i.e., pre-Internet of Things) access control models. We simultaneously highlight challenges for historical access control, describing requirements of these models that reflect 'good' access control. We conclude Chapter 2 by introducing a more formal definition of access control — *the Access Control Unit (ACU)* — which is based on the XACML framework. We show that traditional access control models are expressible using this framework and that they can be characterized as centralized in the sense that they describe systems which only contain a single ACU. In summary, Chapter 2 addresses the first part of **RQ 1** — **RQ 1.1** — concerning the question of what traditional access control has been and what challenges it has faced historically.

In Chapter 3, we focus on Internet of Things access control. Of particular interest in this chapter are accounts from the literature that claim that the Internet of Things poses new challenges to access control. We document these claims and assess them in light of the challenges described in Chapter 2. In concluding Chapter 3, we argue that these challenges are novel only insofar as the Internet of Things, as it currently manifests in the Smart Home, is not a system which contains a single

ACU — it is a decentralized system. Moreover, we give arguments for why this state of affairs should be embraced, not rejected. In summary, Chapter 3 completes our answer to **RQ 1** by addressing **RQ 1.2**, describing what new challenges face Internet of Things access control.

From our analysis of Internet of Things access control in Chapter 3, in Chapter 4, we are led to propose a meta-model (i.e., a model of models) which describes a single system with *multiple ACUs*. We term this model *Smart Object-Oriented Access Control* as it is built from object-oriented principles. By introducing this model we address **RQ 2**, concerning the introduction of a decentralized access control model suitable for representing the current Smart Home. Revisiting the Running Example, we observe that a significant challenge to Internet of Things access control involves global policy conflicts, which emerge from the interaction of multiple ACUs. We propose an algorithm for the automatic avoidance of policy conflicts before they are allowed to emerge. This algorithm involves the systematic communication between ACUs, in which they recursively issue access requests to one another in order to secure the entire system. This part of Chapter 4 addresses **RQ 3**, concerning the introduction of a model for distributed access control.

In Chapter 5 we implement SOOAC in a smart home containing real-world smart devices (**RQ 4**). We assess the performance of this model along the two metrics described in **RQ 4.1** and **RQ 4.2**, respectively: efficiency and efficacy. The former metric assesses whether the implementation of SOOAC runs in a timely manner without negatively impacting the user experience in the Smart Home. We compare these results to a centralized policy conflict resolver. The latter metric assesses to what extent the implementation of SOOAC avoids potential policy conflicts that arise in a real smart home.

Finally, in Chapter 6, we discuss challenges to our account and point to future work to overcome these challenges.

Chapter 2

Access Control: Principles and Foundations

> All models are wrong, but some are useful.
>
> *George Box*

In this chapter:

- we provide a working definition for access control;
- we situate access control within the broader security context, identifying in- and out-of-scope topics;
- we clarify the meaning of common access control terms, such as policies, models, and policy conflicts;
- we describe important historical (i.e., pre-Internet of Things) access control models and the challenges that led to their introduction (**RQ 1.1**);
- we identify access control requirements gleaned from the described models;
- we provide a generalized model that encapsulates the historical models;
- we provide a characterization of traditional access control as centralized.

2.1 Access Control Principles

2.1.1 A Working Definition

Access control is about limiting what can be done to things in order to make those things secure. People have been *doing* access control for as long as we have historical records: the discovered locked boxes of Ancient Egypt [54] are evidence of early attempts, controlling access to just those people with correctly shaped physical keys. In the 1960s and 70s, the rise of digital computing encouraged an *analysis* of access control. This shift from practice to theory was motivated by the need to understand how to control access for multiple users interacting with terminal machines connected to a single mainframe computer.

In particular, Lampson's work on Access Control Matrices [107] is now seen as a crucial model for understanding access control. One reason for this is that it highlights three key elements of access control: *subject*, *object*, and *operation* [142, 53]. These elements form the basis for our initial, working definition for access control:

> *Access control restricts the operations that can be performed by subjects on objects.*

We depict our working definition, along with the additional notion of the *reference monitor*, which is sometimes included in definitions of access control (see e.g., [142, 53, 87]), in Figure 2.1. We describe each of the four elements, with examples, as follows.

- **Subjects** (sometimes referred to as 'principals', or 'agents') are the entities that request access and are granted or denied access. Examples include people ('users'), processes, and devices. We will assume throughout this thesis that subjects have been pre-authenticated; that is, the identities of subjects have been determined prior to access control. This is commonly done in the literature [16], though not universally [77]. We discuss the issue of authentication in Subsection 6.1.6.

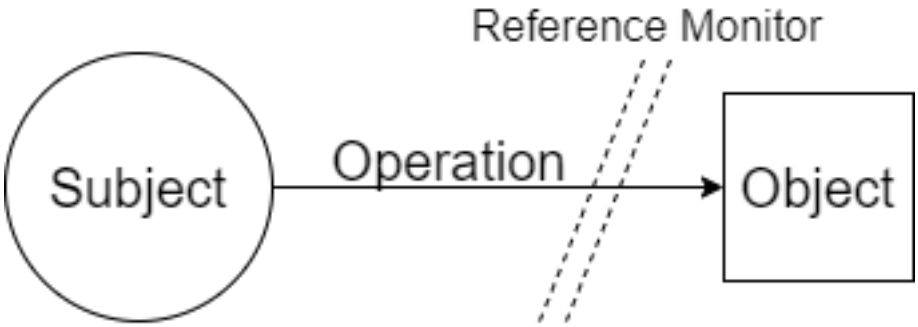

Figure 2.1: The three key elements of access control: access control is about restricting the *operations* that *subjects* can perform on *objects*. In some accounts (e.g., [142, 53, 87]), the mechanism that enforces which operations are performable by which subjects on which objects — the *reference monitor* — is also referenced explicitly.

- **Objects** ('resources', 'assets') are the entities intended to be secured. The nature of objects can be varied, being hardware (e.g., buildings) or software (e.g., files), as can be their complexity, being relatively passive entities that only need securing (e.g., documents) or more active entities that carry out processes themselves (e.g., devices). This latter class of objects will become more relevant when we consider Internet of Things devices in Chapter 3.

- **Operations** ('functions', 'capabilities') are actions that are capable of being performed by subjects on objects. As such, they are dependent on the subjects and objects under consideration, relying in principle (i.e., before access control) on (i) what subjects can do and (ii) what can be done to objects. Simple examples include read and write operations for files and the switching on or off of a device. Usually we will refer to access in terms of access to an operation, but sometimes we will omit the qualifier, referring instead just to access to an object.

- **The Reference Monitor**, introduced by Anderson [15], is the entity responsible for enforcing access control. It may be a physical keycard system in the case of access to a building, or a piece of software in the case of file systems. In information security, it typically falls under the domain of (system) administrators and is closely related to policies, more about which we will discuss

in Subsection 2.1.4.

Before using our working definition to explore historical (i.e., pre-Internet of Things) access control, we will first clarify access control's role in the broader context of security.

2.1.2 The Security Context

As noted in Chapter 1, access control has held a historically important role at the heart of computer security [16]. For several decades now, another prominent concept within computer security has been the so-called "CIA triad", consisting of *confidentiality*, *integrity*, and *availability* [139]. These three properties can be stated using the access control elements described previously: maintaining confidentiality means that objects are not disclosed to unauthorised subjects; maintaining integrity means that objects should not be modified by unauthorized subjects; and maintaining availability means that objects can be accessed by authorized subjects when required. In short, security can be understood as being about keeping things secret, correct, and obtainable.

Despite its widespread usage, the CIA triad as formulated does not by itself provide a comprehensive account of how security is practised. Most obviously, it is opaque as to what constitutes authorised and unauthorised agents. Moreover, security literature is awash with additional terms that are often not defined with respect to the CIA triad. These include 'policies', 'models', and 'policy models'. These terms have occasionally been proposed as extensions to the triad (see, e.g. [128]). This state of affairs shows the need for conceptual clarity. What would be helpful would be an ontology with which to situate these ideas within the broader context of security to give an account of their nature and how they relate to one another.

2.1.3 An Information Security Ontology

Beautement and Pym [24] develop a security ontology based on the distinction between declarative and operational concepts. Declarative concepts describe the security objectives within a given system. These are the security-relevant properties

that system-controllers desire to be the case. In general, they include the CIA triad. More specifically, they consist of statements that describe the character and extent of the confidentiality, integrity, and availability that objects within the system are expected to have. This will be dependent on the system at hand. For example, in government organisations the confidentiality of information is typically prioritised; in banks, the integrity of accounts; and in retail, the availability of goods. Operational concepts, on the other hand, describe the security mechanisms present within a given system. These are the tools designed to achieve the proposed security objectives. Security mechanisms can be physical (e.g. entry barriers, credit cards, scanners), digital (passwords, biometrics), and include policies prescribing desired behaviours (“no tailgating", “do not share your password with anyone"). In essence, while declarative concepts state what security means within a system, operational concepts explain how security is brought about.

The relationship between the declarative and the operational is at the heart of security management. In organisations, security managers are typically tasked with (i) understanding the security objectives, (ii) organising the security mechanisms, and (iii) assessing whether the security mechanisms successfully achieve the security objectives. Using the physical entry barrier example, security managers must consider how such mechanisms achieve the desired security objectives within the organisation. The declarative-operational distinction makes it clear that alternative and additional mechanisms should be considered in light of how they affect these security objectives. If, for example, the security managers propose the policy that key-cards should be worn at all times within the building, this proposal should be considered in light of how it will affect the confidentiality, integrity, and availability of the objects as desired by the organisation.

Outside of organisations, individuals typically carry out procedures (i)–(iii) implicitly. When sharing a document via email, for example, a user may expect the document to be sent only to the recipient specified (confidentiality), expect it to be received as sent (integrity), and expect it to be received in a timely manner (availability). If a user is concerned that traditional email allow documents to be

forwarded to third parties without their knowledge, they may consider showing the recipient the document in person, increasing confidentiality but reducing availability. Again, the declarative-operational distinction makes it clear that operational choices may affect desired security objectives, showing the trade-offs that often occur within the CIA triad. For a fuller discussion on trade-offs and the CIA triad, see [41].

This ontology provides a conceptual basis with which to situate our security terms. It makes clear, for example, that what it means to be authorised or unauthorised depends solely on the security objectives within a given system. It also helps to conceptually guard against making category errors in proposing operational extensions to the CIA triad. For our purposes, the ontology is useful in identifying access control as an operational concept, a point rarely made clear in the literature. We will now look in more detail at how this ontology helps to clarify the most important security concepts in this thesis, namely, policies, models, policy models, and policy conflicts.

2.1.4 Policies, Models, and Policy Conflicts

2.1.4.1 Policies

Aligned with the declarative-operational ontology described, policies come in two flavours. In the literature, this distinction is sometimes described as being between 'high-' and 'low-level' policies, respectively (see, e.g., [6, 23]).

Declarative policies are statements that describe the security objectives of a given system. At the highest level, these type of policies cover the CIA triad described previously. They can also refer to more specific policies that nonetheless still describe security objectives of a system. For example, a declarative policy in an office environment might state that the confidentiality of a specific set of documents should be maintained.

On the other hand, *operational policies* are statements that describe how declarative policies can be brought about. In this thesis, of central concern will be the operational policies involved with access control. These are sometimes referred to as 'access control policies' (see, e.g., [170]) or even 'access control rules' (see,

e.g., [76]). Given our definition of access control in Subsection 2.1.1, these policies are statements that specify the restrictions of subject-object operations. For example, an operational policy in an office environment might be that a set of documents (branded 'confidential') cannot be removed from the premises.

In this thesis, we will throughout use the words 'policy' and 'policies' to refer to operational (low-level) access control policies. At times when we need to refer to declarative (high-level) policies, we will make this alternative usage clear.

2.1.4.2 Models

Models are reasoning aids that can be used at both the declarative level and the operational level. In Subsection 2.2, we will see how they are used at the operational level to reason about access control and access control policies. These models allow practitioners to better understand the security mechanisms implemented within a given system.

Policy models are a particular type of model that help security practitioners bridge the gap between declarative and operational policies, ensuring that the security objectives of a system are met by the security implementations. In 2.2.1.3, we will consider some historically important policy models, but the main focus on models will be at the operational level rather than at the declarative level on policy models.

Another distinction that is helpful in this context is between *descriptive* and *prescriptive* models. The former refer to representations of how security systems are, while the latter are specifications for how they should be. When looking at historical models, we will often find a relationship between these two types of model. This is because representations were often introduced in order to capture some system at hand and then developed on in order to improve the security of that system in implementation. This distinction is natural given security's dual role as a science and a technology [150]. When we come to introduce our own model in Chapter 4, it will also be a model that is both descriptive and prescriptive.

2.1.4.3 Policy Conflicts

In this thesis, we will use the term 'policy conflict' to refer to an inconsistent set of access control policies — that is, when a contradiction follows from the assumption that a set of specified (operational) access control policies are true. This definition is common in the literature (see, e.g., [8, 82, 161, 169]), though sometimes comes under the slightly different heading: 'access control conflict'. The simplest kind of policy conflict is when a subject is both granted and denied access to an object operation.

Much like policies themselves, policy conflicts can arise at the declarative as well as operational level. Alkhabbas et al [9] implicitly use the distinction (in the context of the Internet of Things) when they distinguish between 'Goal Level' and 'Things Level' conflicts. In the former (declarative) case, conflicts arise from user intentions, for instance arising when users *"request goals that can not be achieved simultaneously."* In the latter (operational) case, conflicts arise when devices are requested to be in multiple states simultaneously, being *"assigned to multiple roles and requested to perform [multiple] capabilities."*

In addition to these two types of policy conflict, a third (see, e.g., [42, 38]) involves the relationship between operational and declarative policies and therefore relates closely to policy models. More specifically, this is when a set of specified (operational) access control policies fail to align with the declarative security policies intended for a system. We will consider this alternative type of policy conflict in Chapter 6. However, when we refer to 'policy conflicts' we will always mean operational policy conflicts that arise from the setting of contradictory access control rules.

In the next section we look in detail at historical models that have been applied to systems that predate the Internet of Things, beginning with Lampson's access control model. This account will draw attention to requirements deemed important historically for access control and will help us to characterise the traditional systems for which they have been applied.

Table 2.1: An example of an access control matrix. The subjects (Alice, Bob, and Charlie) have access to different operations (read, write, execute) on different objects (File 1, File 2, and File 3).

	File 1	**File 2**	**File 3**
Alice	Read, Write	-	Execute
Bob	Read	Read, Write, Execute	Read, Write, Execute
Charlie	Write	Write	Read

2.2 Access Control Foundations: Historical Models

The Access Control Matrix (ACM), introduce by Lampson [107] is a tabular model that represents subjects as rows, objects as columns, and operations within table elements (i.e., at the intersection of rows and columns). If an operation exists at the intersection of a row (subject) and column (object), that operation can be performed by the subject on the object; otherwise, the operation cannot be performed. In other words, access to an object is granted to a subject, in terms of some specific operation, if that operation is listed as a subject-object's table element; otherwise, access is denied.

Table 2.1 depicts an example ACM. This shows how (low-level) policies can be recovered from an ACM. For instance, Alice has read and write access to File 1, but neither for File 2; while Bob has only read access to File 1, but full access (read, write, and execute) to File 2. These policies can be stated explicitly in list form (e.g., 'Alice has read access to File 1', 'Alice has write access to File 1', and so on), but the ACM provides an efficient representation of the same information.

Table 2.1 is indicative of Lampson's original use case. It is also indicative for where ACMs remain in use today, namely in file systems. Implementations of ACMs are present in the so-called permission bits of Unix-based systems (e.g., Linux, Android, and iOS) and Microsoft Windows' NTFS [83].

It is important to stress that the key problem that ACMs solve in the context of file systems is the delineation of access for *multiple subjects* (i.e., access control for multi-user systems). While an ACM can have utility in systems containing just a single subject (for example, it can restrict a user from overwriting their own documents by mistake), the historical motivation was to isolate files that exist within

one file system to specific users.

In Subsection 2.1.1, we said that that ACMs are foundational for access control in highlighting the three key elements of subject, object, and operation. They are also foundational as the starting point for two broad research threads: Capability Lists and Access Control Lists. As Gollmann [76] puts it, these two threads are two perspectives on access control: *"what a principal [i.e., subject] is allowed to do, or what may be done with an object."* As we will see, this focus on subjects on one hand, and objects on the other, is a helpful way of separating out historical access control models.

2.2.1 Two Threads of Access Control: Subject-focused and Object-focused

In their simplest formulation, Capability Lists associate with each subject a set of (object, operation) pairs, while Access Control Lists (ACLs) associate with each object a set of (subject, operation) pairs. Insofar as ACMs contain the information present in both, the claim that Capability Lists and ACLs are conceptually distinct has been questioned in the literature (see, e.g., [44, 78]). As others [118, 49, 154] have noted though, the distinction is relevant in implementation: if we want to know what access a given subject has, it is more efficient to use Capability Lists; if we want to know what has access to a given object, it is more efficient to use ACLs. In any case, the problems associated with each are distinct and have historically given rise to distinct new models.

2.2.1.1 From Capability Lists to Role-Based Access Control

Capability Lists are an example of the larger category of Identity-Based Access Control (IBAC): the decision of whether to grant access or not is based on the identity of the subject. In the organizational context in particular, IBAC is now considered overly restrictive [100]. A standard issue for system administrators responsible for access control in organizations is setting policies for the regular inflow of new users. If policies depend on the identity of a subject, then each time a new user arrives, a new set of policies for that user has to be created. This creates a lot of

work for an administrator and does not scale well as the number of users increase. The same can be said for when an administrator needs to change policies for many users simultaneously. In short, IBAC cannot easily handle changes in access over time; they do not cater for *dynamic* access control.

To address this issue, Role-Based Access Control (RBAC) was developed [141], whereby policies refer to subject roles — which need changing less often — rather than subject identities. When an administrator needs to set policies for a new subject using using RBAC, they simply give that subject the necessary role (or roles, as the case may be), and the policies themselves do not need changing. Similarly, when an administrator needs to apply a change for many subjects, they can simply apply the policy change to the role. Conceptually, a role can be understood as the property of a subject, the property of belonging to a specific group of subjects. This was perhaps the earliest way of increasing the dynamicity of access control. It has since been recognized as an important requirement for any access control approach [105, 122, 71]. There is evidence that arbitrarily new roles, however can lead to so-called role explosion, where administrators have to manage permissions for too many roles, leading back to the original lack of dynamicity problem [62]. This indicates that measures introduced to improve dynamicity cannot be relied on blindly, but must be managed with care. In the particular case of role-explosion, for example, roles should only be created when necessary, or some automated process should aid role management [62].

2.2.1.2 From ACLs to Attribute-Based Access Control

ACLs have experienced their own extensions, brought about by their own problems. One such limitation that was quickly noticed was that the complex nature of some objects calls for more fine-grained control over operations. Take, for example, the problem of controlling a text document for collaboration: is it possible to give read access to a title and write access to the rest of the document? If the only available operations to be controlled are read/write/execute for the entire document (as we saw in Table 2.1), this is not possible. The earliest attempt to provide such fine-grained, line-by-line access was developed by Shen and Dewan [145].

Like dynamicity, *fine-grainedness* has since been recognised as a key requirement for any access control model [108]. As object complexity increases, so does the need for further fine-grained controls. Having said this, more controls are not always better. Dalal et al [51] give empirical evidence that users find overly complex controls unhelpful. As Beautement et al [25] and Anderson et al [14] argue, if the security burden on users becomes too great, they are likely to not use any controls and behave insecurely. What appears more appropriate, therefore, is that access control requires *responsible* fine-grained controls: controls that empower users to perform complex tasks simply.

Fine-grainedness is proportional to the number of unique operations that can be performed on an object. This depends on how complex the object is —— on what the attributes of the object are. Exploring this idea encouraged an important extension of ACLs, namely, Attribute-Based Access Control (ABAC) [90]. One way to understand this extension is through analogy: ABAC is to ACLs what RBAC is to IBAC. More specifically, ABAC is a model that allows for the attributes of objects to be relevant to the access control decision. Even more than this, ABAC allows for arbitrary properties of subjects, objects, operations, and the system (the 'environment') to be considered in access control decisions. It can therefore represent any of the previously mentioned models. In this sense, ABAC is the culmination of historical access control. We give an overview of the models discussed so far in Figure 2.2.

2.2.1.3 Orthogonal Models

Parallel to the timeline of access control models described, there exist a number of orthogonal approaches to access control. These include policy models and broad approaches covering policies.

Perhaps the most important policy model developed at a similar time to Lampson's ACM is the Bell-LaPadula policy model [26]. This was designed to reason about, and improve on, the US Department of Defense's method of sharing classified documents. Each document (i.e., object) and subject within this system is allocated a classification — 'classified', 'secret', and 'top secret' — which obey

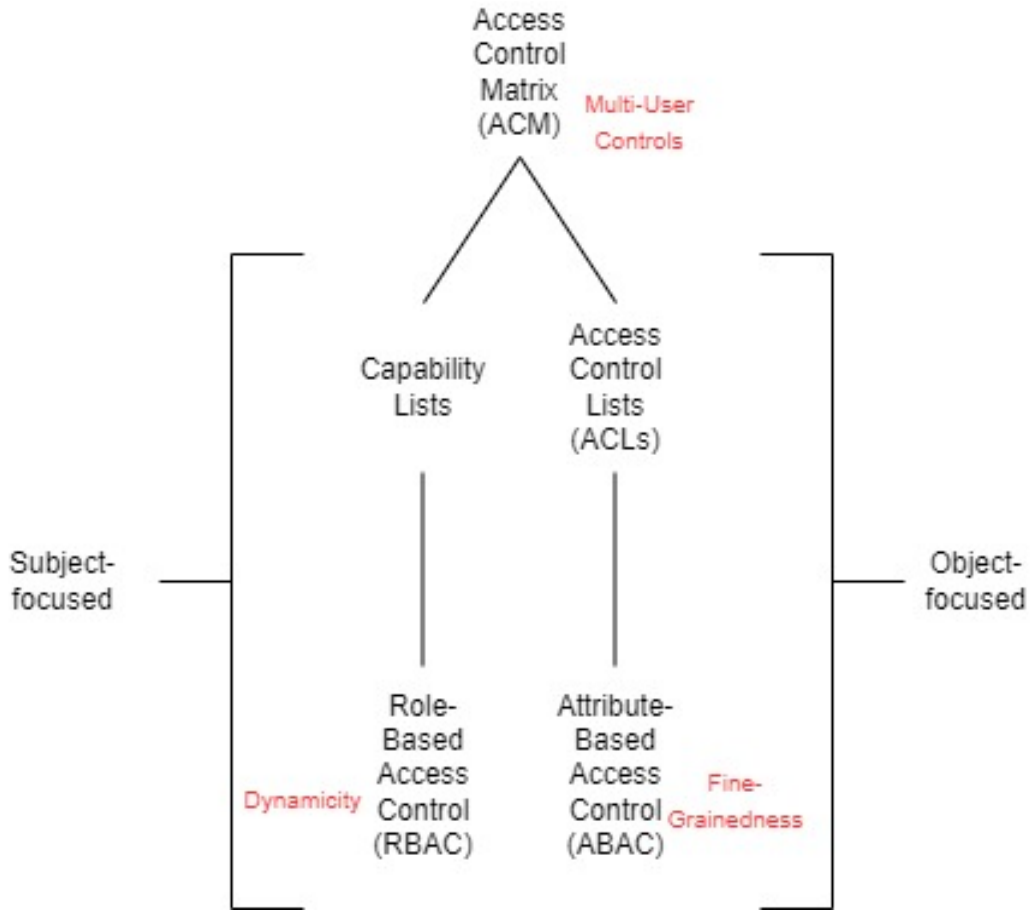

Figure 2.2: Two threads of access control originating from the Access Control Matrix: subject-focused and object-focused. Challenges that drove the particular models are shown in red. In that Attribute-Based Access Control is expressive enough to represent the other models, it is the culmination of historical (i.e., pre-IoT) access control.

a strict ordering. Access is controlled to two possible object operations, read and write, through two associated operational policies: 'no write down', prohibiting a subject of a given classification (e.g., secret) writing to a lower classification (e.g., classified) ; and 'no read up', prohibiting a subject of a given classification (e.g., secret) reading a document of a higher classification (e.g., top secret). Bell and LaPadula showed formally that, if these two policies are adhered to, the system will satisfy the declarative policy that all documents should be confidential. The Biba policy model [32] is closely related to the Bell-LaPadula model, focusing on the integrity declarative policy rather than confidentiality.

The Clark-Wilson policy model [48] is the commercial analogue to the

government-based Bell-LaPadula model. This model is notable for advancing what we would now refer to as access control *transparency*. This is the idea that it should be possible to see what objects have been accessed by subjects and in what way. Transparency is achieved by storing a log, which starts life as a blank document, that records all 'transactions' that take place in the system (i.e., operations on objects). As the policy model is stateful, this allows a full reconstruction of any state at a previous time, allowing auditors to check for policy infringements, that is policy conflicts relating to a mismatch between declarative and operational policies. Transparency has been further advanced by Povey [134], in which it is argued that access can be granted to all objects by default, as long as all historical operations to an object are recorded. If unwanted behaviour occurs, the resource can be returned to a previous state, and those responsible can be potentially denied access from that point on. Implementation of this retrospective style of access control is best seen in Wikipedia [99].

Transparency is now considered an ideal access control requirement for both individuals and organizations [127]. A related requirement is *break-glass mechanisms*. As described at length by Petritsch [133], this is the idea that, akin to breaking the glass cover of an alarm in an emergency, default policies may occasionally be violated in order to carry out a task that must be done immediately. It has been stressed in the literature that they must only be applied in exceptional circumstances. The case of Windows UAC, where users bombarded with permission requests are more likely to accept them blindly, shows that they cannot be the norm [55]. It is also vital for auditing that they are recorded, and are hence transparent. Further considerations may be about whether anyone is able to break-glass or just certain individuals. The work in [35] shows how to extend historical access control models to allow for break-glass mechanisms.

Finally, due to its notoriety, it is worth mentioning the distinction between *mandatory and discretionary access control* (MAC and DAC, respectively). These are operational (but broad) policies, the former stating that access be controlled only by the controller of the system (i.e., the system administrator), while the lat-

ter allows object owners to set their own policies. MAC has been associated with government systems (i.e., the system that Bell-LaPadula is a model of), while DAC is associated with file systems (the systems ACMs are models of). It is because of these close associations to particular models that it is common to see MAC and DAC misrepresented. It is common, for example, to hear words of the effect that MAC, DAC, and RBAC presented as *"the three main approaches to access control"* [96, 108]. Taken literally, this statement is making a category error: RBAC is a model, through which policies can stated; while MAC and DAC are policies. Indeed, it has been clearly shown in [124] that RBAC can be made to implement MAC or DAC.

Mistakes of this nature are symptomatic of the more general ontological issues that have emerged in the field of access control as a consequence of the proliferation of new models [21]. While it is understandable why new approaches to access control appear — they are often needed when trying to understand existing systems and create new ones — if they are orthogonal, they should be described as such, and not be put into direct comparison. A case in point is the practice of describing models in terms of abstractions. It is perfectly reasonable to speak of mathematically modelling Bell-LaPadula implementing MAC as lattice-based access control (LBAC) [140], but it is unhelpful to say BLP is MAC is LBAC. Similarly, while RBAC can be mathematically understood in terms of graphs, RBAC and graph-based access control are not equivalent. Mathematical abstractions, models, policy models, and policies are helpful terms in distinguishing these different approaches.

2.2.2 Access Control Requirements

We have looked at the historical models of ACMs, Capability Lists, ACLs, Role-Based Access Control, and Attribute-Based Access Control, and the orthogonal models of Bell-LaPadula, Clark-Wilson and Petritsch. From looking at these two sets of models, we have identified a number of requirements. The former of these can be considered necessary requirements for 'good' access control; the latter are additional requirements for 'ideal' access control. We summarise these two sets of requirements as follows.

2.2.2.1 Requirements for 'good' access control

- **Multi-User**. An access control model should capture many subjects interacting with potentially the same objects within one system.

- **Dynamic**. An access control model should capture changes in subjects, objects, and operations over time.

- **Fine-Grained**. An access control model should capture the range of operations possible, dependent on what operations are afforded by the objects, and are performable by the subjects, within the system.

2.2.2.2 Additional requirements for 'ideal' access control

- **Transparency**. An access control model should store records of historical policies and accesses for auditing at a later date.

- **Break-Glass Mechanisms**. An access control model should, in time-critical or other emergency situations, allow for policies to be overridden, given that necessary safeguards are in place to avoid abuse.

2.3 Generalizing Access Control

2.3.1 The ACU Model

We have seen a number of historical access control models culminating in ABAC. We have also defined a set of 'good' requirements that are desirable by any access control model. We will now define the *Access Control Unit* (ACU) and show how modelling an ACU can be used to model ABAC and capture the 'good' requirements. An ACU has four abilities:

- **Enforcement**: the ability to receive access requests and to return responses to subjects that request access;

- **Information retrieval**: the ability to retrieve relevant attributes (e.g., subject attributes, object attributes, and system attributes);

- **Administration**: the ability to write policies, store policies, and return responses about policies when queried;
- **Decision-making**: the ability to make an access control decision based on the access request, existing policies, and information retrieved.

These four abilities correspond to the well-established 'points' of an access control system described in the XACML architecture [152]: policy enforcement points (PEPs), policy information points (PIPs), policy adminstration points (PAPs), and policy decision points (PDPs), respectively. Put succinctly, an ACU has a PEP, a PAP, a PIP, and a PDP. Figure 2.3 shows the main components of the ACU Model and how they relate to one another.

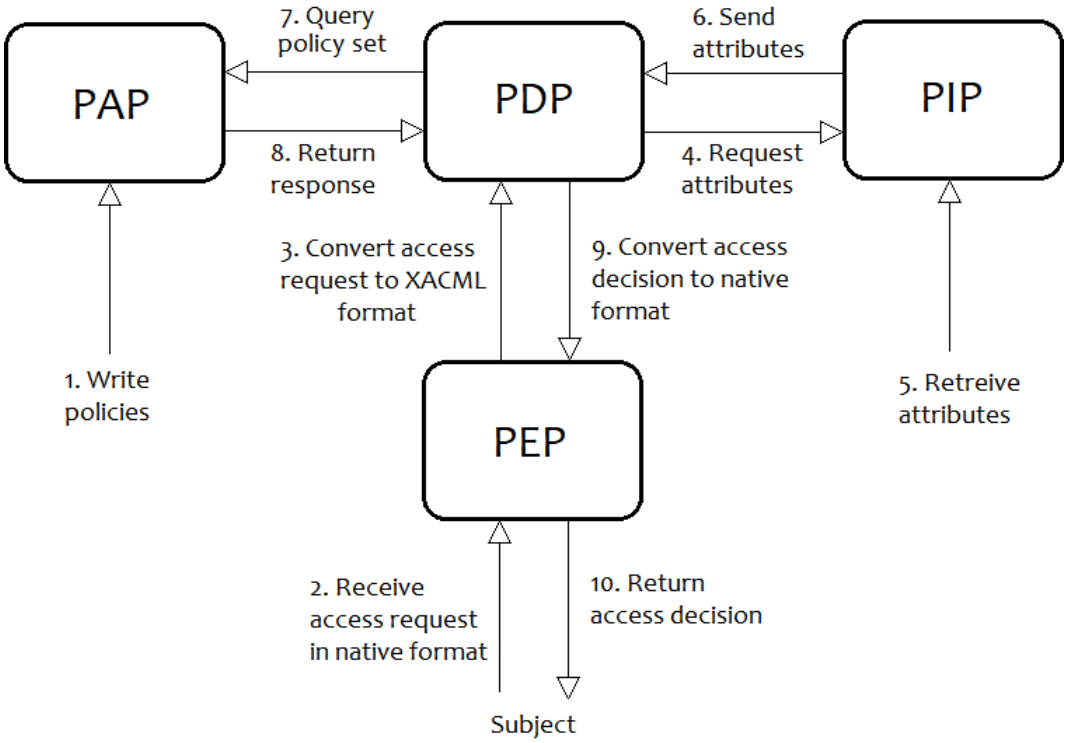

Figure 2.3: The Access Control Unit (ACU) Model showing how the main components of an ACU (PEP, PDP, PAP, PIP) relate to one another in order to decide upon an access control request.

2.3.2 Representing Historical Models Through the ACU Model

An ACU thus defined can be used to represent a range of access control models [18]. For example, it can define ABAC in the following way. Policies (technically,

in the XACML format) relating to subject, object, operation, and system attributes are written at the PAP (Step 1). An access request (in a native format, specific to the subject) is sent by the subject to the PEP (Step 2). This request is converted to the XACML format and sent to the PDP (Step 3), which requests information about the subject, object, operation, and system from the PIP (Step 4). Once the PIP retrieves this information (Step 5), it sends it back to the PDP (Step 6). The PDP then queries the policy set stored at the PAP as to whether the specified policy exists (Step 7). If the policy exists, the PAP returns its response (e.g., 'True') to the PDP (Step 8). The PDP converts the response to the native format and sends it to the PEP (Step 9), which returns the access decision to the subject (Step 10).

Given that ABAC can be defined through the ACU Model, and ABAC is expressive enough to model the historical models, it follows that the ACU Model is capable of defining the historical models.

It should be noted that XACML is first and foremost a policy language. In the present context, what is most important is the architecture described, which is a model. This means that certain features described will not generally be relevant; for example, the fact that a PEP has to convert an access request into the XACML format will not generally be true for all ACUs. This point will be particularly relevant in Chapter 4. For this reason, we refer to the model as the 'ACU Model' rather than the 'XACML Framework'.

2.3.3 Centralized Nature of Historical Access Control

Now that we have seen how the historical models can be defined through the ACU Model, it is possible to characterize these models through the ACU Model.

Most importantly, the ACU Model represents just a single ACU. This means that for all subjects and objects within a system, there is at most one PAP, PDP, PIP, and PEP. As a consequence, the historical models are only capable of representing systems in which there is one authority that carries out the process of controlling access within that system. We refer to this property by saying that access control is *centralized* for historical access control.

This characteristic is understandable given access control's historical founda-

tions: models emerged from attempts to reason about file systems (ACM) and documents within government environments (Bell-LaPadula). These are highly controlled environments with clear boundaries separating that which is inside the system and that which is outside the system. A second observation is that files and documents are passive entities: typically understood, they have no internal security mechanisms themselves. As such, objects require controlling from an external, system-level mechanism. As we will see in the next chapter, neither of these observations hold for the Internet of Things.

2.4 Summary

In this chapter, we gave a working definition for access control (Subsection 2.1.1), situated access control within the broader context of security (Subsection 2.1.2), and described important historical (i.e., pre-Internet of Things) access control models (Section 2.2). From the historical models, we gleaned a set of access control requirements that encapsulate 'good' access control and considered further optional requirements (Subsection 2.2.2). Describing these requirements satisfied the question posed by **RQ 1.1**.

We improved on our initial working definition by defining the Access Control Unit (ACU) Model, based on the XACML Framework (Subsection 2.3.1). We showed how this model is general — that is, capable of representing the historical models and embodying their requirements (Subsection 2.3.2). In this sense, the ACU Model can be understood as the culmination of historical access control prior to the Internet of Things. Finally, we argued that this model is characterized by a notion of centralization: it depicts the operating of a single ACU within a system. As such, the ACU secures all objects within that system and is effectively independent from them (Subsection 2.3.3). In the next chapter, we will see how this characterization becomes crucial for access control in the emerging context of the Internet of Things.

Chapter 3

Internet of Things Access Control

> The Internet gave us access to everything; but it also gave everything access to us.
>
> *James Veitch*

In this chapter:

- we give a definition of the IoT based on existing definitions in the literature;
- we give an overview of current smart home access control;
- we identify challenges to smart home access control proposed in the literature;
- we critically analyse said challenges according to the traditional requirements described in Chapter 2 (**RQ 1.2**);
- we provide definitions for decentralized and distributed access control;
- we introduce the distinction between local and global policy conflicts;
- we consider the main challenge for decentralized access control, in the Running Example;
- we give a set of requirements for smart home access control;
- we assess related literature based on these requirements.

3.1 The Internet of Things

3.1.1 Defining the Internet of Things

Numerous definitions for the Internet of Things (IoT) have been proposed [132, 156, 73, 84]. There have also been attempts to provide taxonomies for the IoT [162, 164], as well as descriptions of different 'waves' of the IoT [19].

The term was initially coined in a narrow sense by Kevin Ashton in 1999 as a label for his work in applying Radio Frequency Identification (RFID) to supply chains [17]. It has since entered common usage to describe something much broader in scope, both in terms of communication protocols (including RFID, WiFi, BlueTooth, MQTT, CoAP) and application environments (health care, agriculture, industry, the automotive sector, critical infrastructure, and the home).

In order to capture as much of the term as it is commonly used, we provide the following broad definition, which is similar to definitions given by Bertino et al [28] and Sfar et al [143]:

The Internet of Things consists of physically embedded devices interacting via internet-like networks.

By referring to IoT devices as 'physically embedded' we indicate that they have some physical component (the 'thing' in the IoT), including sensors and actuators through which they can sense and act in their surroundings. Through these capabilities, IoT devices can react to physical stimuli and enact physical change directly in the world. For example, in the Smart Home, smart speakers can receive and respond to voice commands spoken by members of the household.

By referring to 'internet-like networks', we are accounting for the fact that IoT devices have a digital component, typically existing on and transmitted through the internet (the 'internet' part of the IoT). As well as this, we also include devices that have an indirect or minimal connection to the internet (e.g., through a hub) or no connection to the broader internet, but are nevertheless interconnected with other devices (e.g., on Local Area Networks). The latter class of devices includes those that can behave as if online (e.g., SCADA systems).[1] In the Smart Home, users

[1] As Stuxnet showed (see, e.g., [151]), even devices designed to be offline can be attacked. In

have become increasingly concerned about having their IoT devices connected to the internet and so some are favouring just local connections [121]. Such devices require securing (e.g., through access control) just as those with direct connections to the internet do [60].

As a consequence of this definition, some devices that pre-date the IoT, such as phones and laptops, will qualify as IoT devices. In some contexts employing such a broad definition for the IoT might be problematic.[2] In the context of this thesis, however, the breadth of definition here is a benefit because these device that would otherwise be excluded are typically involved in access control; of particular relevance is the role phones play for administrators to *control* access and users to *have* access to devices in the Smart Home.

3.1.2 Access Control in the Smart Home

IoT devices are now all around us (they are *ubiquitous*) and often blend seamlessly into our daily lives (they are *embedded*) [67]. This is perhaps no more evident than in our homes, in which a variety of so-called 'smart'[3] devices make up the so-called Smart Home. An overview of a smart home is shown in Figure 3.1.

Typically, a smart home will have a single access point, through which IoT devices can send and receive data. For offline-capable IoT devices, a smartphone connected to the WLAN (generated by the access point) will have an app or apps through which IoT devices are controlled and policies are 'written'. This process will typically involve downloading an app, setting up an account, and pairing a de-

general, this case reminds us of the warning: "don't believe in airgaps!"; that is, it is incredibly hard to create systems in which devices are truly separated from the internet.

[2]This does not just include the issue of anachronism in labelling devices as IoT that existed before 'the IoT' was a term; there can also be technical reasons for avoiding such a broad definition for the IoT. For example, in the area of device identification (see, e.g., [144, 47, 146]), producing algorithms that can identify general purpose devices (like phones and laptops) as well as operationally narrow devices (like temperature sensors and smart locks) is a challenge, for one reason because the general purpose devices may be carrying out some of the operations of the operationally narrow devices; it may therefore make sense in this context to use a narrower definition for the IoT in order to use different algorithms for the two classes of device. We will return to the issue of identifying IoT devices when we discuss authentication in Chapter 6.

[3]This term is rarely defined. In common usage it is often used to mark out devices whose primary function pre-date the IoT (e.g., smart TVs, smartphones, smart kettles). In agency access control papers (see, e.g., [106, 59, 69, 68, 70]), the term 'smart object' refers to devices that exhibit context awareness in IoT environments. We leverage this idea in our usage of 'smart' in the Smart Object-Oriented Access Control model we propose in Chapter 4.

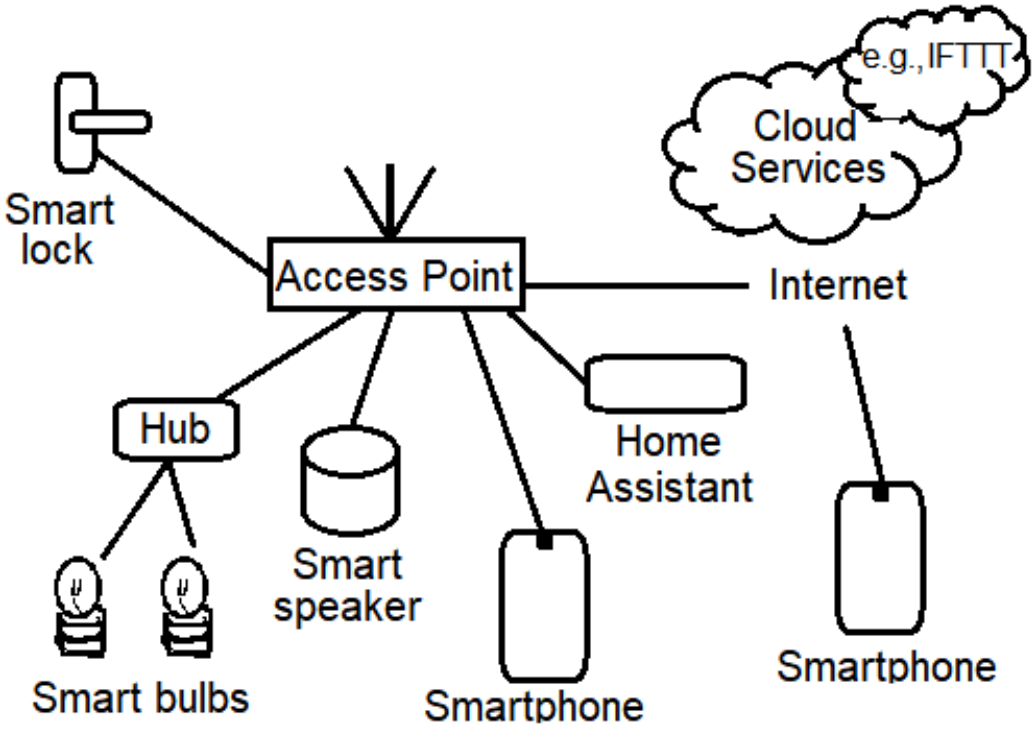

Figure 3.1: Smart Home Overview. Typically, a smart home will have one access point, through which wired and wireless devices communicate to each other and the Internet. Some resource constrained devices (e.g., the smart bulbs shown) may first communicate to a hub. Software (e.g., Home Assistant) may be present on local hardware that controls access to certain devices, and cloud services (e.g., IFTTT) may perform a similar role outside the Smart Home on the Internet. Operations on devices are typically controlled (inside or outside the Smart Home) through smartphones.

vice. Levels of access to specific operations on each device will depend on the app at hand. If multiple IoT devices are made by one manufacturer (especially the larger manufacturers), it is likely that a single app will afford controls to each of them. Hubs may also be involved that use their own (wired or wireless) protocol to communicate with their respective devices (the smart bulbs shown in Figure 3.1). Standalone physical devices (e.g., Home Assistant [159] running on a PC or Raspberry Pi) may allow the smartphone to control many IoT devices (irrespective of manufacturer) within one app.

If the access point is also connected to the internet (the most common smart home setup), it will usually be possible to use the smartphone app(s) without being connected to the WLAN (i.e., remotely). With connection to the internet it is also

possible to leverage cloud services. For example, with IFTTT [93], operations carried out on one device can be chained together with operations on other devices for automation purposes.

The location at which access control requests are deliberated upon varies. Most often, it will take place in the cloud on server databases [148]. It can also, in principle, be controlled at the 'edge' (i.e., within devices themselves) [88]. Using an external device for access control (e.g., running Home Assistant) falls between these two extremes, allowing access to be controlled within the physical bounds of the home but away from the specific IoT devices themselves.

At a basic level, the role access control plays in the smart home is no different from from the role it plays in other environments (as described in Subsection 2.1.1): access control restricts what subjects (e.g., users, IoT devices, processes) have access to what objects (e.g., users, IoT devices, processes, data).

Despite this, it is widely stated in the literature that the Smart Home raises a number of novel challenges for access control. Whether our understanding of access control needs to change in light of the IoT rests precisely on the question of how novel these challenges are; that is, the degree to which the challenges are genuinely new will dictate whether our access control models — be they descriptive or prescriptive (as described in Subsection 2.1.4.2) — need only modification or overhaul. This is still very much an open question [125].

3.1.3 Overview of Proposed Challenges for Internet of Things Access Control

In the next section we will assess to what extent proposed challenges from the IoT access control literature answer this question. These challenges (3.2.1.1–3.2.1.5) are commonly cited in surveys (see, e.g., [135]) as well as individual papers. We will assess these challenges, bearing in mind the following points of clarification.

First, there is no widespread agreement on the meaning of the terms used to describe these challenges and, as we will see, terms are often conflated with one another or used collectively; in short, the syntactic agreement in terminology is rarely reflected semantically. We will therefore have to be careful to describe what specific

challenges are being posed for each term. In cases where challenges involve the traditional requirements (e.g., Multi-User, Dynamic, and Fine-Grained) described in Subsection 2.2.2, we will make this link clear. When we assess that a proposed challenge is covered by a traditional requirement, we will use this as an argument for IoT access control not being significantly novel and therefore not requiring an overhaul.

Second, it is not always explicit in the literature whether proposed challenges are challenges for the IoT broadly or for access control specifically. In general, we will only consider as genuine challenges those challenges that involve access control, so we will be careful to draw out the two cases where necessary.

Third, it is often unclear in the literature whether proposed challenges are challenges for smart home access control in principle, or just challenges for how smart home access control is currently implemented. As we will see, this is problematic, as there is a significant mismatch between what papers in the literature expect and what consumer implementations achieve. When a proposed challenge is covered by a traditional requirement, but not implemented in today's smart homes, we will again use this as an argument for IoT access control not being significantly novel and therefore not requiring an overhaul.

Each of these three points will therefore be relevant as to whether challenges posed in the literature call for new access control models for the IoT. We will summarise these challenges in Subsection 3.2.2, leading to an analysis of what we *do* think is a novel challenge.

3.2 Challenges for Internet of Things Access Control

3.2.1 Proposed Challenges from the Literature

3.2.1.1 Smart Home Multi-User Controls

For Rotondi et al [138], the challenge of multi-user controls in the Smart Home arises from the *"unbounded number of interacting subjects (devices, applications, humans)."* In that access control should be scalable both in the sense of number of subjects and number of types of subjects, this challenge is captured by the Multi-

User requirement. It also bears on the Dynamic requirement, whereby the setting of policies should be scalable with respect to the subjects and objects within the Smart Home.

As Mohammad et al [120] point out, it is not uncommon for a smart home to be designed as *"a single-user domain."* This means that in practice, typical smart homes not only fail to meet the Multi-User requirement; they also fail to meet the Fine-Grained requirement, because a single user might as well have full access to all devices; and the Dynamic requirement, because the *"home owner is the only user responsible for having control over smart devices"* [120]. Similarly, the need for users to access resources anytime and anywhere falls within the definition of the Dynamic requirement.

The specific variety of users in the Smart Home has led to a body of literature in human-computer interaction. One challenge in this area involves the fact that administrators in the smart home are typically not technology experts [173]. This means that they may be *"unaware of security vulnerabilities and the protection mechanisms needed for the home environment"* [31]. This state of affairs is quite unlike the organizational environment, in which trained administrators are dedicated to understanding and implementing the security goals of the organization.

Relatedly, the challenge raised (e.g.,) by Zeng et al [174], is that *"smart homes can be focal points of conflict between people in the home, both due to explicit malice (e.g., abuse) and due to ordinary conflicts between household members."* They continue, citing examples involving *"conflicts arising due to differences in opinion on thermostat setting [75, 173], due to conflicting goals between parents and teens in the context of entryway surveillance [167], or due to the potential use of recorded evidence in household disputes [46]."* Though conflicts of this nature are not unheard of in organizational settings, we recognize that the relationship between users in the Smart Home is different from that between colleagues in organizations, and the Smart Home is therefore likely to give rise to new kinds of conflict.

Despite this, the conflicts described are declarative in nature (see Subsubsection 2.1.4.3), involving intended, higher-level policies. They therefore fall out of

scope from direct analysis in this thesis, though we will return to discuss them in Chapter 6. Where transparency is raised as a potential solution to the power imbalances between users in the Smart Home — for example, to give greater power to users to know what access controls are available to them (see, e.g., [174]) — this issue touches on the Transparency requirement, which we will again discuss in Chapter 6.

3.2.1.2 Smart Home Fine-Grained Controls

Writing in 2012, Rotondi et al [138] state that *"existing smart home platforms mostly offer binary control mode where a user gets all the control or no control at all."* As Sikder et al [149] observe in 2020, this state of affairs has continued in the intervening years. Rather than meeting the Fine-Grained requirement, it is instead the case that coarse-grained access control is the standard in consumer IoT [136, 65]. He et al [86] offer relevant examples:

"Current home IoT devices have relatively limited affordances for access control. For example, the Nest Thermostat supports a binary model where additional users either have full or no access to all of the thermostat's capabilities. The August Smart Lock offers a similar model with guest and owner levels. Withings wireless scales let users create separate accounts and thus isolate their weight measurements from other users. On Apple HomeKit, one can invite additional users, restricting them to: (a) full control, (b) view-only control, (c) local or remote control. Some devices offer slightly richer access-control-policy specification. The Kwikset Kevo Smart Lock allows access-control rules to be time-based; an owner can grant access to a secondary user for a limited amount of time."

Where this challenge refers to limits of controls on IoT devices (e.g., the binary model of the Nest Thermostat capabilites), this falls under the Fine-Grained requirement; where the challenge is about delegating rights to other users (the Kwikset Kevo Smart Lock case), this falls under the Dynamic requirement.

Because IoT devices have a physical as well as a digital component, it is unsurprising that it is a challenge to afford users the scale of operations possible within the Smart Home. For example, this challenge does not simply involve controlling

access to sense and action operations on devices, it can also involve restricting the access to data from the device [91] (e.g., the historical entry logs from a smart lock).

3.2.1.3 Dynamic Smart Home Controls

Affording administrators appropriate policy management tools is also a commonly cited challenge. Similar to multi-user controls, this challenge is sometimes phrased in relation to the unbounded number of devices present in a smart home. For example, Pereira et al [129] indidicate that *"as the number of devices proliferates, and as chance encounters between unfamiliar devices become the norm, manual specification of access control policies becomes unscalable."*

Elsewhere, this challenge is identified as being about the structure of the Smart Home. Ravidas et al [136] declare that an *"IoT ecosystem is dynamic by design wherein the network topology and connectivity can constantly change [163]. For instance, [IoT devices] can leave the system or new [IoT devices] can join the system [52]."* Because new devices are often added in an ad-hoc manner, policy management itself becomes ad-hoc [31]. Insofar as these challenges relate to the quick, easy, and accurate writing of policies in the face of an ever-changing environment, these challenges are captured by the Dynamic requirement.

3.2.1.4 Smart Home Heterogeneity

'Heterogeneity' often, but not always, refers to two distinct features of the Smart Home, which we will refer to as *device heterogeneity* involving the varied capabilities of IoT devices (e.g., that thermostats, locks and TVs do different things), and *protocol heterogeneity* involving the varied mechanisms that IoT devices use to communicate to one another, access points, and the internet (e.g., that 802.11, ethernet and MQTT are different ways of sending and receiving data). Occasionally, these two versions of heterogeneity are not distinguished, being defined collectively (see, e.g., [119, 103]).

Device heterogeneity is a broad challenge that often falls out of scope of access control. This is because it is a general design challenge to equip IoT devices with the wide range of operations that users expect [138]. Where it does fall within scope of access control (see, e.g., [147, 37]), device heterogeneity boils down to the

following question: 'within one system (i.e., the Smart Home), how can we *control access* to the wide range of operations afforded by IoT devices?' This question is captured by the Fine-Grained requirement.

Similarly, protocol heterogeneity, sometimes referred to as 'interoperability', often falls out of scope of access control, describing the challenge of how devices can be designed to communicate together [172, 65]. In some papers, which we will discuss in Section 3.3, this challenge is more particular, posing the question of how IoT devices can communicate their access control decisions.

Ur et al [167] use the term 'heterogeneity' in a third sense, relating it instead to users and their potentially distinct needs. They state: *"The home is a heterogeneous environment. Access control must account for guests [98], children [36], and all manner of temporary workers and visitors [104]."* Insofar as access control must account for different users, this definition of heterogeneity is captured by the Multi-User requirement; insofar as different users require different controls, including temporary controls, this definition of heterogeneity is captured by the Fine-Grained requirement.

3.2.1.5 Other Posed Challenges

Other prospective challenges are raised in the literature relating to the IoT, though they are less common with respect to the Smart Home than those so far discussed. For example, in the IoT literature more broadly, the challenge of *resource constrained devices* is common, though this is usually raised in the context of industrial IoT, in which it is a challenge to design expressive operations and communications for computationally limited sensors and actuators [37]. Another context in which resource constraints are often pointed to is with respect to authentication, the point being that encryption is a strain on processing for resource constrained IoT devices [120]. While we will return to authentication in Chapter 6, it is beyond access control, as we have defined it (and as it is commonly defined), and is therefore beyond the main scope of this thesis.

Finally, it is sometimes mentioned that access control policies in the Smart Home should be flexible, allowing for rare exceptions. For example, Mohammad

et al [120] state that smart home access control *"should be be more tolerant with some changeable attributes and not too strict with the rules. For instance, a user may be out of the country and want to access smart home resources."* Insofar as this need reflects rare exceptions to specific policies, this is captured by the Break-Glass Mechanisms requirement; insofar as it calls for quick and easy policy management, it is captured by the Dynamic requirement.

3.2.2 Summary

As described in Chapter 2, Multi-User, Dynamic, and Fine-Grained controls (and Transparent controls and Break-Glass Mechanisms) are requirements of traditional access control and were recognised, in industry and academia, decades prior to the IoT becoming established. That these challenges have not been met through consumer IoT suggests a failure of engagement between academia and industry.

It also suggests that these challenges are challenges of extent, rather than of kind: the Smart Home may involve more complex interactions between users, it may require more complex controls, and more complex policy management, but these are not fundamentally new challenges for access control. That 'scalability' — in its various guises (e.g., that the Smart Home involves more users, more controls, and more policy setting) — is so often mentioned bears witness to this. As a consequence, if these were the only challenges for IoT access control, it seems more likely that existing models should be modified rather than rejected.

Insofar as these challenges, and other challenges (e.g., heterogeneity and resource constraints) are distinct from the three requirements, they either fall outside of the scope of access control (e.g., in the design of heterogeneous devices, or providing authentication for resource constrained devices), or point at something that is, genuinely, different in kind, namely decentralization, which we will address in the next section.

3.3 From Centralized to Distributed Access Control

As we touched on in our discussion of Figure 3.1, the typical Smart Home of today will involve the orchestration of multiple smartphone apps. This is suggestive of a

critical difference between smart home access control and traditional access control, namely that *the current smart home is not an environment that contains a single access control system*. This means that, even when there is a sole administrator (e.g., home owner), whose role is to set policies across the entire smart home, there are nevertheless multiple decision-making authorities that determine whether access requests are granted or denied. While this point is occasionally mentioned in the literature (e.g., *"we find that each device has its own siloed access-control system"* [167]; that there is *"user annoyance that may be caused by two apps"* [31]), it is rarely cited as a challenge in itself, and it is therefore rarely a challenge that is attempted to be resolved directly.

3.3.1 Defining Decentralized and Distributed Access Control

We can spell out what this challenge is in more detail by using the ACU model defined in Chapter 2. This will also allow us to distinguish these definitions from related definitions found in the literature.

- **Access control is *centralized*, within a system, if the system contains just one ACU.**
- **Access control is *decentralized*, within a system, if the system contains multiple ACUs.**
- **Access control is *distributed*, within a system, if the system contains multiple ACUs that communicate with one another.**

Roman et al [137] use a similar three-fold distinction between centralized, decentralized[4], and distributed access control. Unlike our distinction, theirs is based partly on the location at which access control decisions are made:

- Centralized access control is provided by one entity, typically in the cloud, remote from IoT devices. IoT devices are themselves "*passive: their only task is to provide data.*"

[4]While Roman et al initially frame their distinction in terms of 'decentralized' and 'distributed systems', their final taxonomy instead opts for the terms 'connected' and 'distributed systems'.

- Decentralized access control is provided by IoT devices that can "*actually process local information,*" but they do not share information between themselves.

- Distributed access control is provided by IoT devices that can "*retrieve, process, combine, and provide information and services to other entities.*"

The location at which access control decisions are made – be it in the cloud, at the 'edge' (i.e., within IoT devices), or in the 'fog' (somewhere in between, as described e.g., in [101]) – is, for us, not relevant to the distinction. Instead, our distinction rests on the number of, and connections between, the entities responsible for access control: if one entity is responsible for deciding access on behalf of all IoT devices in the system (e.g., the Smart Home), it is centralized; if multiple entities decide, it is decentralized; and if multiple entities collaborate to decide, it is distributed.

Consequently, we would, for example, describe a smart home with two devices, that defer their access control decisions to two remote servers in the cloud, as decentralized, while Roman et al would describe the same system as centralized.

However, insofar as Roman et al distinguish between decentralized and distributed access control in terms of the respective absence and presence of collaboration between decision-making entities, our definitions agree. This means that, for example, both our definitions would deem two IoT devices making their own access control decisions as decentralized, and the same two devices communicating to make an access control decision as distributed.

In summary, the difference between our definitions can be put in terms of decision-making architectures: our definitions describe the *logical* decision-making architectures within a system, involving *how* the entities that make decisions connect; Roman et al describe the *physical* decision-making architectures within a system, involving *where* the entities that make decisions connect.

While Roman et al's definitions have gained some traction in the literature (see, e.g., [125, 135, 92, 126]), we think that our definitions are more useful for modelling access control in the Smart Home. This can be borne out by looking at

the advantages of decentralized access control over centralized access control.

3.3.2 Arguments For Decentralized Access Control

The arguments for decentralized access control over centralized access control fall into two categories: *necessary* arguments, that relate to how we need to understand the current Smart Home; and *contingent* arguments, that relate to how we can develop smart home access control in the future. Using the model terminology introduced in Subsection 2.1.4.2, the former case is suggestive of new descriptive models, while the latter is suggestive of new prescriptive models.

The necessary arguments relate to the fact that access control in the current Smart Home is fundamentally decentralized. In spite of this fact, there exist a number of limited solutions that attempt to reduce decentralization (i.e., attempts to reduce the the number of access control systems needed to control all devices in one smart home). Examples include Home Assistant software [159] running on dedicated physical hubs to control multiple devices locally, as well as web-services such as IFTTT [93] to manage connections between devices online. Moreover, devices made by the larger manufacturers are typically controllable via one app, creating 'ecosystems' of devices [3, 79, 12]. However, not all devices currently on the market are controllable through just one physical hub [160], are compatible with IFTTT [95], or fall within one manufacturer's ecosystem [2, 81, 11]. It is therefore incredibly difficult to construct a smart home that is centralized, so most are not. It follows that in order to model the current Smart Home faithfully, we have little choice but to embrace decentralized access control.[5]

While we found that the proposed challenges in the literature, given in 3.2.1.1–3.2.1.5, were not credibly novel challenges, we do think that having an accurate descriptive model for the current Smart Home would help to alleviate these challenges. For example, designing separate ACUs for devices (as opposed to a single ACU for all) encourages less of a 'one size fits all' solution, allowing for a specific device to have levels of granularity of control appropriate to that device.

[5]Note that we are unable to describe the current Smart Home as 'decentralized' in the Roman et al sense: the Smart Home is not currently filled with IoT devices that are carrying out access control at the edge.

Even if it was possible in practice for access control in the smart home to be fully centralized (i.e. that access to every device could be controlled by one ACU), it would not necessarily be desirable. These are the contingent arguments for decentralized access control.

One clear benefit of decentralization relates to efficiency: no single access control system has to store every policy and decide every access request [85]. This means that the access control systems can be computationally simpler and run faster. Another benefit relates to reliability: there is no single point of failure if one access control system stops working [13].

The challenge of scalability of users and devices described in the previous section is also more likely to be met by decentralized access control, because decentralized access control is more compatible with the ad-hoc nature of the Smart Home, which involves dynamically changing users and devices. Moreover, because our (logical) definition of decentralization is flexible enough to be compatible with the location definition of decentralization found in Roman et al, we can also benefit from further advantages: for example, the increased security of fully local access control that is segmented away from the internet. While time will tell whether IoT access control will continue to become more fragmented, moving closer to the edge, based on current trends this appears to be the case.

However, this does not mean that embracing decentralization is a panacea that will solve all access control problems in the Smart Home; decentralization also has disadvantages relative to centralization. In general, we think that the most significant challenge to decentralized access control involves policy management. In particular, policy conflict detection and avoidance is considerably harder for decentralized access control. Intuitively, this is because different apps can collectively have contradictory policies that affect all users and devices in a single smart home. The next section will spell out how these conflicts arise, and why they are difficult to resolve.

3.3.3 Policy Conflicts in the Smart Home

Within one ACU, policy conflicts are inherently easier to detect and avoid, as all policies are stored in one place (the PAP) and can be analysed collectively (by the PDP and PIP). In contrast, when policies exist across multiple ACUs, reasoning about whether there is a conflict is not trivial. This can be explained by distinguishing between *local* and *global* policy conflicts.

3.3.3.1 The Global/Local Policy Conflict Distinction

A local policy conflict occurs when there is an inconsistent set[6] of policies within one access control system; a global policy conflict occurs when there is an inconsistent set of policies across multiple access control systems. More specifically, a local policy conflict occurs when the set of policies in a single ACU's PAP is inconsistent, while a global policy conflicts occurs when the union of sets of policies across multiple ACU' PAPs is inconsistent.

For example (see top of Figure 3.2), suppose a smart TV app allows control of the TV to be set automatically, according to the time of day. Two policies are set within the app (which, for simplicity, will be assumed to be the access control system): *'If it's 5PM, then turn the TV on'*, and *'If it's 5PM, turn the TV off'*. This gives rise to a local policy conflict, as these two policies cannot simultaneously hold within the one access control system: at 5PM the TV will need to both switch on and switch off, which is impossible.

In contrast (see bottom of Figure 3.2), suppose a smart air conditioner app allows an air conditioning unit to be controlled automatically, according to the temperature in the room; and a smart window app allows a window to be opened and closed automatically, according to the temperature in the room and whether the air conditioning unit is on. Three policies are set: the policy *'If it's above* $25°C$ *turn the air conditioning unit on'* is set on the air conditioning unit app, and the policies *'If the temperature is above* $25°C$*, then open the window'*, and *'If the air conditioning unit is on, then close the window'* are set on the smart window app. This gives rise

[6] In general, a set is inconsistent when the assumption that all elements of that set are true leads to a contradiction [34].

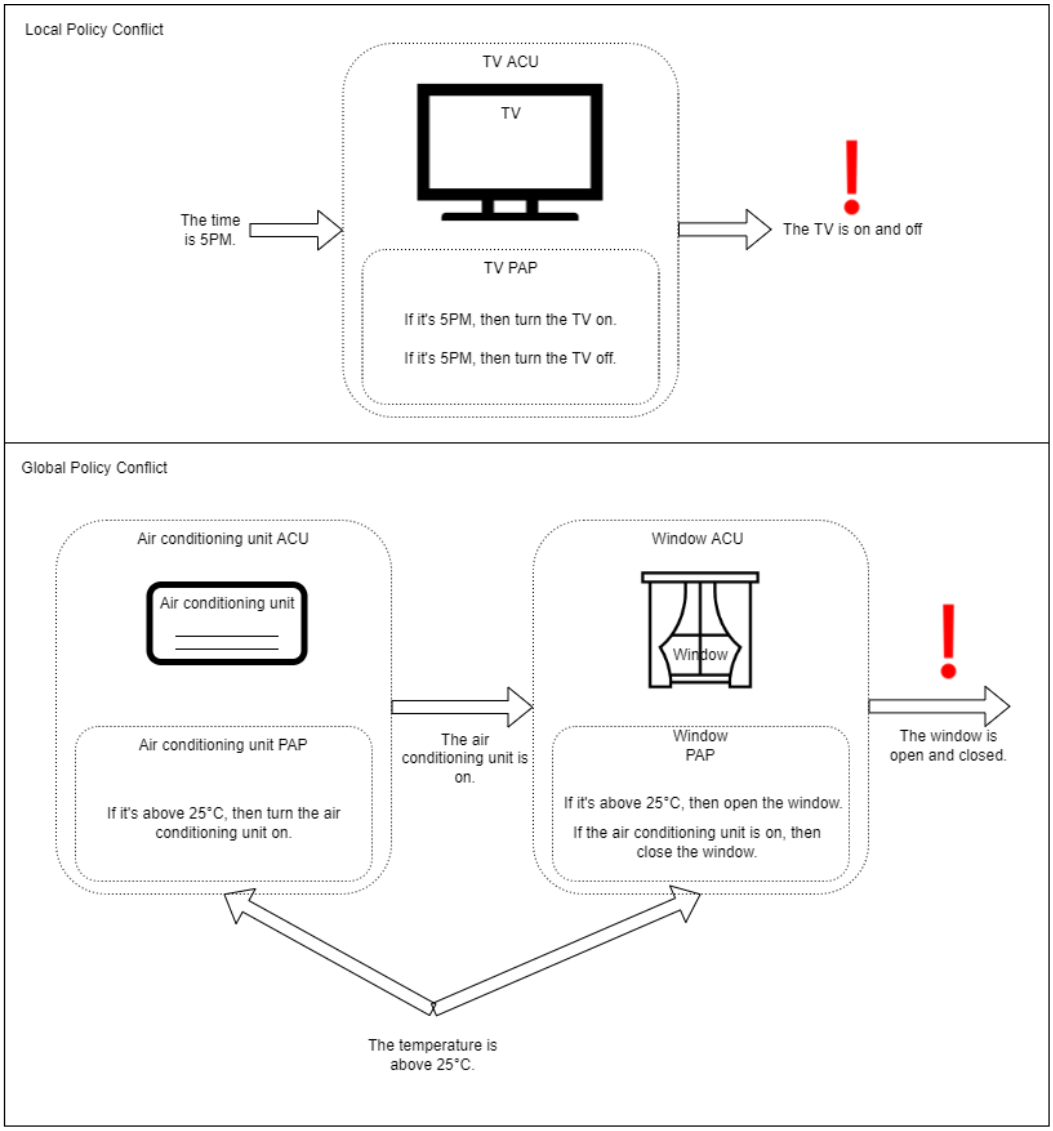

Figure 3.2: Examples showing the distinction between local and global policy conflicts. In the top example, there are two contradictory policies existing in one ACU (the TV PAP). The TV PAP is therefore inconsistent and, when the time is 5PM, the TV will be in the impossible state of being both on and off. In the bottom example, no individual PAP is inconsistent, but the union of the air conditioning unit PAP and the window PAP *is*. As a result, when the temperature is above 25°C, the window will be in the impossible state of being both open and closed.

to a global policy conflict, as these three policies cannot all hold: if the temperature in the room is above $25°C$, the air conditioning unit will turn on, and the window will need to both open and close, which is impossible.

In the latter case, the feature that makes this an example of a global policy conflict is that the three policies exist across more than one access control system. If there was only one app (or physical hub, or cloud-based server) which stored and checked against policies for both devices, the policy conflict would be local. Indeed, the local variant of this example is adapted from Alharithi [8] and is an example of a policy conflict within IFTTT.

In what follows, we will focus on one example of a global policy conflict, first introduced in Chapter 1 (shown again in Figure 3.3). This example is a form of privilege escalation [168] and has been recognised in the literature [109, 171]. It is most readily seen as affecting the confidentiality and integrity of the smart home. More generally, policy conflicts can affect the confidentiality, integrity, availability, and safety of information systems [166]. Detecting and resolving policy conflicts is therefore an important security and safety issue.

In discussing the Running Example in Chapter 1, we noted that the admin of the smart home wants to give the smart speaker access to the smart lock (Policy 1, grey arrow), the guest access to the smart speaker (Policy 2, grey arrow with question mark), and deny the guest access to the smart lock (Policy 3, grey arrow with cross).

Reasoning about transitive access suggests to the admin that there is policy conflict in this system: the guest can use its access to the smart speaker (Policy 2) to access the smart lock (through Policy 1), giving the guest access to the smart lock; this means that Policy 3 both holds and does not hold, and a conflict has emerged. The admin should therefore decide against giving the guest access to the smart speaker. However, we argued that this solution is not general, relying on the admin's ability to reason about transitive access, which may be very complicated with many subjects and objects in the system.[7]

[7]It may be questioned whether a better solution to the problem may be to give fine-grained control to the guest's access to the smart speaker. For example, if the guest were only given access to play

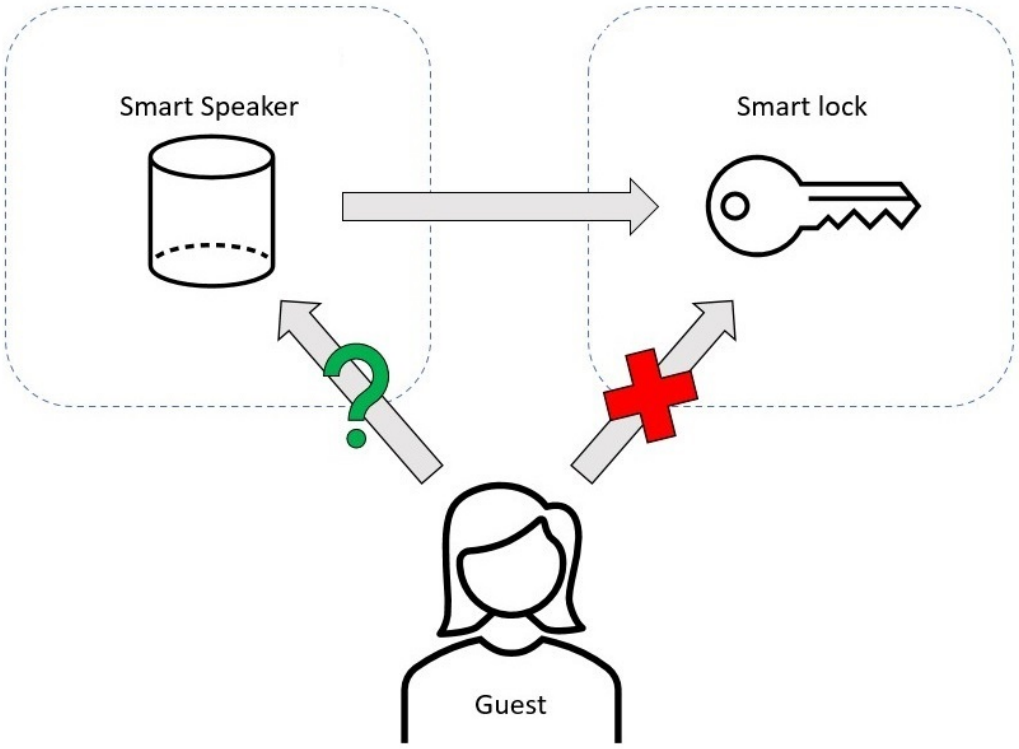

Figure 3.3: The main problem of the thesis: resolving policy conflicts for decentralized access control. Setting policies for devices in separate ecosystems can lead to conflicts in the broader system of the Smart Home. How can these conflicts be resolved in a systematic way?

In a centralized system, resolving the conflict is relatively straightforward as the collective policies are stored in one place and can be reasoned about as a whole. Indeed, this is essentially the reasoning carried out by the admin. The real question is how such a process can work when the policies are stored separately. A solution to this question necessarily involves some information sharing between the separate ACUs.

music on the smart speaker, then they wouldn't be able to use the smart speaker's access to the smart lock. This is an ad hoc solution that only pushes back the real problem, leaving it open to appear again. For example, it may be that allowing the guest access to the smart speaker's music-playing functionality also gives that guest access to previously played songs, when they played them, and other data that the admin might consider sensitive. The point is that the same problem of transitive access arises for fine-grained controls, just as it does for the simple case of unconditioned access involved in the Running Example. Solving the simple case therefore informs us as to the broader solution involving fine-grained controls.

3.3.4 Summary of Smart Home Access Control Requirements

In order to reason about the Running Example — and more generally, to understand how global policy conflicts can be resolved locally — we will need a model. This model will first need to be capable of representing the smart speaker and smart lock's access control systems. Second, the model will need to be capable of representing both access control systems simultaneously. Finally, the model will need to be capable of representing communication between the smart speaker and smart lock's access control systems that facilitates the resolution of policy conflicts.

A model that meets these requirements will be an access control meta-model; that is, a model of access control models [22]. We generalize the requirements for this meta-model as follows.

- **Expressive Requirement:** the proposed model should be capable of capturing the traditional requirements described in Chapter 2;
- **Decentralized Requirement:** the proposed model should be capable of representing multiple ACUs simultaneously;
- **Distributed Requirement:** the proposed model should be capable of representing the communication between ACUs that allow for their collaboration, in order to resolve global policy conflicts.

It is now possible to assess to what extent related work in the literature meet these requirements.

3.3.5 Related Work on Distributed Access Control and Policy Conflicts in the Smart Home

Policy conflict resolution in the smart home is considered by Liang et al [110], with further developments in [157, 42, 113]. These papers focus in particular on conflicts generated from IFTTT [93]. As IFTTT provides a centralized web-based service for the formulation of access control policies, these papers do not consider or account for global policy conflicts that arise from multiple ACUs. They therefore fail to meet the *decentralized* and *distributed* requirements.

Celik et al [42] look at the 'violations' that occur in multi-app environments (i.e., global policy conflicts). However, the graphical models they use for avoiding these conflicts are constructed at run time in a centralized device (the 'IoTGuard').

Liu et al [113] present a more sophisticated approach to statically resolving IFTTT policy conflicts that goes beyond simply *"reporting the conflict [...] and blocking the conflicting behaviour"* to provide so-called 'remedial actions'. Remedial actions are alternative, operationally conflict-free access requests that bring about the same intended behaviour (i.e., declarative policies). While their approach still rests on a centralized architecture, their methods may be general enough to extend any policy conflict resolver (centralized or decentralized). We will return to this issue in Chapter 6.

Similarly, most access control models developed for reasoning about the smart home have been centralized [136]. One exception includes the model provided by Hernandez-Ramos et al [88], in which capability-based access control is implemented in resource constrained devices, enabling multiple devices to carry out their own access control. It is not clear that their approach is general enough to model the complexity of devices found in a typical smart home. It is therefore unlikely to meet the *expressive* requirement. They also do not consider the resolution of global policy conflicts, and therefore fail to meet the *distributed* requirement.

In a position paper, Calo et al [39] argue that current access control is inappropriate for the IoT and should be replaced with a localised approach in which *"the intelligence to provide access control [is] embedded with the resources themselves"* so that *"resources generate and manager their access control policies dynamically on their own"*. Their focus on collections of access control systems is consistent with our own view. While their paper raises the issue of policy conflicts emerging from the *"dynamic interactions among different intelligent systems"*, they do not go into any detail about how they arise, nor do they describe how they should be detected and resolved.

The ACU model, in its guise as the XACML architecture [152], is putatively a model that considers policy conflicts. The XACML specification does not de-

scribe a decentralized architecture; instead, it describes a non-normative centralized architecture [56]. Also, as Mazzoleni et al [116] point out, the pre-defined policy combining algorithms (e.g., 'deny-override') are not applicable to resolving policy conflicts when multiple entities are involved. They therefore fail to meet the *decentralized* and *distributed* requirements.

There have been a number of attempts to decentralize parts of the XACML architecture. Lorch et al [114] describe a model suitable for grid computing, consisting of a centralized PDP that checks policies within decentralized PAPs. We think it unlikely that partial access control decentralization is suitable for the Smart Home, which involves multiple access control systems (i.e., multiple PAPs *and* PDPs). Similarly, Diaz-Lopez et al [58] describe decentralized 'master' and 'slave' PAPs in order to model access control in hierarchical organizations. Though they highlight the need to avoid policy conflicts that arise from interactions between security domains, they leave this issue for future work.

Finally, Charaf et al [45] call for a fully decentralized extension to the XACML architecture appropriate for modelling the decentralized IoT. Introducing the notion of a XACML 'module' that consists of a PEP, PIP, PDP, and PAP, they propose that each IoT device should be assigned *"its own XACML module in order to manage users' requests"*. As we will describe in the next chapter, we think this is precisely the sort of model that is needed for the Smart Home. However, Charaf et al give no description of how multiple modules (i.e., ACUs) should interact and therefore how they may avoid policy conflicts collaboratively. They therefore fail to meet the *distributed* requirement.

Chapter 4

Smart Object-Oriented Access Control

Make everything as simple as possible, but not simpler.

A clever person solves a problem. A wise person avoids it.

Albert Einstein

In this chapter:

- we identify and define the key principles of object-orientation based on object-oriented programming and design;
- we use the key principles of object-orientation to build a new meta-model for access control (**RQ 2**);
- we clarify the relationship to previous object-oriented access control work in the literature;
- we apply the meta-model to resolve the policy conflict described in the Running Example (**RQ 3**).

In Chapter 2, we described prominent historical access control models and gleaned from them desirable model properties, such as multi-user controls, fine-grained controls, and dynamic controls. We concluded Chapter 2 showing that the Access Control Unit (ACU) Model, based on the XACML architecture, is capable of capturing these historical models and their desirable properties.

In Chapter 3, we argued that the key challenge to the ACU Model, with regards to the Smart Home, is not, as some authors have suggested, the absence of any one of the desirable properties gleaned from historical access control, but instead concerns centralization: the Smart Home of today contains multiple access control systems, so no one model (i.e., a model that represents just one ACU) can faithfully describe it.

While, in principle, smart homes can be constructed in which access to every device can be controlled in one place (i.e., centrally), we argued that in practice this is very difficult (e.g., because of the wide range of current vendor's independent ecosystems). We referred to this argument against centralization as necessary and descriptive, as it rests on how access control within typical smart homes exists today.

Moreover, we argued, even if centralization were possible in practice, disadvantages such as having a single point of failure and problems relating to decisions being made away from devices, such as longer times for access decisions to be made, would make the Smart Home less secure than if access were controlled in multiple places simultaneously. We referred to this argument as contingent and prescriptive, as it states that, irrespective of how the Smart Home currently exists, the advantages to decentralized access control outweigh the advantages to centralized access control. Furthermore, we noted that recent trends in fog and edge computing suggest that Internet of Things access control is likely to become more decentralized, not less so.

Finally, we observed that the significant challenge for decentralized access control involves the processing of access requests in multiple locations with an awareness of the security of the smart home as a whole. This challenge is embodied in the Running Example, in which the global policy conflict of a transitive

access attack is shown.

Based on these arguments, at the conclusion of Chapter 4, we defined three requirements for modelling Smart Home access control: the model must be Expressive (i.e., capable of capturing desirable historical model properties), Decentralized (i.e., capable of representing multiple ACUs within one system), and Distributed (i.e., capable of representing appropriate communication between ACUs).

Chapter 2 showed that the ACU model satisfies this first requirement, and Chapter 3 showed the applicability of the first requirement to the Smart Home. Creating a model that *theoretically* satisfies the latter two requirements will be the focus of this chapter. (Showing that this model *practically* satisfies the latter two requirements by implementing it within a prototype smart home will be the focus of Chapter 5.)

To this end, we propose a novel access control meta-model — Smart Object-Oriented Access Control (SOOAC) — which models the communication of multiple ACUs within one Smart Home. In Section 4.1 we lay the conceptual foundations for this model by describing the object-oriented paradigm, its principles, and its relation to distributed access control. We also clarify the relationship between our usage of these principles and existing literature which sometimes refers to 'object-oriented access control'. We then apply object-oriented principles to develop a simplified ACU model that is capable of representing the local version of the Running Example. We build on this model to develop SOOAC, which addresses the (full) Running Example. Finally, in Section 4.2.3, we outline how this simplified ACU model can be extended to account for more complex forms of access.

4.1 Object-Orientation

To meet the Decentralized and Distributed requirements, we follow design principles from the object-oriented paradigm. These principles were initially developed for programming languages [50], but have since emerged as a more general set of design principles [27]. As Achaur [7] notes, *"the object-oriented paradigm has attracted much interest because it achieves data abstraction [and] software modu-*

larity in a natural way. Operations on an object are invoked by sending messages to the object. Thus, communicating among objects maps readily to the communication in a distributed system." In overview, it is precisely object-orientation's ability to represent modules that communicate that we will leverage in the construction of our own meta-model. We describe the principles more specifically in what follows.

4.1.1 Principles of Object-Orientation

Abstraction. This principle encourages *"taking away inessential features, until only the essence of [concepts] remain"* [89]. We will use this principle repeatedly in the construction of our model, but we will be careful to state whenever it is applied and give justifications for its application. Where features are removed that might be considered essential, we will describe either how they are not, or how they can be recovered. In this way, we adhere to Einstein's maxim quoted at the beginning of the chapter: we *"make everything as simple as possible, but not simpler."*

Modularity. This principle states that our designs should consist of objects that have all the properties (or 'fields') and behaviours ('methods') necessary to be self-contained entities [117]. This principle will mainly be applied to represent access control units (ACUs) as objects in order to satisfy the Decentralized model requirement. These will not be the only types (or 'classes') of object we will consider, though. In particular, IoT devices can be seen as objects. For programming, modular code should behave the same way wherever it is put (i.e., it will be 'portable'); for devices, we want them to be secure wherever they are (i.e., to always fall within the reach of an ACU).[1]

Encapsulation. This principle states that objects cannot be interacted with arbitrarily, but only through their pre-defined methods [117]. Such encapsulated systems consist of objects that interact through message-passing. For programming, this has the consequence that well-behaving objects will continue to behave well no matter how they are interacted with; in the present context, the intention is that

[1]The principle of Modularity is particularly well-suited to the idea that devices carry around with them their own ACU (i.e., when access control occurs 'at the edge'), but we do not insist on this restriction; we will only assume that, insofar as an access control system contains everything it needs to make an access decision, an ACU will always determine access to all devices in the smart home.

secure objects will continue to be secure no matter how they are interacted with. This principle will be particularly useful when we come to satisfy the Distributed model requirement for collaborating ACUs. It is also useful throughout, though, to focus attention on the encapsulation boundaries (i.e., interfaces) between objects.

Persistence. This principle states that objects will continue to act as objects through time. Put another way, this principle ensures that an object always retains its structure (fields) and capabilities (methods). For example, an ACU (as it is an object) will continue to have a policy set through time, though the content of this policy set can change. Another specific case of this principle is that object names will remain unique and constant. This will be particularly useful when we come to the implementation in Chapter 5.

The impact of each of these principles will be more clearly apparent as we develop our model. Other object-oriented principles, such as inheritance and polymorphism, may also be useful in the security context of the Smart Home, but we leave this for future research. An alternative formulation of some of these principles, and its applicability to access control, is given in Cattermole et al [40].

4.1.2 Distinguishing SOOAC from Object-Oriented Access Control in the Literature

There is a considerable body of literature in the field of access control for object-oriented systems [29, 74, 30, 66, 158]. Much of this work took place in the 1990s and stemmed from the challenge of adapting and extending existing access control mechanisms in databases for use in object databases. A survey of this literature can be found in [111]. Somewhat confusingly, approaches in this area have sometimes been referred to in terms of (or similar to) 'object-oriented access control' [63, 64, 155]. It is therefore easily misunderstood to be access control that is object-oriented — the approach taken in this thesis — and not access control for object-oriented systems. The distinction, though, becomes clear when looking at the details: *"Object-oriented access controls [OOAC] simply deal with controlling the flow of messages among objects of an object database"* [63]; *"OOOAC intercepts the message sending process and applies access controls in that it evaluates*

whether the source objects is allowed to send the particular message to the target object" [47]. Moreover, unlike the present work, it is also clear that *"OOAC are not intended to replace legacy access control mechanisms which mainly have been designed and applied in non-object environments. Instead, they provide the basis for applying these concepts in true object-oriented environments"* [63]. As such, this work is still very much in a tradition of centralized access control at the level of systems.

Work in this field, and in the same spirit, is continuing to be done. For example, in [155], it is shown how role-based access control can be extended and applied to object-oriented software projects. An explicitly stated goal of this work is *towards* the centralization of access control and not away from it, and is hence quite different from the approach taken in this thesis. Similar work, with the same goal of centralization, is being done in databases for CRUD operations (create, read, update, delete) [130, 131, 165]. There are also related approaches to securing access in object-oriented programming languages, for example in Java [64] and JavaScript [102].

Again, this body of work must be clearly distinguished from the present one. The distinction is between access control for object-oriented systems on the one hand, and access control that is constructed from object-oriented principles (which allows multiple ACUs to communicate appropriately) on the other hand. Unlike the existing literature in this area, SOOAC is a meta-model firmly captured by the latter description.

4.2 Theory of Automated Policy Conflict Resolution

From the outset, we have argued that the significant challenge for decentralized access control involves the resolution of global policy conflicts. Before we address this problem head-on, we will first address the simpler, but related problem of resolving local policy conflicts. By modelling a centralized ACU, we will develop a theory of automated local policy conflict resolution and show explicitly how the local version of the Running Example can be resolved. We will build on this theory

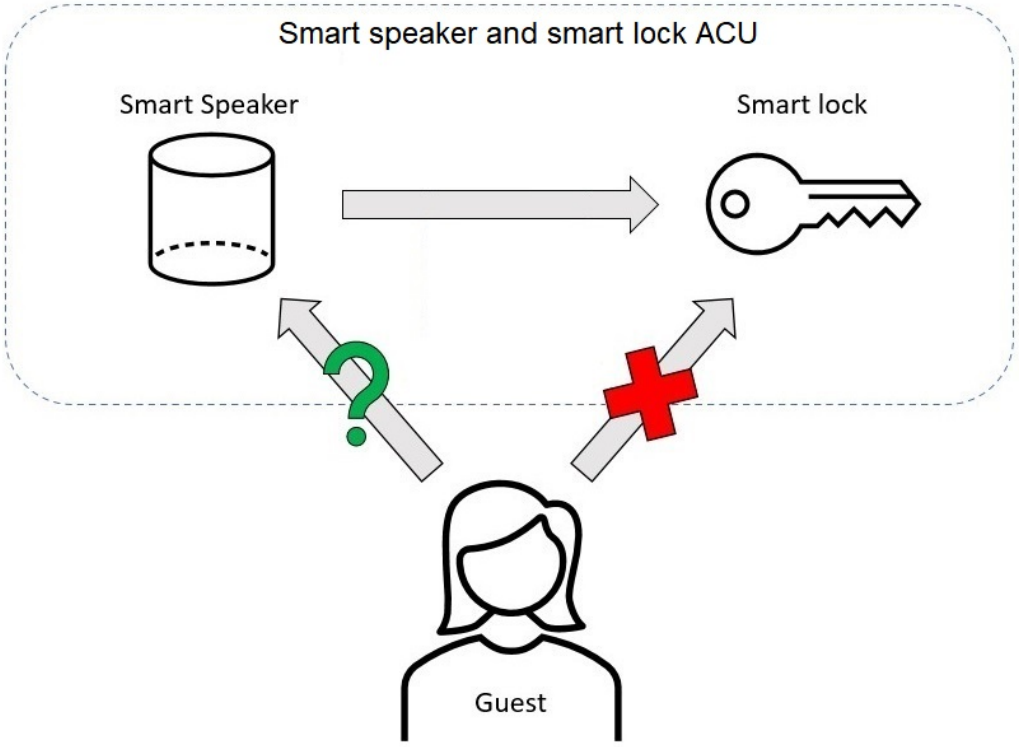

Figure 4.1: The local version of the running example. A single ACU controls access to a smart speaker and smart lock. By policy, the smart speaker has access to the smart lock, and the guest does not have access to the smart lock. How can the guest's access to the smart lock be determined automatically, so as to avoid a local policy conflict?

in Subsection 4.2.2 to develop the full theory which will address the (full) Running Example.

4.2.1 Automated Policy Conflict Resolution in Centralized Access Control

The local version of the Running Example is shown in Figure 4.1. The only difference between it and the (full, global) Running Example is that access to the smart speaker and smart lock is determined by a single ACU. Other than this, the admin faces the same problem: they have granted the smart speaker access to the smart lock; they have denied the guest access to the smart lock; and they want to automatically avoid the policy conflict of giving the guest transitive access to the smart lock via the smart speaker.

Much like the Running Example, this example involves unconditioned access and not access involving operations, roles, or attributes. The access control system we need to model is therefore very simple. By the object-oriented principle of Abstraction, we will restrict our focus to just this form of access. In Section 4.2.3, we will discuss how more complex forms of access (e.g., access to specific operations) can be recovered.

We define policies and access requests as pairs of device[2] *names.* For example, (a,b) indicates that the device a has access to the device b. Where appropriate, we will write $(a,b)?$ to indicate that a question is being asked about whether a has access to b. This can occur in two ways: when an access request is received by a PEP (as defined by Enforcement in Chapter 2), and when a policy set is queried for elementhood (as defined by Administration in Chapter 2). The distinction between these two cases, and the distinction between them and (non-interrogative) policies (e.g., (a,b)) will be clear from context. By the object-oriented principle of Persistence, we will require that device names remain constant and unique through time. Without this requirement we could not rely on the consistent interpretation of access requests and policies.

Also by the principle of Abstraction, we will restrict our ACUs to contain only PDPs and PAPs. This will provide clarity to our model without rendering it unfaithful at representing the local version of the Running Example. PIPs are superflous in this context, as we assume that an access request contains the subject's name; and we can assume that the role of a PEP in receiving access requests and returning access decisions is carried out by a PDP. As we are not using the XACML language as the format for our internal messages within an ACU (we are simply using pairs of device names), there is no need for the PEP's conversion functionality. When we come to implement our model in Chapter 5, it will be clear that the PEP's conversion functionality is not even necessary when resolving policy conflicts in practice, but we will consider this issue in more detail in Chapter 6. We will first

[2]Even when we refer to human users (e.g., admin, guest), we will always have in mind some device belonging to them. This is made more explicit when we come to implementation in Chapter 5: users will be their smartphones.

define the capabilities of a centralized PDP.

Centralized PDP Capabilities

A centralized PDP can:

(1) take in an access request (of the form (a,b)?);

(2) send an access request to a PAP (of the form (a,b)?);

(3) return an access decision ('True' or 'False');

(4) transform an access request, by substituting its first entry with its second entry, and its second entry with a variable[3] (e.g., (a,b)? will become (b,x)?);

(5) transform an access request by substituting its first entry with the first entry of a policy (e.g., (a,b)? and (c,d) will become (c,b)?).

Capabilities (1)–(3) above refer to standard access control abilities described in Chapter 2: (1) and (3) are captured by the Enforcement ability, and (2) is captured by the Decision-Making ability (it involves basing a decision on existing policies). We introduce the new capabilities (4) and (5). These involve transforming requests and policies, respectively. Intuitively, capability (4) enables the PDP to ask what further access the object of an initial access request has. In other words, when given a request (a,b)?, it enables the PDP to ask what b has access to. Capability (5) enables the PDP to generate the appropriate transitive request, given a previous request and a policy. We have specified capability (5) in general terms involving four devices (a,b,c,d), but in practice, as we will see, a and d will always refer to the same device. Capability (5) is therefore better understood as taking a chain $(c \to a \to b)$ (i.e., the policies (c,a) and (a,b)) and skipping a to form $c \to b$ (i.e., (c,b)). This will make more sense by considering how the PDP interacts with the PAP.

[3] A variable should not refer to any named device.

Centralized PAP Capabilities

A centralized PAP can:

(1) take in (i.e., 'write') a policy (of the form (a,b));

(2) store policies in a policy set (e.g., $\{(a,b),(b,c)\}$);

(3) take in an access request from the PDP and return a response ('True' or 'False'), based on whether the policy is in its policy set or not (e.g., if (a,b)? is the access request and $\{(a,b),(b,c)\}$ the policy set, then the PDP will return 'True');

(4) take in an access request from the PDP of the form (a,x)? and return all policies of the form (a,d) (e.g., if the policy set is $\{(a,b), (b,c)\}$ and (a,x)? is taken in, (a,b) will be returned).

Again, capabilities (1)–(3) refer to standard access control abilities described in Chapter 2: they are all captured by the Administration ability. We introduce the new capability (4) which enables the PAP to find new policies that link a known device to other devices. This capability acts between the PDP capabilities (4) and (5) in the following way. Suppose there is a chain $(a \rightarrow b \rightarrow c)$ consisting of the policies (a,b) and (b,c). The PDP takes the policy (a,b) and transforms it, by PDP capability (4), to form the new request (b,x)?. This request is sent to the PAP, returning the policy (b,c), by PAP capability (4). The PDP transforms this policy to the new request (a,c)? which will be sent back to the PAP. This process checks whether transitivity holds for the chain $a \rightarrow b \rightarrow c$. As shown, this approach is in the object-oriented spirit that *"the structure of the software should mirror the structure of the problem"* [27].

A centralized ACU, containing one centralized PDP and one centralized PAP, detects and resolves local policy conflicts by carrying out *Centralized In-Access Recursion*. This algorithm is shown in Algorithm 1 and is described as follows.

Algorithm 1: Centralized In-access Recursion algorithm for resolving an access request for a centralized ACU

1 An access request (a, b)? is sent to the PDP.
2 The PDP sends (a, b)? to the PAP.
3 The PAP checks whether (a, b) is in its policy set.
  3.1 If it is not, the PAP sends the PDP 'False'.
    3.1.1 The PDP returns 'False'.
  3.2 If it is, the PDP transforms (a, b) into (b, x).
    3.2.1 The PDP sends (b, x)? to the PAP.
    3.2.2 The PAP returns all policies of the form (b, c) in its policy set to the PDP.
    3.2.3 The PDP transforms all (b, c) into (a, c).
    3.2.4 The procedure returns to Step 2, with (a, c)? instead of (a, b)?.
4 The PDP returns 'True'.

When an access request is made (Step 1) involving a subject device (a) and object device (b), it is first checked that the request exists as a policy (Steps 2 and 3). If it does not exist (Steps 3.1 and 3.1.1), the ACU returns 'False' and the procedure terminates. If it does exist, it is checked whether any policies exist that specify access between the original object (b) and other objects (c) (Steps 3.2, 3.2.1, and 3.2.2). New requests (of the form (a, c)?) are created between the original subject (a) and other objects (c) (Step 3.2.3). These new requests return to Step 1 in order to be recursively checked themselves (Step 3.2.4). If, once this process has been completed, no access request has returned 'False', the ACU returns 'True' (Step 4).

At each step of the algorithm it is possible to specify which PDP or PAP capability is being put to use: Step 1 relies on PDP (1); Step 2 relies on $PDP(2)$; Steps 3 and 3.1 rely on PAP (3); Step 3.1.1 relies on PDP (3); Step 3.2 relies on PDP (4); Step 3.2.1 relies on PDP (2); Step 3.2.2 relies on PAP (4); Step 3.2.3 relies on PDP (5); Step 3.2.4 relies on PDP (1); and Step 4 relies on PDP (3).

Centralized In-Access Recursion forces an access control decision to be withheld by an ACU until every recursive chain of requests derivable from existing policies have themselves been decided upon. If either the original request or any subsequent request fails to be satisfied by a policy, access is denied; otherwise, access is granted.[4] This process occurs at 'runtime', that is, when an access request is made to an ACU.[5]

Intuitively, this process is perhaps best understood in terms of trust, which is a relationship that is converse to access (i.e., a has access to b if and only if b trusts a). For a device b to trust a device a, it is must be checked what devices also trust b. If a device c trusts b, then it should be checked whether c trusts a. This process is necessarily recursive, as any device that trusts c should also be checked, and so on.

Centralized In-Access Recursion makes formal the informal reasoning used by the admin in the Running Example, allowing for the local policy conflict to be detected and guest access denied.

Let a, b, c refer to the guest, the smart speaker, and the smart lock, respectively. The local version of the Running Example is resolved as follows (see in parallel with Figure 4.2).

[4]It should be noted that we do not consider access chains that form loops. For example, if a request (a,b)? is made and the policy set contains the policies (a,b) and (b,a), In-Access Recursion will enforce the checking of (a,b)? and then the reflexive request (a,a)?. Access will be denied as (a,a) is not in the policy set. If we allow reflexive access to always hold (as would make intuitive sense), In-access Recursion will create the access request (a,b)? again, and an endless cycle will ensue. Accounting for this is not problematic: once a reflexive access request is made, access should automatically be granted and no further recursion should occur (on that chain). For simplicity, as per the object-oriented Abstraction principle, our definition of In-Access Recursion avoids this minor complication by ignoring loops.

[5]For a more detailed discussion about the distinction between static (pre-runtime) and dynamic (runtime) policy setting, see Dunlop [61]. See also Chapter 6 for our own contribution to this issue.

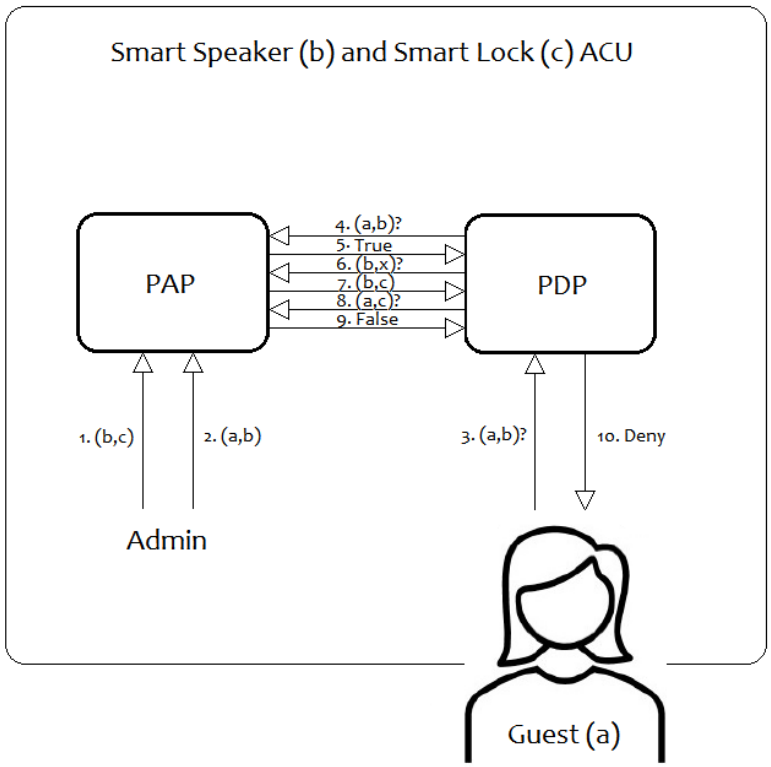

Figure 4.2: Flow chart showing how Centralized In-Access Recursion detects and resolves the Running Example in a centralized manner. Note that there is only one ACU, containing one PDP and PAP; that is, access to the smart speaker and smart lock is controlled by a single ACU.

1. The admin writes the policy (b,c).
2. The admin writes the policy (a,b).
3. The guest sends the access request (a,b)? to the PDP.
4. The PDP sends (a,b)? to the PAP.
5. The PAP checks whether $(a,b) \in \{(a,b),(b,c)\}$. It is, so the PAP returns 'True' to the PDP.
6. The PDP transforms (a,b) into (b,x)? and sends it to the PAP.
7. The PAP returns (b,c) to the PDP.
8. The PDP transforms the policy (b,c) and the access request (a,b) into (a,c)? and sends it to the PAP.
9. The PAP checks whether $(a,c) \in \{(a,b),(b,c)\}$. It is not, so the PAP returns 'False' to the PDP.
10. The PDP returns denies access to the guest.

4.2.2 Automated Policy Conflict Resolution in Distributed Access Control: SOOAC

We have designed a centralized ACU with a single PDP and PAP. The formulation of a centralized ACU was guided by the object-oriented principles of Abstraction and, to some extent, Modularity, but only in so far as an ACU is understood as a standalone security object that has everything it needs to make an access control decision. The full power of the Modularity principle comes when we understand the ACU as a class, of which multiple instances can exist within the same model. This satisfies the Decentralized model requirement. This alone, however, will not be enough to avoid policy conflicts: it is not enough to have multiple disconnected ACUs; they will need to interact. It is here where the object-oriented principle of Encapsulation will be useful. To understand how, we will need to consider what design choices we have.

Given how we've defined an ACU, to link two of them together, we only really have two choices: PDP-PDP connections or PDP-PAP connections. (PAP-PAP connections are not possible as a PAP has no ability to do anything itself; it can only respond to requests from a PDP.) Although a PAP can return policies to a PDP, this occurs within a single ACU, through *internal* message-passing. By the principle of Encapsulation, message-passing must occur using existing object methods. Hence we do not allow a PDP from one ACU to communicate to a PAP from another ACU. Instead, we rely on the *external* message-passing ability already present in a PDP, namely the ability to receive access requests. As described in Subsection 4.1.1, this will help ACUs to be more secure. The new capability of a PDP is given given below.

Distributed PDP Capabilities

A distributed PDP can:

(1)-(5) do everything a centralized PDP can;

(6) *make access requests to other ACUs' PDPs.*

ACUs will now be able to speak to each other, satisfying the Distributed requirement, but they will not necessarily have the right things to say. This is because, by default, an ACU will only store policies that relate to itself as the object of an access control request (i.e., ACU_b will only store policies of the form (a,b)). This means that an ACU will not, by default, be able to issue a request suitable for resolving the Running Example. For example, if PAP_b has a policy (a,b), and PAP_c has a policy (b,c), ACU_b and ACU_c will be connected, but only insofar as ACU_c has a policy that connects them. This means that when PDP_b receives the acecss request (a,b)?, it will not be able to make the request (a,c)?, because it has no policy (b,c) to transform. To rectify this, we introduce the following restriction to policy writing at a PAP.[6] (PAPs are otherwise unchanged.)

Simultaneous Policy Restriction

Whenever a policy (a,b) is written at PAP_b, it must be simultaneously written at PAP_a.

The simultaneous policy restriction ensures that every ACU stores all policies that refer to itself as object *or subject*. The restriction should hold for any devices intended to be guarded against transitive access conflicts. Enforcement of the restriction will depend on the policy-making authorities within the system at hand.[7]

The object-oriented principle of Persistence is also important for distributed ACUs: every named device must stay constant and unique through time. Without

[6]Interestingly, if subjects — rather than objects — were to hold the authority in access control decision-making (e.g., that an object a alone held the policy (a,b)), the Simultaneous Policy Restriction would not be necessary. This would, however, go against a seemingly unbreakable rule of access control, namely, that the things which want access should not be responsible for deciding access. It is nevertheless interesting to ponder whether certain trusted environments (which perhaps includes the Smart Home) give subjects such power.

[7]For example, in the transitive access attack example, the admin is assumed only to be the policy maker for the smart speaker and smart lock, not for the guest. This means that the simultaneous policy restriction only holds for these two devices. In general, the simultaneous policy restriction creates trusted zones of devices, outside of which, devices cannot be assumed to be free of policy conflicts. Where multiple policy-making authorities exist within one smart home (for example, in flat-shares), an additional mechanism will be required to enforce the simultaneous policy restriction. One potential candidate is sticky policies [43].

this, not only will internal policies stored at one ACU be inconsistent, but so will an ACU's ability to reliably send messages to other ACUs.

With these additions to PDPs and PAPs, we can provide the step-by-step procedure for resolving the transitive access example. To do this, we define *In-Access Recursion*, shown in Algorithm 2.

Algorithm 2: Decentralized, Distributed In-Access Recursion algorithm for resolving an access request sent to a decentralized, distributed ACU

1 An access request (a, b)? is sent to PDP_b.
2 PDP_b sends (a, b)? to PAP_b.
3 PAP_b checks whether (a, b) is in its policy set.
  3.1 If it is not, PAP_b sends PDP_b 'False'.
    3.1.1 PDP_b returns 'False'.
  3.2 If it is, PDP_b transforms (a, b) into (b, x).
    3.2.1 PDP_b sends (b, x)? to PAP_b.
    3.2.2 PAP_b returns all policies of the form (b, c) in its policy set to PDP_b.
    3.2.3 PDP_b transforms all (b, c) into (a, c).
    3.2.4 PDP_b sends all access requests (a, c)? to PDP_c.
      3.2.4.1 If PDP_b is sent 'False' for any request (a, c)?, PDP_b returns 'False'.
      3.2.4.2 Otherwise, PDP_b returns 'True'.

When an access request is made to ACU_b (Step 1) between a subject (a) and object (b), it is first checked that the request exists as a policy (Steps 2 and 3). If it does not exist (Steps 3.1 and 3.1.1), ACU_b returns 'False'. If it *does* exist, it is checked whether any policies exist that specify access between the original object (b) and other objects (c) (Steps 3.2, 3.2.1, and 3.2.2). New requests (of the form (a, c)) are created between the original subject (a) and other objects (c) (Step 3.2.3). These new requests are sent to every ACU_c in order to be recursively checked themselves (Step 3.2.4). If any ACU_c returns 'False' to PDP_b, ACU_b returns 'False' (Step 3.2.4.1); otherwise, ACU_b returns 'True' to the original request (Step 3.2.4.2).

There are two ways to interpret the communication between new objects that occurs in Steps 3.2.4 and 3.2.4.1: *In Series* and *In Parallel*. In the former case, which we will follow, the initial ACU (i.e., ACU_b) waits to hear back a response from the first new object (ACU_{c_1}), before the second (ACU_{c_2}), and so on. In the latter case, which we consider in Chapter 6, ACU_b can issue a request to ACU_{c_2} while it is waiting to hear back from ACU_{c_1}. With the In Series interpretation, In-Access Recursion carries out a depth-first search through devices in an access chain, stopping the search immediately if the initial ACU does not have transitive access. In effect, this means that an access control decision is withheld by an ACU until every recursively connected ACU has been asked whether it too would have granted the request if they had been asked. The work of checking policies and creating transitive chains is therefore outsourced from any one single ACU to all ACUs that have policies involved in the access chain. This ensures that if the initial ACU is not transitively closed with respect to other devices in the access chain, access is denied.

It is now possible to show explicitly how the Running Example can be avoided automatically. Let a, b, c refer to the guest, the smart speaker and the smart lock, respectively. Let their ACUs, PDPs and PAPs be indexed accordingly. The step-by-step process in which the global policy conflict is avoided is given as follows (see in parallel with Figure 4.3).

1. The admin writes the simultaneous policy (b, c) to PAP_b and PAP_c.
2. The admin writes the policy (a, b) to PAP_b.
3. The guest sends the access request $(a, b)?$ to PDP_b.
4. PDP_b sends $(a, b)?$ to PAP_b.
5. PAP_b checks whether $(a, b) \in \{(a, b), (b, c)\}$. It is, so PAP_b return 'True' to PDP_b.
6. PDP_b transforms (a, b) into $(b, x)?$ and sends it to PAP_b.
7. PAP_b returns (b, c) to PDP_b.
8. PDP_b transforms the policy (b, c) and the access request (a, b) into $(a, c)?$ and sends it to PDP_c.

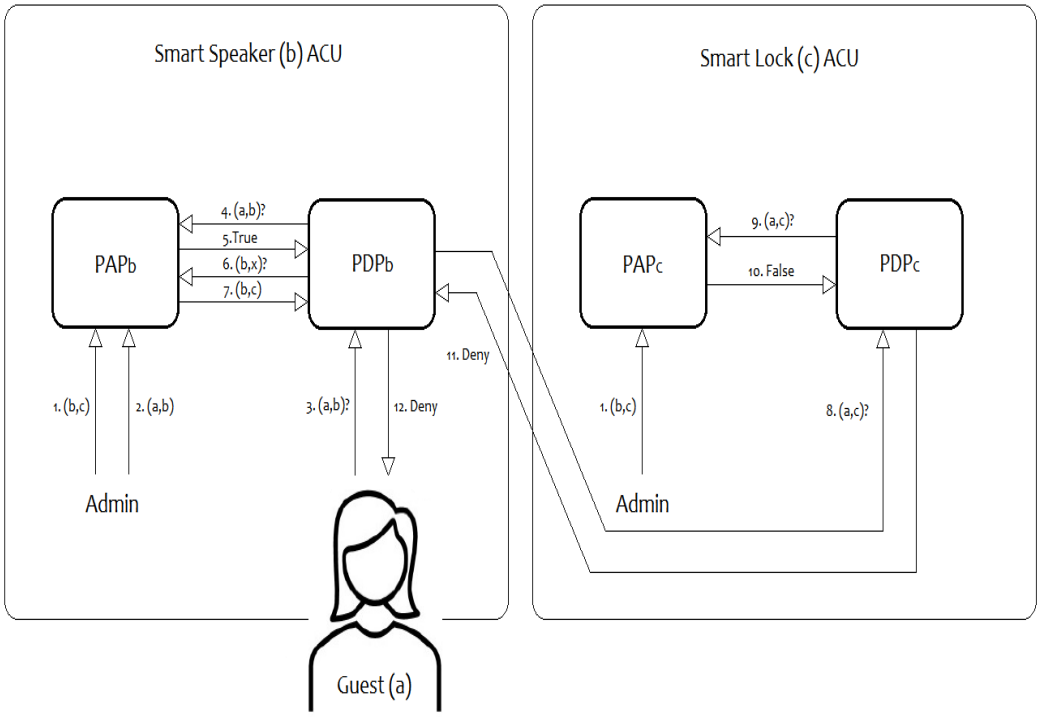

Figure 4.3: Flow chart showing how In-Access Recursion detects and avoids the transitive access example conflict. Note that there are two ACUs (with their own PDPs and PAPs) that communicate to avoid the policy conflict.

9. PDP_c sends $(a,c)?$ to PAP_c.
10. PAP_c checks whether $(a,c) \in \{(b,c)\}$. It is not, so PAP_c returns 'False' to PDP_c.
11. PDP_c returns 'False' to PDP_b.
12. PDP_b denies access to the guest.

4.2.3 Extension to Arbitrary Global Policy Conflicts

As shown, decentralized, distributed In-Access Recursion detects and resolves the global policy conflict in the Running Example. A crucial part of this process is that the smart speaker rewrites its policy (b,c), using the initial request $(a,b)?$, to form the new request $(a,c)?$, which is then sent to the smart lock to be checked. In general, *resolving global policy conflicts that arise from arbitrary models requires arbitrary policy rewriting.*

For conflicts arising from ACM models, in which permitted access to operations can be specified, policy rewriting should be indexed according to opera-

tions. That is, if an initial request is of the form (a,b,op), then the recursively generated new requests should be of the form (b,x,op). Returning to the example given in 3.3.3.1, if an initial request is $(Guest, SmartSpeaker, PlayMusic)$? and this exists as a policy, it should be checked whether any policies exist of the form $(SmartSpeaker,x,PlayMusic)$. As a result, if there exists a policy $(SmartSpeaker, SongList, PlayMusic)$, this will be rewritten to form the new request $(Guest, SongList, PlayMusic)$. If the admin is reluctant to allow the guest to play previous songs, this can be stated as a policy and the conflict can be automatically detected and resolved by denying the guest access to the *PlayMusic* operation on the smart speaker. Similarly, for ABAC, in which permitted access to operations can be specified according to attributes of the subject, object, and environment, new requests should be indexed according to the operation and all attributes.

For IFTTT-like conditional statements that specify the pre- and post-conditions of some event, new requests will be formed by the rewriting of events according to their pre- and post-conditions. Returning to the example in 3.3.3.1, if an initial access request $(> 25°, AC.ON)$? is made, this exists as a policy at the AC app, so it will be checked whether any policies of the form $(AC.ON,x)$ exist. There exists the policy $(AC.ON, window.close)$ at the AC app, so the new request $(> 25°, window.close)$? will be sent to the window app. This request will be denied by the window app as only the contradictory policy $(> 25°, window.open)$ exists, resulting in the initial request being denied by the AC app.

As described, the same fundamental elements needed to resolve the characteristic transitive access attack present in the Running Example are present when resolving arbitrary policy conflicts. In this way, the principles of Smart Object-Oriented Access Control can be applied beyond the Running Example to resolve arbitrary global policy conflicts. To formalise this precisely would require a general policy language, of which the XACML language is a good candidate. We leave this as further work.

Chapter 5

Implementation and Evaluation

> Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality.
>
> *Nikola Tesla*

In this chapter:

- we describe how Smart Object-Oriented Access Control (SOOAC) can be embedded within a prototype smart home;
- we show how the policy conflict present in the Running Example can be resolved in practice;
- we calculate the efficiency of SOOAC (**RQ 4.1**) and compare it to a centralized access control policy conflict resolver;
- we calculate the effectiveness of SOOAC (**RQ 4.2**) in a protype smart home;
- we summarise the advantages of SOOAC in quantitative and qualitative terms based on the implementation.

In Chapter 4, we described Smart Object-Oriented Access Control (SOOAC) and showed how it models IoT devices that detect and resolve global policy conflicts locally. SOOAC is based on the idea that policy conflicts can be avoided if devices, when issued with an access request, can rewrite their relevant policies appropriately and issue them as new requests to other devices, before they allow or deny access themselves. This process is recursive in the sense that every device issued a new request along the access chain carries out the same process. In this chapter we will embed the SOOAC model within a prototype smart home and compare its performance to centralized access control. More specifically, in Section 5.1, we show how individual ACUs can be assigned to IoT device in a prototype smart home. In Section 5.2, we describe the setups and results for two experiments that compare SOOAC with centralized access control. Finally, in Section 5.3, we conclude our comparative analysis by drawing on the advantages and disadvantages of distributed access control that we described in Chapter 3.

5.1 Embedding SOOAC in the Smart Home

We set up a smart home (in a number of different configurations, shown in Figure 5.1) consisting of seven smart devices: two iOS smart phones (admin and guest), an Ener-J smart lock, a Homepod Mini Siri smart speaker, a BT Hive smart bulb, a TP-Link smart plug, and a Samsung smart TV.

As the access control systems used by the various manufacturers are not open source, we are not able to directly equip the smart devices with the ability to recursively issue access requests. Instead, we assign each smart device (excluding the smartphones) its own Raspberry Pi that acts as the device's ACU. By the principle of Modularity, each Pi is a standalone entity which can completely determine access to its associated device, according to the design of distributed ACUs given in Subsection 4.2.2.

Each Pi (excluding the smart speaker's) interacts with their corresponding smart device through public APIs[1]. For the smart speaker, a more complex im-

[1] APIs allow for information to be sent to and received from IoT devices. In centralized systems, such as Home Assistant and IFTTT, APIs are used to issue commands to IoT devices in one place

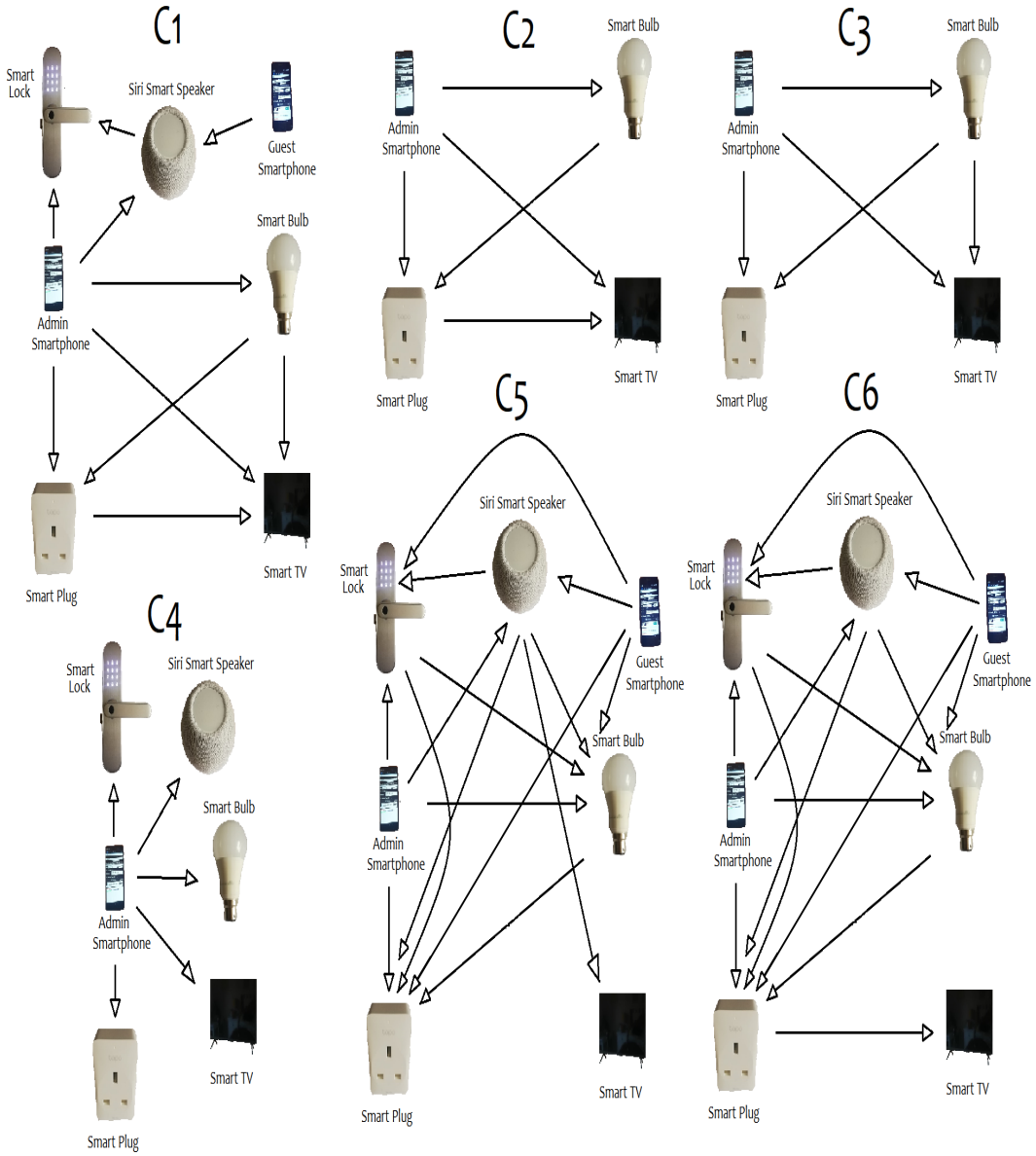

Figure 5.1: Overview of smart home setups showing configurations (C1–C6) of policies (arrows) for seven smart devices. Two experiments are performed on these configurations in Section 5.2: Experiment 1 gathers response times based on each possible policy as an initial request, and Experiment 2 checks for erroneous access decisions based on random initial requests.

plementation is necessary, as the speaker acts as its own interface (i.e. through its voice interface). This will be explained at the end of this section. By the principle of Abstraction, the behaviour of each smart device can be understood as being completely determined by the Pi which determines access to it. The distinction between each Pi and its corresponding smart device is therefore not significant in our implementation. We will therefore, from time to time, refer to smart devices when, technically, we mean their respective ACUs.

Each Pi runs an identical Python script (given in Appendix B) that performs the In-Access Recursion algorithm described in Algorithm 2 in Subsection 4.2.2. This script also contains the ACU's set of local policies, implemented as a list of pairs of IP addresses. By the principle of Persistence, the Pis are assigned permanent names, which we implement as static IP addresses. These addresses are used to issue requests reliably between the Pis.

By the principle of Encapsulation, each Pi is capable of receiving requests as messages. As we argued in Subsection 4.2.2, this ensures that the ACUs will be secure, as they cannot arbitrarily interfere with another and can only message one another. To facilitate a uniform request language, we implement a simple UDP protocol.[2] The Pis can receive three types of input message: a request to perform a device operation ('ACTION'), an access request ('REQUEST'), and a response to an access request ('RESPONSE'). REQUESTS and RESPONSES work exactly as described in Subsection 4.2.2. The two smartphones can issue ACTIONS from the "UDP Sender/Receiver" app [5] of the form "ACTION operation-name", indicating the operation the user wishes to perform. As our prototype does not facilitate fine-grained controls on operations, we only consider one operation on each device (see Appendix B for the operations for each device). REQUESTS are of the form "REQUEST IP-address-sender IP-address-receiver". RESPONSES are "RESPONSE True" or "RESPONSE False". All the Pis can issue as well as receive these three

[1, 94]; in keeping with our overall approach, for SOOAC, we use APIs in a more decentralized manner: each Pi associated with a device has an API that allows the Pi to control it. It should be stressed that besides their specific APIs and local policies, each Pi is otherwise identical.

[2]All Pis use the Python *socket* module [4]. Alternate protocols could be used (e.g., TCP, SSL, MQTT). In a deployed implementation, the payloads of communications should be encrypted, but this is beyond the scope of this thesis. We return to the challenge of using UDP reliably in Chapter 6.

types of message.

Each Pi transitions through a series of states, embodying the SOOAC model:

- they start in a listen state, waiting for an ACTION, REQUEST, or RESPONSE;
- if they receive an ACTION, this is converted to a REQUEST with the sender IP address and their own IP address;
- they check their internal policy set for the existence of this policy;
- if no such policy exists, they issue the response "RESPONSE False" to the IP address of the original sender;
- if the policy exists, they check their policy set for any related policies;
- they issue the appropriate access request to every IP address named in the related policy;
- if no response returns "RESPONSE False", the access request is accepted and the operation called for is performed.

The admin smartphone, as it acts as its own interface from which to send and receive ACTIONS, REQUESTS, and RESPONSES, is not embedded with SOOAC. The same is true, in Experiment 2, for the guest smartphone. As the Homepod smart speaker also acts as its own interface (by receiving and sending audio messages), it also does not have its own Pi. To capture the desired behaviour as described in our running example, our setup relies on individual voice profiles being set up on both iOS smartphones. These smartphones contain a python script (using the Pythonista app [123]) which is run automatically when the voice command, "open smart lock" is said. This is carried out as a 'Shortcut' through the 'Home' app on each device. Similar solutions to the problem of embedding SOOAC for other smart speakers should be possible. For example, with Alexa, this should be possible as a 'Skill' [10], and with Google Assistant, this should be possible using the SDK platform [80]. Having said that, we have not tested these different approaches.

5.1.1 ACTIONS and REQUESTS

Typically, the distinction between a request to carry out an operation ('ACTION') and a request to know whether an operation can be performed ('REQUEST') is not significant. For SOOAC, this distinction is necessary because the order that operations are performed on a successful request will not be the same as the order that the requests were received. To see this in action, we look at configuration C3, and specifically at ACTIONs between the admin smartphone, the smart bulb and the smart plug. This can be seen as a decentralized version of an IFTTT-style policy. The intention is that when the admin switches the smart bulb on, the smart plug automatically switches on too.

In Figure 5.2, we show the terminal dialogue boxes associated with both Pis after the admin attempts to switch the smart bulb on through an ACTION. Acting as the smart bulb's ACU, smart-bulb-pi first converts the ACTION into a REQUEST with the appropriate IP addresses. smart-bulb-pi checks whether this access request exists as a policy in its policy set. It exists, so smart-bulb-pi checks its policy set for any related policies. It finds the policy linking itself to smart-plug-pi, and duly requests access to smart-plug-pi on behalf of the admin. smart-plug-pi checks that this request exists within its local policies. It does, so smart-plug-pi checks its policies for any related policies. There are none, so smart-plug-pi responds to smart-bulb-pi 'True'. smart-bulb-pi receives 'True' and grants the admin's operational request, by issuing the operation through the BT Hive API associated with the smart bulb. This causes the smart bulb to switch on. Performing this operation causes the smart-bulb-pi to issue an ACTION to smart-plug-pi. smart-plug-pi converts this message into a REQUEST and checks its policy set for the existence of the request. It exists, and there are no related policies, so the smart-plug-pi, calls the TP-Link API, performs the operation, and switches the smart plug on.

This case shows why ACTIONS are required and why there needs to be a distinction between requesting access to a device operation and requesting that a device operation be performed. If we do not allow for ACTIONS and do not maintain the distinction, it is not clear how we could reliably implement operations while also

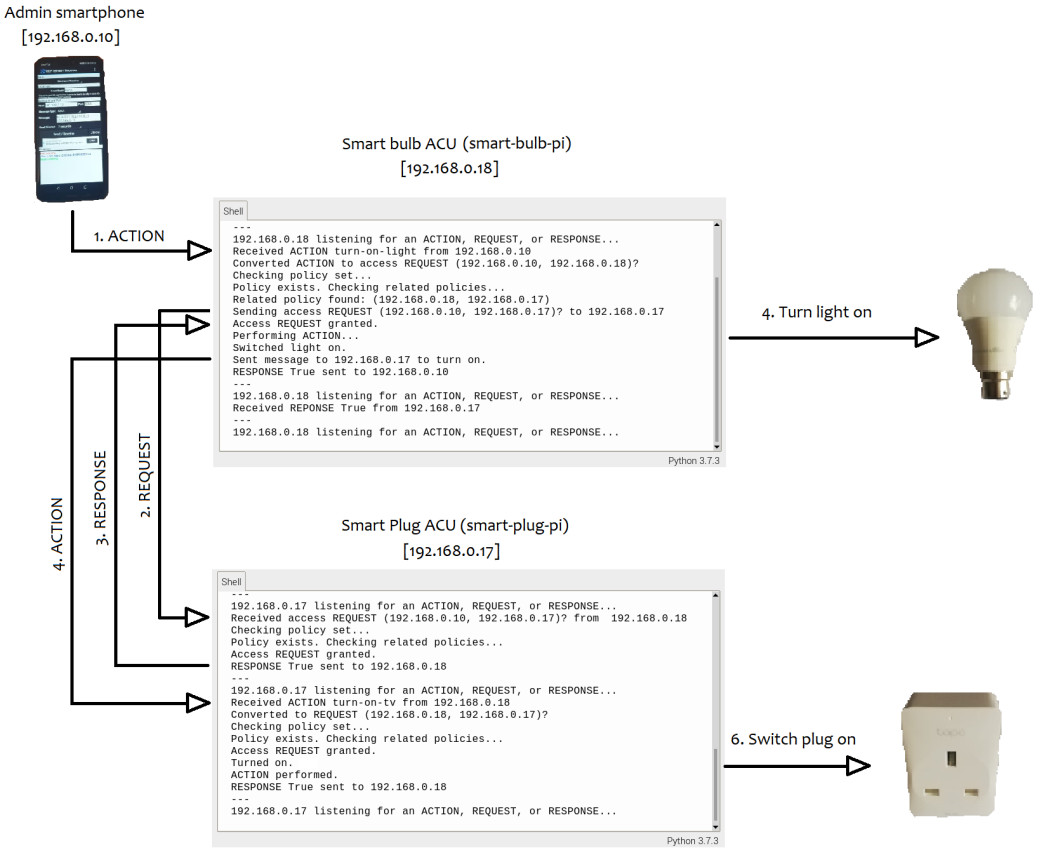

Figure 5.2: Python script terminals for the smart bulb and smart plug Raspberry Pis (smart-bulb-pi and smart-plug-pi, respectively) showing the main steps of a successful operational request by the admin. In Step 1, the admin issues the command to smart-bulb-pi to switch the smart bulb on. This command is converted into an access request which is recursively sent to smart-plug-pi in Step 2. After the request is granted by the smart-plug-pi in Step 3, smart-bulb-pi performs the operation by switching on the smart bulb in Step 4. Performing this operation causes a new operational request, to switch the smart plug on, to be sent by smart-bulb-pi to smart-plug-pi in Step 5. This request is granted by smart-plug-pi, which switches the plug on in Step 6.

resolving policy conflicts. For example, if we assume that requesting access to an operation performs that operation, the smart plug would actually switch on before the smart bulb! Even worse, if there was another device linked by a policy to the smart bulb that denied access, the smart plug would switch on and the smart bulb would not, thereby going against what should occur.

It should be noted that ACTIONS demand further recursion of access requests and therefore more processing power and time. This is not ideal, but appears unavoidable, either for SOOAC or a centralized conflict resolver.

5.2 Performance Experiments

To test devices' ability to avoid different policy conflicts efficiently and effectively under the SOOAC model, we performed two experiments. Experiment 1 tested whether access requests are responded to in a timely manner, not significantly slower than the equivalent centralized system. Experiment 2 tested whether using the connectionless UDP protocol affects the number of accurate responses.

5.2.1 Experiment 1: Testing the Efficiency of SOOAC

We formulated a list of policies, and formed six policy configurations (C1–C6) as subsets of this list. These configurations were chosen to show some variety of smart home setups, while giving some potential IFTTT-style connections between real-world devices that would be useful to users, using the seven devices we have. The full list of policies are given in Appendix A. The configurations are shown in Figure 5.1.

5.2.1.1 Experiment 1 Design

For each policy in a configuration, we used a desktop PC (acting as a user) to send an initial access request for each policy and measure the time taken to receive a response. For example, in C1, the PC issued the initial requests P1–P6, P11, P18, and P20. The PC then waited for a response to the request. As we are using UDP, we set a long time-out value (5s), and if a request was not received within that time, no data was collected and the same request was re-sent, so as to get accurate data for the system working under optimal conditions. The experimental design is shown in

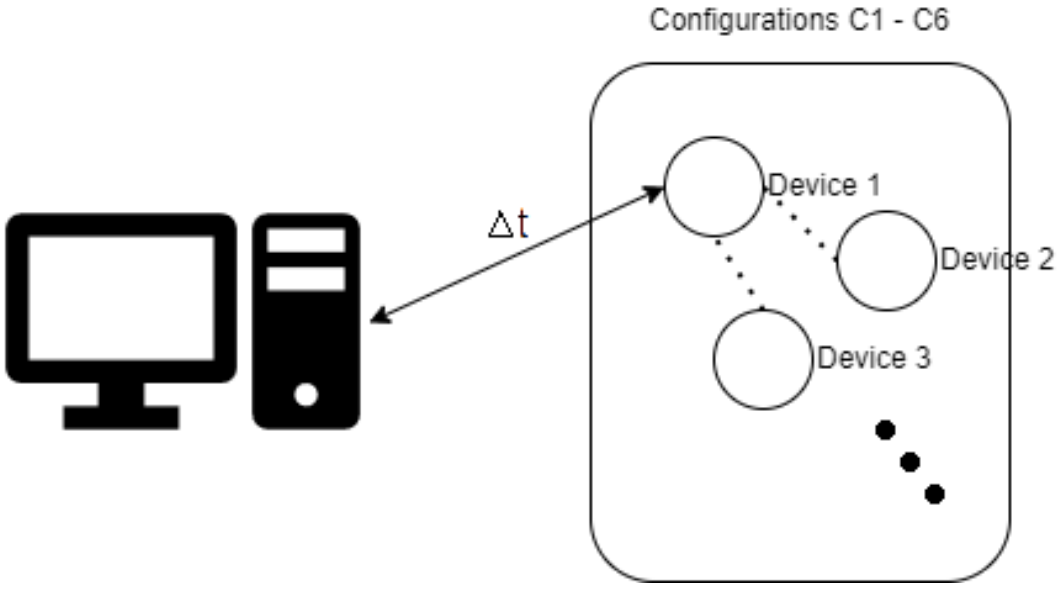

Figure 5.3: Experiment 1: testing the efficiency of SOOAC. A desktop PC issued access requests according to the policies in each configuration (C1–C6). The total times measured by the PC were recorded and then averaged, giving an average response time (Δ t) for the configuration. If responses were not received before the end of a time-out period (5s), they were ignored, giving results for the system working under optimal conditions.

Figure 5.3.

We gathered data (shown in Table 5.1) according to a number of metrics, which are described as follows:

- For each policy configuration (Row 1), we recorded the *no. of devices* (Row 2), and the total *no. of policies* across the smart home for that configuration (Row 3);
- The *centralized no. of policies* (Row 4) is the number of policies necessary to represent the same smart home configuration under a centralized access control system.
- The *no. of conflicts* (Row 5) is the number of unique pair-wise requests that are denied access in a configuration. Put another way, this is the number of policies in the configuration that, when issued as initial requests by the external device, are denied.
- Each initial request may lead on to zero or more recursive requests before receiving a response. This is the number of unique arrows that can be made

from one starting arrow, before either reaching a contradiction or exhausting all arrows (Row 6).

- For each configuration, we sum together these numbers to form the *no. of requests for response* (Row 7).
- When an access chain contains multiple devices that have access to each other, it is possible to have redundant access requests. These are transitive checks that have already been made while following the access chain for one initial request. The total number of redundant checks in one configuration is given as the *no. of redundant requests* (Row 8).
- Finally, we measured the time taken for the PC to hear back a response for each request. The experiment was performed 100 times to generate an accurate response time average for each possible initial request. These numbers were then summed and divided by the number of possibles requests (i.e., policies) to form the *av. response time* (Row 9).

All of these metrics, except *av. response time* are derivable from the configuration graphs in Figure 5.1. To ensure our system was working correctly, we verified these numbers during the running of Experiment 1. Because of SOOAC's decentralized approach, some of these metrics can be given in finer granularity, such as the number of policies for each specific device, and the response times for each request. The complete data set is given in Appendix A.

5.2.1.2 Experiment 1 Results

As shown in Table 5.1, the *no. of policies* in each configuration is at most double the number of policies present in a centralized system. This is a consequence of the simultaneous policy restriction, introduced in Subsection 4.2.2. For configurations in which it is less than double (C1, C5, C6), this is due to the guest smartphone, which does not adhere to this restriction. The low values for *av. no. of policies per device* is a consequence of the Modularity principle we used in the design of ACUs. The specific numbers of policies present in each device can vary from this value. For example, in C5, the smart speaker has 6 policies, while the smart TV has just 1 (see Appendix A, C5 Results).

Table 5.1: Results of Experiment 1 primarily showing the efficiency of SOOAC (in **bold**).

Policy configuration	C1	C2	C3	C4	C5	C6
No. of devices	7	4	4	6	7	7
No. of policies	19	10	10	10	26	26
Centralized no. of policies	10	5	5	5	15	15
Av. no. of policies per device	2.71	2.50	2.50	1.67	3.71	3.71
No. of conflicts	1	2	0	0	2	16
No. of requests for response	17	9	7	5	42	45
No. of redundant requests	1	0	0	0	11	0
Av. response time for request (s)	**0.03**	**0.03**	**0.03**	**0.02**	**0.06**	**0.06**

Table 5.1 also shows that the *no. of conflicts* within a configuration is independent of the number of devices and policies within a configuration (e.g., compare C2 to C3 and C5 to C6). The lack of conflicts for C3, despite its superficial similarity to C2, can be explained by it containing two transitively closed chains, while C2 has only one chain that is not transitively closed. The high number of conflicts for C6 can be explained by the fact that, because the final device in the chain is only connected to the penultimate one, every other device in the chain fails to have transitive access. In contrast, because C5 has a device early in the chain that is connected to a device connected to no others, only two devices fail to have transitive access. This also explains why the *no. of redundant requests* is high for C5 and low for C6: in C6, because the In-Access Recursion algorithm carries out a depth-first search, the conflict is quickly found, while in C5, many identical requests are checked multiple times before a conflict is found. The *no. of requests for response* appears to be approximated by the number of policies in a configuration, but also appears inversely proportional to the number of redundant checks.

Finally, Table 5.1 suggests that while response times do tend to increase with

the number of policies present in a configuration, they do not increase at a significant rate. C4 can be understood to be the minimum response time to any request (roughly 0.002s), as this configuration has no chains of devices. This value is also the baseline response time for centralized access control. The longest response time for any request (given in Appendix A, Results C5), involves the admin smartphone and smart speaker (0.234s). This is still well within standard user tolerances [42], and provides no significant increase in time compared to centralized access control.

5.2.2 Experiment 2: Testing the Effectiveness of SOOAC

As UDP is a connectionless protocol, dropped packets may result in requests and responses never reaching their targets. This may occur when a message is sent when an ACU is out of its listen state, or it may occur when multiple initial requests are sent to the same listening device at the same time. The latter will occur when there are multiple users in the smart home.[3] We tested how well the system performs under these conditions by investigating how likely it is that an initial request is not responded to. We chose C2 and C5 as representative for the smart home, being the simplest (excluding the trivial configuration C4) and most complicated configurations, respectively.

5.2.2.1 Experiment 2 Design

We used an external device to issue initial requests to any of the devices in the system by randomly choosing a policy for that configuration. In Test 1, we use one external device that issues 1000 initial requests. After one request is sent, this device listens for a response, and then issues another initial request, and so on. We employ a time-out on this device and all smart devices within the system, so that if no response is received within 2s, it is presumed that at least one packet along the chain has been lost. (2s was chosen as this is significantly longer than any response time from Experiment 1.) We keep a record of these with the integer denoted *Fails*. By default, a time-out returns False to any request, so no request can be allowed

[3] It may also occur when a device sends an initial request due to some internal state (e.g., based on its internal clock) but as the effect is the same, cases such as this are also covered in Experiment 2.

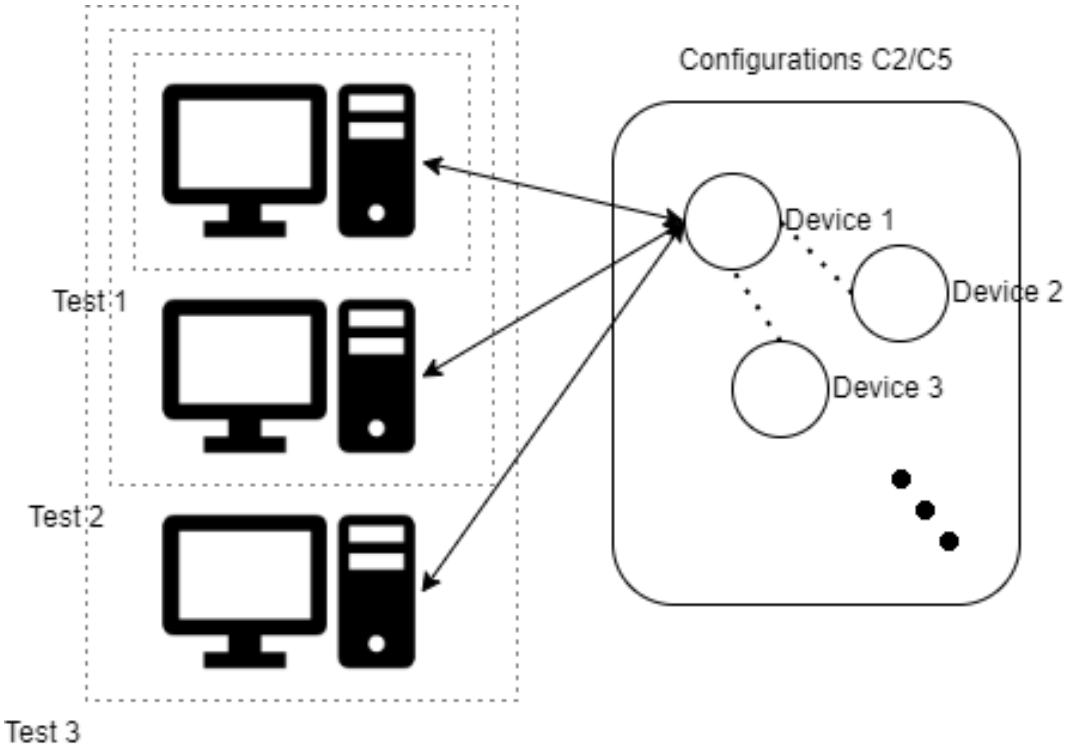

Figure 5.4: Experiment 2: testing the effectiveness of SOOAC. In Test 1, a desktop PC issued 1000 access requests according to the policies in Configurations C2 and C5. The number of Successes (i.e., received a correct response — allow or deny as appropriate — within the time-out period) and Fails (received a deny when it should have been an allow due to falling outside of the time-period) was measured, giving a measure for how effective SOOAC is. In Test 2 we introduced a second desktop PC issuing 1000 more requests, and in Test 3, we introduced a third desktop PC issuing 1000 more requests, simulating the scalability of effectiveness of SOOAC as the number of users in the system increases.

that should be denied (i.e., no false positives). Out of order UDP packets can also never give a false positive. It may be possible though, that when a time-out is reached, the initial request should have been allowed but was not (i.e., there may be false negatives). Every initial request that is responded to within 2s will increment the integer denoted *Successes*. In Tests 2 and 3, we recorded how the number of Successes and Fails change as we increase the number of users. In each case we increased the number of requests issued by 1000. An overview of the design for Experiment 2 is given in Figure 5.4.

It should be noted that this experiment places SOOAC under pressure beyond normal operating conditions, as a typical smart home would not receive so many

Table 5.2: Results of Experiment 2 showing the effectiveness of SOOAC.

C2	Test 1	Test 2		Test 3		
	User 1	User 1	User 2	User 1	User 2	User 3
Successes	985	973	959	961	934	952
Fails	15	27	41	39	66	48

C5	Test 1	Test 2		Test 3		
	User 1	User 1	User 2	User 1	User 2	User 3
Successes	973	943	938	929	902	913
Fails	27	57	62	71	98	87

initial requests within such a short space of time. It is therefore best understood as a ‘stress test’.

5.2.2.2 Experiment 2 Results

The baseline results given under the Test 1 column indicate that, even at the high request rates provided by the external devices, SOOAC is capable of getting over 97% of access requests responded to correctly, even in the most complicated configuration C5. As discussed, when SOOAC does give an incorrect response, it is only ever an incorrect denial. This means that in rare cases when access is denied incorrectly, the user can just try the request again and they will likely be allowed access on the second attempt.

As expected, the effectiveness of SOOAC is dependent on the complexity of the smart home (i.e., the number of access chains present), as shown by the lower numbers of successes across tests in C5 compared to C2. Nevertheless, the success numbers are high enough to suggest that a smart home involving dozens of devices would still be very usable under SOOAC, not significantly negatively affecting the user experience.

As the number of external devices (Users) increases (Test 1, Test 2, Test 3), the number of failed requests increases, but not drastically so. This shows that our approach accommodates typical numbers of users in a smart home (three residents [153] and several guests).

These results show that taking a simple approach by using UDP (avoiding the need to manage multiple connections simultaneously between the various devices)

does not significantly undermine the decentralized nature of our approach: access requests can be sent freely without significant impact on users.

5.3 Summary of advantages of SOOAC over centralized access control

In Chapter 3 (specifically, in Subsection 3.3.2) we gave arguments for decentralized access control in the IoT. We separated out these arguments into necessary and contingent arguments, the former relating to the need for a model that accurately describes current access control in the Smart Home, and the latter relating to the benefits of prescribing a model in which multiple access control systems exist in one smart home. Carrying out the experiments described in the present chapter have given us qualitative and quantitative empirical evidence justifying these arguments.

With respect to the necessary arguments, the diversity of device manufacturers involved in the Smart Home Configurations C1–C6 necessitate a decentralized model. Only the smart speaker and smartphones fall under one manufacturer (Apple), meaning that by default constructing these configurations would involve managing multiple independent apps. Even if attempts were made to reduce this decentralization by using, for example, IFTTT, doing so is not, at the time of writing, possible because the Ener-J smart lock is not compatible with IFTT. It is therefore not currently possible to construct a faithful centralized model for these configurations. While the Ener-J smart lock may, in time, be made compatible with IFTTT, it is impossible that every new device on the market will be. Faithfully describing the Smart Home configurations chosen therefore necessitates a decentralized model like SOOAC.

With respect to the contingent arguments, we see the efficiency and reliability arguments played out in practice. In Experiment 1, we saw that the average number of policies required for each device is significantly lower than the total that would exist in one centralized ACU. This means that individual requests can be checked faster and simultaneously across devices (not possible with a centralized ACU). It also means that an ACU for an individual device may break and yet not bring down

the Smart Home as a whole. We gained further empirical evidence in Experiment 2 that many requests can be made to devices in a short space of time without reducing the user experience.

Chapter 6

Conclusion

In the preceding chapter, the experimental results collected from implementing SOOAC in a prototype smart home addressed our final research question, **RQ 4**. More specifically, we showed that SOOAC is efficient (**RQ 4.1**): it adds no significant increase in time for access requests to be deliberated on compared to the centralized case, and therefore does not negatively impact the user experience. We also showed SOOAC to be effective (**RQ 4.2**): even using the connectionless UDP protocol, almost all access requests were responded to correctly, and when they were not, there were no false positives (i.e., it only denied access when it should have granted it; it never granted access when it should have denied it). However, there remain a number of improvements that can be made to SOOAC — brought about by its limitations — that lead to further work.

6.1 Limitations and Further Work

6.1.1 Resolving Arbitrary Conflicts

In Chapter 4, we argued that modelling the Running Example only requires a simple policy language in which ('unconditioned') access control policies can be represented as pairs of devices. We showed how the Running Example can be resolved if ACUs can rewrite these policies and issue them as new access requests. Generalising our approach to avoid all global policy conflicts requires arbitrary policy rewriting. To do this requires extending the present work with a more complex access control policy language. This language should be capable of expressing

fine-grained policies, should allow for negative policies to be expressed explicitly, and enable ACUs to carry out more logical operations (beyond the transformations we have described).

A full treatment of decentralized policy conflicts would involve extending the present model to cover the 'good' and 'ideal' requirements described in 2.2.2. Future work should be carried out to investigate whether the XACML policy language can serve this desired purpose. Another avenue for future research would be to use languages specifically designed for distributed environments (see, e.g., [57]).

6.1.2 Reducing Run-Time Delays

Our results show that SOOAC introduces some delay to the time it takes for a response to be received from an access request. This is a consequence of the In-Access Recursion algorithm (see Subsection 4.2.2). Dunlop et al [61] argue that policy conflict detection should ideally be carried out statically (i.e., before run time), so as to not burden users with delays. SOOAC can allow for this by modifying In-Access Recursion to *Pre-Access Recursion*. When setting a policy, the same recursive requests should be formed. If these requests rely on policies external to an ACU, these policies should be attempted to be written to the initial ACU. If writing the policy invokes a contradiction, it should not be written; otherwise, it should be. Then, when making a request, only local policies need to be checked. As a consequence, time delays will no longer be felt by users — but by admins, who write the policies, instead.

It may be the case that certain Internet of Things environments are more applicable to one or the other type of recursion. For example, while Pre-Access Recursion may be faster for users in relatively static environments, more dynamic environments may cause additional waiting times (the additional time it takes for admins to update policies). Further research should be carried out to systematically compare In-Access and Pre-Access Recursion. Ideally, this research would involve human-centred studies to explore which is better for users and admins in real-world Internet of Things environments.

6.1.3 In-Parallel Requests and PDP-PAP Encapsulation

It is a limitation of our work that we chose the In Series rather than In Parallel issuing of access requests (see Subsection 4.2.2). This design choice led us to using UDP as the communication protocol in our implementation of SOOAC, forcing devices to enter listening states. If, instead, requests can be checked and communicated in parallel, more concurrent processes can occur, improving efficiency. This is implementable using TCP, but requires socket threading, a more complicated transition between states for ACUs, and care to avoid racing conditions. Significant optimization benefits will result from such an implementation [97], which we plan to carry out.

Similarly, it is worth investigating whether modifying our design choice of PDP-PDP connections (see Subsection 4.2.2) to PDP-PAP connections would affect the performance of SOOAC in implementation. By choosing PDP-PDP connections, we force (initially requested) ACUs to outsource policy-checking to other ACUs; PDP-PAP connections would require ACUs to do all policy-checking themselves. Like transitioning to In Parallel Recursion, significant care would need to be made to avoid racing conditions, as multiple ACUs could then require information from the same PAP simultaneously. Further work should compare these design choices.

6.1.4 Decentralized Policy Setting

One important assumption we have made is that smart home policies are set individually within each ACU. Setting policies in this way is time consuming and is a disadvantage of any decentralized approach. Indeed, as we noted in Section 3.2, this disadvantage is already experienced by smart home admins when they have to manage many apps. One solution is to allow ACUs to intermittently update their policies from a centralized policy store. While this shift towards centralization would go against the spirit of our approach, many of the benefits of decentralization (e.g., efficient response time to requests, no single point of failure) would still be held. Further research, ideally human-based studies, should be carried out to explore whether this idea is workable in practice.

6.1.5 Going Beyond Avoidance of Policy Conflicts

At present, SOOAC avoids policy conflicts by effectively ignoring policies that would lead to conflict. Ideally, though, information should be present to admins and users as to why certain policies are ignored. As Mare et al [115] put it:

"Ask for post-failure actions. When a user creates an automation, asking the user what the smart home should do when the automation fails may serve two goals: i) educating the user by informing that the automation could fail – something the user may not be aware of – and the system could offer some information on how that automation could fail; and ii) increasing user confidence in the smart home's reliability, by educating the user and giving her more control."

An appropriate user interface for admins (when setting policies) and users (when desired behaviour is not allowed) should be implemented for SOOAC. Moreover, work should be carried out to go beyond the avoidance of operational policy conflicts (see Subsection 2.1.4 for the distinction between declarative and operational policy conflicts). Ideally, the desired intentions of users should be met by SOOAC automatically, meaning that operational policy conflicts are avoided while adhering to declared policies. For example, rather than simply refusing access to all operations on a certain device, some subset of operations should be allowed. Celik et al [42] have carried out similar work involving centralized policy conflicts that may be adaptable to the decentralized case.

6.1.6 Authentication

In Chapter 2, when first defining access control, we were clear that, for us, access control is about authorization and not authentication; that is, we assumed that when considering what access some given subjects have, those subjects' identities are known. This move is a common one in the access control literature [16], and is sometimes made more explicit by referring to subjects as *principals* (i.e., pre-authenticated entities). In Chapter 4, we embodied this idea within the principle of Persistence, ensuring that all identities within a system remained unique and constant over time, and in Chapter 5, we implemented it by defining policies as pairs of IP addresses. This effectively allows each ACU to correctly know the identities of

subjects within the system.

For models that attempt to represent centralized systems, distinguishing authorization and authentication — and focusing only on authorization — is less controversial than for doing the same in decentralized systems. This is because in centralized systems, there is presumably only one trustworthy entity responsible for identifying subjects in a system; for us, by modelling multiple ACUs, we would presumably require multiple trustworthy authenticating entities.

It is therefore a lacuna in our account as to what mechanism can ensure that each ACU can trust the identities of subjects within a system. Potential candidates may include blockchain technologies [125, 112, 20] or consensus-based approaches more broadly [33]. Other promising approaches may involve automated device recognition [144, 47, 146], sometimes used to detect network intrusion in the Internet of Things. Further work should be carried out to implement such methods in tandem with the access control framework we have described. Crucially, the mechanism chosen should retain the decentralized nature of the present work so that ACUs do not lose the benefits afforded to them by being independent. This work should also be careful to account for resource constrained devices.

6.2 Conclusion

This thesis has presented a novel meta-model for access control that is applicable to decentralized systems, particular to the Internet of Things and the Smart Home. This model was borne out of an analysis of historical models for access control and the challenges that guided their introduction, as well as an analysis of the proposed challenges facing Internet of Things access control. We have argued that any faithful analysis of the Smart Home of today requires an embracing of decentralization, not a rejection of it. We have drawn attention to the new challenge of policy conflict resolution between access control systems (i.e., global policy conflicts), and we have given a distributed algorithm (In-Access Recursion) in which devices communicate with one another to address this challenge.

We have also indicated improvements to our approach that lead to future work.

These have been outlined in this chapter and elsewhere in the thesis. They include: the resolution of policy conflicts involving arbitrary policies, not just simple, unconditioned access between devices; resolution that goes beyond avoidance, to enable users to carry out their desired intentions in an operationally secure manner; the investigation of alternate placement of recursive checks to place the time burden away from users; the issue of updating decentralized policies efficiently; and the challenge of coupling a decentralized authentication mechanism with the proposed access control model.

The Internet of Things seems destined to grow, its devices becoming ever more part of our lives. Their security cannot be left behind. As devices become smarter — more human — so should their security. This means giving devices the power to make their own security decisions. Our work in enabling devices to themselves automatically avoid policy conflicts is a step towards that ideal.

Appendix A

Policy Configurations and Results from Chapter 5

A.1 List of smart devices

Code	Device name
a	Admin smartphone
b	Guest smartphone
c	Smart speaker
d	Smart lock
e	Smart bulb
f	Smart plug
g	Smart TV

A.2 List of Policies (/initial access requests)

Code	Description	Pair
P1	The admin smartphone can access the smart speaker.	(a,c)
P2	The admin smartphone can open the smart lock.	(a,d)
P3	The admin smartphone can switch on the smart bulb.	(a,e)
P4	The admin smartphone can switch on the smart plug.	(a,f)
P5	The admin smartphone can switch on the smart TV.	(a,g)
P6	The guest smartphone can access the smart speaker.	(b,c)
P7	The guest smartphone can open the smart lock.	(b,d)
P8	The guest smartphone can switch on the smart bulb.	(b,e)
P9	The guest smartphone can switch on the smart plug.	(b,g)
P10	The guest smartphone can switch on the smart TV.	(b,g)
P11	The smart speaker can open the smart lock.	(c,d)
P12	The smart speaker can switch on the smart bulb.	(c,e)
P13	The smart speaker can switch on the smart plug.	(c,f)
P14	The smart speaker can switch on the smart TV.	(c,g)
P15	If the smart lock is open, then switch the smart bulb on.	(d,e)
P16	If the smart lock is open, then switch on the smart plug.	(d,f)
P17	If the smart lock is open, the switch on the smart TV.	(d,g)
P18	If the smart bulb is on, then switch on the smart plug.	(e,f)
P19	If the smart bulb is on, then switch on the smart TV.	(e,g)
P20	If the smart plug is on, then switch on the smart TV.	(f,g)

A.3 List of policy configurations

Code	Description
C1	P1—6, P11, P18—20
C2	P3—5, P18, P20
C3	P3—5, P18, P19
C4	P1—5
C5	P1—4, P6—9, P11—16, P18
C6	P1—4, P6—9, P11—13, P15, P16, P18, P20

A.4 Results

CONFIGURATION 1 (C1) RESULTS

	Smart devices										
	Admin phone					Guest phone	Speaker	Lock	Bulb		Plug	TV
Size of policy set	5					0*	3	2	3		3	3
Number of possible access requests sent from	5					1	1	0	1		1	0
Number of possible access requests sent to	0					0	2	2	2		2	3
Number of blocked access requests (no. of policy conflicts)	0					1	0	-	0		0	-
Number of redundant access requests	1					0	0	-	0		0	-
Device linked by policy as object	Speaker	Lock	Bulb	Plug	TV	Speaker	Lock	-	Plug	TV	TV	-
Number of requests needed for response	2	1	4	2	1	2	1	-	2	1	1	-
Allowed access? (i.e., conflict free?)	Yes	Yes	Yes	Yes	Yes	No	Yes	-	Yes	Yes	Yes	-
Number of redundant access requests	0	0	1	0	0	0	0	-	0	0	0	-
Time taken for response	0.0239	0.0054	0.1411	0.0235	0.0061	0.0245	0.0061	-	0.0217	0.0070	0.0070	-

CONFIGURATION 2 (C2) RESULTS

	Smart devices					
	Admin phone			Bulb	Plug	TV
Size of policy set	3			2	3	2
Number of possible access requests sent from	3			1	1	0
Number of possible access requests sent to	0			1	2	2
Number of blocked access requests (no. of policy conflicts)	1			1	0	-
Number of redundant access requests	0			0	0	-
Device linked by policy as object	Bulb	Plug	TV	Plug	TV	-
Number of requests needed for response	3	2	1	2	1	-
Allowed access? (i.e., conflict free?)	No	Yes	Yes	No	Yes	-
Number of redundant access requests	0	0	0	0	0	-
Time taken for response	0.0782	0.0224	0.0074	0.0138	0.0066	-

CONFIGURATION 3 (C3) RESULTS

	Smart devices						
	Admin phone			Bulb		Plug	TV
Size of policy set	3			3		2	2
Number of possible access requests sent from	3			2		0	0
Number of possible access requests sent to	0			1		2	2
Number of blocked access requests (no. of policy conflicts)	0			0		-	-
Number of redundant access requests	0			0		-	-
Device linked by policy as object	Bulb	Plug	TV	Plug	TV	-	-
Number of requests needed for response	3	1	1	1	1	-	-
Allowed access? (i.e., conflict free?)	Yes	Yes	Yes	Yes	Yes	-	-
Number of redundant access requests	0	0	0	0	0	-	-
Time taken for response	0.0895	0.0078	0.0128	0.0079	0.0125	-	-

CONFIGURATION 4 (C4) RESULTS

	Smart devices								
	Admin phone					Speaker	Lock	Bulb	Plug	TV
Size of policy set	5					1	1	1	1	1
Number of possible access requests sent from	5					0	0	0	0	0
Number of possible access requests sent to	0					1	1	1	1	1
Number of blocked access requests (no. of policy conflicts)	0					-	-	-	-	-
Number of redundant access requests	0					-	-	-	-	-
Device linked by policy as object	Speaker	Lock	Bulb	Plug	TV	-	-	-	-	-
Number of requests needed for response	1	1	1	1	1	-	-	-	-	-
Allowed access? (i.e., conflict free?)	Yes	Yes	Yes	Yes	Yes	-	-	-	-	-
Number of redundant access requests	0	0	0	0	0	-	-	-	-	-
Time taken for response	0.0182	0.0148	0.0196	0.0187	0.0171	-	-	-	-	-

CONFIGURATION 5 (C5) RESULTS

	Smart devices																			
	Admin phone				Guest phone				Speaker				Lock		Bulb		Plug		TV	
Size of policy set	4				0*				6				5		5		5		1	
Number of possible access requests sent from	4				4				2				2		1		0		0	
Number of possible access requests sent to	0				0				4				3		4		5		1	
Number of blocked access requests (no. of policy conflicts)	1				1				0				0		0		-		-	
Number of redundant access requests	5				5				1				0		0		-		-	
Device linked by policy as object	Speaker	Lock	Bulb	Plug	Speaker	Lock	Bulb	Plug	Lock	Bulb	Plug	TV	Bulb	Plug	Plug		-		-	
Number of requests needed for response	9	3	2	1	9	3	2	1	4	2	1	1	2	1	1		-		-	
Allowed access? (i.e., conflict free?)	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		-		-	
Number of redundant access requests	4	1	0	0	4	1	0	0	1	0	0	0	0	0	0		-		-	
Time taken for response	0.2340	0.0903	0.0541	0.0054	0.1955	0.0817	0.0604	0.0062	0.0447	0.0346	0.0077	0.0083	0.0638	0.0074	0.0062		-		-	

CONFIGURATION 6 (C6) RESULTS

	Smart devices																			
	Admin phone				Guest phone				Speaker				Lock		Bulb		Plug			
Size of policy set	4				0*				5				5		5		6			
Number of possible access requests sent from	4				4				3				2		1		1			
Number of possible access requests sent to	0				0				2				3		4		5			
Number of blocked access requests (no. of policy conflicts)	4				4				3				2		1		0			
Number of redundant access requests	0				0				0				0		0		0			
Device linked by policy as object	Speaker	Lock	Bulb	Plug	Speaker	Lock	Bulb	Plug	Lock	Bulb	Plug		Bulb	Plug	Plug		TV			
Number of requests needed for response	5	4	3	2	5	4	3	2	4	3	2		3	2	2		1			
Allowed access? (i.e., conflict free?)	No	No	No	No	No	No	No	No	No	No	No		No	No	No		Yes			
Number of redundant access requests	0	0	0	0	0	0	0	0	0	0	0		0	0	0		0			
Time taken for response	0.1729	0.0828	0.0655	0.0293	0.1767	0.1057	0.0776	0.0165	0.0627	0.0529	0.0188		0.0508	0.0169	0.0136		0.0055			

Appendix B

Python Code

Listing B.1: Generic ACU Python Code (example given is for the smart bulb in C1)

```
from send import send_message
from receive import get_message

import random, time

# ====================================
# Static IP addresses for smart devices (their Pis). Change as appropriate.

admin = "192.168.1.151"
guest = "192.168.1.152"
speaker = "192.168.1.153"
lock = "192.168.1.154"
bulb = "192.168.1.155"
plug = "192.168.1.156"
tv = "192.168.1.157"

this_machine = bulb # Change as appropriate.

# ====================================
# Local policies. Change as appropriate.


policy_set =[

    (admin, bulb),
    (bulb, plug),
    (bulb, tv)

    ]

# ====================================
# Main program (includes SOOAC)

def access(subj, obj):
    if (subj, obj) not in policy_set:
        return False
    for policy in policy_set:
        if policy[0] == obj:
            new_obj = policy[1]
            send_message(policy[1], "REQUEST", subj, policy[1])
```

```
            if get_message(this_machine, 3)[0][1] == "False":
                return False
    return True

while True:
    current_udp = get_message(this_machine)
    print(".")
    requester = current_udp[1][0]
    if current_udp[0][0] == "REQUEST":
        subj = current_udp[0][1]
        obj = current_udp[0][2]
        if access(subj, obj):
            send_message(requester, "RESPONSE", "True")
        else:
            send_message(requester, "RESPONSE", "False")
```

Listing B.2: Generic ACU send.py Python code

```
import socket

def send_message(address, category, *args):
    if category == "REQUEST":
        message = args[0] + "␣" + args[1]
    if category == "RESPONSE":
        message = args[0]
    message = category + "␣" + message
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(message.encode('utf-8'), (address, 5005))
    return
```

Listing B.3: Genereric ACU receive.py Python code

```
import socket

def get_message(address, timeout = 2):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((address, 5005))
    if timeout != "None":
        sock.settimeout(timeout)
    try:
        data, requester = sock.recvfrom(1024)
        list_message = [entity.decode("utf-8") for entity in data.split()]
        return list_message, requester
    except:
        return (("RESPONSE", "False"), "None")
```

Listing B.4: Experiment 1 Python code

```
from send import send_message
from receive import get_message
import random, time

# ====================================
# Static IP addresses for smart devices (their Pis)

admin = "192.168.1.151"
guest = "192.168.1.152"
```

```
speaker = "192.168.1.153"
lock = "192.168.1.154"
bulb = "192.168.1.155"
plug = "192.168.1.156"
tv = "192.168.1.157"

desktop_pc = "192.168.1.150"
this_machine = desktop_pc # change according to what machine this is

# ====================================

policy_set = [

(admin, speaker), (admin, lock), (admin, bulb), (admin, plug), (admin, tv),
(guest, speaker),
(speaker, lock),
(bulb, plug), (bulb, tv),
(plug, tv)

]

policy_set_names = [

("admin", "speaker"), ("admin", "lock"), ("admin", "bulb"), ("admin", "plug"), ("admin", "tv"),
("guest", "speaker"),
("speaker", "lock"),
("bulb", "plug"), ("bulb", "tv"),
("plug", "tv")

    ]

total_exp_length = 100

totals = [0.0 for i in range(len(policy_set))]
for duration in range(total_exp_length):
    for i in range(len(policy_set)):
        request = policy_set[i]
        check = False
        while check != True:
            time_start = time.time()
            send_message(request[1], "REQUEST", request[0], request[1])
            reply = get_message(this_machine)
            response_time = time.time() - time_start
            if response_time < 2:
                check = True
                totals[i] += response_time

            print(policy_set_names[i], response_time)
            print(reply)
            print("")

totals = [i/total_exp_length for i in totals]

print("====RESULTS=====")

for i in range(len(totals)):
    print(policy_set_names[i], totals[i])
```

Listing B.5: Experiment 2 Python code

```
from send import send_message
from receive import get_message
```

```
import random, time

# ====================================
# Static IP addresses for smart devices (their Pis)

admin = "192.168.1.151"
guest = "192.168.1.152"
speaker = "192.168.1.153"
lock = "192.168.1.154"
bulb = "192.168.1.155"
plug = "192.168.1.156"
tv = "192.168.1.157"

# this line is only for performance tests in Subsection 5.3
desktop_pc_1 = "192.168.1.150"
desktop_pc_2 = "192.168.1.158"
desktop_pc_3 = "192.168.1.159"

this_machine = desktop_pc_1 # change according to what machine this is

# ====================================
# Each policy set (a list) corresponds to a Configuration
# i.e., policy_sets[0] is C1, policy_set[1] is C2, etc.

policy_set = [

(admin, bulb), (admin, plug), (admin, tv),
(bulb, plug),
(plug, tv)

]


def qm(subj, obj):

    time_start = time.time()
    a = (subj, obj)
    to_be_sent = a[1], "REQUEST", a[0], a[1]
    send_message(a[1], "REQUEST", a[0], a[1])
    x = get_message(this_machine)
    return time.time() - time_start

bad_count = 0
good_count = 0
for i in range(1000):

    i = random.choice(policy_set)
    temp = qm(i[0], i[1])
    print(temp)
    if temp > 2.0:
        bad_count += 1
    else:
        good_count += 1
    print(float(good_count)/float(good_count + bad_count))
```

Bibliography

[1] Home Assistant API. https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/. Last accessed: May 8, 2023.

[2] Samsung compatible devices. https://www.samsung.com/sg/support/mobile-devices/what-devices-are-compatible-with-samsung-smartthings/. Last accessed: May 8, 2023.

[3] Samsung Smart Things. https://www.samsung.com/uk/smartthings/. Last accessed: May 8, 2023.

[4] Socket - low-level networking interface. https://docs.python.org/3/library/socket.html. Last accessed: May 8, 2023.

[5] UDP sender / receiver - apps on Google Play. https://play.google.com/store/apps/details?id=com.jca.udpsendreceive. Last accessed: May 8, 2023.

[6] ABRAMS, M., AND BAILEY, D. Abstraction and refinement of layered security policy. *Information Security: An Integrated Collection of Essays* (1995), 126–136.

[7] ACHAUER, B. The DOWL distributed object-oriented language. *Communications of the ACM 36*, 9 (1993), 48–55.

[8] ALHARITHI, F. *Detecting Conflicts among Autonomous Devices in Smart Homes*. PhD thesis, 2019.

[9] ALKHABBAS, F., SPALAZZESE, R., AND DAVIDSSON, P. An agent-based approach to realize emergent configurations in the internet of things. *Electronics 9*, 9 (2020), p. 1347.

[10] AMAZON. Alexa Skills. `https://www.amazon.co.uk/b?ie=UTF8&node=10068517031`. Last accessed: September 1, 2022.

[11] AMAZON. Alexa smart home compatible devices. `https://developer.amazon.com/en-US/alexa/devices/connected-devices`. Last accessed: May 8, 2023.

[12] AMAZON. Amazon smart home. `https://www.amazon.co.uk/b?ie=UTF8&node=14526211031`. Last accessed: September 1, 2022.

[13] ANDALOUSSI, Y., EL OUADGHIRI, M. D., MAUREL, Y., BONNIN, J.-M., AND CHAOUI, H. Access control in iot environments: Feasible scenarios. *Procedia computer science 130* (2018), 1031–1036.

[14] ANDERSON, G., MCCUSKER, G., AND PYM, D. A logic for the compliance budget. In *International Conference on Decision and Game Theory for Security* (2016), Springer, pp. 370–381.

[15] ANDERSON, J. P. Computer security technology planning study. Tech. rep., ANDERSON (JAMES P) AND CO FORT WASHINGTON PA FORT WASHINGTON, 1972.

[16] ANDERSON, R. *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2020.

[17] ASHTON, K., ET AL. That 'internet of things' thing. *RFID journal 22*, 7 (2009), 97–114.

[18] ATLAM, H. F., ALASSAFI, M. O., ALENEZI, A., WALTERS, R. J., AND WILLS, G. B. Xacml for building access control policies in internet of things. In *IoTBDS* (2018), pp. 253–260.

[19] ATZORI, L., IERA, A., AND MORABITO, G. Understanding the internet of things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks 56* (2017), 122–140.

[20] BANERJEE, M., LEE, J., AND CHOO, K.-K. R. A blockchain future for internet of things security: a position paper. *Digital Communications and Networks 4*, 3 (2018), 149–160.

[21] BARAANI-DASTJERDI, A. Access control in object-oriented databases.

[22] BARKER, S. The next 700 access control models or a unifying meta-model? In *Proceedings of the 14th ACM symposium on Access control models and technologies* (2009), pp. 187–196.

[23] BASKERVILLE, R., AND SIPONEN, M. An information security meta-policy for emergent organizations. *Logistics information management* (2002).

[24] BEAUTEMENT, A., AND PYM, D. J. Structured systems economics for security management. In *WEIS* (2010), Citeseer, pp. 1–20.

[25] BEAUTEMENT, A., SASSE, M. A., AND WONHAM, M. The compliance budget: managing security behaviour in organisations. In *Proceedings of the 2008 New Security Paradigms Workshop* (2008), pp. 47–58.

[26] BELL, D. E., AND LAPADULA, L. J. Secure computer systems: Mathematical foundations. Tech. rep., MITRE CORP BEDFORD MA, 1973.

[27] BENNETT, S., MCROBB, S., AND FARMER, R. *Object-oriented systems analysis and design using UML*. McGraw-Hill, 2005.

[28] BERTIN, E., HUSSEIN, D., SENGUL, C., AND FREY, V. Access control in the internet of things: a survey of existing approaches and open research questions. *Annals of Telecommunications 74*, 7 (2019), 375–388.

[29] BERTINO, E., AND MARTINO, L. Object-oriented database management systems: concepts and issues. *Computer 24*, 4 (1991), 33–47.

[30] BERTINO, E., AND WEIGAND, H. An approach to authorization modeling in object-oriented database systems. *Data & Knowledge Engineering 12*, 1 (1994), 1–29.

[31] BEZAWADA, B., HAEFNER, K., AND RAY, I. Securing home iot environments with attribute-based access control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control* (2018), pp. 43–53.

[32] BIBA, K. J. Integrity considerations for secure computer systems. Tech. rep., MITRE CORP BEDFORD MA, 1977.

[33] BISWAS, S., SHARIF, K., LI, F., MAHARJAN, S., MOHANTY, S. P., AND WANG, Y. Pobt: A lightweight consensus algorithm for scalable iot business blockchain. *IEEE Internet of Things Journal 7*, 3 (2019), 2343–2355.

[34] BOOLOS, G. S., BURGESS, J. P., AND JEFFREY, R. C. *Computability and logic*. Cambridge university press, 2002.

[35] BRUCKER, A. D., AND PETRITSCH, H. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies* (2009), pp. 197–206.

[36] BRUSH, A. B. It's used by us: Family friendly access control. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Workshop on Technology for Today's Family* (2012), Citeseer.

[37] BUTUN, I. Privacy and trust relations in internet of things from the user point of view. In *2017 IEEE 7th annual computing and communication workshop and conference (CCWC)* (2017), IEEE, pp. 1–5.

[38] CALO, S., LUPU, E., BERTINO, E., ARUNKUMAR, S., CIRINCIONE, G., RIVERA, B., AND CULLEN, A. Research challenges in dynamic policy-based autonomous security. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 2970–2973.

[39] CALO, S., VERMA, D., CHAKRABORTY, S., BERTINO, E., LUPU, E., AND CIRINCIONE, G. Self-generation of access control policies. In *Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies* (2018), pp. 39–47.

[40] CATTERMOLE, T., DOCHERTY, S., PYM, D., AND SASSE, M. A. Asset-oriented access control: Towards a new iot framework. In *Proceedings of the 9th International Conference on the Internet of Things (IOT'19)* (2019).

[41] CAULFIELD, T., AND PYM, D. Improving security policy decisions with models. *IEEE Security & Privacy 13*, 5 (2015), 34–41.

[42] CELIK, Z. B., TAN, G., AND MCDANIEL, P. D. Iotguard: Dynamic enforcement of security and safety policy in commodity iot. In *NDSS* (2019).

[43] CHADWICK, D. W., AND LIEVENS, S. F. Enforcing" sticky" security policies throughout a distributed application. In *Proceedings of the 2008 workshop on Middleware security* (2008), pp. 1–6.

[44] CHANDER, A., MITCHELL, J. C., AND DEAN, D. A state-transition model of trust management and access control. In *CSFW* (2001), vol. 1, pp. 27–43.

[45] CHARAF, L. A., ALIHAMIDI, I., ADDAIM, A., AND ABDESSALAM, A. A distributed xacml based access control architecture for iot systems. In *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)* (2020), IEEE, pp. 1–5.

[46] CHOE, E. K., CONSOLVO, S., JUNG, J., HARRISON, B., PATEL, S. N., AND KIENTZ, J. A. Investigating receptiveness to sensing and inference in the home using sensor proxies. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), pp. 61–70.

[47] CHU, Y. H., AND CHOI, Y. K. A deep learning based iot device recognition system. *Journal of the Semiconductor & Display Technology 18*, 2 (2019), 1–5.

[48] CLARK, D. D., AND WILSON, D. R. A comparison of commercial and military computer security policies. In *1987 IEEE Symposium on Security and Privacy* (1987), IEEE, pp. 184–184.

[49] CLOSE, T. ACLs don't. *HP Laboratories Technical Report* (2009).

[50] DAHL, O.-J., AND NYGAARD, K. SIMULA: an ALGOL-based simulation language. *Communications of the ACM 9*, 9 (1966), 671–678.

[51] DALAL, B., NELSON, L., SMETTERS, D. K., GOOD, N., AND ELLIOT, A. Ad-hoc guesting: When exceptions are the rule. *UPSEC 8* (2008), 1–5.

[52] DAR, K., TAHERKORDI, A., ROUVOY, R., AND ELIASSEN, F. Adaptable service composition for very-large-scale internet of things systems. In *Proceedings of the 8th Middleware Doctoral Symposium* (2011), pp. 1–6.

[53] DE CAPITANI DI VIMERCATI, S., PARABOSCHI, S., AND SAMARATI, P. Access control: principles and solutions. *Software: Practice and Experience 33*, 5 (2003), 397–421.

[54] DE VRIES, M. J., CROSS, N., AND GRANT, D. P. *Design methodology and relationships with science*. Springer, 1993.

[55] DEVAAN, J. Update on uac.

[56] DHANKHAR, V., KAUSHIK, S., WIJESEKERA, D., AND NERODE, A. Evaluating distributed xacml policies. In *Proceedings of the 2007 ACM workshop on Secure web services* (2007), pp. 99–110.

[57] DI VIMERCATI, S. D. C., FORESTI, S., JAJODIA, S., AND SAMARATI, P. Access control policies and languages in open environments. In *Secure data management in decentralized systems*. Springer, 2007, pp. 21–58.

[58] DIAZ-LOPEZ, D., DOLERA-TORMO, G., GOMEZ-MARMOL, F., AND MARTINEZ-PEREZ, G. Managing xacml systems in distributed environments through meta-policies. *Computers & Security 48* (2015), 92–115.

[59] DO NASCIMENTO, N. M., AND DE LUCENA, C. J. P. Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. *Information Sciences 378* (2017), 161–176.

[60] DOURISH, P., GRINTER, R. E., DELGADO DE LA FLOR, J., AND JOSEPH, M. Security in the wild: user strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing 8*, 6 (2004), 391–401.

[61] DUNLOP, N., INDULSKA, J., AND RAYMOND, K. Methods for conflict resolution in policy-based management systems. In *Seventh IEEE International Enterprise Distributed Object Computing Conference, 2003. Proceedings.* (2003), IEEE, pp. 98–109.

[62] ELLIOTT, A., AND KNIGHT, S. Role explosion: Acknowledging the problem. In *Software Engineering Research and Practice* (2010), Citeseer, pp. 349–355.

[63] ESSMAYR, W., PERNUL, G., AND TJOA, A. M. Access controls by object-oriented concepts. In *Database Security XI*. Springer, 1998, pp. 325–340.

[64] EVERED, M. Object-oriented access control in jarrah. *Journal of Object Technology* (2005).

[65] FERNANDES, E., RAHMATI, A., EYKHOLT, K., AND PRAKASH, A. Internet of things security research: A rehash of old ideas or new intellectual challenges? *IEEE Security & Privacy 15*, 4 (2017), 79–84.

[66] FERRARI, E., SAMARATI, P., BERTINO, E., AND JAJODIA, S. Providing flexibility in information flow control for object oriented systems. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)* (1997), IEEE, pp. 130–140.

[67] FOGG, B. J. *Persuasive technology: Using computers to change what we think and do*. Morgan Kaufmann, 2003.

[68] FORTINO, G., GUERRIERI, A., AND RUSSO, W. Agent-oriented smart objects development. In *Proceedings of the 2012 IEEE 16th international conference on computer supported cooperative work in design (CSCWD)* (2012), IEEE, pp. 907–912.

[69] FORTINO, G., GUERRIERI, A., RUSSO, W., AND SAVAGLIO, C. Towards a development methodology for smart object-oriented iot systems: A metamodel approach. In *2015 IEEE international conference on systems, man, and cybernetics* (2015), IEEE, pp. 1297–1302.

[70] FORTINO, G., RUSSO, W., SAVAGLIO, C., SHEN, W., AND ZHOU, M. Agent-oriented cooperative smart objects: From iot system design to implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 48*, 11 (2017), 1939–1956.

[71] FREUDENTHAL, E., PESIN, T., PORT, L., KEENAN, E., AND KARAMCHETI, V. drbac: distributed role-based access control for dynamic coalition environments. In *Proceedings 22nd International Conference on Distributed Computing Systems* (2002), IEEE, pp. 411–420.

[72] FRUHLINGER, J. What is iot? the internet of things explained. `https://www.networkworld.com/article/3207535/what-is-iot-the-internet-of-things-explained.html`. Last accessed: May 8, 2023.

[73] FURNESS, A. A framework model for the internet of things. *GRIFS/-CASAGRAS: Hong Kong, China* (2008).

[74] GAL, N., GUDESL, E., AND FERNANDEZ, E. B. A model of methods access authorization in object-oriented databases.

[75] GEENG, C., AND ROESNER, F. Who's in control? interactions in multi-user smart homes. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (2019), pp. 1–13.

[76] GOLLMANN, D. Computer security. *Wiley Interdisciplinary Reviews: Computational Statistics 2*, 5 (2010), 544–554.

[77] GOLLMANN, D. *Computer Security (3. ed.)*. Wiley, 2011.

[78] GONG, L., ET AL. A secure identity-based capability system. In *IEEE symposium on security and privacy* (1989), pp. 56–63.

[79] GOOGLE. Google Nest. `https://store.google.com/gb/category/connected_home`. Last accessed: May 8, 2023.

[80] GOOGLE. Google sdk. `https://developers.google.com/assistant/sdk`. Last accessed: May 8, 2023.

[81] GOOGLE. Google smart home compatible devices. =https://developers.google.com/assistant/smarthome/overview. Last accessed: May 8, 2023.

[82] GORTON, S., AND REIFF-MARGANIEC, S. Towards feature interactions in business processes.

[83] GOVINDAVAJHALA, S., AND APPEL, A. W. Windows access control demystified. *Princeton university* (2006).

[84] HART, M., CASTILLE, C., JOHNSON, R., AND STENT, A. Usable privacy controls for blogs. In *2009 International Conference on Computational Science and Engineering* (2009), vol. 4, IEEE, pp. 401–408.

[85] HE, D., BU, J., ZHU, S., CHAN, S., AND CHEN, C. Distributed access control with privacy support in wireless sensor networks. *IEEE Transactions on wireless communications 10*, 10 (2011), 3472–3481.

[86] HE, W., GOLLA, M., PADHI, R., OFEK, J., DÜRMUTH, M., FERNANDES, E., AND UR, B. Rethinking access control and authentication for the home internet of things (iot). In *27th {USENIX} Security Symposium ({USENIX} Security 18)* (2018), pp. 255–272.

[87] HECKMAN, M. R. Every secure system wants to be a reference monitor.

[88] HERNÁNDEZ-RAMOS, J. L., JARA, A. J., MARIN, L., AND SKARMETA, A. F. Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security (JISIS) 3*, 3/4 (2013), 1–16.

[89] HORSTMANN, C. S. *Practical object-oriented development in C++ and Java*. John Wiley & Sons, Inc., 1997.

[90] HU, V. C., FERRAIOLO, D., KUHN, R., FRIEDMAN, A. R., LANG, A. J., COGDELL, M. M., SCHNITZER, A., SANDLIN, K., MILLER, R., SCARFONE, K., ET AL. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication 800*, 162 (2013), 1–54.

[91] HUR, J. Fine-grained data access control for distributed sensor networks. *Wireless Networks 17*, 5 (2011), 1235–1249.

[92] HUSSEIN, D., BERTIN, E., AND FREY, V. A community-driven access control approach in distributed iot environments. *IEEE Communications Magazine 55*, 3 (2017), 146–153.

[93] IFTTT. IFTTT. `https://ifttt.com/`. Last accessed: May 8, 2023.

[94] IFTTT. IFTTT API requirements. `https://platform.ifttt.com/docs/api_reference`. Last accessed: May 8, 2023.

[95] IFTTT. IFTTT compatible devices. `https://ifttt.com/explore/welcome_to_ifttt?utm_source=IFTTT&utm_medium=Tout&utm_campaign=Signedout_Audience`. Last accessed: May 8, 2023.

[96] JANCZEWSKI, L., AND COLARIK, A. *Cyber warfare and cyber terrorism*. IGI Global, 2007.

[97] JEONG, E., WOOD, S., JAMSHED, M., JEONG, H., IHM, S., HAN, D., AND PARK, K. mtcp: a highly scalable user-level {TCP} stack for multicore

systems. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)* (2014), pp. 489–502.

[98] JOHNSON, M., AND STAJANO, F. Usability of security management: Defining the permissions of guests. In *International Workshop on Security Protocols* (2006), Springer, pp. 276–283.

[99] JOHNSON, M. L., BELLOVIN, S. M., REEDER, R. W., AND SCHECHTER, S. E. Laissez-faire file sharing: Access control designed for individuals at the endpoints. In *Proceedings of the 2009 workshop on New security paradigms workshop* (2009), pp. 1–10.

[100] KARP, A. H., HAURY, H., AND DAVIS, M. H. From abac to zbac: the evolution of access control models. *Journal of Information Warfare 9*, 2 (2010), 38–46.

[101] KHALID, T., ABBASI, M. A. K., ZURAIZ, M., KHAN, A. N., ALI, M., AHMAD, R. W., RODRIGUES, J. J., AND ASLAM, M. A survey on privacy and access control schemes in fog computing. *International Journal of Communication Systems 34*, 2 (2021), p. e4181.

[102] KHAN, F. F., AND MALLIKA, R. A survey of object oriented javascript language. *International Journal of Computer Applications 975* (2014), p. 8887.

[103] KIM, J. E., BARTH, T., BOULOS, G., YACKOVICH, J., BECKEL, C., AND MOSSE, D. Seamless integration of heterogeneous devices and access control in smart homes and its evaluation. *Intelligent Buildings International 9*, 1 (2017), 23–39.

[104] KIM, T. H.-J., BAUER, L., NEWSOME, J., PERRIG, A., AND WALKER, J. Access right assignment mechanisms for secure home networks. *Journal of Communications and Networks 13*, 2 (2011), 175–186.

[105] KNORR, K. Dynamic access control through petri net workflows. In *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)* (2000), IEEE, pp. 159–167.

[106] KORTUEM, G., KAWSAR, F., SUNDRAMOORTHY, V., AND FITTON, D. Smart objects as building blocks for the internet of things. *IEEE Internet Computing 14*, 1 (2009), 44–51.

[107] LAMPSON, B. W. Protection. *ACM SIGOPS Operating Systems Review 8*, 1 (1974), 18–24.

[108] LAZOUSKI, A., MARTINELLI, F., AND MORI, P. Usage control in computer security: A survey. *Computer Science Review 4*, 2 (2010), 81–99.

[109] LEI, X., TU, G.-H., LIU, A. X., ALI, K., LI, C.-Y., AND XIE, T. The insecurity of home digital voice assistants–amazon alexa as a case study. *arXiv preprint arXiv:1712.03327* (2017).

[110] LIANG, C.-J. M., KARLSSON, B. F., LANE, N. D., ZHAO, F., ZHANG, J., PAN, Z., LI, Z., AND YU, Y. Sift: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks* (2015), pp. 298–309.

[111] LIN, C. Object-oriented database systems: A survey. *Online. Online, http* (2003).

[112] LIU, L., FENG, J., PEI, Q., CHEN, C., MING, Y., SHANG, B., AND DONG, M. Blockchain-enabled secure data sharing scheme in mobile-edge computing: an asynchronous advantage actor–critic learning approach. *IEEE Internet of Things Journal 8*, 4 (2020), 2342–2353.

[113] LIU, R., WANG, Z., GARCIA, L., AND SRIVASTAVA, M. Remediot: Remedial actions for internet-of-things conflicts. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2019), pp. 101–110.

[114] LORCH, M., PROCTOR, S., LEPRO, R., KAFURA, D., AND SHAH, S. First experiences using xacml for access control in distributed systems. In *Proceedings of the 2003 ACM workshop on XML security* (2003), pp. 25–37.

[115] MARE, S., GIRVIN, L., ROESNER, F., AND KOHNO, T. Consumer smart homes: Where we are and where we need to go. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications* (2019), pp. 117–122.

[116] MAZZOLENI, P., CRISPO, B., SIVASUBRAMANIAN, S., AND BERTINO, E. Xacml policy integration algorithms. *ACM Transactions on Information and System Security (TISSEC) 11*, 1 (2008), 1–29.

[117] MEYER, B. *Object-oriented software construction*, vol. 2. Prentice hall Englewood Cliffs, 1997.

[118] MILLER, M. S., YEE, K.-P., SHAPIRO, J., ET AL. Capability myths demolished. Tech. rep., Technical Report SRL2003-02, Johns Hopkins University Systems Research ..., 2003.

[119] MIORANDI, D., SICARI, S., DE PELLEGRINI, F., AND CHLAMTAC, I. Internet of things: Vision, applications and research challenges. *Ad hoc networks 10*, 7 (2012), 1497–1516.

[120] MOHAMMAD, Z. N., FARHA, F., ABUASSBA, A. O., YANG, S., AND ZHOU, F. Access control and authorization in smart homes: A survey. *Tsinghua Science and Technology 26*, 6 (2021), 906–917.

[121] NANDY, T., IDRIS, M. Y. I. B., NOOR, R. M., KIAH, L. M., LUN, L. S., JUMA'AT, N. B. A., AHMEDY, I., GHANI, N. A., AND BHATTACHARYYA, S. Review on security of internet of things authentication mechanism. *IEEE Access 7* (2019), 151054–151089.

[122] NAYAK, A. K., REIMERS, A., FEAMSTER, N., AND CLARK, R. Resonance: Dynamic access control for enterprise networks. In *Proceedings of*

the 1st ACM workshop on Research on enterprise networking (2009), pp. 11–18.

[123] OMZ:SOFTWARE, O. Pythonista 3, Jun 2016.

[124] OSBORN, S., SANDHU, R., AND MUNAWER, Q. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC) 3*, 2 (2000), 85–106.

[125] OUADDAH, A., ABOU ELKALAM, A., AND AIT OUAHMAN, A. Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and communication networks 9*, 18 (2016), 5943–5964.

[126] OUADDAH, A., MOUSANNIF, H., AND OUAHMAN, A. A. Access control models in iot: The road ahead. In *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)* (2015), IEEE, pp. 1–2.

[127] PALLAS, F. Information security inside organizations-a positive model and some normative arguments based on new institutional economics. *Available at SSRN 1471801* (2009).

[128] PARKER, D. B. *Fighting computer crime: A new framework for protecting information*. John Wiley & Sons, Inc., 1998.

[129] PEREIRA, H. G., AND FONG, P. W. Sepd: An access control model for resource sharing in an iot environment. In *European Symposium on Research in Computer Security* (2019), Springer, pp. 195–216.

[130] PEREIRA, O. M., AGUIAR, R. L., AND SANTOS, M. Y. Crud-dom: a model for bridging the gap between the object-oriented and the relational paradigms. In *2010 Fifth International Conference on Software Engineering Advances* (2010), IEEE, pp. 114–122.

[131] PEREIRA, O. M., REGATEIRO, D. D., AND AGUIAR, R. L. Distributed and typed role-based access control mechanisms driven by crud expressions. *International Journal of Computer Science Theory and Application 2*, 1 (2014), 1–11.

[132] PESCATORE, J., AND SHPANTZER, G. Securing the internet of things survey. *SANS Institute* (2014), 1–22.

[133] PETRITSCH, H. *Break-glass: handling exceptional situations in access control*. Springer, 2014.

[134] POVEY, D. Optimistic security: a new access control paradigm. In *Proceedings of the 1999 workshop on New security paradigms* (1999), pp. 40–45.

[135] RANJAN, A. K., AND SOMANI, G. Access control and authentication in the internet of things environment. In *Connectivity Frameworks for Smart Devices*. Springer, 2016, pp. 283–305.

[136] RAVIDAS, S., LEKIDIS, A., PACI, F., AND ZANNONE, N. Access control in internet-of-things: A survey. *Journal of Network and Computer Applications 144* (2019), 79–101.

[137] ROMAN, R., ZHOU, J., AND LOPEZ, J. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks 57*, 10 (2013), 2266–2279.

[138] ROTONDI, D., AND PICCIONE, S. Managing access control for things: a capability based approach. In *BodyNets* (2012), pp. 263–268.

[139] SAMONAS, S., AND COSS, D. The cia strikes back: Redefining confidentiality, integrity and availability in security. *Journal of Information System Security 10*, 3 (2014).

[140] SANDHU, R. S. Lattice-based access control models. *Computer 26*, 11 (1993), 9–19.

[141] SANDHU, R. S. Role-based access control. In *Advances in computers*, vol. 46. Elsevier, 1998, pp. 237–286.

[142] SANDHU, R. S., AND SAMARATI, P. Access control: principle and practice. *IEEE communications magazine 32*, 9 (1994), 40–48.

[143] SFAR, A. R., NATALIZIO, E., CHALLAL, Y., AND CHTOUROU, Z. A roadmap for security challenges in the internet of things. *Digital Communications and Networks 4*, 2 (2018), 118–137.

[144] SHAHID, M. R., BLANC, G., ZHANG, Z., AND DEBAR, H. Iot devices recognition through network traffic analysis. In *2018 IEEE International Conference on Big Data (Big Data)* (2018), IEEE, pp. 5187–5192.

[145] SHEN, H., AND DEWAN, P. Access control for collaborative environments. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (1992), pp. 51–58.

[146] SHIN, D.-G., AND JUN, M.-S. Home iot device certification through speaker recognition. In *2015 17th International Conference on Advanced Communication Technology (ICACT)* (2015), IEEE, pp. 600–603.

[147] SHOURAN, Z., ASHARI, A., AND PRIYAMBODO, T. Internet of things (iot) of smart home: privacy and security. *International Journal of Computer Applications 182*, 39 (2019), 3–8.

[148] SIFOU, F., ALSHAHWAN, F., MARWAN, M., HAMMOUD, A., AND HAMMOUCH, A. Implementing policy rules in attributes based access control with xacml within a cloud-enabled iot environment. *Journal of Communications 15*, 1 (2020), 107–114.

[149] SIKDER, A. K., BABUN, L., CELIK, Z. B., ACAR, A., AKSU, H., MCDANIEL, P., KIRDA, E., AND ULUAGAC, A. S. Kratos: Multi-user multi-device-aware access control system for the smart home. In *Proceedings of*

the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (2020), pp. 1–12.

[150] SPRING, J. M. *Human decision-making in computer security incident response*. PhD thesis, UCL (University College London), 2019.

[151] STAJANO, F., AND LOMAS, M. User authentication for the internet of things. In *Cambridge International Workshop on Security Protocols* (2018), Springer, pp. 209–213.

[152] STANDARD, O. extensible access control markup language (xacml) version 3.0, 2013.

[153] STATISTA. Average household size in the united kingdom from 1996 to 2020. `https://www.statista.com/statistics/295551/average-household-size-in-the-uk/`.

[154] STIEGLER, M., AND MILLER, M. How emily tamed the caml. *Hewlett Packard Labs Tech Report* (2006).

[155] STIPEK, P., KRALIK, L., AND SENKERIK, R. Object oriented role-based access control. *Proceedings of SECUREWARE'16* (2016), 76–81.

[156] SUNDMAEKER, H., GUILLEMIN, P., FRIESS, P., WOELFFLÉ, S., ET AL. Vision and challenges for realising the internet of things. *Cluster of European research projects on the internet of things, European Commision 3*, 3 (2010), 34–36.

[157] SURBATOVICH, M., ALJURAIDAN, J., BAUER, L., DAS, A., AND JIA, L. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In *Proceedings of the 26th International Conference on World Wide Web* (2017), pp. 1501–1510.

[158] TACHIKAWA, T., HIGAKI, H., AND TAKIZAWA, M. Purpose-oriented access control model in object-based systems. In *Australasian Conference on Information Security and Privacy* (1997), Springer, pp. 38–49.

[159] TEAM, H. A. C., AND COMMUNITY. Home Assistant. `https://www.home-assistant.io/`. Last accessed: May 8, 2023.

[160] TEAM, H. A. C., AND COMMUNITY. Home Assistant compatible devices. `https://www.home-assistant.io/integrations/homekit_controller/`. Last accessed: May 8, 2023.

[161] TER BEEK, M. H., GNESI, S., MONTANGERO, C., AND SEMINI, L. Detecting policy conflicts by model checking uml state machines. In *ICFI* (2009), pp. 59–74.

[162] TILAK, S., ABU-GHAZALEH, N. B., AND HEINZELMAN, W. A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review 6*, 2 (2002), 28–36.

[163] TÖNJES, R., BARNAGHI, P., ALI, M., MILEO, A., HAUSWIRTH, M., GANZ, F., GANEA, S., KJÆRGAARD, B., KUEMPER, D., NECHIFOR, S., ET AL. Real time iot stream processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications* (2014), sn, p. 10.

[164] TORY, M., AND MOLLER, T. Rethinking visualization: A high-level taxonomy. In *IEEE symposium on information visualization* (2004), IEEE, pp. 151–158.

[165] TRUICA, C.-O., RADULESCU, F., BOICEA, A., AND BUCUR, I. Performance evaluation for crud operations in asynchronously replicated document oriented database. In *2015 20th International Conference on Control Systems and Computer Science* (2015), IEEE, pp. 191–196.

[166] TRYFONAS, T., GRITZALIS, D., AND KOKOLAKIS, S. A qualitative approach to information availability. In *IFIP International Information Security Conference* (2000), Springer, pp. 37–47.

[167] UR, B., JUNG, J., AND SCHECHTER, S. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS)* (2013), vol. 29, HUPS 2014, pp. 209–218.

[168] WANG, D., JIN, H., ZOU, D., XU, P., ZHU, T., AND CHEN, G. Taming transitive permission attack via bytecode rewriting on android application. *Security and Communication Networks 9*, 13 (2016), 2100–2114.

[169] WANG, F., AND TURNER, K. J. Policy conflicts in home care systems. In *ICFI 2007-Ninth International Conference on Feature Interactions in Software and Communication Systems* (2008), IOS Press, pp. 54–65.

[170] WANG, Y., ZHANG, H., DAI, X., AND LIU, J. Conflicts analysis and resolution for access control policies. In *2010 IEEE International Conference on Information Theory and Information Security* (2010), IEEE, pp. 264–267.

[171] YUAN, X., CHEN, Y., WANG, A., CHEN, K., ZHANG, S., HUANG, H., AND MOLLOY, I. M. All your alexa are belong to us: A remote voice control attack against echo. In *2018 IEEE Global Communications Conference (GLOBECOM)* (2018), IEEE, pp. 1–6.

[172] ZDRAVKOVIĆ, M., ZDRAVKOVIĆ, J., AUBRY, A., MOALLA, N., GUEDRIA, W., AND SARRAIPA, J. Domain framework for implementation of open iot ecosystems. *International Journal of Production Research 56*, 7 (2018), 2552–2569.

[173] ZENG, E., MARE, S., AND ROESNER, F. End user security and privacy concerns with smart homes. In *thirteenth symposium on usable privacy and security (SOUPS 2017)* (2017), pp. 65–80.

[174] ZENG, E., AND ROESNER, F. Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study. In *28th USENIX Security Symposium* (2019), pp. 159–176.