An Approach to Enhancing Security and Privacy of the Internet of Things with Federated Identity

Paul Zachary Fremantle

A thesis submitted in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy.

School of Computing
University of Portsmouth

July 2017

Abstract

The Internet of Things (IoT) is the set of systems that enable sensors and actuators to be connected to the Internet. It is estimated that there are already more IoT devices than humans, and that by 2020 there will be 50 billion connected devices. A review of related literature outlines concerns regarding security and privacy of the IoT, demonstrating that IoT devices are creating the opportunity to infringe on security and privacy in numerous ways. One significant challenge is to manage the identity of IoT devices in an effective way. Many IoT systems are built using middleware systems. The main research question of this thesis is whether an improved model for IoT middleware systems — based around federated identity — can provide significant improvements to security and privacy while maintaining reasonable costs in terms of user experience and performance.

In a review of related work, a matrix of IoT threats is presented and from this a number of requirements are identified. A structured survey of literature around IoT middleware systems and platforms identifies 20 systems which are evaluated against those requirements. From this, a set of gaps in IoT middleware systems are identified.

This work addresses a number of these gaps in a novel approach for linking IoT devices to cloud and web systems. A proposed architecture supports an integrated set of privacy preserving controls based on federated identity and access management patterns. In particular, a model introduces device and user registration processes that are adapted to support constrained IoT devices. Federation and de-coupling of systems are incorporated to allow choice of where data is shared with the result that users can choose to avoid sharing data with systems that may infringe privacy. Users are automatically provisioned with a cloud service that manages their devices and data. Summarisation and filtering of data are incorporated to protect raw data and prevent fingerprinting attacks.

A formal model of the approach is presented and properties are proved mathematically, and these properties are used to inform a threat model of the system, which demonstrates benefits of the model in enhancing privacy and security.

The model is implemented in a prototype system and experimental results on this system are presented, including energy usage, cost, scalability and performance. The prototype demonstrates that the approach is both feasible and cost-effective. Performance data demonstrates that the impact on users of the approach is minimal and within norms for such systems. Finally, areas of further research are presented.

Declaration

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

46,548 words

Dissemination

Several papers have been published during the preparation of this thesis, in addition to participation in posters, demos and presentations. This section provides a summary. Where material that has been included within this thesis has previously been jointly published, in all cases I was the lead author, and responsible for the technical content of the papers. The items listed as *Contributions as a Co-Author*, where I was not the lead author, do not form any substantive part of this thesis.

Book Chapters

Paul Fremantle. "A Security Survey of Middleware for the IoT". in: *Engineering Secure Internet of Things Systems*. IET, 2016. Chap. 1

Paul Fremantle. "Federated Identity and Access Management in IoT Systems". In: *Engineering Secure Internet of Things Systems*. IET, 2016. Chap. 6

Journal Articles

Paul Fremantle and Philip Scott. "A survey of secure middleware for the Internet of Things". In: *PeerJ Computer Science* 3 (2017), e114

Conference Proceedings

Paul Fremantle, Benjamin Aziz, and Tom Kirkham. "Enhancing IoT Security and Privacy with Distributed Ledgers - A Position Paper". In: *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*. INSTICC. ScitePress, 2017, pp. 344–349. ISBN: 978-989-758-245-5. DOI: 10.5220/0006353903440349

P. Fremantle and B. Aziz. "OAuthing: Privacy-enhancing federation for the Internet of Things". In: *2016 Cloudification of the Internet of Things (CIoT)*. Nov. 2016, pp. 1–6. DOI: 10.1109/CIOT.2016.7872911

Workshop Proceedings

Paul Fremantle et al. “Federated Identity and Access Management for the Internet of Things”. In: *3rd International Workshop on the Secure IoT*. 2014

Paul Fremantle, Jacek Kopecký, and Benjamin Aziz. “Web API Management Meets the Internet of Things”. In: *The Semantic Web: ESWC 2015 Satellite Events: ESWC 2015 Satellite Events, Portorož, Slovenia, May 31 – June 4, 2015, Revised Selected Papers*. Ed. by Fabien Gandon et al. Cham: Springer International Publishing, 2015, pp. 367–375. ISBN: 978-3-319-25639-9. DOI: 10.1007/978-3-319-25639-9_49. URL: http://dx.doi.org/10.1007/978-3-319-25639-9_49

Selected as Joint Best Paper at the Workshop

Paul Fremantle. “Privacy-enhancing Federated Middleware for the Internet of Things”. In: *Proceedings of the Doctoral Symposium of the 17th International Middleware Conference*. Middleware Doctoral Symposium’16. Trento, Italy: ACM, 2016, 4:1–4:4. ISBN: 978-1-4503-4665-8. DOI: 10.1145/3009925.3009929. URL: http://doi.acm.org/10.1145/3009925.3009929

Presentations And Posters

Paul Fremantle and Benjamin Aziz. “Privacy-enhancing Federated Middleware for the Internet of Things”. In: *Proceedings of the Posters and Demos Session of the 17th International Middleware Conference*. Middleware Posters and Demos ’16. Trento, Italy: ACM, 2016, pp. 33–34. ISBN: 978-1-4503-4666-5. DOI: 10.1145/3007592.3007596. URL: http://doi.acm.org/10.1145/3007592.3007596

OAuthing: Enhancing privacy and security for the Internet of Things, *School of Computing Research Seminar Presentation*, University of Portsmouth, September 2016

Federated Identity and Access Management for the Internet of Things, *School of Computing Research Seminar Presentation*, University of Portsmouth, October 2014

Federated Identity and Access Management for the Internet of Things, *Technology Faculty Research Conference Poster*, University of Portsmouth, July 2014

Towards a Secure Private Middleware for the Internet of Things, *Research Seminar, Institute for the Architecture of Applications and Systems*, Stuttgart, Germany, October 2016

The Identity of Things, *Presentation to Daimler-Benz*, Stuttgart, Germany, October 2016

The Identity of Things, *Presentation to Zühlke Engineering*, Eschborn, Germany, October 2016

Your Thing is Pwned - security challenges for IoT, *Presentation at DSS ITSEC 2016, 7th International Cyber Security Conference*, Riga, Latvia, October 2016

Your Thing is Pwned, *Presentation at RSM Telemedecine and eHealth, Big data, clouds, and the internet of healthy things*, London, June 2016

Technical Reports

Paul Fremantle. *A Reference Architecture for the Internet of Things*. Tech. rep. WSO2, 2014

Contributions As A Co-Author

Benjamin Aziz, Paul Fremantle, and Alvaro Arenas. "A reputation model for the Internet of Things". In: *Engineering Secure Internet of Things Systems*. IET, 2016. Chap. 10

Jacek Kopecký, Paul Fremantle, and Rich Boakes. "A history and future of Web APIs". In: *Information Technology* (2014)

Tom Kirkham et al. "Privacy Aware On-Demand Resource Provisioning for IoT Data Processing". In: *Internet of Things. IoT Infrastructures: Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015, Revised Selected Papers, Part II*. Springer International Publishing. 2016, pp. 87–95

Boris Adryan, Dominik Obermaier, and Paul Fremantle. *The Technical Foundations of IoT*. Artech House, 2017. ISBN: 9781630812515

Acknowledgements

Firstly, I would like to acknowledge my first supervisor, Benjamin Aziz, who has provided clear advice, unswerving support, constant effort, and a wealth of knowledge in security and IoT. Thanks Ben.

Secondly, I'd like to thank the other academics who have provided supervision — informally and formally. Frank Leymann has been a mentor to me for more than 15 years. His advice is always brilliant, and his questions have been even more valuable. Jacek Kopecký is one of the smartest people in the department and every time I talk with him I learn something and come away doubting everything I thought I knew. Philip Scott welcomed me to the department and has been invaluable in helping me become at least a little bit academic. Mads Ohm Larsen provided invaluable advice and proofreading around CSP. Nick Savage always cuts to the chase and sees through the trees to the wood.

Thirdly, I need to thank WSO2: not only for giving me the time to do this — but also for giving me the opportunity to meet so many amazing customers who need identity, security and privacy of IoT and helped me to motivate the requirements. I can't possibly name all the WSO2 team who encouraged me, but I specifically want to thank Prabath Siriwardena for introducing me to the mysteries of OAuth2 and Sanjiva Weerawarana for encouraging me to work towards a doctorate.

Finally, to Jane, Anna and Dan for all your support, patience, and love. The three of you inspire me in everything I do.

Abbreviations

ABAC Attribute Based Access Control

ACE Authentication and Authorisation for Constrained Environments

AES Advanced Encryption Standard

API Application Programming Interface

AuthCode Authorisation Code

BLE Bluetooth Low Energy

CCS Communicating Concurrent Systems

CIA Confidentiality Integrity Availability

CIA+ "CIA Plus"

CoAP Constrained Application Protocol

CSP Communicating Sequential Processes

DCR Dynamic Client Registration

DDoS Distributed Denial of Service

DoS Denial of Service

DTLS Datagram Transport Layer Security

ECC Elliptic Curve Cryptography

EXI Efficient XML Interchange

FDR Failures Divergences Refinement

FIAM Federated Identity and Access Management

FIOT Federation of IoT

GCHQ UK Government Communications Headquarters

GDPR General Data Protection Regulations

GPRS General Packet Radio Service

HIP Host Identity Protocol

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force

IGNITE Intelligent Gateway for Networked Internet of Things Environments

IoT Internet of Things

JSON JavaScript Object Notation

M2M Machine to Machine

MAC Media Access Control

MQTT Message Queueing Telemetry Transport

NFC Near Field Communication

NSA US National Security Agency

OIDC OpenID Connect

PBD Privacy By Design

PCM Personal Cloud Middleware

PIN Personal Identification Number

PKC Public Key Cryptography

PKI Public Key Infrastructure

PoP Proof of Possession

PSK Pre Shared Key

PZH Personal Zone Hub

QR Quick Response

RFID Radio Frequency Identification Device

RSA Rivest, Shamir, and Adleman public key cryptography

SGX Secure Guard Extensions

SOAP Simple Object Access Protocol

SoC System-on-Chip

TCP Transport Control Protocol

TLS Transport Layer Security

TPM Trusted Platform Module

UDP User Datagram Protocol

UI User Interface

UMA User Managed Access

UML Unified Modelling Language

URU User Registration URI

VM Virtual Machine

WAM Web API Management

WEP Wired Equivalency Privacy

WSN Wireless Sensor Network

XACML XML Access Control Markup Language

XML eXtensible Markup Language

Contents

List of Figures

List of Tables

Part I

Introduction, Literature Review and Core Technologies

Chapter 1

Introduction

1.1 Motivation

The Internet of Things (IoT) was originally coined as a phrase by Kevin Ashton in 1990 [19], with reference to "taggable" items that used Radio Frequency Identification Device (RFID) chips to become electronically identifiable and therefore amenable to interactions with the Internet. With the ubiquity of cheap processors and System-on-Chip (SoC) based devices, the definition has expanded to include wireless and Internet-attached sensors and actuators, including smart meters, home automation systems, Internet-attached set-top-boxes, smartphones, connected cars, and other systems that connect the physical world to the Internet either by measuring it or affecting it.

There are a number of definitions of IoT. For the purposes of this work, IoT will be defined in the following way. An IoT *device* is a system that contains either *sensors* or *actuators* or both and supports connection to the Internet either directly or via some intermediary. A sensor is a sub-component of an IoT device that measures some part of the world, allowing the IoT device to update Internet and Cloud systems with this information. A sensor may be as simple as a button (e.g. Amazon Dash Button). Sensors widely deployed include weather sensors (barometers, anemometers, thermometers), accelerometers and GPS units, light sensors, air quality sensors, people-counters, as well as medical sensors (blood sugar, heart rate, etc), industrial sensors (production line monitoring, etc) and many more. Actuators are electronically controlled systems that affect the physical world. These includes lights, heaters, locks, motors, pumps, and relays. Therefore the IoT is the network of such devices together with the Internet systems that are designed to inter-operate and communicate with those devices, including the websites, cloud servers, gateways and so forth.

The number of IoT devices has grown rapidly, with a recent estimate suggesting that

there were 12.5 billion Internet attached devices in 2010 and a prediction of 50 billion devices by 2020 [88]. This brings with it multiple security challenges:

- The sheer scale and number of predicted devices will create new challenges and require new approaches to security.
- These devices are becoming more central to people's lives, including in safety critical systems such as cars. Therefore the security of IoT devices is becoming more important.
- Many IoT devices collect information that may be *fingerprinted* [146] and therefore become personally identifiable. This can lead to privacy concerns.
- Because devices can affect the physical world, there are attacks that can cause physical harm to people and systems.
- These devices, due to size and power limitations, may not support the same level of security that is expected from more traditional Internet connected systems.

Because of the pervasive nature of IoT, privacy and security are important areas for research. In 2016, more than 100,000 IoT devices were conjoined into a hostile botnet named Mirai that attacked the DNS servers of the east coast of the US [283]. The total attack bandwidth of this system was measured at more than 600Gbps. In fact, the number of devices attacked was a small number compared to the potential: previous research [66] has identified several million devices that are available for attack.

Privacy and security are best defined in terms of specific properties that can either be kept or broken by systems or people. Broadly speaking, security properties are those which prevent attackers from breaking, misusing, or stealing access to systems or data. Broadly speaking, privacy properties allow people to act without themselves or their actions being identifiable. A detailed description of security and privacy properties and threats comes in Chapter 2.

Therefore there is a strong motivation to find approaches to improve and enhance the security and privacy of the IoT. Many IoT projects use existing *platforms*, also known as *middleware* to build upon. The Oxford English Dictionary [73] defines middleware as:

> Software that acts as a bridge between an operating system or database and applications, especially on a network.

Such systems can either improve security or reduce it: if the platform is built with privacy and security in mind then such systems can embed best-practices and enable system designers to rapidly create secure systems. If platforms are built without security, or security is added as an after-thought, then it is possible that not only does the platform encourage the creation of insecure, privacy-negating systems, but also that

it may make it more difficult to add security when problems are found. The creation of systems with security and privacy as a key design principle is known as *Privacy by Design* [54]. The security and privacy capabilities and weaknesses of existing IoT middleware are explored in Chapter 3.

The remainder of this chapter explores the thesis from multiple dimensions, including the methodology, the research questions and the contributions. In each case, there are pointers to the remaining chapters. In addition, the overall chapter outline provides a clear outline of each chapter.

1.2 Thesis Approach

The aim of this thesis is to look at challenges in the security and privacy of IoT systems, and to identify improved approaches to address these, especially the creation of a secure, privacy-enhancing platform.

Figure 1.1 shows the overall approach.

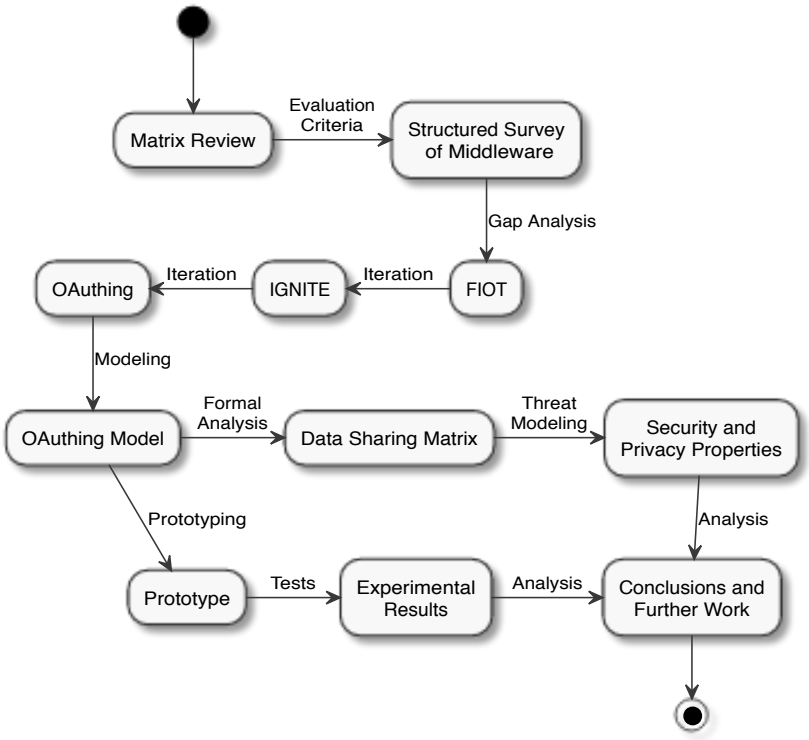

Figure 1.1: Thesis Approach

The thesis structure starts with a broad literature review of IoT threats, challenges and approaches (labelled *Matrix Review*). This approach is enhanced by the creation of a *threat matrix*, which emerged as part of the review and was then used to inform

the review still further. The outcome of this part of the literature review is a set of requirements for IoT systems to implement. This set of requirements is used as part of a literature review of existing IoT middleware (labeled *Structured Survey of Middleware*): this uses a structured survey methodology to identify a set of IoT systems and middleware that are evaluated on the basis of their security attributes and their ability to meet the previously identified requirements. This is used to identify a set of gaps in the available systems. This literature review was published in [93], with an updated version published in [102].

These gaps are used to inform a set of approaches that were created to address them. The first prototype approach that was implemented is named *FIOT*. This work, which was originally published in [103], described experiments and findings in using token-based, federated authentication and authorisation models in IoT. Based on gaps in this work, a second prototype was created (named *IGNITE*). This work, which was published in [101], improves upon the *FIOT* system in resolving specific security concerns, especially around the refreshing of tokens. In addition, experimental testing on *IGNITE* provided comparisons with existing approaches. Both of these prototypes can be considered "preliminary investigations". Aspects of the approach — but not the actual code — of the first IGNITE prototype are re-used as part of the later work.

Based on the evaluations of these first prototypes, a new model, named *OAuthing* was created. This model incorporates a number of important improvements upon the previous approaches and forms the main basis for the research in this thesis. The new model extends the federated identity and access control approach into a wider middleware model for IoT. OAuthing demonstrates device and user registration flows, pseudonymous interactions, and the use of *personal cloud middleware*, where each user has their own runtime to control sharing of data and managing external commands to their devices.

OAuthing is formally modeled in Communicating Sequential Processes (CSP), a process algebra. This formal model is used to prove specific properties of the model, including end-to-end consent-based authorisation and pseudonymity. The proven properties of the model include a *data sharing matrix*, which is then used to create a threat model. The threat model uses side-by-side approaches based on published threat modeling methodologies to analyse the data sharing matrix and identify the enhancements to security and privacy that OAuthing provides.

A prototype of this model was created, including an updated version of IGNITE. In addition a test harness and test system was created, and this was used to perform controlled experiments on the OAuthing system. The results of this are presented. The results of the theoretical threat modeling and experimental testing of the system are analysed, identifying the strengths and weaknesses, together with areas of further research. The initial research into the OAuthing model was published in [95, 97].

Although this approach is presented in a broadly linear format, it is clear that there were multiple iterations. The three prototypes demonstrate that, but in addition, each prototype explored different approaches with further iterations in each case.

The overall flow of this thesis demonstrates a clear consistent, logical progression from: the threats, gaps and requirements identified in the literature review; the model developed and analysed; the threat modeling; through the creation of a prototype of the model; the experimental testing of the prototype; the analysis of the test results; to the conclusions.

1.3 Research Questions

The research sets out to answer specific research questions around the security and privacy of IoT. These research questions are summarised here and then the results of these are addressed in Chapter 11.

RQ1

What are the privacy and security requirements, especially those on identity and access control, for the Internet of Things? How do they differ from the existing requirements on the classical Internet.

RQ2

What is a model and architecture for IoT systems that can meet the requirements identified in RQ1? What are the threats and risks of this compared to the risks and threats identified in RQ1?

RQ3

Is there a practical instantiation of this architecture, and if so, what are the increased costs in complexity, performance, latency and resources compared to existing systems.

RQ1 is addressed in Chapters 2 and 3. RQ2 is addressed in Chapters 6 and 7. RQ3 is addressed in Chapter 9.

1.4 Summary Of Contributions

The contributions of this work in addressing these research questions include:

- A matrix model for evaluating threats to IoT systems (in Chapter 2) and a structured literature review of security of middleware systems for IoT (in Chapter 3).

- An architecture and system model that allows the decoupling of multiple parties: the manufacturer, the identity provider, the device identity management, and cloud services and applications that require access to IoT data (in Chapter 6). This decoupling encourages choice of provider as well as reducing the data available in any given attack. This model includes:
  - Clearly defined device and user registration processes, based on the OAuth2 protocol, that have been extended to support IoT devices and be effective in device scenarios (in Chapter 6).
  - An approach to pseudonymity for users in IoT systems, reducing the chance that leaked data can be tied to users (in Chapter 6).
  - A model and architecture that provides each user with a separate cloud instance to handle sharing device data, and an approach for dynamically provisioning these cloud instances (in Chapters 6 and 8).
  - A threat model based on structured approaches to threat modeling, including the *STRIDE* and *LINDDUN* approaches (in Chapter 7)
- A demonstration of the workability of the model through the creation and testing of a working prototype (in Chapter 8), including:
  - The creation of a cloud-based test harness for emulating devices and collecting test results, as well as a test harness for measuring power and energy consumption in IoT devices (in Chapter 9).
  - Experimental results including energy and power consumption, performance, cost and capacity metrics of the prototype (in Chapter 9).

1.5 Chapter Outline

In Chapter 2, an improved security threat matrix is proposed into which IoT security threats and challenges are modelled. This matrix is used as a basis for a wide ranging review of existing threats and work to address these. Out of this review, criteria are identified for judging IoT middleware systems to evaluate their security and privacy capabilities.

In Chapter 3 a structured literature review approach is used to identify IoT middleware systems and these are evaluated using the criteria from the previous section. The outcome of this analysis is both an understanding of best practices in the IoT middleware space and the identification of gaps in the current best practice.

In Chapter 4, there is an introduction to two core technologies used in this work, specifically the OAuth2 authorisation framework and MQTT.

In Chapter 5, two preliminary investigations (FIOT and IGNITE) into identity and access management for IoT are examined. These systems were prototyped and the results are presented.

In Chapter 6, the creation of an improved model, OAuthing, is presented. This model improves on the preliminary systems and provides enhanced security and privacy. The model is presented in CSP [126] with some non-normative Unified Modelling Language (UML) [227] diagrams added to aid communication. The CSP model is used to prove specific properties of the model.

These properties are then used to create a threat model of the system, which is presented in Chapter 7.

In Chapter 8, a prototype system is presented. The prototype demonstrates full end-to-end flows of the system from secured device through to third-party data recipients.

In Chapter 9 there is a description of the test harness that was created in order to run the experiments, together with information on the test methodology. In this chapter there are also the results of performance tests and cost analysis of the prototype.

In Chapter 10, the approach is compared to related work.

In Chapter 11, strengths and weaknesses of the thesis are presented, and the impact of this work is assessed. Further work is identified and conclusions are drawn.

Chapter 2

Security and Privacy Threats for IoT

2.1 Summary

The rapid growth of small Internet connected devices, known as the Internet of Things (IoT), is creating a new set of challenges to create secure, private infrastructures. The purpose of this chapter is to review the current literature on the challenges and approaches to security and privacy in the Internet of Things.

In order to understand the security threats against the Internet of Things, there is a need for an approach to classifying threats. The most widely used ontology of security threats is the Confidentiality Integrity Availability (CIA) triad [206] which has been extended over time. This extended ontology is now often referred to as the "CIA Plus" (CIA+) model [250]. In the course of reviewing the available literature and approaches to IoT security, a new approach was created. This proposed expansion of the existing ontology is a simple matrix model that specifically targets the IoT space. In particular, this ontology creates a matrix of evaluation where each of the classic security challenges is evaluated against three different aspects: device/hardware, network, and cloud/server-side. In some cells in this matrix, no challenges were identified areas where the IoT space presents a difference to classical Internet systems: in other words, whilst the domain space covered by these cells contains security challenges, those challenges are no different from existing Web and Internet security challenges in that domain. In those cells the challenges are "unchanged". In other cells the challenges that are significantly modified by the unique nature of the Internet of Things are called out.

In addition to the matrix, the *Three Layer Privacy Model* from Spiekermann and Cranor [255] is used to explore privacy concerns in more detail.

Security Characteristic	A. Device / Hardware	B. Network	C. Cloud / Server-Side
1. Confidentiality	A1. Hardware attacks	B1. Encryption with low capability devices	C1. Privacy Data leaks Fingerprinting
2. Integrity	A2. Spoofing; Lack of attestation	B2. Signatures with low capability devices Sybil attacks	C2. No common device Identity
3. Availability	A3. Physical attacks;	B3. Unreliable networks, DDoS, Radio jamming	C3. DDoS (as usual)
4. Authentication	A4. Lack of UI, Default Passwords, Hardware secret retrieval	B4. Default Passwords, lack of secure identities	C4. No common device identity, insecure flows
5. Access Control	A5. Physical access; Lack of local authentication	B5. Lightweight distributed protocols for Access Control	C5. Inappropriate use of traditional ACLs, Device Shadow
6. Non-Repudiation	A6. No secure local storage; No attestation, forgery	B6. Lack of Signatures with low capability devices	C6. Lack of secure identity and signatures

Table 2.1: Matrix of Security Challenges for the IoT

Table 2.1 shows the matrix that will be used for evaluating security challenges. In each cell of the table, the main challenges are summarised that are *different* in the IoT world, or exacerbated by the challenges of IoT compared to existing Internet security challenges. Each cell in the matrix is explored in detail below. Each of the cells is given a designation from *A1* to *C6* and these letters are used as a key to refer to the cells below.

The three aspects (Hardware/Device, Network, Cloud/Server) were chosen because as the available literature was studied, these areas offered a clear way of segmenting the unique challenges within the context of the IoT. These form a clear logical grouping of the different assets involved in IoT systems. Before the matrix details are presented, a short overview of each column heading is provided.

Device and Hardware

IoT devices have specific challenges that go beyond those of existing Internet clients. These challenges come from: the different form factors of IoT devices, from the power requirements of IoT devices, and from the hardware aspects of IoT devices. The rise of cheap mobile telephony has driven down the costs of 32-bit processors (especially those based around the ARM architecture [106]), and this is increasingly creating lower cost microcontrollers and System-on-Chip (SoC) devices based on ARM. However, there are still many IoT devices built on 8-bit processors, and occasionally, 16-bit [280]. In particular the open source hardware platform Arduino [16] supports both 8-bit and 32-bit controllers, but the 8-bit controllers remain considerably cheaper and at the time of writing are still widely used.

The challenges of low-power hardware mean that certain technologies are more or less suitable. The detailed description of each cell below will address and analyse specific details as they pertain to security. In addition, there are specific protocols and approaches designed for IoT usage that use less power and are more effective. In [111] there is a comparison of eXtensible Markup Language (XML) parsing with binary alternatives. The processing time on a constrained device is more than a magnitude slower using XML, and that the heap memory used by XML is more than 10Kb greater than with binary formats. These improvements result in a 15% saving in power usage in their tests. XML security standards such as XML Encryption and the related WS-Encryption standard have significant problems in an IoT device model. For example, any digital signature in XML Security needs a process known as XML Canonicalisation (XML C14N). XML Canonicalisation is a costly process in both time and memory. [40] shows that the memory usage is more than $10\times$ the size of the message in memory (and XML messages are already large for IoT devices). A search was conducted for any work on implementing WS-Security on Arduino, ESP8266 or Atmel systems (which are common targets for IoT device implementations) without success. XML performance on small devices can be improved using Efficient XML Interchange (EXI), which reduces network traffic [157].

Network

IoT devices may use much lower power, lower bandwidth networks than existing Internet systems. Cellular networks often have much higher latency and more "dropouts" than fixed networks [55]. The protocols that are used for the Web are often too data-intensive and power-hungry for IoT devices. Network security approaches such as encryption and digital signatures are difficult and in some cases impractical in small devices. New low-power, low-bandwidth networks

such as LoRaWan [1] are gaining significant traction.

There have been some limited studies comparing the power usage of different protocols. In [157] there is comparison of using Constrained Application Protocol (CoAP) with EXI against HyperText Transfer Protocol (HTTP), showing efficiency gains in using CoAP. In [191], MQTT over TLS is shown to use less power than HTTP over TLS in several scenarios. In [264], there is a comparison of network traffic between CoAP and Message Queueing Telemetry Transport (MQTT) showing that each performs better in different scenarios, with similar overall performance. This is an area where more study is clearly needed, but it is a valid conclusion to draw that traditional protocols such as Simple Object Access Protocol (SOAP) are unsuited to IoT usage.

Cloud/Server-Side

While many of the existing challenges apply here, there are some aspects that are exacerbated by the IoT for the server-side or cloud infrastructure. These include: the often highly personal nature of data that is being collected and the requirement to manage privacy; the need to provide user-managed controls for access; and the lack of clear identities for devices making it easier to spoof or impersonate devices.

2.2 A1. Device Confidentiality

Hardware devices have their own challenges for security. There are systems that can provide tamper-proofing and try to minimise attacks, but if an attacker has direct access to the hardware, they can often break it in many ways. For example, there are devices that will copy the memory from flash memory into another system (known as *NAND Mirroring*). Code that has been secured can often be broken with Scanning Electron Microscopes. Skorobogatov from Cambridge University has written a comprehensive study [252] of many semi-invasive attacks that can be done on hardware. Another common attack is called a *side-channel attack* [293, 165] where the power usage or other indirect information from the device can be used to steal information. This means that it is very difficult to protect secrets on a device from a committed attacker.

A specific outcome of this is that designers should not rely on obscurity to protect devices. A clear example of this was the Mifare card used as the London Oyster card and for many other authentication and smart-card applications. The designers created their own cryptographic approach and encryption algorithms. Security researchers used a number of techniques to break the obscurity, decode the algorithm, find flaws

[1] `https://www.lora-alliance.org/`

in it and create a hack that allowed free transport in London as well as breaking the security on a number of military installations and nuclear power plants [107]. Similarly, relying on the security of a device to protect a key that is used across many devices is a significant error. For example, the encryption keys used in DVD players and XBoX gaming consoles [257] were broken meaning that all devices were susceptible to attack.

A related issue to confidentiality of the data on the device is the challenges inherent in updating devices and pushing keys out to devices. The use of Public Key Infrastructure (PKI) requires devices to be updated as certificates expire. The complexity of performing updates on IoT devices is harder, especially in smaller devices where there is no user interface. For example, some devices need to be connected to a laptop in order to perform updates. Others need to be taken to a dealership or vendor. The distribution and maintenance of certificates and public-keys onto embedded devices is complex [282]. In [197] a novel approach to supporting mutual authentication in IoT networks is proposed. However, this model assumes that each device has a secure, shared key (called *kIR* already deployed and managed into every device. As discussed above, ensuring this key is not compromised is a challenge, as the authors admit: "However, further research is required to realize the secure sharing of *kIR*."

In addition, sensor networks may be connected intermittently to the network resulting in limited or no access to the Certificate Authority (CA). To address this, the use of threshold cryptographic systems that do not depend on a single central CA has been proposed [295], but this technology is not widely adopted: in any given environment this would require many heterogeneous Things to support the same threshold cryptographic approach. This requires human intervention and validation, and in many cases this is another area where security falls down. For example, many situations exist where security flaws have been fixed but because devices are in homes, or remote locations, or seen as appliances rather than computing devices, updates are not installed [125]. The Misfortune Cookie [208] demonstrates that even when security fixes are available, some manufacturers do not make them available to customers and continue to ship insecure systems. It is clear from the number of publicised attacks [169, 142, 125] that many device designers have not adjusted to the challenges of designing devices that will be connected either directly or indirectly to the Internet.

A further security challenge for confidentiality and hardware is the fingerprinting of sensors or data from sensors. In [43] it has been shown that microphones, accelerometers and other sensors within devices have unique "fingerprints" that can uniquely identify devices. Effectively there are small random differences in the physical devices that appear during manufacturing that can be identified and used to recognise individual devices across multiple interactions.

2.3 B1: Network Confidentiality

The confidentiality of data on the network is usually protected by encryption of the data. There are a number of challenges with using encryption in small devices. Performing public key encryption on 8-bit microcontrollers has been enhanced by the use of Elliptic Curve Cryptography (ECC) [145, 174]. ECC reduces the time and power requirements for the same level of encryption as an equivalent Rivest, Shamir, and Adleman public key cryptography (RSA) public-key encryption [223] by an order of magnitude [116, 242, 243]: RSA encryption on constrained 8-bit microcontrollers may take minutes to complete, whereas similar ECC-based cryptography completes in seconds. However, despite the fact that ECC enables 8-bit microcontrollers to participate in public-key encryption systems, in many cases it is not used. It is possible to speculate as to why this is: firstly, as evidenced by [243], the encryption algorithms consume a large proportion of the available ROM on small controllers. Secondly, there is a lack of standard open source software. For example, a search that I carried out (on the 21st April 2015) of the popular open source site Github for the words "Arduino" and "Encryption" revealed 10 repositories compared to "Arduino" and "HTTP" which revealed 467 repositories. These 10 repositories were not limited to network level encryption. However, recently an open source library for AES on Arduino [151] has made the it more effective to use cryptography on Atmel-based hardware.[2]

While ECC is making it possible for low-power devices to be more efficient in performing cryptography operations, in 2015 the NSA made an unprecedented warning against ECC [3]. It is not known why, as of the time of writing. There are differing theories. One known issue with both Prime Numbers and Elliptic Curves is Quantum Computing. In Quantum computers, instead of each bit being 0 or 1, each *qubit* allows a superposition of both 0 and 1, allowing Quantum computers to solve problems that are very slow for classical computers in a fraction of the time. At the moment general purpose Quantum computers are very simple and confined to laboratories, but they are increasing in power and reliability. In 1994, Peter Shor identified an algorithm for Quantum Computers [245] that performs prime factorisation in polynomial time, which effectively means that most existing Public Key Cryptography (PKC) will be broken once sufficiently powerful Quantum computers come online. Given that most Quantum Computers are as yet ineffective, there is some concern that maybe the problem with ECC is actually based on classical computing, but this is all speculation. One thing that that is clear is that ECC is much easier to do on IoT devices, and especially on low-power, 8- or 16-bit systems. Therefore this warning is worrying for

[2]The same search was repeated on the 10th Feb 2017. The number of repositories for "Arduino" and "Encryption" had grown to 21, while for "Arduino" and "HTTP" had reached 941, demonstrating that support for encryption is growing slowly.

[3]`https://threatpost.com/nsas-divorce-from-ecc-causing-crypto-hand-wringing/115150/`

IoT developers.

Another key challenge in confidentiality is the complexity of the most commonly used encryption protocols. The standard Transport Layer Security (TLS) [74] protocol can be configured to use ECC, but even in this case the handshake process requires a number of message flows and is sub-optimal for small devices as documented in [148]. [201] has argued that using TLS with Pre Shared Key (PSK) improves the handshake. PSK effectively allows TLS to use traditional symmetric cryptography instead of Public Key (assymetric) cryptography. However, they fail to discuss in any detail the significant challenges with using PSK with IoT devices: the fact that either individual symmetric keys need to be deployed onto each device during the device manufacturing process, or the same key re-used. In this case there is a serious security risk that a single device will be broken and thus the key will be available.

Some IoT devices use User Datagram Protocol (UDP) instead of the more commonly used Transport Control Protocol (TCP). Both protocols are supported on the Internet. UDP is unreliable, and is typically better suited to local communications on trusted networks. It is more commonly used between IoT devices and gateways rather than directly over the Internet, although, like all generalisations there are exceptions to this rule. TLS only works with TCP, and there is an alternative protocol for UDP. Datagram Transport Layer Security (DTLS) [218] provides a mapping of TLS to UDP networks, by adding retransmission and sequencing which are assumed by TLS. While the combination of DTLS and UDP is lighter-weight than TLS and TCP, there is still a reasonably large RAM and ROM size required for this [141], and this requires that messages be sent over UDP which has significant issues with firewalls and home routers, making it a less effective protocol for IoT applications [22]. There is ongoing work at the IETF to produce an effective profile of both TLS and DTLS for the IoT [270].

A significant area of challenge for network confidentiality in IoT is the emergence of new radio protocols for networking. Previously there were equivalent challenges with Wifi networks as protocols such as Wired Equivalency Privacy (WEP) were broken [50], and there are new attacks on protocols such as Bluetooth Low Energy (BLE) (also known as Bluetooth 4.0). For example, while BLE utilises Advanced Encryption Standard (AES) encryption which has a known security profile, a new key exchange protocol was created, which turns out to be flawed, allowing any attacker present during key exchange to intercept all future communications [228]. One significant challenge for IoT is the length of time it takes for vulnerabilities to be addressed when hardware assets are involved. While the BLE key exchange issues are addressed in the latest revision of BLE, it is expected it to take a very long time for the devices that encode the flawed version in hardware to be replaced, due to the very large number of devices and the lack of updates for many devices. By analogy, many years after the WEP issues were uncovered, in 2011 a study showed that 25% of wifi networks were

still at risk [44].

Even without concerning the confidentiality of the data, there is one further confidentiality issue around IoT devices in the network and that is confidentiality of the metadata. Many IoT systems rely on radio transmission and in many cases they can be fingerprinted or identified by the radio signature. For example, Bluetooth and Wifi systems use unique identifiers called MAC address (Media Access Control). These can be identified by scanning, and there have been a number of systems deployed to do that, including in airports and in cities [281]. These systems effectively can follow users geographically around. If the user then connects to a system, that fingerprint can be associated with the user and the previously collected location information can be correlated with that user. In a similar attack, security researchers recently found [237] that they could fingerprint cars based on transmissions from tyre pressure monitors, and in addition that they could drive behind a car and from up to 40 feet away they could signal to the driver that the tyre pressure was dangerously low when in fact it wasn't. Such an attack could easily be used to get a driver to stop and leave their car.

In [211] a theoretical model of traceability of IoT devices and particularly RFID systems is proposed in order to prevent unauthorised data being accessible. A protocol that preserves the concept of untraceability is proposed.

Many of the same references and issues apply to section *B2* where the use of digital signatures with low power devices is examined.

2.4 C1. Cloud Confidentiality

In the main, the issues around Cloud confidentiality are the same as the issues in non-IoT systems. There are however, some key concerns over privacy that are unique to the Internet of Things. For example, the company Fitbit [90] made data about users sexual activity available and easily searchable online [297] by default. There are social and policy issues regarding the ownership of data created by IoT devices [216, 182]. These issues are addressed in more detail in the cell *C5* which looks at the access control of IoT data and systems in the cloud and on the server-side.

A second concern that is exacerbated by the Internet of Things are concerns with correlation of data and metadata, especially around *de-anonymisation*. In [185] it was shown that anonymous metadata could be de-anonymized by correlating it with other publicly available social metadata. This is a significant concern with IoT data. This is also closely related to the fingerprinting of sensors within devices as discussed in cell *A1*. An important model for addressing these issues in the cloud are systems that filter, summarise and use *stream-processing* technologies to the data coming from IoT devices before this data is more widely published. For example, if a system only

publishes a summarised co-ordinate rather than the raw accelerometer data then it can potentially avoid fingerprinting de-anonymisation attacks.

In addition, an important concern has been raised in the recent past with the details of the government sponsored attacks from the US National Security Agency (NSA) and UK Government Communications Headquarters (GCHQ) that have been revealed by Edward Snowden [52]. These bring up three specific concerns on IoT privacy and confidentiality.

The first concern is the revelations that many of the encryption and security systems have had deliberate backdoor attacks added to them so as to make them less secure [153]. The second concern is the revelation that many providers of cloud hosting systems have been forced to hand over encryption keys to the security services [159]. The third major concern is the revelations on the extent to which metadata is utilised by the security services to build up a detailed picture of individual users [29].

The implications of these three concerns when considered in the light of the Internet of Things is clear: a significantly deeper and larger amount of data and metadata will be available to security services and to other attackers who can utilize the same weaknesses that the security services compromise.

2.5 A2: Integrity & Hardware/Device

The concept of integrity refers to maintaining the accuracy and consistency of data. In this cell of the matrix, the challenges are in maintaining the device's code and stored data so that it can be trusted over the lifecycle of that device. In particular the integrity of the code is vital if one is to trust the data that comes from the device or the data that is sent to the device. The challenges here are viruses, firmware attacks and specific manipulation of hardware. For example, [115] describes a worm attack on router and IoT firmware, where each compromised system then compromises further systems, leaving behind a slew of untrustworthy systems.

The traditional solution to such problems is attestation [229, 46, 241]. Attestation is important in two ways. Firstly, attestation can be used by a remote system to ensure that the firmware is unmodified and therefore the data coming from the device is accurate. Secondly, attestation is used in conjunction with hardware-based secure storage (Hardware Security Managers, as described in [70]) to ensure that authentication keys are not misused. The model is as follows.

In order to preserve the security of authentication keys in a machine where human interaction is involved, the user is required to authenticate. Often the keys are themselves encrypted using the human's password or a derivative of the identification pa-

rameters. However, in an unattended system, there is no human interaction. Therefore the authentication keys need to be protected in some other way. Encryption on its own is no help, because the encryption key is then needed and this becomes a circular problem. The solution to this is to store the authentication key in a dedicated hardware storage. However, if the firmware of the device is modified, then the modified firmware can read the authentication key, and offer it to a hacker or misuse it directly. The solution to this is for an attestation process to validate the firmware is unmodified before allowing the keys to be used. Then the keys must also be encrypted before sending them over any network.

These attestation models are promoted by groups such the Trusted Computing Group [262], and Samsung Knox [233]. These rely on specialized hardware chips such as the Atmel AT97SC3204 [20] which implement the concept of a Trusted Platform Module (TPM) [177]. There is research into running these for Smart Grid devices [200]. However, whilst there is considerable discussion of using these techniques with IoT, during this literature review no evidence was found of any real-world devices apart from those based on mobile-phone platforms (e.g. phones and tablets) that implemented trusted computing and attestation.

2.6 B2: Network Integrity

Maintaining integrity over a network is managed as part of the public-key encryption models by the use of digital signatures. The challenges for IoT are exactly those that are already identified in the cell *B1* above where the challenges of using encryption from low-power IoT devices are described.

However, there is a further concern with IoT known as the Sybil Attack [76]. A Sybil attack[4] is where a peer-to-peer network is taken over when an attacker creates a sufficiently large number of fake identities to persuade the real systems of false data. A Sybil attack may be carried out by introducing new IoT devices into a locality or by suborning existing devices. For example, it is expected that autonomous cars may need to form local ephemeral peer-to-peer networks based on the geography of the road system. A significant threat could be provided if a Sybil attack provided those cars with incorrect data about traffic flows.

[4]Named after a character in a book who exhibits multiple personality disorder

2.7 C2: Cloud Integrity

The biggest concern in this area is the lack of common concepts and approaches for device identity. Integrity relies on identity - without knowing who or what created data, one cannot trust that data. This is addressed in cells *A4, B4* and *C4*. One specific aspect of trust in cloud for IoT scenarios is where the device lacks the power to participate in trust and must therefore trust the cloud server. One key example of this is where a *blockchain* [184] is being used in respect of IoT devices. Blockchains are cryptographically secure ledgers that typically require a significant amount of memory, disk space and processor power to work [41]. These requirements go beyond typical IoT devices and even beyond more powerful systems in IoT networks such as hubs. One option to address this is to use remote attestation, but as yet there is little or no work in this space.

2.8 A3: Hardware Availability

One of the significant models used by attackers is to challenge the availability of a system, usually through a Denial of Service (DoS) or Distributed Denial of Service (DDoS) attack. DoS attacks and availability attacks are used in several ways by attackers. Firstly, there may be some pure malicious or destructive urge (e.g. revenge, commercial harm, share price manipulation) in bringing down a system. Secondly, availability attacks are often used as a pre-cursor to an authentication or spoofing attack.

IoT devices have some different attack vectors for availability attacks. These include resource consumption attacks (overloading restricted devices), physical attacks on devices. A simple availability attack on an IoT device might be to force it to use more power (e.g. by initiating multiple key exchanges over Bluetooth) and thereby draining the battery. Another even more obvious availability challenge would be to simply physically destroy a device if it is left in a public or unprotected area.

2.9 B3. Network Availability

There are clearly many aspects of this that are the same as existing network challenges. However, there are some issues that particularly affect IoT. In particular, there are a number of attacks on local radio networks that are possible. Many IoT devices use radio networking (Bluetooth, Wifi, 3G, General Packet Radio Service (GPRS), LoRa and others) and these can be susceptible to radio jamming. In [180] there is a survey of jamming attacks and countermeasures in Wireless Sensor Network (WSN). Another

clear area of attack is simply physical access. For example, even wired networks are much more susceptible to physical attacks when the devices are spread widely over large areas.

2.10 C3: Cloud Availability

The challenges here are not new: DoS attacks and DDoS attacks have been already discussed. The biggest challenge here is the use of IoT devices themselves to create the DDoS attack on the server, as in the Mirai botnet.

2.11 A4: Device Authentication

Authentication of the device to the rest of the world is considered in cells B5 and C5. In this cell of the matrix one must consider the challenges of how users or other devices can securely authenticate to the device itself. These are however related: a user may bypass or fake the authentication to the device and thereby cause the device to incorrectly identify itself over the network to other parts of the Internet.

Some attacks are very simple: many devices come with default passwords which are never changed by owners. In a well-publicised example [125], a security researcher gained access to full controls of a number of “smart homes”. As discussed above, the Mirai attack took control of devices that used default or easily guessed passwords.

Similarly many home routers are at risk through insecure authentication [14]. Such vulnerabilities can then spread to other devices on the same network as attackers take control of the local area network.

A key issue here is the initial *registration* of the device. A major issue with hardware is when the same credential, key, or password is stored on many devices. Devices are susceptible to hardware attacks (as discussed above) and the result is that the loss of a single device may compromise many or all devices. In order to prevent this, devices must either be pre-programmed with unique identifiers and credentials at manufacturing time, or must go through a registration process at setup time. In both cases this adds complexity and expense, and may compromise usability. The use of the OAuth2 Dynamic Client Registration process is presented in Chapter 5 (as published in [101]) to create unique keys/credentials for each device. A significant part of the work described in Chapter 6 is a well-defined and secure process for device and user registration that allows users to take control of devices in scenarios where the device itself offers no User Interface (UI) or a very basic UI.

2.12 B4: Network Authentication

Unlike browsers or laptops where a human has the opportunity to provide authentication information such as a userid and password, IoT devices normally run unattended and need to be able to power-cycle and reboot without human interaction. This means that any identifier for the device needs to be stored in the program memory (usually SRAM), ROM or storage of the device. This brings two distinct challenges:

- The device may validly authenticate, but the program code may have been changed, and therefore it may behave incorrectly.
- Another device may steal the authentication identifier and may *spoof* the device.

In the Sybil attack [188] a single node or nodes may impersonate a large number of different nodes thereby taking over a whole network of sensors. In all cases, attestation is a key defence against these attacks.

Another defence is the use of *reputation* and reputational models to associate a trust value to devices on the network. Reputation is a general concept widely used in all aspects of knowledge ranging from humanities, arts and social sciences to digital sciences. In computing systems, reputation is considered as a *measure* of how trustworthy a system is. There are two approaches to trust in computer networks: the first involves a "black and white" approach based on security certificates, policies, etc. For example, SPINS [205], develops a trusted network. The second approach is probabilistic in nature, where trust is based on reputation, which is defined as a probability that an agent is trustworthy. In fact, reputation is often seen as one measure by which trust or distrust can be built based on good or bad past experiences and observations (direct trust) [137] or based on collected referral information (indirect trust) [1].

In recent years, the concept of reputation has shown itself to be useful in many areas of research in computer science, particularly in the context of distributed and collaborative systems, where interesting issues of trust and security manifest themselves. Therefore, one encounters several definitions, models and systems of reputation in distributed computing research (e.g. [105, 137, 249]).

There is considerable work into reputation and trust for wireless sensor networks, much of which is directly relevant to IoT trust and reputation. The Hermes and E-Hermes [302, 301] systems utilise Bayesian statistical methods to calculate reputation based on how effectively nodes in a mesh network propogate messages including the reputation messages. Similarly, [57] evaluates reputation based on the packet-forwarding trustworthiness of nodes, in this case using fuzzy logic to provide the evaluation framework. Another similar work is [173] which again looks at the packet forwarding reputation of nodes. In IoT, [27] utilizes the concept of a *Utility Function* to create a reputational model for IoT systems using the MQTT protocol.

2.13 C4: Cloud Authentication

The IETF has published a draft guidance on security considerations for IoT [193]. This draft does discuss both the bootstrapping of identity and the issues of privacy-aware identification. One key aspect is that of bootstrapping a secure conversation between the IoT device and other systems, which includes the challenge of setting-up an encrypted and/or authenticated channel such as those using TLS, Host Identity Protocol (HIP) or Diet HIP. HIP [179] is a protocol designed to provide a cryptographically secured endpoint to replace the use of IP addresses, which solves a significant problem – IP-address spoofing – in the Internet. Diet HIP [178] is a lighter-weight rendition of the same model designed specifically for IoT and Machine to Machine (M2M) interactions. While HIP and Diet HIP solve difficult problems, they have significant disadvantages to adoption. Secure device identity models that work at higher levels in the network stack, such as token-based approaches, can sit side by side with existing IP-based protocols and require no changes at lower levels of the stack. By contrast, HIP and Diet HIP require low-level changes within the IP stack to implement. As they replace traditional IP addressing they require many systems to change before a new device using HIP can successfully work. In addition, neither HIP nor Diet HIP address the issues of federated authorisation and delegation.

In [96, 103], which is also covered in Chapter 5, I proposed using *federated* identity protocols such as OAuth2 [117] with IoT devices, especially around the MQTT protocol [32]. The IOT-OAS [59] work similarly addresses the use of OAuth2 with CoAP. Other related works include the work of Augusto et al. [23] have built a secure mobile digital wallet by using OAuth together with the XMPP protocol [230]. In [101], (also covered in Chapter 5) I extended the usage of OAuth2 for IoT devices to include the use of Dynamic Client Registration [232] which allows each device to have its own unique identity, which was discussed as an important point in Cell A1.

A contradictory aspect of IoT Authentication is the proposal to use secure *Pseudonyms*. A pseudonym is also sometimes referred to as an *Anonymous Identity*. Effectively, a secure pseudonym is a way of a user securely interacting with a system without giving away their real identity. This overlaps with Cell C5 where access control for cloud systems was examined. It has been seen from well-publicised cases that systems may be compromised and offer personal information, even years after that information was originally stored [294]. In one case, two suicides have been attributed to an attack that compromised personal identities [35]. Pseudonyms are an approach that can be considered to treat the sharing of meta-data as important as sharing of data. Also see Section 2.20 where another model of privacy is presented.

In [226] a capability-based access system is described that allows anonymous identities to be used. [38] provides an Architecture Reference Model for an approach that

supports anonymous identities. Neither of these systems separate the provision of anonymous identities from the data-sharing middleware. A concept called Zooko's Triangle [194] proposed that it is only possible to support two out of the following three capabilities in a system: human-readable names; decentralised infrastructure; and security. Recent papers, such as [11], claim that the blockchain construct proves Zooko's hypothesis wrong. In [120] the concept of anonymous identities for blockchains is explored, which will have significant impact as blockchains become more prevalent in IoT. I have published a proposal for extending the current work to suppport blockchains in IoT in [100], and this is further discussed in Section 11.4.

2.14 A5: Device Access Control

There are two challenges to access control at the device level. Firstly, devices are often physically distributed and so an attacker is likely to be able to gain physical access to the device. The challenges here (hardware attacks, NAND mirroring, etc) were already discussed in Cell A1.

However, there is a further challenge: access control almost always requires a concept of identity. One cannot restrict or allow access except in the most basic ways without some form of authentication to the device. As discussed in the review of Cell A4, this is a significant challenge. To give a real life example, certain mobile phones have recently started encrypting data based on the user's own lock-screen Personal Identification Number (PIN) code [236]. This guarantees the data cannot be read without the user's PIN code. However, using NAND Mirroring, it has been demonstrated that the controls that stop repeated attempts at PIN codes can be overcome [251], with the result that a 4 digit PIN can easily be broken within a reasonable amount of time.

Systems such as Webinos [72] have proposed using policy-based access control mechanisms such as XML Access Control Markup Language (XACML) [113] for IoT devices. However, XACML is relatively heavyweight and expensive to implement [273], especially in the context of low power devices. To address this, Webinos has developed an engine which can calculate the subset of the policy that is relevant to a particular device. Despite this innovation, the storage, transmission and processing costs of XACML are still high for an IoT device. Another approach based around a standard called UMA is covered in Cell C5.

2.15 B5: Network Access Control

There are a number of researchers looking at how to create new lightweight protocols for access control in IoT scenarios. [168] describe a new protocol for IoT authentication and access control is proposed based on ECC with a lightweight handshake mechanism to provide an effective approach for IoT, especially in mobility cases. [124] propose a non-centralised approach for access control that uses ECC once again and supports capability tokens in the CoAP protocol.

2.16 C5: Cloud Access Control

The biggest challenge for privacy is ensuring access control at the server or cloud environment of data collected from the IoT. There is some significant overlap with the area of confidentiality of data in the cloud as well (Cell *C1*).

I argued strongly in [103] that existing hierarchical models of access control are not appropriate for the scale and scope of the IoT. There are two main approaches to address this. The first is *policy-based* security models where roles and groups are replaces by more generic policies that capture real-world requirements such as "A doctor may view a patient's record if they are treating that patient in the emergency room". The second approach to support the scale of IoT is user-directed security controls, otherwise known as consent. This is the approach taken in this thesis. In [272] a strong case is made for ensuring that users can control access to their own resources and to the data produced by the IoT that relates to those users. In [272], an approach for using UMA together with OAuth2 is proposed for constrained devices. This approach also addresses Cell A5. While this approach has a lot of capabilities and power, there is a slow uptake of UMA in real-world services and even less in IoT. The complexity of this approach is a significant factor in the lack of widespread adoption.

[284] argues that contextual approaches must be taken to ensure privacy with the IoT. Many modern security systems use context and reputation to establish trust and to prevent data leaks. Context-based security [176] defines this approach which is now implemented by major Web systems including Google and Facebook. This is closely related to reputation-based models which was discussed above.

2.17 A6: Device Non-Repudiation

The biggest challenge in the non-repudiation network with IoT devices is the challenge of using *attestation* for small devices. Attestation is discussed in detail in Cell *A2*.

Without attestation, one cannot trust that the device system has not been modified and therefore it is not possible to trust any non-repudiation data from the device.

2.18 B6: Network Non-Repudiation

The same challenges apply here as discussed in cells *B2, B3*. Non-repudiation on the wire requires cryptography techniques and these are often hindered by resource restrictions on small devices. In [198] a non-repudiation protocol for restricted devices is proposed.

2.19 C6: Cloud Non-Repudiation

This area is unchanged by the IoT, so it is not discussed any further.

In the previous eighteen sections a significant number of threats and challenges have been identified, and this matrix has been used to assess the most relevant current work in each space. Before summarising this work, an orthogonal model is presented that more closely focusses on user privacy.

2.20 Privacy

One area that crosses most or all of the cells in the matrix is the need for a holistic and studied approach to enabling privacy in the IoT. As discussed in a number of cells, there are significant challenges to privacy with the increased data and metadata that are being made available by IoT-connected devices. An approach that has been proposed to address this is *Privacy by Design* [54]. This approach suggests that systems should be designed from the ground up with the concept of privacy built into the heart of each system. Many systems have added security or privacy controls as “add-ons”, with the result that unforeseen attacks can occur.

Privacy can be inherently contradictory to security. For example, non-repudiation implies that a user can be proved to have done something. A contrary privacy requirement is *plausible deniability*. For example, a user may wish to have non-repudiation for their e-commerce activities, but may desire plausible deniability for their secret cosplay hobby.

2.20.1 Privacy Properties

There are a set of core privacy properties identified in [71]. This model is named *LINDDUN* after the threats identified against these properties. The privacy properties and corresponding threats identified in the paper are:

- **Unlinkability** — refers to hiding the link between two or more actions, identities or pieces of information. The corresponding threat is **linkability**.
- **Anonymity and Pseudonymity** — Anonymity is unlinkability between identity and other properties. **Pseudonymity** is the idea that you can use a anonymous identity. In addition, many pseudonymous systems allow you to use different identities for different purposes, thereby allowing unlinkability between different pseudonyms. The corresponding threat is **Identifiability**.
- **Plausible Deniability** — is the ability to deny that an action took place. The threat is **non-repudiation**.
- **Undetectability and unobservability** — is the ability to act without the action being known. For example, to be able to be in a particular place without being observed. The threat is **Detectability**.
- **Confidentiality** — is as defined above: hiding data or controlling the release of data. The threat is **Disclosure**.
- **Content Awareness** — is the ability for users to understand the data that is being shared by that user. For example, the fact that users were unaware that Fitbit was collecting information about their sexual activity is an example. The threat is **content Unawareness**.
- **Policy and Consent compliance** — refers to the ability of the system to inform users of privacy policies, offer consent-based systems, and behave according to the policies and consents. The threat is **policy and consent Non-compliance**.

In the paper they identify that the latter two properties capture *soft* privacy properties, while the other properties form *hard* properties. This distinction comes from [65]. Broadly speaking, soft privacy is based on the assumption that data is given away and there is the use of policies, regulations and consent to manage it. Hard privacy is the aim of directly controlling data before it is given away. The paper makes this interesting comment:

> Hard privacy technologies are active in research but poor in deployment, due to cost and technical evolvement restrictions (such as cryptography). Soft privacy technologies are state-of-art and have fewer research activities.

Spiekermann and Cranor [255] offer a model for looking at user privacy. In their model, they identify three spheres: the *User Sphere*, the *Joint Sphere* and the *Recipient Sphere*. The User Sphere is completely in the control of the user (e.g. a laptop). The Joint Sphere refers to areas that may seem to be in the user's control, but may have some significant control by a third-party. For example, a cloud email account may seem like the user can delete emails, but the cloud provider may in fact back these up and keep a copy. Finally, once data has been transferred to a third-party, it is assumed to be in the Recipient Sphere, where the only controls are legal and contractual.

In the model, a device that offers the user full control is firmly in the *User Sphere*. However, it can be argued that many current devices are actually in the *Joint Sphere*: this is where the device appears to be in the control of the user but in fact is in the control of a third-party. To give an example, the Google Nest device offers users the opportunity to apply smart heating controls to their house. While a number of user-centred controls give the user the impression that it is in the User Sphere, there are two key reasons to counter this: firstly, the data logged by the device is extensive and cannot be controlled by the user; secondly, the device auto-updates itself based on commands from Google rather than based on user input [187]. Applying the concepts of hard and soft privacy, it can be seen that hard privacy can apply in the User Sphere, soft privacy in the Recipient Sphere and some combination may apply in the Joint Sphere, although this will mainly be soft privacy.

Using this model, a set of clear approaches that strengthen each of the privacy and security controls available in each sphere are proposed. Figure 2.1 provides an overview of this model, and its applicability to the IoT domain.

2.20.2 User Sphere

Moving privacy and security controls back to the users inherently strengthens the User Sphere and provides greater choice, thereby allowing more secure approaches to flourish.

As discussed above, devices need to have secure identities, and currently these are either not provided, or provided by the device manufacturer.

A second, related issue, is the ownership of devices. The Mirai botnet spread because dictionary attacks allowed attackers to take ownership of devices. Some systems offer models of taking ownership securely (e.g. Bluetooth, Near Field Communication (NFC)). In Chapter 6.2, a model where a QR code is used in conjunction with a Web-based system is proposed.

A third issue within the User Sphere is updating the device firmware. A number of attacks have originated in lack of updates. One issue is that device manufacturers

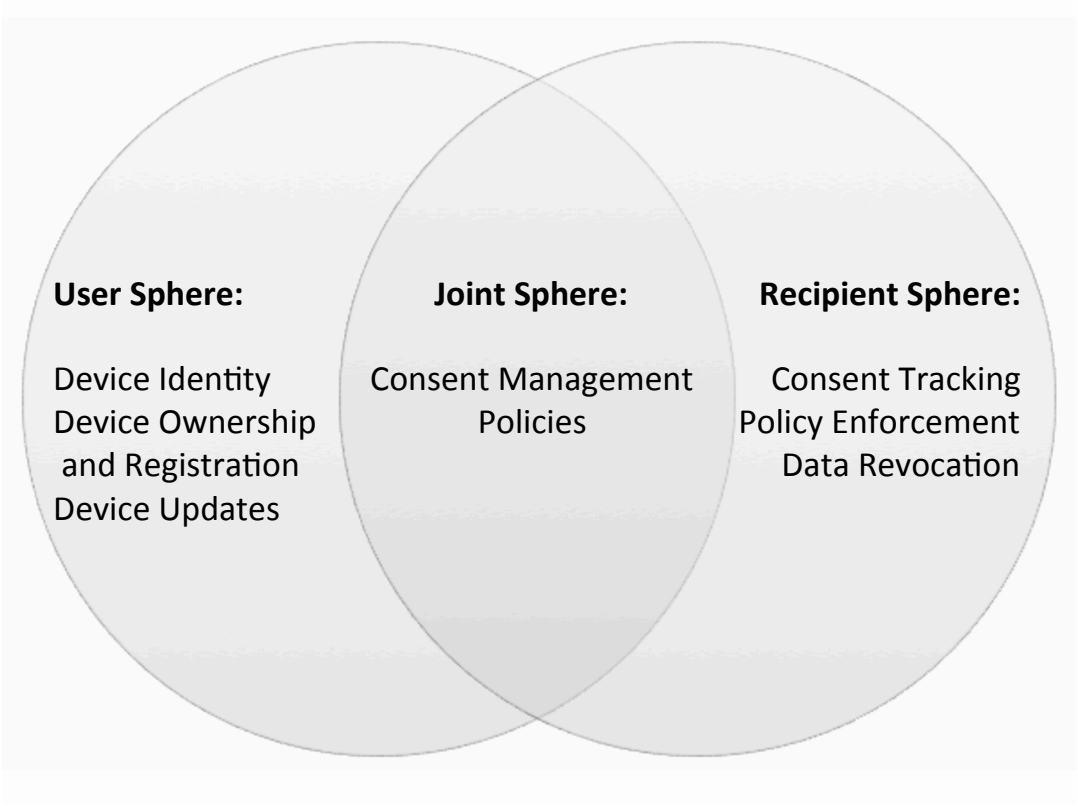

Figure 2.1: Three Layer Privacy Model Applied to IoT

are incentivised to create new products but not to update old products. In [268], a model is proposed whereby devices can pay for updates using a blockchain-based cryptocurrency such as Bitcoin. The model presented in Chapter 6 offers an approach whereby devices can be updated based on secure identity and consent approaches.

2.20.3 Joint Sphere

Recall that the Joint Sphere is the parts of the system where the user has some form of control over their data and systems, but the provider also shares control. For example, a health-monitoring device may upload data to an Internet-based system and then may offer users controls on how they share data. A major change in legislation around this is the European Union's General Data Protection Regulation (GDPR) [276] which requires much stronger consent controls. Many systems offer forms of user consent for sharing data with third parties, but these lack significant requirements. For example, many users are not aware of how to revoke consent. Similarly, there is no clear place a user can identify all the consents they have approved across different devices. Consent is not just about privacy. IoT devices often include actuators that can act based on *Commands*, and the security of a device includes ensuring that only authorised systems can issue commands to devices.

A related area is that of policies. In this meaning a policy is a computer-readable expression of rights and obligations. For example, a consent approval may refer to a policy: the user might approve sharing of data to a website based on the fact that the website promises not to share the data to any other body. Languages such as XACML [113] allow complex access control policies to be encoded in XML or JavaScript Object Notation (JSON). This is discussed in Cell C5.

2.20.4 Recipient Sphere

The Recipient Sphere is the area where the user's data is now out of their control. Ultimately, the user must rely on legislation or legal contracts in order to maintain control of this data. Of course, it is hard to police this recipient sphere: it is possible that the third-party website will share data following policies. In addition, many organisations have such complex and poorly worded policies that users are unaware of the rights they are giving up to their data. Spotting illicit data shares can possibly be done using a concept of a *Trap Street*. This is the habit that map-makers have of including incorrect data to see if others copy it. Similarly, IoT devices could deliberately share incorrect data to specific parties to see if it leaks out against the agreed policy.

2.21 Summary Of The Review Of Security Issues

In this chapter a widened ontology has been proposed for evaluating the security issues surrounding the Internet of Things. The existing literature and research in each of the cells of the expanded matrix has been examined. These issues have been incorporated into Spiekermann and Cranor's Three Layer Privacy Model. This is an important basis for the next section where the provisions around security and privacy that are available in available middleware for the Internet of Things are examined.

In reviewing these areas, a list of security properties and capabilities that are important for the security and privacy of IoT were collected. This list will be used in Chapter 3 to evaluate a set of middleware systems on their provision of these capabilities.

- **REQ1 - Integrity and Confidentiality** The requirement to provide integrity and confidentiality is an important aspect in any network and as discussed in cells *A1-B2* there are a number of challenges in this space for IoT.
- **REQ2 - Access Control** Maintaining access control to data that is personal or can be used to extract personal data is a key aspect of privacy. In addition, it is of prime importance that actuators are not allowed unauthorised access to control aspects of the world.

  - **REQ2.1 - Consent** As described in cells A5-C5, traditional hierarchical models of access control are ineffective for personal data and IoT systems. Consent approaches – such as OAuth2 and UMA – are a key requirement.
  - **REQ2.2 - Policy-Based Access Control** As discussed in cells A5-C5, policy-based access control models such as XACML enable privacy considerations and rules to be implemented effectively in IoT scenarios, although in many cases models such as XACML are too heavyweight to deploy into devices.

- **REQ3 - Authentication** As discussed in numerous of the cells, IoT systems need a concept of authentication in order to enable integrity, confidentiality, and access control amongst other requirements.
  - **REQ3.1 - Federated Identity** As argued in cells *A4-C4*, there is a clear motivation for the use of federated models of identity for authentication in IoT networks.
  - **REQ3.2 - Secure Device Identity** Managing the security of devices requires unique credentials to be embedded into each device and secure registration processes as discussed in cell A4.
  - **REQ3.3 Anonymous Identities** In order to guard against de-anonymisation and other leakages of personally identifiable information, anonymous identities/pseudonyms can offer individuals clearer consent as to when they wish to actively share their identity, as discussed in A4.

- **REQ4 - Attestation** Attestation is an important technique to prevent tampering with physical devices (as discussed in cells in column A) and hence issues with integrity of data as well as confidentiality in IoT.
- **REQ5 - Summarisation and Filtering** The need to prevent de-anonymisation is a clear driver for systems to provide summarisation and filtering technologies such as stream processing.
- **REQ6 - Context-Based Security and Reputation** Many modern security models adapt the security based on a number of factors, including location, time of day, previous history of systems, and other aspects known as context. Another related model is that of the reputation of systems, whereby systems that have unusual or less-than-ideal behaviour can be trusted less using probabilistic models. In both cases there are clear application to IoT privacy as discussed above.

While Privacy By Design (PBD) is considered an important aspect, it can be argued that it is a *meta-requirement*: it effectively covers the need to implement the major security and privacy requirements from the initial design outwards.

There are of course many other aspects to IoT security and privacy as have been demonstrated in the matrix table and accompanying description of each cell. However, these specific aspects form an effective set of criteria by which to analyse different systems, as shown below in the next Chapter.

Chapter 3

Secure Middleware for the Internet of Things

3.1 Introduction

Middleware has been defined as computer software that has an intermediary function between the various applications of a computer and its operating system [119]. Middleware that is specifically designed or adapted to provide capabilities for IoT networks is explored here.

There are a number of existing surveys of IoT middleware. Bandyopadhyay et al. [30, 31] review a number of middleware systems designed for IoT systems. While they look at security in passing, there is no detailed analysis of the security of each middleware system. [56] calls out the need for security, but no analysis of the approaches or existing capabilities is provided. [21] is a very broad survey paper that addresses IoT middleware loosely. [215] is another wide-ranging survey of IoT middleware that provides a simple analysis of whether the surveyed systems have any support for security or privacy, but does not address detailed requirements.

It is clear then, that a detailed evaluation of security in IoT middleware is a useful contribution to the literature. Therefore a set of middleware systems to study were identified using a structured methodology.

3.2 Review Methodology

This set was identified through a combination of the existing literature reviews on IoT middleware [30, 56, 215] together with a new search for middleware systems that explicitly target IoT scenarios. Some of the systems that were included in these papers

were excluded from the list on the basis that they were not middleware. For example, [56] lists TinyREST [166] as a middleware, but in fact this paper defines a standard protocol rather than a middleware and was therefore excluded.

The search strategy was to use a search for the terms (”IoT” OR ”Internet of Things”) AND ”Middleware”. Only the subject terms were searched and the search was restricted to academic papers written in English. The search was carried out by the Portsmouth University Discovery system which is a metasearch engine. The list of databases that are searched is available at [277]. The search was originally issued on June 6th, 2015, identifying 152 papers. It was repeated on December 1st, 2016, and 213 papers were identified, showing a significant growth in IoT middleware papers over the intervening period.

The abstracts of the 213 papers were then reviewed to identify a list of functioning middleware systems as opposed to papers that describe other aspects of IoT without describing a middleware system. This produced a list of 55 middleware systems.

In this study, the security and privacy requirements listed in section 2.21 were looked for. It was also identified if the middleware had a clearly defined security model and/or security implementation. Out of the 55 middleware systems identified, it was found that 34 had no published discussion or architecture for security, or such a minimal description that it was not possible to identify any support for the selected security requirements. These are labelled as *non-secured* systems.

3.3 Non-Secured Systems

A brief description of each of the non-secure middleware systems follows:

ASIP The Arduino Service Interface Programming model (ASIP) [36] is a middleware for Arduino hardware.

ASPIRE ASPIRE Project (Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications) [209] is a EU-funded project that created an open, royalty-free middleware for RFID-based applications.

Autonomic QoS Management [33] offers a middleware that autonomically manages Quality of Service (QoS) in IoT scenarios. While this does address some aspects related to security (i.e. accuracy and availability), there is no discussion of how security is handled.

CASCOM In [202] a semantically-driven configuration model is built on top of existing middleware systems such as GSN [2]. The authors state their intention of addressing privacy in future work.

CIRUS CIRUS [207] is a cloud-based middleware for ubiquitous analytics of IoT data.

Cloud-based Car Parking Middleware In [135] the authors describe an OSGi-based middleware for smart cities enabling IoT-based car parking.

Context Aware Gateway [13] provides a reference architecture for using context-awareness in IoT scenarios. The middleware itself does not address security or privacy and the authors plan to address this in further work.

DAMP In [8] there is a middleware – Distributed Applications Management Platform – that can configure systems based on Quality of Service characteristics (QoS). These characteristics can include security, but the system itself does not offer any security model.

Dioptase Dioptase [39] is a RESTful stream-processing middleware for IoT. Dioptase does address one useful aspect for privacy: intermediate stream processing of data, summarisation and filtering. However, there is no detailed security architecture and description and the security model is left as an item of future work.

EDBO [83] describes the Emergent Distributed Bio-Organization: a biologically-inspired platform for self-organising IoT systems.

EDSOA An Event-driven Service-oriented Architecture for the Internet of Things Service Execution [150] describes an approach that utilizes an event-driven SOA.

EMMA The Environmental Monitoring and Management Agent (EMMA) is a proposed middleware based on CoAP [78]. It does not offer any security architecture.

GSN The GSN framework [2] (Global Sensor Networks) defines a middleware for the Internet of Things that requires little or no programming. The security architecture of the system is not described in any detail: there are diagrams of the container architecture which point to proposed places for access control and integrity checks, but unfortunately there is not sufficient discussion to be able to categorize or evaluate the approach taken.

Hi-Speed USB middleware [24] offers a middleware based on USB.

Hitch Hiker Hitch Hiker 2.0 [214] is a prototype middleware environment built on Contiki OS.

LMTS In [172] a middleware system for asset tracking (Laptop Management and Tracking System) is described.

Middleware for Environmental Monitoring and Control [292] defines a middleware for environmental monitoring and control.

Middleware for Industrial IoT [275] describes a middleware for Industrial IoT based on OpenDDS, which is a middleware that implements the Data Distributions Services (DDS) protocol. At the time of writing the DDS security model was in development and hence the architecture does not address security.

MIFIM Middleware for Future Internet Models (MIFIM) [28] is a Web Service-based architecture that uses Aspect-Orientation to allow for simpler reconfiguration.

MOSDEN MOSDEN (Mobile Sensor Data Processing Engine) [203] is an extension of the GSN approach (see above) which is explicitly targeted at *opportunistic* sensing from restricted devices.

M-Hub [261] describes a middleware for Mobile IoT applications built on top of another middleware (Scalable Data Distribution Layer). In [114] this work is enhanced to create a middleware for Ambient Assisted Living. In [278] there is another middleware based on M-Hub. There is no support for security or privacy described.

PalCom Palcom [258] is a middleware designed for pervasive computing, including IoT systems. It supports ad-hoc composition of services. There is no discussion of security beyond a statement that traditional security models may be added in future.

POBICOS Platform for Opportunistic Behaviour in Incompletely Specified, Heterogeneous Object Communities (POBICOS) [274] is a device middleware designed to run on small devices. In [260] there is a description of migrating aspects of the middleware to a proxy to enable support for smaller devices.

PROtEUS PROtEUS is a process manager designed to support Cyber-Physical Systems [239]. It describes a middleware for complex self-healing processes.

RemoteUi [53] offers a middleware for Remote user interfaces.

SBIOTCM In *A SOA Based IOT Communication Middleware* [299] is a middleware based on SOAP and WS. There is no security model described.

Service Oriented access for Wireless Sensor Networks [104] provides a service-oriented middleware for IoT and Wireless Sensor Network data.

Smart Object Middleware [123] describes a Smart Object middleware based on Java.

symbIoTe In [254] a roadmap is laid out for a new EU funded project to allow vertical IoT platforms to interoperate and federate. There is no plan for security presented.

Thingsonomy Thingsonomy [121] is an event-based publish-subscribe based approach that applies semantic technology and semantic matching to the events published

within the system.

UBIROAD The UBIROAD middleware [263] is a specialisation of the UBIWARE project specifically targeting traffic, road management, transport management and related use-cases.

UBISOAP ubiSOAP [51] is a Service-Oriented Architecture (SOA) approach that builds a middleware for Ubiquitous Computing and IoT based on the Web Services (WS) standards and the SOAP protocol.

VEoT [10] describes a Virtual Environment of Things which is a middleware for Virtual Reality engagement with the Internet of Things.

WHEREX WhereX [110] is an event-based middleware for the IoT.

3.4 Secured Systems

20 middleware systems were identified that implement or describe sufficient security architecture that they could be evaluated against the requirements that were identified in section 2.21. These systems are described as *secured*. In addition to the requirements identified above in Section 2.21, it is also identified whether the systems had explicit support or adaptation for IoT specific protocols: MQTT, CoAP, DDS, Bluetooth or Zigbee. As discussed above, in Section 2.1, these protocols have been specifically designed for low-power devices. This requirement is labeled as **REQ7**. Table 3.1 shows the summary of this analysis.

For each of the secured middleware system the core published papers were reviewed, but in addition any further available documentation was explored. Below are the specific details of each middleware system.

3.4.1 &Cube

In [296] they describe a middleware, &Cube, that is designed to offer RESTful APIs as well as MQTT connections to integrate with IoT devices. The system offers a security manager providing encryption, authentication and access control. No further details are available on the techniques used.

3.4.2 Device Cloud

In [217] there is a blueprint for a middleware that applies Cloud Computing concepts to IoT device middleware. A more detailed exposition is given in [144]. The approach

	REQ1 -Integrity and Confidentiality	REQ2 -Access Control	REQ2.1 -Consent	REQ2.2 -Policy-based security	REQ3 - Authentication	REQ3.1 -Federated Identity	REQ3.2 -Secure Device Identity	REQ3.3 -Anonymous Identities	REQ4 -Attestation	REQ5 - Summarisation and Filtering	REQ6 -Context-based security/Reputation	REQ7 -IoT-specific Protocol Support
&Cube	Y	Y			Y							Y
Device Cloud	Y	Y	Y		Y	Y						Y
DREMS	Y	Y			Y							Y
DropLock		Y	Y		Y	Y						Y
FIWARE	Y	Y	Y	Y	Y	Y						Y
Hydra/Linksmart	Y	Y			Y		Y					
Income	Y	Y		Y	Y						Y	
IoT-MP	Y				Y							
NAPS	Y	Y			Y							
NERD	Y				Y							Y
NOS	Y	Y			Y						Y	Y
OpenIoT					Y	Y						
SensorAct		Y		Y								
SIRENA	Y				Y							
SMEPP	Y	Y			Y							
SOCRADES	Y	Y			Y							
UBIWARE				Y								
WEBINOS	Y	Y		Y	Y	Y	Y					
XMPP	Y	Y			Y	Y						
VIRTUS	Y	Y			Y	Y						

Table 3.1: Summary of Reviewed Middleware Systems and Major Properties

supports OAuth2.0 to provide tokens to devices. It also supports encryption and access control. There is no support for summarisation, filtering, or consent-based access control described in the publications.

3.4.3 DREMS

Distributed RealTime Managed Systems (DREMS) [158] is a combination of software tooling and a middleware runtime for IoT. It includes Linux Operating System extensions as well. DREMS is based on an actor [7] model has a well-defined security model that extends to the operating system. The security model includes the concept of multi-level security (MLS) for communications between a device and the actor. The MLS model is based on *labelled* communications. This ensures that data can only flow to systems that have a higher *clearance* than the data being transmitted. This is a very powerful security model for government and military use-cases. However, this approach does not address needs-based access control. For example, someone with *Top Secret* clearance may read data that is categorised as *Secret* even if they have no business reason to utilise that data. The weaknesses of this model have been shown with situations such as the Snowden revelations.

3.4.4 DropLock

In [154] the authors describe a middleware specifically built for IoT systems and Smart Cities. The DropLock system is designed to enable secure smartphone access to a smart locker, allowing delivery personnel secure access to drop off packages. The system uses secure tokens to allow access to devices. The tokens are passed to the secure locker using Bluetooth.

3.4.5 FIWARE

FIWARE [112] is a middleware designed to be the basis of a Future Internet, sponsored by the European Union under the FP7 programme. FIWARE is one of the few systems that claim to have used PBD as a basis for design [279]. FIWARE has a concept of plugins, known as Generic Enablers (GE). The security model is implemented through GEs including the Identity Management (IdM GE), the Authorisation Policy Decision Point (PDP) GE, and the Policy Enforcement Point (PEP) Proxy. The standard approach within FIWARE is based on OAuth2 and XACML. It also supports interoperable standards for exchanging identities with other systems. The overall security design of FIWARE fits into modern authentication and authorisation models. However, there is no significant adaption of this architecture to the specifics of IoT systems in the core descriptions of the security architecture. IoT devices are catered for in the FIWARE Architecture through a gateway model. The IoT devices connect to the gateway using IoT specific protocols. The gateway is part of the *IoT Edge*. This communicates via the standard FIWARE protocols into an *IoT Backend* where there are components supporting Device Management and Discovery. The FIWARE documentation does not describe any specific adaptation of security or support for security between devices and the gateway.

3.4.6 Hydra / Linksmart

Hydra [82] was a European Union funded project which has since been extended and renamed as LinkSmart [149]. The Hydra team published a detailed theoretical model of a policy-based security approach [4]. This model is based on using lattices to define the flow of information through a system. This model provides a language-based approach to security modelling. However, whilst this paper is published as part of the Hydra funded project, there is no clear implementation of this in the context of IoT or description of how this work can benefit the IoT world. Hoever, because Hydra / Linksmart is an Open Source project with documentation beyond the scientific papers, it is possible to understand the security model in greater detail by review of this project.

The Hydra and LinkSmart architectures are both based on the Web Services (WS) specifications, building on the SOAP protocol [253], which in turn builds on the XML Language [290]. The security model is described in some detail in the LinkSmart documentation [161]. The model utilises XML Security [77]. There are significant challenges in using this model in the IoT world, as discussed above in section 2.1. The Hydra/Linksmart approach also uses symmetric keys for security which is a challenge for IoT because each key must be uniquely created, distributed and updated upon expiry into each device creating a major key management issue.

Hydra / Linksmart offers a service called the TrustManager. This is a system that uses the cryptographic capabilities to support a trusted identity for IoT devices. This works with a Public Key Infrastructure (PKI) and certificates to ensure trust. Once again there are challenges in the distribution and management of the certificates to the devices which are not addressed in this middleware.

The Hydra middleware does not offer any policy based access control for IoT data, and does not address the secure storage of data for users, nor offer any user-controlled models of access control to user's data.

In [199] there is a specific instantiation of LinkSmart applied to energy efficiency in buildings. There is no further extension to the security model.

3.4.7 INCOME

INCOME [15] is a framework for multi-scale context management for the IoT, funded by the French National Research Agency. The aim of INCOME is to fuse together context data from multiple levels to provide a high-level set of context data from IoT systems that can be applied to decision making, including trust, privacy and security decisions. MuDebs and MuContext [160] are frameworks built on top of INCOME that add Attribute Based Access Control (ABAC) and Quality of Context (QoC) processing. MuDebs utilises XACML policies to implement ABAC. MuContext validates QoC and enables privacy filtering.

3.4.8 IoT-MP

The IoT Management Platform (IoT-MP) is a middleware system described in [84]. IoT-MP offers a security module that implements attribute-based access control (ABAC) against systems. IoT-MP has a model whereby an *Agent* is registered for each class of *Things*, creating the concept of a *Managed Thing*. Agents have unique secure identities. The IoT-MP does not define how devices are identified to agents.

3.4.9 NAPS

The *Naming, Addressing and Profile Server* (NAPS) [162] describes a heterogeneous middleware for IoT based on unifying data streams from multiple IoT approaches. Based on RESTful APIs, the NAPS approach includes a key component handling Authentication, Authorisation, and Accounting (AAA). The design is based on the Network Security Capability model defined in the ETSI M2M architecture [86]. However, the main details of the security architecture have not yet been implemented and have been left for future work. There is no consideration of federated identity or policy based access control.

3.4.10 NERD

No Effort Rapid Development (NERD) [63] is a middleware designed for human IoT interfaces, especially around Bluetooth LE systems and iBeacon discovery. It does not add any new security measures but uses the existing security models in Bluetooth and HTTP.

3.4.11 NOS

NetwOrked Smart objects (NOS) [247, 248] takes an interesting approach to security where the aim is to provide each item of data with a reputational score based on a quality analyser and a security analyser. A machine learning algorithm is used to learn the behaviour of systems in the network and adjust the scores based on the potential attacks and the applied countermeasures. The system incorporates keys and key-based authentication, encryption and complex passwords.

3.4.12 OpenIoT

OpenIoT is an open cloud-based middleware for the Internet of Things, funded by the European Union FP7 programme. It also extends the GSN framework. The *Security Module* uses OAuth2 as the main authentication and authorisation model for web-based systems. No details are given of how sensors are authenticated or authorised.

3.4.13 SensorAct

SensorAct [17] is an IoT middleware specifically aimed at providing support for Building Management Systems (BMS). It supports fine-grained access control through the

use of a rules engine to implement access control policies. No details are provided of the authentication models at the device level or the web interface.

3.4.14 SIRENA

SIRENA (Service Infrastructure for Real-time Embedded Networked Devices) [42] is a SOAP/WS-based middleware for IoT and embedded devices. While there is little description of the security framework in SIRENA, it does show the use of the WS-Security specification. As previously discussed, this approach is very heavyweight, has issues with key distribution, federated identity and access control.

3.4.15 SMEPP

Secure Middleware for P2P (SMEPP) [37] is an IoT middleware explicitly designed to be secure, especially dealing with challenges in the peer-to-peer model. SMEPP security is based around the concept of a group. When a peer attempts to join a group, the system relies on challenge-response security to implement mutual authentication. At this point the newly joined peer is issues a shared session key which is shared by all members of the group. SMEPP utilizes elliptic key cryptography to reduce the burden of the security encryption onto smaller devices. Overall SMEPP has addressed security effectively for peer-to-peer groups, but assumes a wider PKI infrastructure for managing the key model used within each group. In addition, there is no discussion of access control or federated identity models, which are important for IoT scenarios. The model is that any member of the group can read data published to the group using the shared session key.

3.4.16 SOCRADES

SOCRADES [69] is a middleware specifically designed for manufacturing shop floors and other industrial environments. Based on SOAP and the WS stack it utilizes the security models of the WS stack, in particular the WS-Security standard for encryption and message integrity. There is no special support for federation, tokens or policy-based access control (instead relying on role-based access control). The resulting XML approach is very heavyweight for IoT devices and costly in terms of network and power [79]. In addition, the lack of explicit support for tokens and federated security and identity models creates a significant challenge in key distributions and centralized identity for this approach.

3.4.17 UBIWARE

The UBIWARE project is a smart semantic middleware for Ubiquitous Computing [238]. The security model for UBIWARE is not clearly described in the original paper, but an additional paper describes a model called Smart Ubiquitous Resource Privacy and Security (SURPAS) [186], which provides a security model for UBIWARE. UBIWARE is designed to utilize the semantic Web constructs, and SURPAS utilises the same model of semantic Web as the basis for the abstract and concrete security architectures that it proposes. The model is highly driven by policies and these can be stored and managed by external parties. In particular the SURPAS architecture is highly dynamic, allowing devices to take on board new roles or functions at runtime. While the SURPAS model describes a theoretical solution to the approach, there are few details on the concrete instantiation. For example, while the model defines a policy-based approach to access control, there are no clearly defined policy languages chosen. There is no clear model of identity or federation, and there is no clear guidance on how to ensure that federated policies that are stored on external servers are protected and maintain integrity. The model does not address any edge computing approaches or filtering/summarisation of IoT data. However, the overall approach of using ontologies and basing policies on those ontologies is very powerful.

3.4.18 WEBINOS

The Webinos [72] system has a well-thought through security architecture. The documentation explicitly discussed PBD. The Webinos system is based around the core concept of devices being in the personal control of users and therefore having each user having a “personal zone” to protect. This is a more advanced concept but in the same vein as the protected sub-domains in VIRTUS. In the Webinos model, each user has a cloud instance - known as the Personal Zone Hub (PZH) that supports their devices. The Personal Zone Hub acts as a service to collect and offer access to data and capabilities of the user’s devices. The PZH acts as a certificate authority, issuing certificates to the devices that are used for mutual authentication using TLS. User’s authenticate to the PZH using the OpenID protocol. On the device, a communications module known as the Personal Zone Proxy (PZP) handles all communications with the PZH.

The idea of the Personal Zone may have significant issues however, when a single device is used by many different people (for example, the in-car system in a taxi as opposed to a personal vehicle). These issues are not addressed in Webinos, though they are called out in the lessons learnt.

Webinos utilizes policy-based access control modelled in the XACML [113] language.

The system pushes XACML policies out to devices to limit the spread of personal and contextual data.

Webinos addresses the issue of software modification using an attestation API, which can report whether the software running is the correct level. This requires the device to be utilising Trusted Platform Module (TPM) hardware that can return attestation data.

Webinos also addresses the issue of using secure storage on devices where the device has such storage.

While the Webinos project does address many of the privacy concerns of users through the use of the Personal Zone Hub, there is clearly further work that could be done. In particular the ability for users to define what data they share with other users or other systems using a protocol such as OAuth2 [117], and the ability to install filters or other anonymising or data reduction aggregators into the PZH are lacking. One other aspect of Webinos that is worth drawing attention to is the reliance on a certain size of device: the PZP that is needed on the device is based on the *node.js* framework and therefore the device needs to be of a certain size (e.g. a 32-bit processor running a Linux derivative or similar) to participate in Webinos.

3.4.19 VIRTUS

The VIRTUS middleware [60] utilizes the core security features of the XMPP protocol to ensure security. This includes tunnelling communications over TLS, authentication via SASL, and access control via XMPP's built-in mechanisms. SASL is a flexible mechanism for authentication which supports a number of different systems including token-based approaches such as OAuth2 or Kerberos, username/password, or X.509 certificates. For client-to-server based communications, it is not clear from the description which of these methods is actually implemented within VIRTUS. For server-to-server communications there is specified the use of SASL to ensure full server federation.

While the VIRTUS model does not describe the challenges of implementing a personal instance of middleware for single users or devices, there is a concept of edge computing described, where some interactions may happen within an edge domain (e.g. within a house) and lower security is required within that domain while higher security is expected when sharing that data outside. This model is fairly briefly described but provides an interesting approach. One challenge is that there are multiple assumptions to this: firstly, that security within the limited domain needs less security, when there may be attackers within that perimeter. Secondly, that the open channel to the wider Internet cannot be misused to attack the edge network. The ability to

calculate, summarise and/or filter data from the edge network before sharing it is also not discussed except in very granular terms (e.g. some data are available, other data are not).

3.4.20 XMPP

The paper [131] describes how the XMPP architecture can be applied to the challenges of M2M and hence the IoT, together with a proof-of-concept approach. The system relies on the set of XMPP extensions around publish/subscribe and the related XMPP security models to implement security. This includes TLS for encryption, and access control models around publish-subscribe. There is also a discussion about leakage of information such as *presence* from devices. The proof-of-concept model did not include any federated identity models, but did utilize a One-Time Password (OTP) model on top of XMPP to address the concepts such as temporary loans of devices.

3.5 Summary Of IoT Middleware Security

In reviewing both the security and privacy challenges of the wider IoT and a structured review of more than fifty middleware platforms, key categories that can be applied across these areas have been identified.

The first set is the majority of the identified systems that did not address security, left it for further work, or did not describe the security approach in any meaningful detail were identified. There were other systems (such as UBIWARE and NAPS) that offer theoretical models but did not demonstrate any real-world implementation or concrete approach.

The next clear category are those middlewares that apply the SOAP/Web Services model of security. This includes SOCRADES, SIRENA, and Hydra/Linksmart. As has been discussed in the previous sections there are significant challenges in performance, memory footprint, processor power and usability of these approaches when used with the IoT.

Two of the approaches delegate the model to the XMPP standards: VIRTUS and XMPP [60, 131]. XMPP also has the complexity of XML, but avoids the major performance overheads by using TLS instead of XML Encryption and XML Security. In addition, recent work on XMPP using EXI makes this approach more effective for IoT.

This finally leaves a few unique approaches, each of which brings their own unique benefits.

DREMS is the only system to provide Multi-level security based on the concept of security clearances. While this model is attractive to government and military circles (because of the classification systems used in those circles), it can be argued that it fails in many regards for IoT. In particular there are no personal controls, no concept of federated identity and no policy based access controls in this model.

SMEPP offers a model based on public key infrastructures and shared session keys. It can be argued that this approach has a number of challenges scaling to the requirements of the IoT. Firstly, there are significant issues in key distribution and key revocation. Secondly, this model creates a new form of perimeter - based on the concept of a shared session key. That means that if one device is compromised then the data and control of all the devices in that group are also compromised.

Only Dioptase supports the concept of stream processing in the cloud, which is a serious requirement for the IoT. The requirement is to be able to filter, summarise and process streams of data from devices to support anonymisation and reduction of data leakage.

FIWARE has a powerful and extensible model for authentication and access control, including support for federated identity and policy-based access control.

Finally, the most advanced approach identified is that proposed by Webinos. Webinos utilizes some key technologies to provide a security and privacy model. Firstly, this uses policy-based access control (XACML). The model does not however support user-guided access control mechanisms such as OAuth2 or UMA.

Webinos does support the use of Federated Identity tokens (OpenID), but only from users to the cloud, as opposed to devices to the cloud. I and others have proposed the model of using federated identity tokens from the device to the cloud in [103, 101, 59].

The contribution of the Webinos work with the largest potential impact is the concept of Personal Zone Hub, which is a cloud service dedicated to a single user to handle the security and privacy requirements of that user. There is, however, further research around this area: the PZH model from Webinos does not examine many of the challenges of how to implement the PZH in real life. For example, user registration, cloud hosting, and many other aspects need to be defined in more detail before the Webinos PZH model is practicable for real world projects. In addition there are challenges using the PZH model with smaller devices, because of the requirement to use the PZP.

3.5.1 Overall Gaps In The Security Of Middleware

When the requirements for security and privacy of the Internet of Things are examined there are some gaps that are not provided by any of the reviewed middleware systems.

- Only two of the middleware systems explicitly applied the concept of PBD in designing a middleware directly to support privacy, although Webinos did exhibit many of the characteristics of a system that used this approach.
- Only two of the systems applied any concepts of context-based security or reputation to IoT devices.
- User consent was only supported in three of the systems.
- None of the systems supported anonymous identities or attestation.
- None of the systems satisfied all the requirements identified.

3.6 Discussion

In this literature review a two-phase approach to reviewing the available literature around the security and privacy of IoT devices has been taken.

In the first part a matrix of security challenges was created that applied the existing CIA+ model to three distinct areas: device, network and cloud. This new model forms a clear contribution to the literature. In each of the cells of the matrix were identified threats, challenges and/or approaches, exacerbated by the IoT, or unchanged. Further, Spiekerman and Cranor's three-layer privacy model was used to analyse the privacy requirements of IoT. This overall analysis was used to identify seven major requirements and five subsidiary requirements.

In the second part, a structured search approach was used to identity 55 specific IoT middleware frameworks and then the security models of each of those was analysed. The twelve requirements from the first phase were used to validate the capabilities of each system. While there are existing surveys of IoT middleware, none of them focussed on a detailed analysis of the security of the surveyed systems and therefore this has a clear contribution to the literature.

In this survey, clear gaps in the literature and practice have been identified. Over half the surveyed systems had either no security or no substantive discussion of security. Out of 55 surveyed systems very few address a significant proportion of the major challenges that are identified in the first section. Some aspects are not addressed by any of the surveyed systems.

This creates an opportunity for research that is the basis of the rest of this thesis:

- First, to define a model and architecture for IoT middleware that is designed from the start to enable privacy and security (Privacy by Design).
- Secondly, to bring together the best practice into a single middleware that includes: federated identity (for users and devices), policy-based access control, user managed access to data, summarisation and filtering, and other requirements.
- Thirdly, there is considerable work to be done to define a better model around the implementation challenges for the concept of a personal cloud service (e.g the Webinos PZH). This includes the hosting model, bootstrapping, discovery and usage for smaller devices.

Chapter 4

Core Technologies

The model and architecture proposed in this thesis is heavily based on the OAuth2 specification. In addition, the prototype is built on the MQTT protocol. In this chapter there is a short introduction to each of these as well as looking at existing work in each case.

4.1 OAuth2

The OAuth 1.0 Protocol [80], and its successor, the OAuth 2.0 Framework [81], are protocols designed to solve the privacy and access control issues related to large scale Internet-connected applications. The original OAuth protocol emerged in response to requirements to allow users to enable third-party websites to post tweets on their behalf using APIs [118], and OAuth2 was developed at the IETF as a replacement. OAuth and OAuth2 allows users to delegate access to specified function to third-party websites. It also allows users to share identification across websites without sharing their credentials across those websites.

A typical use-case for OAuth2 is as follows. Social networking websites often used to ask users for access to their email contact lists in order to bootstrap or extend the user's social network. In order to implement this, the social network would as the user for the username and password to their email provider (such as Hotmail, Google Mail, or Yahoo). Then, using an undocumented Application Programming Interface (API) or via HTML screen-scraping, the social network website would login as the user and access the contact list. This approach had significant concerns:

- There was no fine-grained access control: once the social network had access to the username and password they could do anything – for example, posting emails or spam – and not just the task in hand (accessing the contact list).

- There was no time limitation: unless the user then changed the password then the social network could store this data and re-use it later. It is assumed that most users do not change their passwords after this step.
- This breaks the normal terms and conditions of websites that do not allow sharing of credentials.

The OAuth 1.0 Protocol and OAuth 2.0 Authorisation[1] Framework are designed to solve these problems. OAuth2 has been available for a number of years now and is widely used, including by major websites like Facebook, Github, Twitter, Google and Yahoo.

In OAuth2, there are typically four roles:

- **Resource Owner** — this is the person who owns the resource that is going to be accessed. In this work the alternative *User* will be used for the same concept.
- **Authorisation Server (AS)** — this server issues the tokens and handles user consent flows. This component is sometimes co-located with the Resource Server.
- **Resource Server** — the server that controls access to a resource owned by the User.
- **Client** — the client is the system that wishes to access the user's resource. In many cases the client is itself an application running in a web server. The client is uniquely identified by a *ClientID*. The most common authentication uses this in conjunction with a *ClientSecret*.

In the social networking example: the Resource Owner is the user; the Resource Server hosts the contact list; the Authorisation Server is the identity system managing the users credentials for the contact list; and the Client is the social networking site.

Before anything happens, the two sites (the Client and the Authorisation Server) must communicate, and the Client is issued with a Client ID and Client Secret. These are used by the Client to authenticate to the Authorisation Server.

The user is browsing the social network and the opportunity to expand their network is offered. The social networking site (Client), makes a request to the Resource Owner to access their contact list. In most cases, this is actually done indirectly via the Authorisation Server - what will happen is that the user's browser will have an HTTP redirect to the Authorisation Server. The Authorisation Server asks the user to login (with the same credentials that they use for their email site). An important point is that they only share their credentials with the site that owns them. In many cases, if

[1] For consistency across this document, the UK English "authorisation" is used. However, the original spelling by the IETF uses the US English "authorization"

they are already logged into that email site in another window or with a cookie, this step will be automatic.

Once they are logged into their email site, the Authorisation Server asks if they would like to permit the Client (the social network) access to their contact list. There are two important aspects to this: firstly, there is a *scope* associated with this, and secondly there is a time limit. For example, in this case, the scope would be a URI that indicates access to just the contact list. The user might be asked ”Do you want to authorise this just once or always”, which would allow them to specify how long the authorisation should last for.

At this point the Client is issued an Authorisation Grant or code. The Client then uses their Secret together with the Authorisation code to request a token. This token identifies the application making the request (the Client or social networking site) together with the user. The token can then be used to access the Resource Server (the API that allows access to the contact list).

The important features of this model are:

- The user only shares their email credentials with the email site (or its associated Authorisation Server).
- The Client only has access to a specific scope – in this case the contact list.
- The token identifies both the Client and the User: this is not a general token allowing any system access to the contact list, but a very specific authorisation of a specific application or site to access a particular scope.
- The token has a time limit, and the authorisation itself has a time limit as well. In the case where the token expires, but the authorisation still continues, the client may refresh the token using a further process.
- As a further point, the authorisation grant is *dynamic* – the user authorises access and within seconds their social network is enlarged.
- The model is completely *federated* – the Client may interact with multiple Authorisation Servers, and vice-versa, and each identity is trusted as needed by other sites within a trust framework.

This particular flow describes one of several options that the OAuth specifications allow, and these different flows are known as Authorisation Grant Types:

- **Authorisation Code** — in this flow, the AS acts as an intermediary between the client and the user. The client redirects the user to the AS, who then issues an Authorisation Code (AuthCode) to the client (once the user has authorised or consented to the access). The client must then present its credentials together with the AuthCode to the AS and is then issued with a token. This flow is the

only flow that is used throughout this work. It is shown in a sequence diagram in Figure 4.1. This is sometimes known as a “three-legged” flow.

- **Implicit** — this flow is designed for clients such as client-side JavaScript clients. In this flow, the client does not authenticate using ClientID and ClientSecret.
- **Resource Owner Password Credentials** — This is a model where the client is trusted enough that the user shares their password directly with the client.
- **Client Credentials** — This is a model sometimes known as “two-legged”. In this model, the client is typically the resource owner, and therefore the client’s credentials are valid to issue a token.

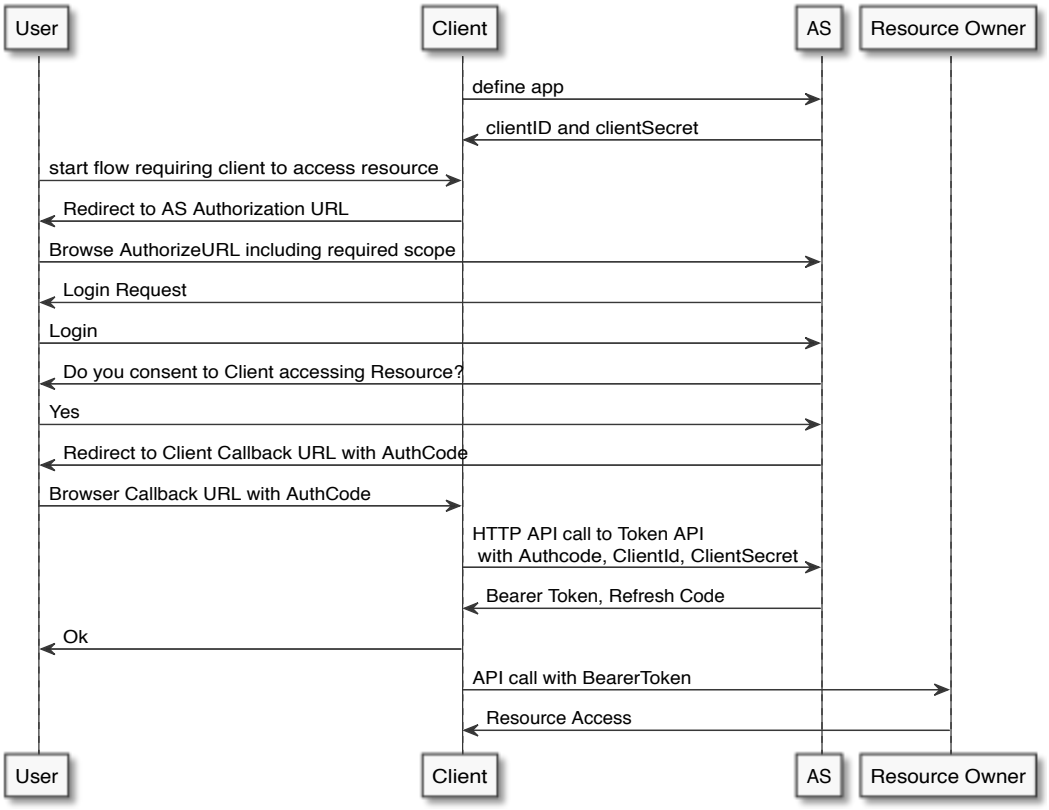

Figure 4.1: Authorisation Code flow

4.1.1 Related Standards

The OAuth2 framework has a number of related standards and proposals for standards in the Internet Engineering Task Force (IETF) and elsewhere. In Table 4.1, the main related protocols are shown that will be used or discussed throughout this work.

The User Managed Access (UMA) from the Kantara Initiative enhances the OAuth specification to provide a rich environment for users to select their own data sharing

Name	Description	Citation
OAuth 2.0 Authorisation Framework	This is the main OAuth2 protocol as described above	[81]
OAuth2 Introspection	This protocol standardises the connection between the Resource Server and the Authorisation Server, allowing the Resource Server to interrogate the AS for information about a token	[219]
Dynamic Client Registration	This protocol standardises an API for registering a new client with an AS.	[222]
User Managed Access (UMA)	UMA is a protocol that allows users to tune the permissions that the client is granted, instead of simply being able to deny or authorise a given scope	[140]
Proof of Posession (PoP)	This standard allows a client to prove that is holds a key thereby removing security issues around replay of bearer tokens.	[129]

Table 4.1: OAuth2 and Related Protocols

preferences [140]. It can be argued strongly that this overall concept of user-directed access control to IoT data is one of the most important approaches to ensuring privacy. However, despite the availability of this standard, very few systems implement it compared to the less complex approach defined in OAuth2. In [272], an approach for using UMA together with OAuth2 is proposed for constrained devices.

IOT-OAS [59] addresses the use of OAuth2 with the CoAP protocol. In [192] there is a demonstration of the OAuth1 protocol with MQTT, favouring OAuth1 over OAuth2 for IoT devices. The reasons for choosing the older OAuth protocol are obviated by the mapping of the refresh flow which OAuthing offers. In [212] and in [85] there are platforms that support OAuth2 for IoT devices that communicate via HTTP and WebSockets. None of these works address automated registration processes, and none provide the privacy controls of anonymous identifiers or isolated personal cloud instances.

OAuth2 has some analysis of security. OAuth2 has had a formal analysis in ProVerif [34], which identified specific threats to OAuth2. Another work is Pai et al [196] which utilized the Alloy Framework [133] to analyse the security constraints of the OAuth protocol. In [164] there is a threat model for OAuth2., many of which are applicable to this work as well. While these threats show that OAuth2 security has its concerns, overall the widespread use of OAuth in many of the world's busiest web systems including Google, Twitter and Facebook shows that it can be effectively se-

cured. In [103], I addressed the use of OAuth2 with MQTT, and this is covered in significant detail below in Section 5.2. In IOT-OAS [59], Cirani et al. address the use of OAuth2 with CoAP, an alternative IoT protocol based on RESTful principles [244]. The mapping of the OAuth2 Token API to support IoT devices using the CoAP protocol is being formalised in Authentication and Authorisation for Constrained Environments (ACE) [130], and is described in [271]. In addition, there is early work mapping ACE to MQTT [240], which explicitly references the work presented in Chapter 5.

I chose to experiment with OAuth 2.0 as the basis for IoT federation because:

- OAuth is a widely implemented standard and is well understood.
- OAuth is used in many cases to control access to personal information.
- OAuth permits users the ability to direct the access control of their own information.
- OAuth is specifically designed for machine-to-machine and Web API based scenarios where humans are involved in the initial setup, but beyond that all interactions are based on tokens.

4.2 MQTT

The MQTT protocol was originally devised by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom Systems [181] as a protocol for telemetry over constrained networks. Other messaging systems, such as IBM MQ Series [109], assumed high speed networks with low-costs per byte transmission. In many M2M, IoT and Telemetry examples, the networks are constrained (including GPRS and satellite communications) with high costs per byte. Therefore the MQTT specification was designed to have a very low message overhead, with as little as two bytes extra per message.

MQTT has been donated to the OASIS standards organisation, and standardized by the OASIS MQTT Technical Committee. The initial aim of the technical committee was to clarify any semantics but to preserve wire-level compatibility. The committee has now moved on to updating the standard with new wire-level features, and this will be published as *MQTT 5.0* when this work is complete. MQTT is used in many IoT scenarios, and there are libraries for microcontrollers-based systems such as Arduino [16] that make it easy to utilize.

MQTT is based around a *publish and subscribe* model [87], often known as *pub/sub*. In this model, there are one or more central servers, known as *brokers*, that clients connect to. A client may publish information to the broker, subscribe to receive infor-

mation, or both. The publishers are unaware of the subscribers, and vice-versa. Each publication and subscription is tied to a *topic*. The topic is a named virtual resource that is used to decouple publishers and subscribers.

As well as decoupling publishers and subscribers from knowing about each other, the pub/sub model also decouples them in time, because all interactions are asynchronous.

Most pub/sub systems utilize a tree-based model for topics, where there is a hierarchy and wildcard-based matching, which allows subscribers to subscribe to all topics in a branch of the tree, and MQTT also follows this model. For example, a subscriber can subscribe to the topic string "devices/uk/#" which would match topics including "devices/uk/hampshire/emsworth" as well as "devices/uk/sussex". The "#" identifies a wildcard that matches any number of levels within the hierarchy. The "+" character identifies matching only a single level.

Publishers cannot publish to a wildcard - they must publish to a fully-qualified topic name.

MQTT has a very simple security model. For authentication, it currently only allows the use of a username and/or password. Some implementations also support the use of a Pre-Shared Key (PSK) which can be used for authentication as well as encryption. This requires the client and server to have an agreed private key, which imposes a large burden on setup and configuration, especially with small, cheap IoT clients. For encryption and transport-level security, the Transport Layer Security (TLS) standard is recommended, although this is not always suitable for small devices. The specification does not describe or recommend any authorisation models, but certain implementations support access control lists on specific topics.

Perez [204] has modelled and analysed the performance of MQTT, and Lee et at [156] have analyzed message loss in MQTT networks and the correlation to the level of Quality of Service (QOS) requested. Mengusoglu and Pickering [170] have created an Autonomic Management system utilizing MQTT as the messaging protocol, whereas Stanford-Clark and Wightwick [256] have demonstrated the applicability of MQTT for environmental monitoring and control systems. Recently, a formal analysis of MQTT's quality of service semantics also has been undertaken [25], which demonstrated some ambiguities related to those semantics.

There has been little research into the security of MQTT. The OASIS Technical Committee has offered a document analysing MQTT security and offering some best practices [47].

Facebook Messenger has said that it uses MQTT on mobile devices [48] and Facebook is also a user of OAuth [89], but there is no published information on whether they

utilize the OAuth tokens for MQTT authentication and authorisation.

MQTT was chosen as the basis of this work for the following reasons:

- MQTT is a standard protocol used by a variety of IoT and M2M systems.
- There are several open source implementations, making it easy to work with.
- The central broker model creates an easy place to implement authorisation and access control measures.
- The topic model is highly flexible and offers scope for authorisation control.

Part II

Part II - Preliminary Investigations

Chapter 5

Investigations into Federated Identity and Access Management for IoT

5.1 Summary

In this chapter, two preliminary investigations into the use of Federated Identity and Access Management (FIAM) for IoT are presented. The first investigation, Federation of IoT (FIOT), was initially published in [103, 96] and was the first published work that looked at using federated tokens with IoT devices. The second work — Intelligent Gateway for Networked Internet of Things Environments (IGNITE) — addressed the wider issue of API management in the context of IoT devices, and was published in [101].

5.1.1 Motivation For Federated Identity And Access Management In IoT

The traditional approach to access control is based on the concept of roles, and is typically managed in a hierarchical, top-down approach [234]. This approach has distinct drawbacks for the Internet of Things. Firstly, it was designed without millions or billions of devices in mind. That would not be an issue, if every device has the same access requirements. However, a fundamental tenet of privacy is that users can decide (and understand) who can share their data. Consumers demand, for example, to allow a specific application access to only certain data. A user might allow their weight-loss club access to a rolling 7-day average but not to individual days weight values. This argues towards a highly controllable model where users can specify authorisation for specific devices and/or applications.

The second concern with the traditional model is that it utilizes a centralized model of identity and authentication. When there are many devices manufactured by many different organisations and operating in many environments, this is an unrealistic requirement. There are also significant scaling issues with centralized identity models, which are obviated by using a federated model.

A third important concern that is not answered by traditional role-based security models is that of a mechanism for delegation of authority. In many IoT scenarios, there are machines that are operating on behalf of a user, and also scenarios where a device may operate on a third-party's behalf for a specific period of time. For example, the owner of a smart lock may authorise a friend's mobile phone to unlock that door for the next week so that the friend may feed the owner's cat. This argues for a model where the user can *delegate* certain permissions to specific resources to a machine for a limited time.

For this to happen, users must have effective controls over their own data and the ability to specify how this data is shared. This is a major motivation for the use of FIAM for IoT.

In particular there are significant challenges around *privacy* with IoT. Many IoT devices are collecting personal data: including human activity, sleep patterns, health information, home automation usage, geographic locations, etc. The result is that access to that data or ability to manipulate those devices may infringe on privacy. As a real example, in 2011, it was publicized that the sexual activity of users of the FitBit activity tracker was public by default [297]. A key concern here was that many users were unaware that the data collected by Fitbit could be used to identify sexual activity. Similarly, concerns have been raised around the data that a Google Nest device collects [136], and whether consumers are aware of the implications of how that data can be analysed to understand their lives.

5.1.2 Research Questions And Contributions

The work in this preliminary research addresses some key questions for the use of FIAM and IoT:

- What is the importance of FIAM in the IoT space?
- Can one adapt existing Web-based FIAM technologies to IoT and in particular to lower power binary IoT protocols?
- What are the challenges of using FIAM in the IoT context and what changes need to be made to support its use?
- What architectures can support FIAM with IoT?

In addition, this work addressed the new problem of adapting the principles and technologies of Web API management to the landscape of the IoT, which poses challenges stemming from the great numbers and low power of IoT devices, compared to typical full-fledged clients for Web APIs. The problems addressed can be clearly stated:

- What is the impact of the Internet of Things onto Web APIs and Web API Management?
- How do IoT devices identify themselves to Web APIs over IoT protocols?
- How can one add IoT protocol support to existing Web API Management systems?
- What is the impact of adding identity, usage control and analytics to existing IoT protocol interactions?

The preliminary work presented in this chapter provides a number of contributions.

- This work provided the first published implementation of an FIAM protocol with IoT devices.
- Challenges were identified around using existing Web-based protocols (particularly OAuth2) with IoT devices as well as IoT servers.
- Prototype systems were created that implemented these approaches, demonstrating that this approach can work in running systems.
- Through the analysis of the first prototype and the development of the second work, significant challenges were identified for using FIAM with IoT, especially with regard to potential hardware attacks on devices, and these are addressed through the inclusion of an additional component (Dynamic Client Registration). This was the first published work to explicitly address this requirement in IoT devices to the knowledge of the author.
- A performance analysis of the IGNITE system was presented and this showed that FIAM technologies can be implemented in an effective way for IoT that performs within acceptable ranges.

5.1.3 Outline Of The Chapter

The remainder of this chapter is structured as follows. In Section 5.2 it is presented how to use OAuth2 and FIAM with IoT systems, including a first prototype - *FIOT* - that provides a framework for analysing how FIAM and IoT intersect. In Section 5.3 the model of the first prototype is improved into a second model - *IGNITE* - that also explores how FIAM and IoT intersect with the concepts and architecture of Web API

Management. In Section 5.4 performance analysis of the IGNITE prototype is presented and the impact of this performance on real world systems is analysed. In Section 5.5 the work is reviewed, conclusions are drawn, and further work is proposed.

5.2 Federated Identity And Access Management For IoT

This section describes the work done to create a system called FIOT which utilizes the OAuth2 protocol to provide authentication and access control for IoT devices and networks that use the MQTT protocol.

5.2.1 Research Questions Of The FIOT Work

The main research aim is to explore whether it is feasible and effective to use OAuth2 as part of the MQTT protocol flow and within an MQTT broker to make federated, user-directed access control decisions. This aim translates in terms of a number of research questions that include:

- Identifying whether there are any significant issues to using OAuth2 with MQTT.
- Identifying parts of the OAuth2 specification that are amenable to be used with MQTT and whether there are any mismatches or areas where existing specifications need to be modified or enhanced to support the new model.
- Understanding which message formats could work and how could the OAuth2 tokens can be used in the flow, what the overall flow itself would look like and generally whether there are improvements to the implementation systems that would help support the aims.
- Understanding if there are any wider impacts or lessons to be learnt when applying Federated Identity and Access Management to IoT.

The approach taken for using OAuth2 tokens with MQTT in FIOT is as follows. Firstly, the Bearer and Refresh tokens are created using a normal HTTP flow. These are then embedded into the flash memory of the device, and then passed to the broker at the MQTT CONNECT in the userid/password field. At publish or subscribe, the broker uses the OAuth2 introspection API on the Authorisation Server to evaluate if the scopes supported by the token allowed the requested action, and if the token is valid.

5.2.2 FIOT Implementation

In order to implement the system, several existing open source projects were used that provided a set of building blocks. This allowed focus on the core concerns of

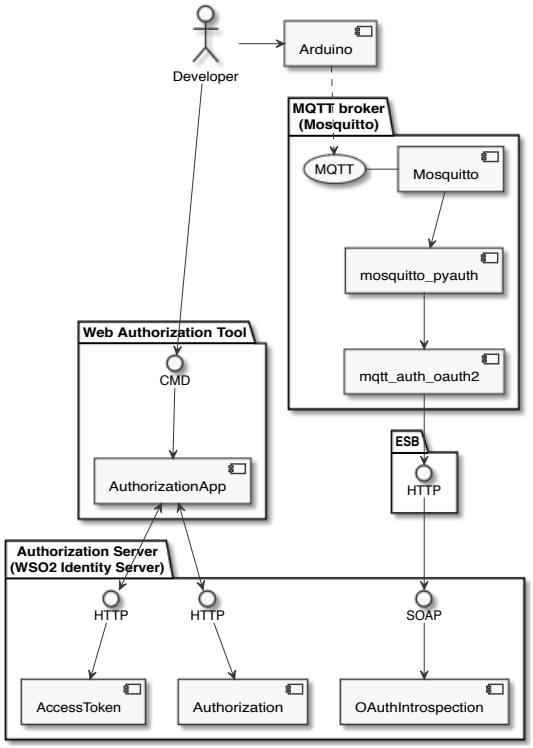

Figure 5.1: Component Diagram of FIOT

authorisation and security without spending too much time on implementing aspects that were already understood. There are some areas where the choice of existing technologies provided limitations, and those areas are called out in the results section below.

The overall system consists of five major components:

- the MQTT broker: based on the open source Mosquitto project [12], together with extensions created for this work
- the Authorisation Server, based on the open source *WSO2 Identity Server* [288].
- the Web Authorisation Tool, a system created to enable developers to initiate tokens.
- the Device, based on an 8-bit Arduino open hardware device [16].
- the Message Viewer, an MQTT client populated with a valid bearer token subscribing to data from the device.

Figure 5.1 shows the major components of the system and the major interactions between them.

In order to capture authorisation scopes that a token is given access to, a JSON [138]

encoding was created. The following example shows a scope that encodes a client who can write to any subtree of */topic/paul* and read/write to */scratch*:

```
[   {rw:"w", topic:"/topic/paul/#"},
    {rw:"rw", topic:"/scratch"} ]
```

This model uses the same syntax for wildcards as the MQTT specification, which is the logical and most effective approach. Because the scope list in OAuth2 is space-delimited, this JSON cannot be used as-is to form the authorisation scopes. To solve this, a Base64-encoding was added to create simple strings that are used as scopes. There are other solutions that could have been used, but this was expedient as it made parsing the scopes simple for the broker.

The Base64-encoded scopes were not humanly-readable. This meant it was harder for the user to validate the meaning of the scope request. However, in a real consumer facing system, the scopes should be human-readable names with proper descriptions which would be mapped by the broker into the topic hierarchy.

The Web Authorisation Tool (WAT) allows developers to request tokens for a given scope against the Authorisation Server. This tool generates a textual version of the token, suitable for cutting and pasting into other systems; and generates C code for the Arduino, which can be cut and pasted into the Arduino tooling to update the Arduino. In a real environment as opposed to this prototype, this tool could potentially be extended to write this directly into the persistent flash memory of a connected device.

The device utilizes a 9-axis inertial measurement unit (IMU) that provides acceleration, rotation and compass data – each in 3 dimensions, to track the movement of the device. Such devices, when attached to a person, provide significant data on the users position, activity and exercise levels. Such data exemplifies the problem space here: users wish to share this kind of data, but only in precisely controlled ways, and each user may have radically different approaches and concerns about both data sharing and privacy.

One important point to note is that there is no implementation of any TLS or encryption from the device to the MQTT broker. This was because of space requirements in the Arduino device. However, this issue is orthogonal to the other concerns as OAuth2 and MQTT layer cleanly sits over TLS, without requiring any modification to the flows.

The MQTT broker was extended with an authorisation plugin, which validates the OAuth2 bearer token which the client passes over. The plugin sends this OAuth token to the *OAuth Introspection Service*, that validates the token and returns a set of authorised scopes for this token. The authentication/authorisation plugin then validates if the requested action (reading or writing to a topic) is valid against this scope.

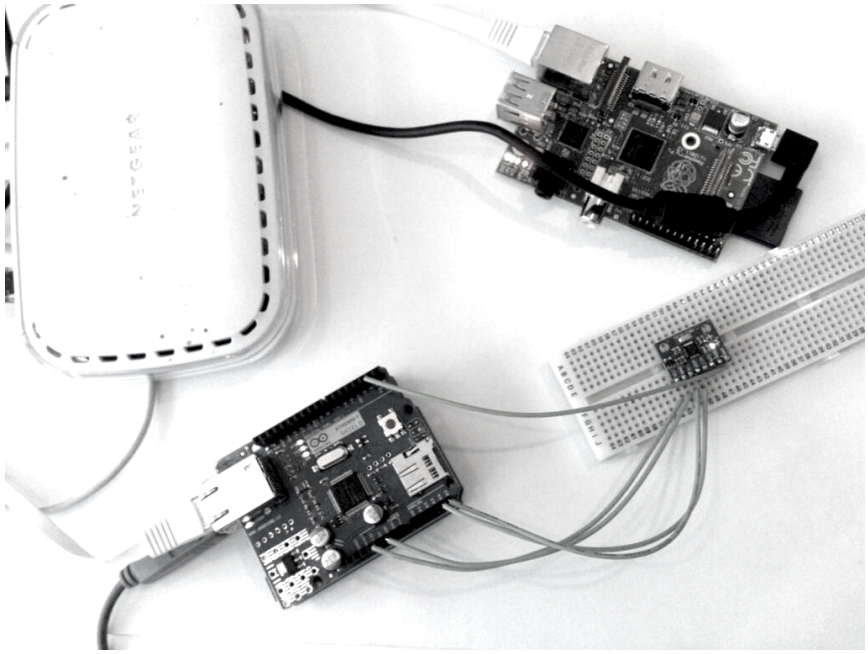

Figure 5.2: FIOT Arduino Device Prototype with 9-axis IMU

A picture of the prototype is shown in Figure 5.2, where the device is connected to a Mosquitto broker running on Raspberry Pi hardware. Two sequence diagrams show the system interactions: Figure 5.3 shows the sequence diagram of interactions during the bootstrap phase, and Figure 5.4 shows the sequence diagram of the interactions during the use of the device and the Message Viewer.

5.2.3 Results Of The Prototyping Of The FIOT System

Overall the system worked as intended, and showed the following aspects:

- Both IoT and non-IoT clients were able to use OAuth tokens to authenticate to the broker.
- The broker was able to connect using a RESTful interface to the OAuth Authorisation Server to validate tokens.
- The broker was able to introspect the token via the RESTful interface on the Authorisation Server to retrieve a list of appropriate scopes.
- The broker was able to use the scopes to decide on whether to authorise a publish or subscribe operation.
- The Web Authorisation Tool was able to implement the OAuth 2.0 bootstrap process to create Access Tokens, which were then embedded into the MQTT clients.

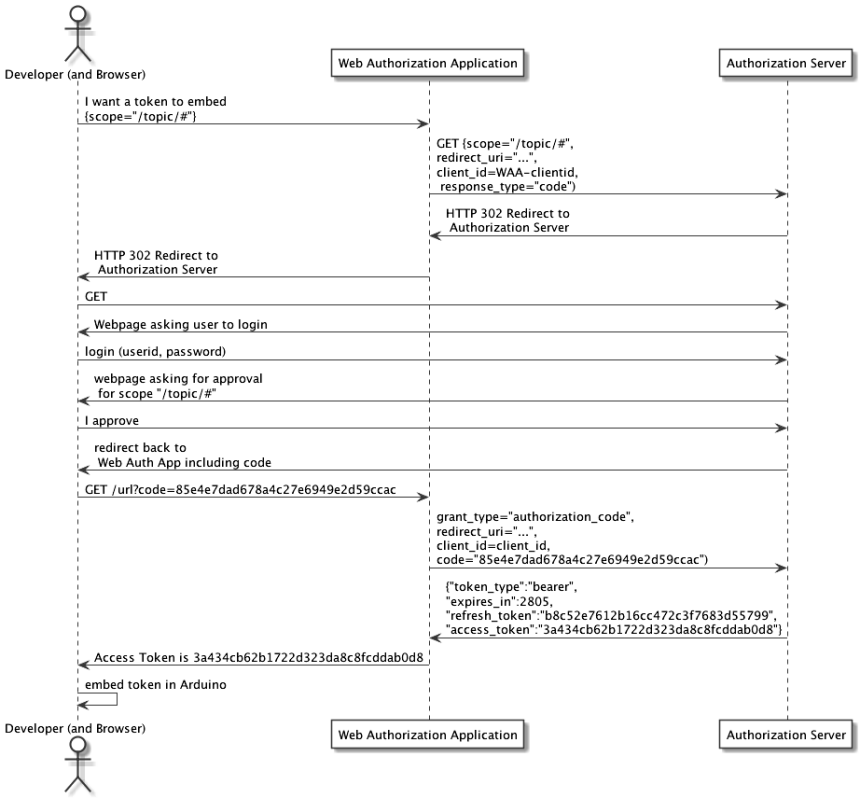

Figure 5.3: UML Sequence Diagram of the Bootstrap in FIOT

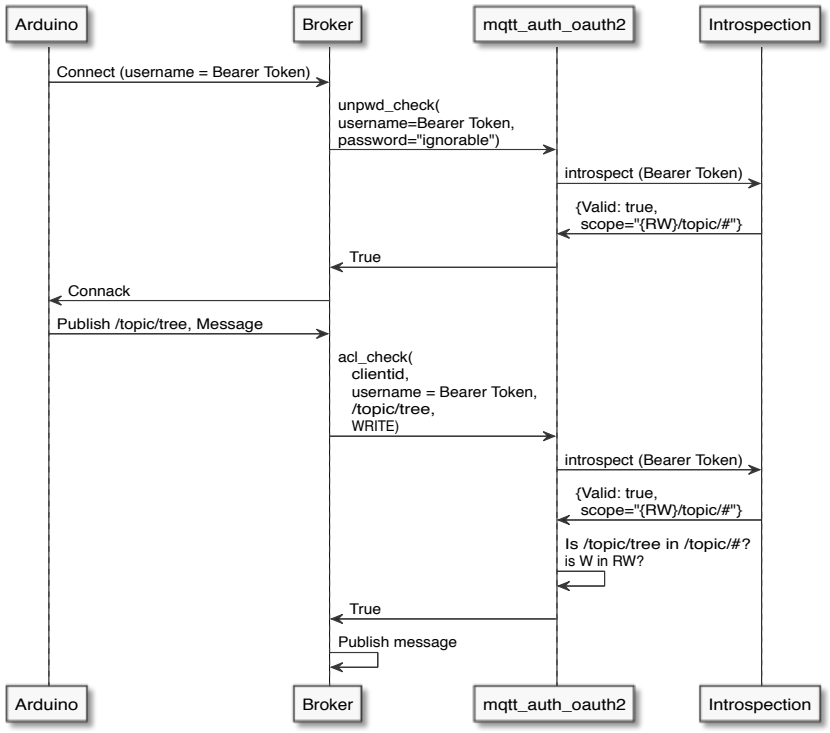

Figure 5.4: UML Sequence Diagram of OAuth Access Control in FIOT

However, this work did identify several concerns with the overall model during implementation. Those concerns are more fully documented in the original paper. The most serious concern was with the modeling of Clients. In the FIOT approach, the WAT was the OAuth2 Client, and each device had a token issued by that client. In order to refresh the tokens, the ClientID and ClientSecret were deployed in a separate server that was specifically designed to handle refresh flows on behalf of clients. After the publication of the original paper, I identified a significant security flaw in the refresh flow, based on the fact that the refresh flow was handled in MQTT. The MQTT specification allows a new connection using the same MQTT client identifier to usurp a previous connection, meaning that an attack could potentially usurp a refresh flow and steal the updated bearer token. This led to the design in IGNITE and later work, where each IoT device is treated as a separate OAuth2 client with its own ClientID and ClientSecret.

5.2.4 Conclusions Of The First Phase

In this section, a standardized federated, dynamic, user-directed authentication and authorisation model has been shown to be adapted from the Web for use in IoT devices. It has been argued that this model is important for the concept of privacy with respect to IoT devices and the data that they generate and use. A number of issues have been identified, including both minor implementation issues as well as more fundamental issues further research is proposed This was the first such implementation of OAuth2 with MQTT. One particularly concerning aspect was the issue around trust in the refresh broker, and the fact that each device did not have its own Client ID and Client Secret. This is a key area that is improved in the next iteration, the IGNITE system (see Section5.3).

5.3 Exploration Of FIAM And API Management In IoT

In this section an improved approach is outlined to applying federated identity and access control to MQTT and IoT devices. This is integrated into existing Web API management systems to provide a wider context for understanding how IoT devices communicate in the overall Web.

Web APIs are capabilities offered across the Web that are designed to be accessed by software rather than people. Unlike traditional APIs, Web APIs are inherently public or semi-public in that they are designed to be used over the public Internet and not solely over private networks or VPNs. The public nature of Web APIs poses a number of challenges addressed by the emerging area of Web API Management (WAM).

Inevitably the Internet of Things will need to engage with Web APIs. For now, most IoT devices connect to services that are created by the provider of the hardware, and so are using private APIs. Public APIs are an increasingly important factor. There are a set of companies that are providing common cloud services and corresponding APIs for IoT (such as Xively [289]), and there are emerging API standards for IoT communication (such as HyperCat [155]). Much of the envisioned strength of the IoT will emerge when data from multiple sources can be aggregated, analysed and acted upon. This will increase the demand for IoT devices to communicate with open Web APIs.

This section looks specifically at how WAM intersects with IoT.

5.3.1 Related Work On Web API Management

While there is a great deal of industrial effort and research on Web API management, the academic literature is sparse. In the industrial sector, much of the literature is provided by vendors. However, the report by Forrester [122] provides a good overview. In the academic literature, Raivio et al. [213] explore the business models around Open APIs for the telecommunications industry. In [147], Kopecky, Boakes and I discussed the challenges and approaches of managing Web APIs.

In the IoT space, there are a number of efforts around creating open APIs for IoT: for instance, HyperCat [155] is a JSON-based catalogue format for exposing IoT information over the Web, developed by a consortium of academic and industrial partners, and ZettaJS [298] is an open source Web API for IoT devices.

There are a number of existing IoT gateways, including [300, 67, 58], that deal with the problem of connecting wireless devices to the wider Internet. They typically bridge multiple low-power devices in a house or factory into a traditional Internet connection. However, the literature search did not identify any server-side gateways or *reverse proxies*[1] specifically designed for IoT.

As discussed above, a new approach was chosen where each device has a unique ClientID and ClientSecret. It is impractical to think that these client keys will be issued manually to the IoT devices: this process must be automated. This is enabled by the extension to the OAuth2 specification called Dynamic Client Registration (DCR) [231]. DCR automates the process that a developer would go through on the API portal to gain OAuth2 credentials on behalf of their API client. The second IGNITE prototype was built to explore the use of DCR in IoT scenarios.

Looking at existing systems and literature, there was no identification of any API yet in production where millions of devices each have their own API key, their own set

[1] A reverse proxy is a proxy that sits at the server-side and is transparent to the clients

of throttling measures, etc. It can therefore be seen that API management systems will need to evolve to support very large numbers of clients, with millions or even tens of millions of concurrently connected devices. In the later work described in Chapter 8 the system was designed specifically to cope with very large numbers of OAuth2 Clients without performance issues.

In summary, this work is addressing how to adapt the existing Web API management capabilities to support:

- Large numbers of clients, each with their own credential.
- Devices communicating with public APIs via binary and low-energy protocols such as MQTT and CoAP.
- Usage control, access control, throttling and other API management techniques applied to IoT scenarios.
- Federated Identity and Access Management for IoT devices.
- How to apply these capabilities orthogonally to existing systems.

5.3.2 IGNITE - An API Gateway For IoT Protocols

To solve these issues a system was prototyped that allows using the capabilities of existing API management solutions with IoT protocols. The system is called IGNITE (Intelligent Gateway for Network IoT Events)[2]. The initial work focuses on the MQTT protocol, but it could be extended to CoAP.

For the proof-of-concept prototype, three existing open source projects were used:

- The WSO2 API Manager [287] project provides the main capabilities for Web API Management including a developer portal, subscription management system, key server, API gateway, access control, throttling, monitoring and analytics system;
- The MITREid-Connect project implements of OAuth2 and OpenID Connect [221] and includes new capabilities such as Dynamic Client Registation and Token Introspection.
- The Mosquitto MQTT broker provides an open source messaging broker for the MQTT protocol.

IGNITE takes a different approach to FIOT for adding OAuth2 authorisation to IoT device flows. Instead of extending the MQTT broker with authorisation logic, the IGNITE approach is to add a new authorisation gateway between the client and the

[2]The source code is available at `https://github.com/pzfreo/ignite`

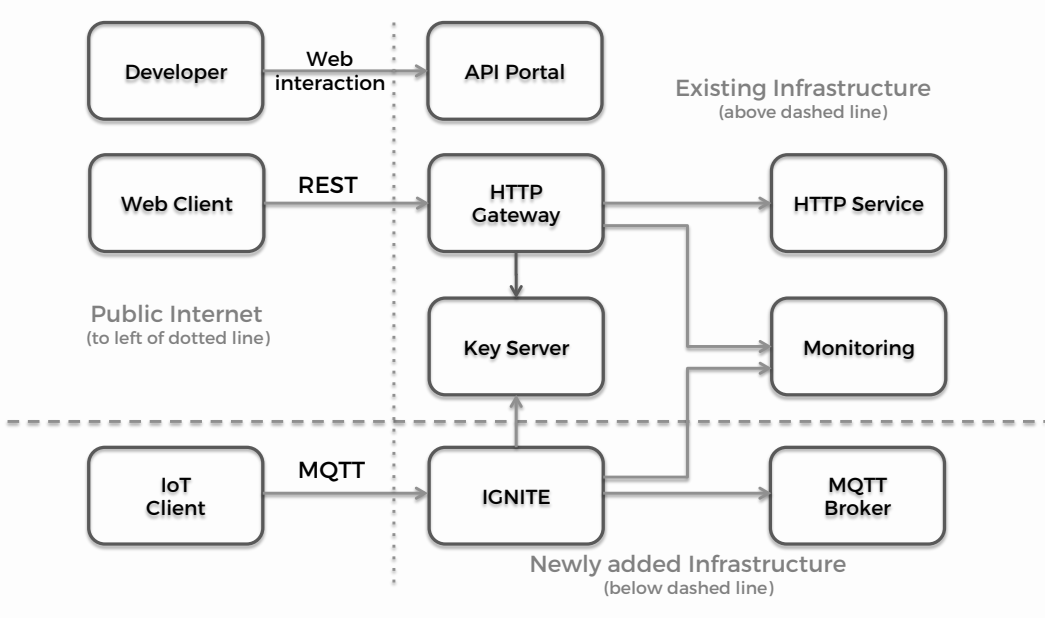

Figure 5.5: IGNITE System Architecture

broker. This gateway follows a well-understood pattern — the *reverse proxy*. This sits side-by-side with the existing API gateway and shares the same tokens. In a future version this could be integrated into the HTTP-based API gateway.

A first prototype of this gateway was written in Python and there was some initial work to port it to Java to improve performance. It will be seen in Chapter 8 that in the end the decision was made to move this to JavaScript instead. Figure 5.5 shows the overall architecture with the capabilities of the existing projects plus the added capabilities of this work.

The IGNITE component implements the following logic: On a `CONNECT` packet arriving, it extracts the OAuth2 Bearer token from the username field in the packet. It then invokes the Token Introspection service on the MITREid-Connect server to validate the token. If the token is valid, the gateway replaces the token in the request with the userid returned from the introspection call, and forwards the request on to the existing MQTT Broker, which may implement its own validation checks as well. If the token is invalid or no longer active, the IGNITE responds to the client with a packet that indicates that the credential was invalid (a `CONNACK` packet with `ReturnCode=5`).

5.4 Results

To test the system, the performance of this system was evaluated compared to a direct call to the MQTT broker. In this case, the MQTT broker was not running any authentication of its own, so the comparison is not completely like-for-like. Figure 5.6 shows

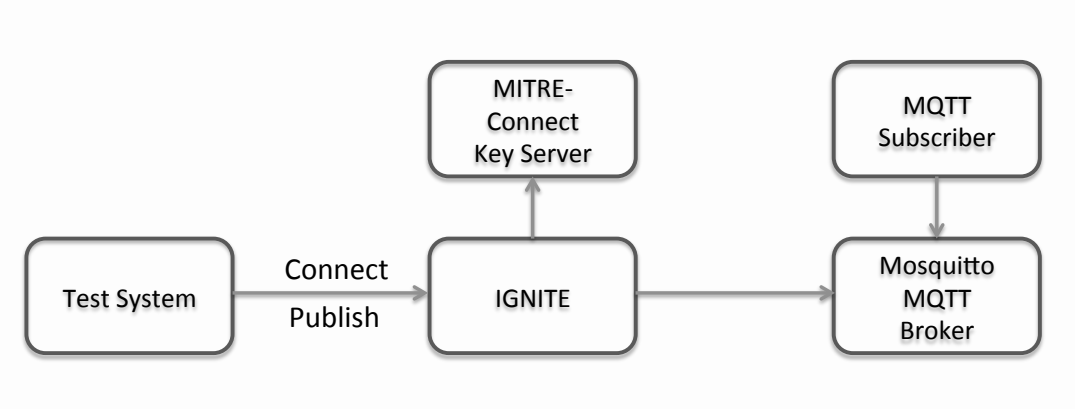

Figure 5.6: IGNITE Test Architecture

the architecture of the test set-up.

For the tests two flows were sampled: A `CONNECT` flow and a `PUBLISH` flow. For `PUBLISH` all three levels of QoS were tested: fire-and-forget (QoS0), at least once (QoS1) and exactly-once (QoS2). QoS1 and QoS2 involve multiple packets transferring between the client and the server.

The tests were all run on a single machine[3] using the localhost networking. The gateway tests include both the more functional Python prototype of IGNITE and an early prototype of the Java version. The tests show the average result over 1,000 `CONNECT/CONNACK` messages and 10,000 `PUBLISH` messages, in both cases giving the system time to warm up before capturing timing data. The QoS 1 and 2 tests inherently capture the use of `PUBACK`, `PUBREC`, `PUBREL`, and `PUBCOMP` messages. The focus on connection was because the authentication step during connection is where the most work takes place, and on publication because this is the most used flow in MQTT, as subscriptions are infrequent compared to publications.

The `CONNECT` results are shown in Figure 5.7. The results show that the overhead of using the Python IGNITE for `CONNECT` is around 7,700µs per request. Given that this includes a HTTP REST call to the key server this is not unexpected. In the WSO2 API Manager this overhead has been reduced by implementing a binary key validation protocol instead of HTTP. However given that MQTT is a persistent connection compared to existing Web API gateways and HTTP where each request needs to be validated, this is a very effective result. There was no implementation of caching of token introspection results which could improve this considerably.

The `PUBLISH` numbers (Fig. 5.8) show a much lower overhead. For QoS0 the overhead of going through the IGNITE is around 11µs. The QoS2 case has a significant higher overhead due to considerably more complex message flow. Even in this case the overhead is less than 1500µs and the preliminary data from the Java implementation

[3]Mac OS/X 10.10 running on a 3Ghz Intel Core i7 with 16Gb RAM and SSD storage

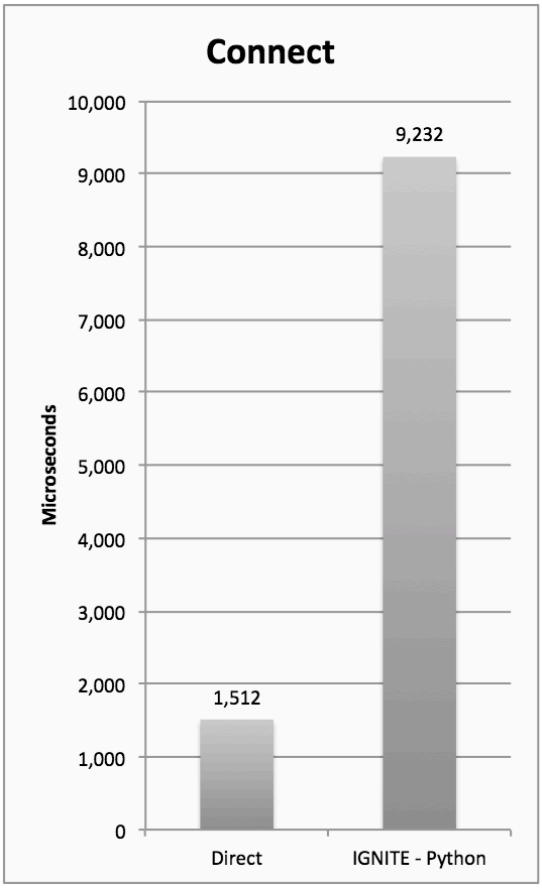

Figure 5.7: CONNECT Performance Test with IGNITE

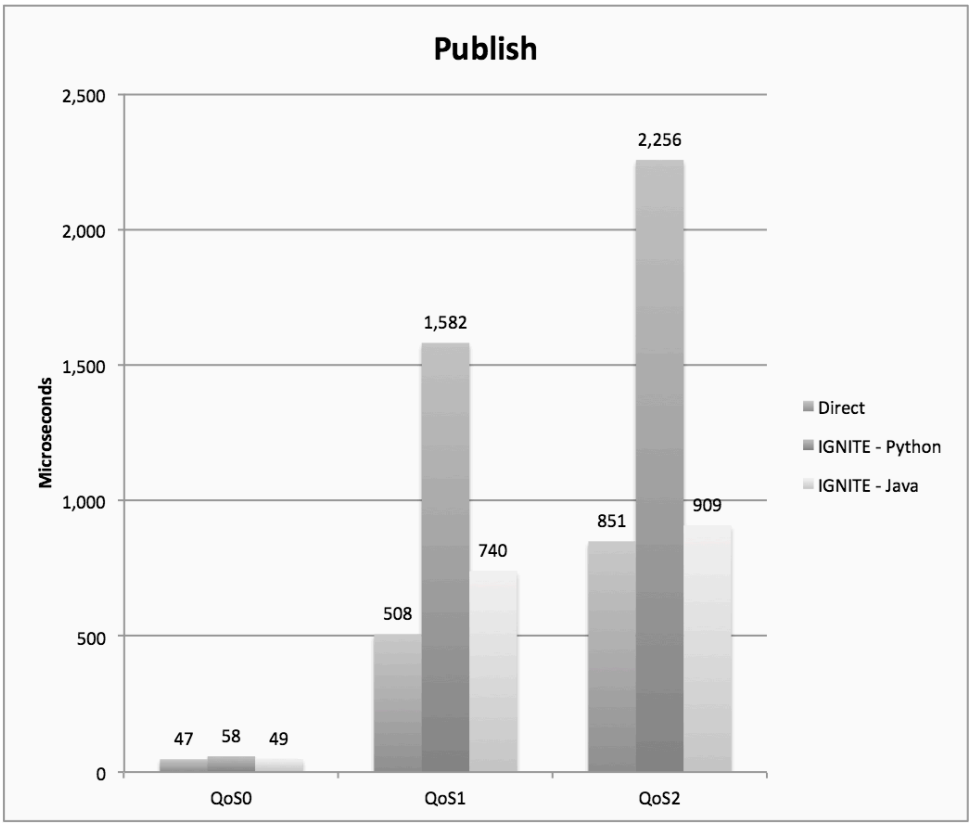

Figure 5.8: PUBLISH Performance Test with IGNITE

shows an overhead of less than 60µs. Note that at this stage there is no implementation of usage control and monitoring into the `PUBLISH` flow so these numbers do not yet reflect the full workload required.

To put these numbers into perspective, the typical overhead of such a gateway in the HTTP world is around 500µs without implementing any OAuth2 token introspection [286]. In addition, these numbers are all likely to be dwarfed by average Internet latencies. For example, the speed of light requires a minimum latency of 40,000µs between the East and West Coasts of the USA, and typical real-world latencies are twice this. Even with prototype code and no optimisation, these numbers are respectable and would fit into the tolerance of many existing IoT projects. Therefore it can be concluded that this approach is eminently practicable.

5.5 Discussion

In this chapter two key aspects have been addressed. Firstly, the concepts and realities of Federated Identity and Access Management for IoT were examined. Secondly, two approaches for implementing this using OAuth2 and MQTT were presented. The second approach offers a significant improvement in the model and security of the

system via the use of Dynamic Client Registration.

As a result of this work, there are several recommendations that are proposed. These include the need for clear standardisation of where to put the token as well as limiting the OAuth2 token size to some reasonable limit - at least for IoT use. Also there is a need to define a clear MQTT flow for refresh. Thirdly there is a requirement to support DCR to ensure that each device has unique credentials. This ensures that a compromise to a single device does not compromise other devices. A concern was identified with the refresh token being transmitted unreliably, and therefore it is recommended that refresh tokens are not updated unless there is a reliability mechanism in place for the refresh flow.

While there were some issues with implementing FIAM for IoT using OAuth2 with MQTT, the benefits of building on existing widely implemented and deployed protocols are significant. Many years of work and review have gone into the security of OAuth2, and from this work it can be stated that it is possible to re-use this work with IoT devices and new protocols.

In addition, a promising approach was identified to unify IoT devices and MQTT into existing Web API Management technologies and architecture. This model demonstrates enhancing existing Web API management systems with a new gateway — IGNITE — that focuses on IoT protocols and demonstrates how protocols such as MQTT can be integrated into existing API Management models with some success, in a completely orthogonal manner. In addition, the model of the server-side IoT gateway that is introduced with IGNITE offers a considerable number of possibilities for managing usage control, access control, monitoring, etc.

Preliminary data on performance which shows that the overheads of such approaches are reasonable, even before optimisation, caching and other techniques are introduced.

Part III

Part III - Main Research

Chapter 6

Model

In this chapter, the model and architecture developed for this work is presented and examined. The model and architecture are aiming to satisfy the requirements that were identified from the literature review in Chapter 2. This model is an iterative development of the two approaches outlined in Chapter 5. In particular, those approaches did not address device registration and bootstrap, and they did not attempt to deal with pseudonymity. In addition, this model includes the use of *Personal Cloud Middleware*.

6.1 Introduction

The model is normatively defined in CSP algebra. CSP is a process algebra with strongly defined denotational and trace semantics [126]. In addition to the CSP there is an introduction presented with the aid of UML. UML offers a graphical modelling approach that is more familiar to software developers, architects and engineers, which complements the more formal approach used in CSP.

The aim of producing this model is threefold. Firstly, the CSP allows one to reason about the model and therefore understand the attributes of the model, and in this way identify if the proposed approach offers improvements to the security and privacy of IoT networks. In particular, through both mathematical proof and the CSP technique of *trace refinement*, it can be proved that the model has specific properties. Secondly, the model provides a clear and unambiguous presentation of the approach for evaluators and implementors of this approach. The algrebraic approach provides the most clarity and the least ambiguity and is therefore the normative model. However, UML is more widely used and the graphical representation can help communicate the model more effectively, and therefore this work adds in a set of UML diagrams that capture aspects of the model. These UML models were created through a combination of

manually inspecting the CSP model, together with documenting machine-generated traces of the CSP model. The UML models are designed to communicate aspects of the approach rather than be the definitive model and therefore they are non-normative. Finally, the aim of this model is to encourage implementation of the approach. The implementation of a prototype is described in Chapter 8.

One of the benefits of CSP is that it provides a powerful *refinement checker* — Failures Divergences Refinement (FDR) [108], that can validate models against specifications. This work uses a combination of mathematical proof using the laws of CSP, and the model checker to prove a set of assertions about data sharing in the system. These assertions are used to create a matrix of data propagation. This matrix, together with the model, are then used as input to a threat modelling exercise, in Chapter 7, where the threats against this system will be explored.

The rest of this chapter is laid out as follows. In Section 6.2 there is a short, informal overview of the model designed to give a high-level understanding. In Section 6.3 there is an introduction to CSP process algebra as well as a discussion of why CSP was chosen. In Section 6.4 the model is defined in an algrebraic format using the CSP language. In Section 6.5 there are a set of trace-refinement proofs that prove properties of the CSP model. In Section 6.6 the chapter is summarised.

6.2 Informal Description Of The Model

While the CSP description of the model provides a rigorous and clear description of the system and allows one to reason about properties of the system, it is easy to become buried in the detail. Therefore this chapter starts with a high-level informal description of the model, aided by a few chosen UML diagrams.

6.2.1 Assumptions And Boundaries

In this model, and the subsequent prototype, the concept of device is restricted to systems that can directly contact the Internet. For example, the prototype uses a Wifi connected chip, which allows direct connection to Internet systems. This excludes systems that connect via Bluetooth or other non-IP networking. It is possible to think of such Bluetooth connected systems as sensors or actuators connected wirelessly to a "device" (such as a mobile phone), that does participate in this model. For example, a connected car might be seen as a "device" in the terms of this model, and some sensors or actuators might be wirelessly connected over various non-IP protocols to the central processing unit of the car. In addition, this model assumes that devices have at least intermittent connections to the Internet, whereby tokens can be refreshed. Both of

these areas could potentially be supported by extending this work, and this is left for further work.

6.2.2 Participants

Figure 6.1 shows the current situation for many IoT systems, where there is no federation and the device talks to a single service that manages identity, stores data, provides a user web interface, etc.

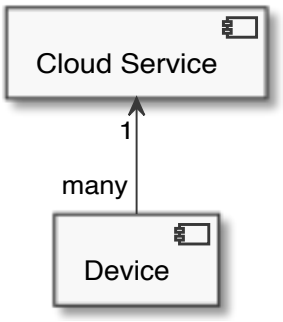

Figure 6.1: Existing Model

By comparison, the federated model presented in Figure 6.2 allows different federated parties to provide different services that work together. This approach is called *OAuthing*.

The participants of the OAuthing model are:

- **The User Identity Provider (UIdP)**: this is an existing login system where Users present their credentials (e.g. Google, Facebook, Github, Twitter, or any OIDC login).
- **The User**: A User may own one or more Devices. A User must have at least one identity with a UIdP.
- **The Device Identity Provider (DIdP)**: this is an *Identity Broker* that first authenticates a User with a UIdP using existing federated identity protocols including OAuth2, OpenID Connect (OIDC), or SAML2. Once the identity is validated it then creates a secure random anonymous identity which is used in all further processing. This anonymous identity is not shared except with the Intelligent Gateway. Devices and Cloud Services are issued with random tokens that give permission to perform certain actions but do not identify users in any way. Currently, each instance of OAuthing has a single DIdP.

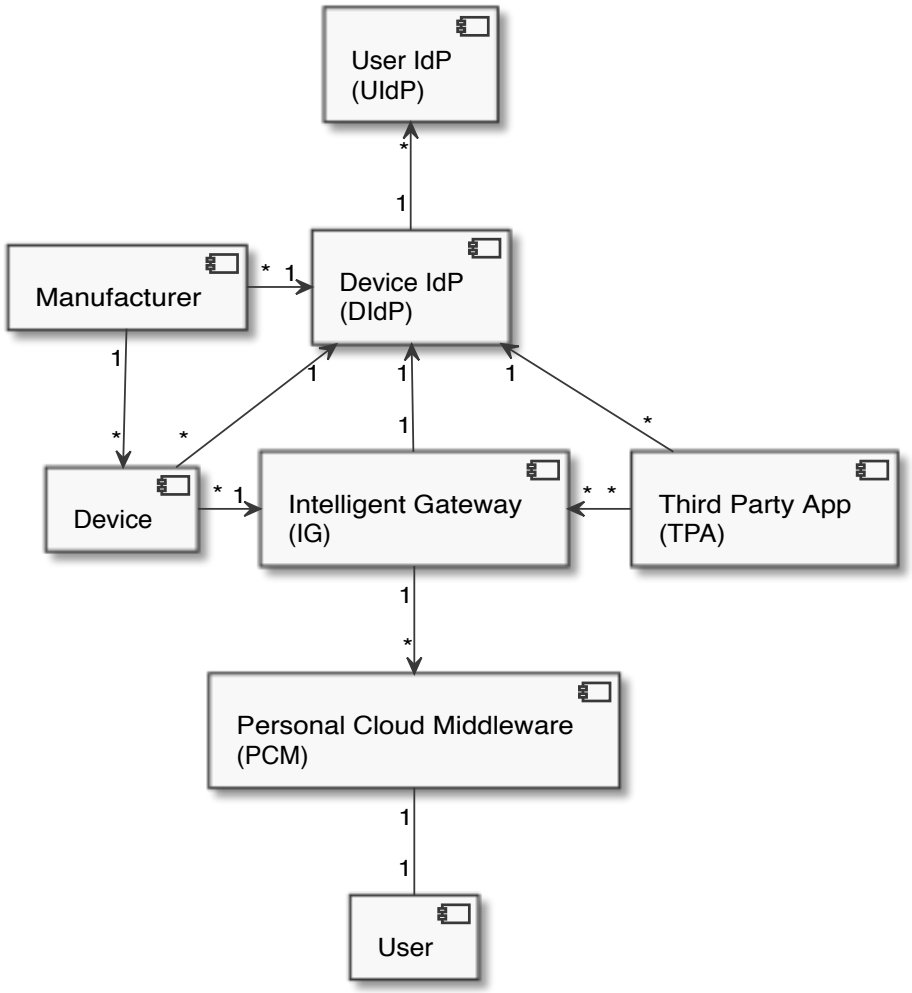

Figure 6.2: Proposed Model

- **Personal Cloud Middleware (PCM)**: this is an isolated broker that shares data between devices and Third Party Applications (TPAs) on behalf of the user. The PCM talks to the Devices and the TPAs. Within the remit of a single OAuthing instance there is one PCM per user. A cloud environment is used to dynamically launch PCM instances on behalf of users as needed.
- **Intelligent Gateway (IG)**: The IG interfaces with the DIdP to validate identities and access authorisation policies and to the cloud infrastructure to instantiate new PCMs. Devices and CSs connect to the IG, and it routes requests to each user's PCM.
- **Third Party Application (TPA)**: A device is an IoT device if and only if it shares or receives data and commands with an Internet service. Users control which TPAs can access their sensor data or control their actuators by explicitly consent-

ing to authorise a TPA. Any third party can provide a TPA. If no TPA is authorised by the user then a Device's data is neither shared nor stored.

- **The Device**: The device consists of one or more sensors and actuators together with a controller. The device is issued with a Client ID at manufacturing time. Once the device is registered with a user, it stores a token that identifies the user, the Client ID and the scopes of access that the user has authorised.
- **The Manufacturer**: The Manufacturer is the logical organisation that creates and markets the Device, irrespective of whether they actually outsource any part of the physical manufacturing to a third party. In this model, the Manufacturer configures each device with a single DIdP.

Figure 6.3 shows the UML sequence diagram of a runtime interaction between a device and a third-party application.

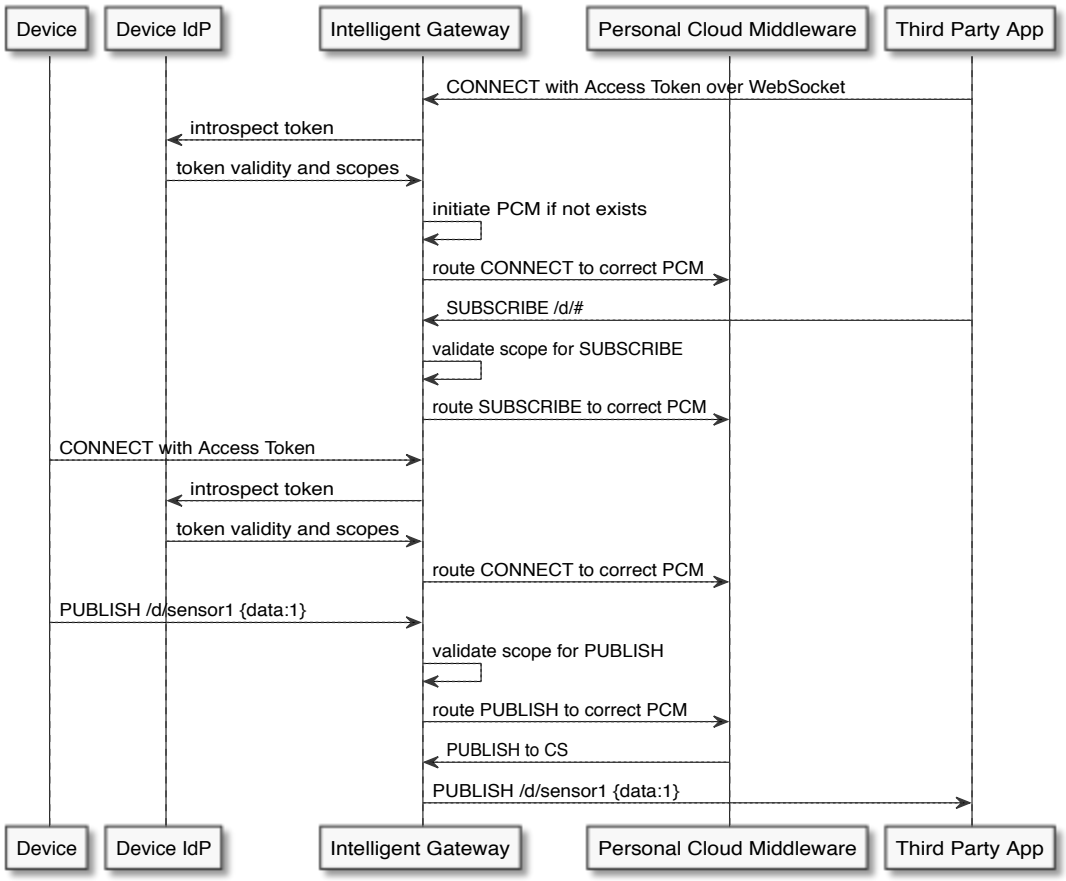

Figure 6.3: Device Publishing Data to App

This model utilises the OAuth2 model as a basis for the identity and ownership of devices. One concern with IoT is that hardware devices can be compromised and secrets read from them. It is therefore important that each device has its own credentials. Each device is to be a unique OAuth2 Client, and the system uses the OAuth2

Client ID as a secure device ID that is only ever shared with the DIdP. Ownership of a device is defined by the user authorising the issuance a security token to the device giving it permission to act on the user's behalf.

6.2.3 Lifecycle

The UML lifecycle diagram is shown in Figure 6.4.

Once the device is initially flashed it is connected to a manufacturing server. The manufacturer then uses the DCR API into the DIdP to request a Client ID and Secret. These are configured into the device by the manufacturing server. At the same time, the DIdP returns a unique User Registration URI (URU), that is printed onto the device (usually as a Quick Response (QR) code) by the manufacturer.

When the user buys the device, they scan the QR code or otherwise access the URU. This directs the user to the DIdP which presents a choice of UIdPs to the user. Once the user is authorised with their existing UIdP, they are asked in turn to authorise the device. The resulting OAuth2 refresh token is then stored on the device, and represents the logical ownership of the device.

If at some future point the user sells the device they can revoke the OAuth2 token - either by resetting the device back to its initial state or by using a web interface at the DIdP.

6.2.4 Personal Cloud Middleware

A key part of the model is the concept of a personal hub: where each user's data is routed to its' own hub, protecting the data from multi-tenant attacks. Each hub is run in its own virtualised Cloud environment. When a request comes in from a device or CS, the pseudonym associated with the bearer token is used to route the request to an instance that is specific to that user. If there is no cloud server available, the routing system makes a call to the cloud management system to instantiate a new PCM "on-demand", and then waits until the instance is running before routing the request to the PCM. In the model the PCM supports routing, distribution of data and commands, as well as summarisation and filtering of data. These capabilities have an important role in protecting users privacy: firstly, the runtime does not inherently share data such as IP addresses or MAC addresses that can be used to identify devices or users. Secondly, by filtering or summarising data, the PCM can avoid many fingerprinting attacks on devices [146]. The PCM can also provide protocol mapping and device shadow capabilities, meaning that it is simpler for TPAs to connect to devices.

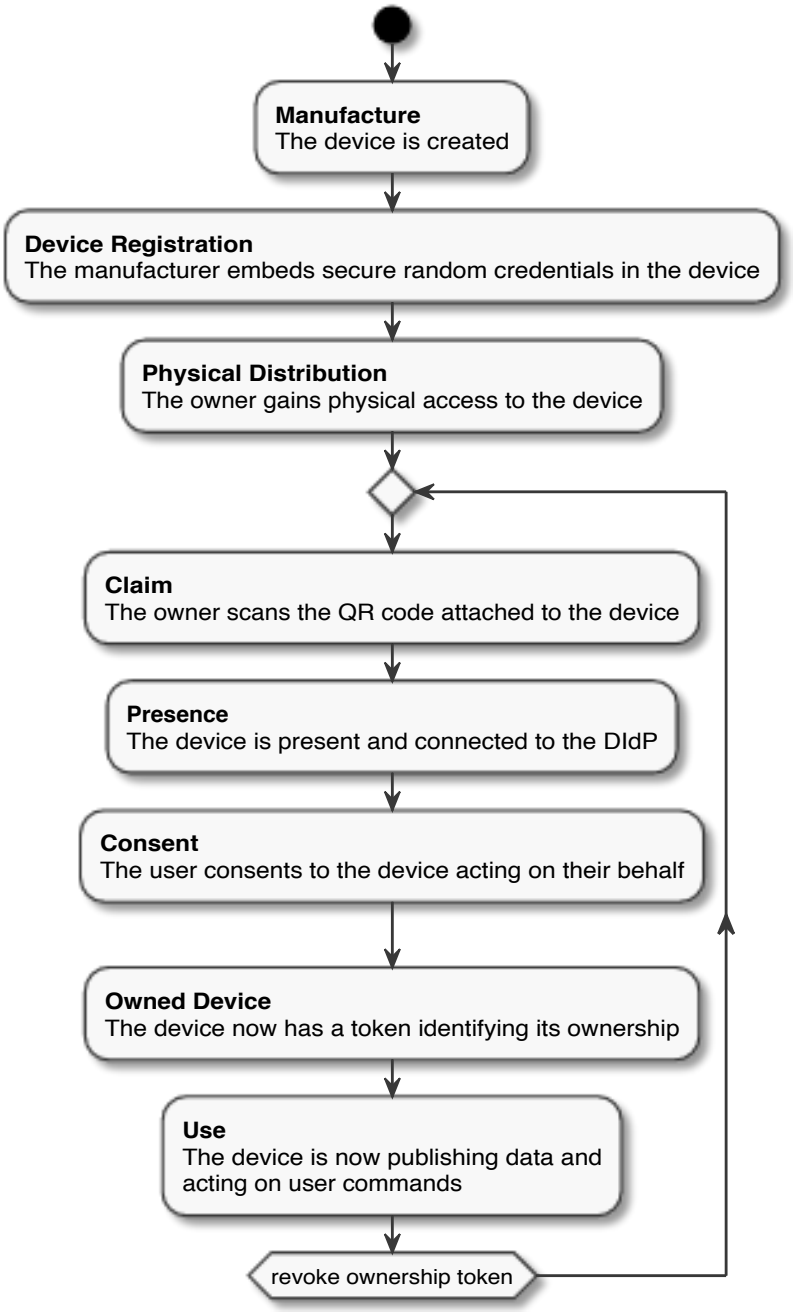

Figure 6.4: Lifecycle of a Device

6.2.5 Scopes

The DIdP implements consent-based authorisation policies called *scopes*. The concept of scopes comes from the OAuth2 specification. Each scope controls access to a set of APIs. These APIs may be implemented in multiple protocols. Users may consent to a third-party to have access to a specific scope, which is captured in a token.

One of the outcomes of defining scopes as part of this model is that there is a clean mapping between the different roles in the system and the scopes which each role requires access to, which is shown in Table 6.1.

Role	Scopes	Description of roles and scopes
UIdP	N/A	This IdP is the primary source of identity to the Device IdP and does not have any OAuth2 scope permissions
DIdP	openid ***(or UIdP Specific)***	The Device IdP is the "source" of scopes to the other roles. It requires access to the third-party IdPs, which may define their own scopes.
Manu-facturer	dcr	Dynamic Client Registration (DCR): allows caller to create new ClientIDs using the DCR API
Intelligent Gateway	intro	Introspection: allows the IG to ask the DIdP for the pseudonym and scopes for a given Bearer Token
TPA	Rd, Pc	Read/Subscribe to Data (Rd) and Publish Commands (Rc) The TPA may be allowed one or other or both
Device	Pd, Rc	Publish Data (Pd). Read/Subscribe to Commands (Rc).

Table 6.1: Mapping of Scopes to Participants

6.3 Formal Modelling

Since the system is fundamentally a distributed, federated system of communicating processes, CSP offers a clear and unambiguous approach for describing these, as well as a powerful refinement checking tool, FDR. CSP offers a mathematical and set-theoretic approach to defining processes and how they interact. In CSP processes communicate through message passing and not through shared state, which is how distributed and federated systems also work, thereby making this a good fit for a modelling approach in this case. CSP dates back to the initial paper in 1978 from Hoare [128]. Recently, the popularity of the Go[1] programming language has renewed interest in CSP, as Go relies heavily on CSP for its parallelism model. The FDR tool uses trace refinement modelling to validate that a defined process behaves as a specification. This is accomplished by defining two different processes, and showing that the finite traces of one process are a subset of the finite traces of the other. While this does not handle infinite traces, in many cases the finite model checking is sufficient.

[1] `https://golang.org`

In this way it is possible to validate that the model meets specifications. In Figure 6.5 there is a screenshot of the FDR system showing a counterexample, where the screen is zoomed into part of the flow. FDR supports both graphical and command-line operation.

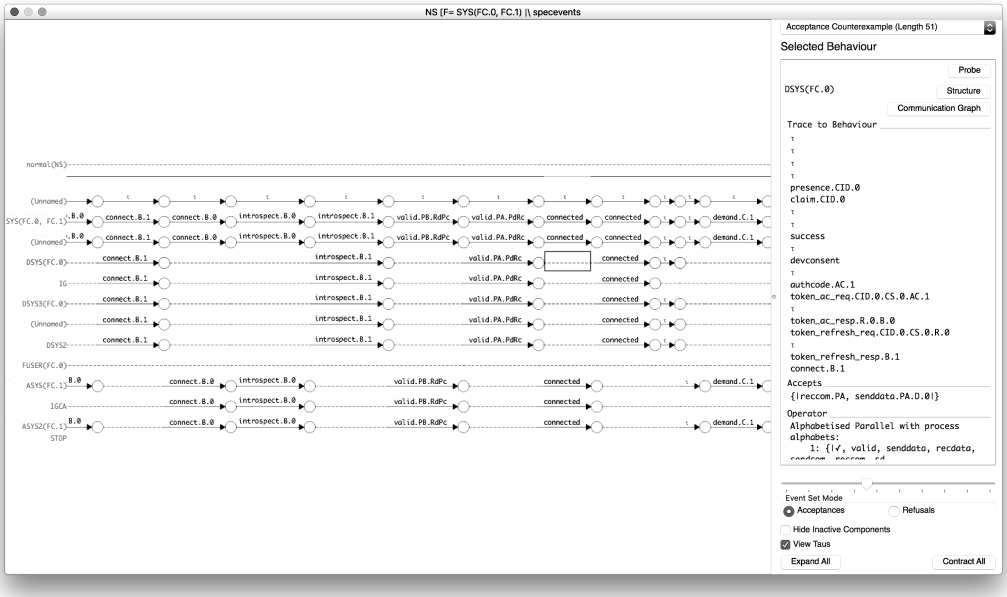

Figure 6.5: FDR in Use

6.3.1 Alternatives To CSP

There are several alternative formalisms that could be used for this work, and some were evaluated, including *Z* [285] and *Event-B* [3]. Out of these two, Z lacks clear support for events and inter-process communications and therefore would entail significant complexity to capture these aspects. The π-calculus [175] is an alternative process algebra that has significant usage. The π-calculus emerged out of Communicating Concurrent Systems (CCS), which was contemporaneously developed with CSP. The advantage of the π-calculus over CCS is the support for *mobile processes* whereby work can move around a network. One aspect of the OAuthing model where that would help is in modelling HTTP redirects, such as in OpenID Connect (OIDC) where the login server is chosen by the user, and the login process can be seen as moving across the network. In CSP, Roscoe has led work to model mobility [224]. Event-B, the π-calculus and CSP all offer useful and clear process algebra with well-defined semantics. In [235], there is a derivation of Event-B refinement in CSP, demonstrating the equivalence of both approaches. In [225] there is work to show that the π-calculus can be represented in CSP and therefore have its semantics defined in terms of CSP. Event-B supports both state-based and symbolic tooling. The π-calculus tooling is

based on symbolic theorem provers. There have been attempts to build symbolic provers around CSP [132, 49], but the productivity of the FDR tool has meant that it remains the leading tool for CSP analysis.

In truth, any of these three systems are good candidates for the modelling approach. The author had personal experience of both CSP and Event-B and in the end CSP was chosen for the expressivity and concinnity of the algebra, together with the productivity of FDR. Because of the aforementioned works that demonstrate that the semantics of Event-B and π-calculus are both derivable in CSP, it can be inferred that any model that could have been produced in those systems can also be modelled in CSP.

There are several works that are used to define the underlying semantics of UML using CSP [64, 190, 291]. There is some early work on automatically generating UML diagrams from CSP [189], but the UML diagrams in this work are generated manually from inspecting the CSP traces produced by the FDR tool.

6.3.2 A Brief Introduction To CSP

CSP is an algebraic approach for reasoning about components that communicate via messages. In CSP, components are called *processes*, although this does not necessarily map to the operating system concept of a process. CSP allows reasoning about individual processes and also communicating groups of processes. Each process communicates using *events*. Now follows a short introduction to CSP. For more in-depth details, [127, 224] both offer detailed introductions.

In CSP the messages between components are called *events*. For example, the process P accepts an event e, and then ($\rightarrow$) stops (accepts no further events):

$$P = e \rightarrow STOP$$

The process *STOP* is the well-defined process that accepts no events. *STOP* effectively captures deadlock, so to distinguish *successful* termination, the event $\checkmark$, (pronounced "success") identifies successful termination. The process *SKIP* is defined as a process that does nothing but terminate successfully.

The process Q accepts event e and then continues as Q, showing recursion. In effect, Q can accept any number of events e:

$$Q = e \rightarrow Q$$

A process can be parameterised, leading to a set of processes:

$$Q(d) = e.d \rightarrow Q(d)$$

In this process, $e.d$ indicates that event e transfers data d.

In addition, CSP supports *pattern matching*, which is a common approach in functional programming languages. For example, the following three lines of CSP syntax define the function $f(x)$ for all values of x:

$$f(0) = \textit{True}$$

$$f(1) = \textit{False}$$

$$f(\_) = \textit{Error}$$

An equivalent definition of the same function as a bijection would be:

$$f = \left\{ (0, True), (1, False), (x, Error) \;\middle|\; \forall\, x \notin \{0, 1\} \right\}$$

Pattern matching is also applicable to processes:

$$P(0) = e.A \rightarrow P(1)$$

$$P(1) = e.B \rightarrow P(0)$$

$$P(\_) = e.C \rightarrow P(0)$$

This defines a set of interlinked processes. For an example of where pattern matching is used in the model, see Definition 6.4.31.

An event can have associated data. Previously, $e.d$ was defined as event e parameterised with data d. In addition it is possible to indicate sending or receiving data with events: sending data is written $e!d$, receiving data as $e?d$. $e\$d$ indicates that d may be any arbitrary non-deterministic choice of data from the range of allowed values of d. This range can be restricted: if D is a set, then $e.d : D$ allows $e.d$ where $d \in D$. The same is applicable with $e\$d : D$ and $e!d : D$ as well.

If P, Q are processes, $P \,\Box\, Q$ is the choice between P and Q where the environment chooses which process continues based on the initial event that is received. This is called *external choice*. For example, $(a \rightarrow P) \,\Box\, (b \rightarrow Q)$ will behave as P after an event a, or Q after an event b. Informally it could be said that the "incoming event" chooses the path that the process takes.

CSP also supports general external choice across a set of processes:

$$\Box_{x \in S} P_x$$

is equivalent to

$$P_a \Box P_b \Box \ldots \Box P_m$$

where $a, b, \ldots, m \in S$.

$P \mid\mid\mid Q$ is the process where P and Q *interleave*: that is, they operate independently with no synchronisation of events.

$P \parallel Q$ is the process formed by running P and Q in parallel, *synchronising* on all events. In other words, they must both accept the same event at the same time.

The $\parallel$-step law captures this most clearly:

$$(?x : A \to P) \parallel (?y : B \to Q) = ?z : A \cap B \to (P \parallel Q)$$

One can say that P and Q only synchronise on events in set E using the following construct:

$$P \underset{E}{\parallel} Q$$

In many cases there is the need to define the combination of two processes that each offer certain events, whilst synchronising on yet others. The construct:

$$P \; {}_X\!\parallel_Y \; Q$$

indicates that P handles events in X except those in Y (notated $X \setminus Y$), Q handles events in $Y \setminus X$ and the processes synchronise on events in $X \cap Y$.

In some cases, one needs to rename events:

$$Q = P[[^{to}/_{from}]]$$

to indicate that the process Q acts like P with event *from* renamed to event *to*.

CSP also allows the *hiding* of events. For example:

$$(P \underset{E}{\parallel} Q) \setminus E$$

is the process where P and Q synchronise on events E, but only events that are not in set E are visible outside the process.

The opposite of hiding is *projection*:

$$P \upharpoonright E$$

is the process P where only the events in E are visible.

Hiding events is the most obvious cause of *non-determinism*. For example, if there is a process:

$$(a \rightarrow b \rightarrow P \,\Box\, c \rightarrow d \rightarrow Q) \setminus \{a, c\}$$

then the external observer is not aware of whether internal (i.e., hidden) events are taking the process down the a path or the c path. This gives rise to the concept of *Internal choice* ($R \sqcap S$), where there is non-deterministical choice between the processes R and S.

Therefore

$$(a \rightarrow b \rightarrow P \,\Box\, c \rightarrow d \rightarrow Q) \setminus \{a, c\} = b \rightarrow P \sqcap d \rightarrow Q$$

The concept of *linked parallel* processes encapsulates renaming, hiding and parallel processes into a single concise definition.

$$R = P[c \leftrightarrow d, e \leftrightarrow f]Q$$

indicates that the process R behaves like P interleaved with Q, except that event c from P is renamed as d for Q and vice-versa. Similarly, P sees event f from Q as e and vice-versa. The overall process synchronises on these events (c/d, e/f) and they are hidden in R.

Effectively, if f is a fresh unused name:

$$P[c \leftrightarrow d]Q = (P[[^f/_c]] \underset{f}{\parallel} Q[[^f/_d]]) \setminus \{f\}$$

Another useful notation is:

$$R = P \,\Theta_E\, Q$$

which indicates that R behaves like P until an *exception event* from the set of events E occurs, after which it behaves like Q.

6.3.3 Refinement

CSP defines the concept of *refinement* ($\sqsubseteq$), based on *traces*. The traces are the possible patterns of visible events that a process will accept. One can say that P is *trace-refined*

by Q (written $P \sqsubseteq_{\mathrm{T}} Q$) means that every finite trace of Q is also a finite trace of P:

$$P \sqsubseteq_{\mathrm{T}} Q \Leftrightarrow \mathit{traces}(P) \supseteq \mathit{traces}(Q)$$

This model offers an operational semantics for CSP based on the concept of *Labelled Transition Systems* [269]. The CSP tool FDR [108] allows one to evaluate these trace refinements. The *trace* refinement model does not however, fully show that the implementation process properly implements the specification. To say that all the traces of the implementation are also traces of the specification is not enough. One also needs to show that the implementation has the same failures as the specification (i.e., that it refuses the same events that the specification does). The failures of a process P are formally defined as the set of pairs (s, X), such that the process can follow the sequence of events s (written P/s) and then refuse event X:

$$\mathit{failures}(P) = \{(s, X) \mid s \in traces(P) \wedge X \in \mathit{refusals}(P/s)\}$$

$$P \sqsubseteq_{\mathrm{F}} Q \Leftrightarrow \mathit{failures}(P) \supseteq \mathit{failures}(Q) \wedge \mathit{traces}(P) \supseteq \mathit{traces}(Q)$$

There is one more assertion one can make. A process can *diverge* if it follows a finite trace and then ends up in a state where it can perform an infinite number of *internal events*. As well as the visible events of a process, it may have internal events that are hidden from the external world. In CSP, these events all have the same name τ. They have the same name since they are indistinguishable from each other. A process P is *failures-divergence refined* by Q ($P \sqsubseteq_{\mathrm{FD}} Q$) if the failures and divergences of Q are also failures and divergences of P.

Refinement can be used in two slightly different ways. Firstly, one can define a higher-level model that is refined into a more detailed model. This allows one to show an abstraction away from the details and ensure that the system meets that abstraction. For example, this work defines a device abstractly and then shows that the more complex device lifecycle is a refinement of the simpler concept. Secondly, one can specifically define behaviours that either the model implements or does not implement. The FDR trace refinement analysis can prove that the model either refines or doesn't refine these behaviour specifications. Note that there is a fine line between these two approaches and some of the analyses in this work may straddle that line.

6.4 The Model In CSP

Each of the different participants — Device, Manufacturer, Device Identity Provider, Gateway, Personal Cloud Middleware, User, and Application — are modeled in CSP.

The whole system is modeled as a composition of these components.

In addition to the model presented here, there is a machine readable version of the CSP definition, which was used to prove theorems of the system using FDR. The following resources are considered supplementary resources to this work:

- The machine readable model:
  `https://freo.me/oauthing-model`
- The logs of running the FDR assertions:
  `https://freo.me/oauthing-model-logs`

6.4.1 Assumptions And Boundaries Of The Model

It is assumed that all distributed communications channels are encrypted. This is discussed in more detail in the threat model in Section 7.1. In this model this applies to all the process interactions except the PCM-to-PCM communications (see Definition 6.4.31) where the model treats a PCM as CSP processes that communicate internally. Since this PCM-to-PCM communications is internal, it is not encrypted.

The model does not address discovery of the DIdP, UIdP or IG. These could be addressed as further work. Similarly, there is no modeling of change of ownership of a device, which could also be addressed in further work.

6.4.2 Devices

A *sensor* is a system that can emit an event containing *data* about the world. A sensor can be defined as a process that senses the world and then emits sensor data events. Once it has done this is can once again sense.

Definition 6.4.1. Sensor

$$SENSOR = sense\$d \rightarrow sd!d \rightarrow SENSOR$$

The notation $sense\$d$ indicates that this process may *internally choose* (i.e. non-deterministically) between all values of data. The notation $sd!d$ indicates that the event sd *sends* the value d. Non-deterministic choice correctly models a sensor since the real-world events that prompt the sensor data are hidden from the model, and therefore come "internally" within the sensor.

Similarly, an *actuator* is defined as a system that receives a *command* and then acts upon it.

Definition 6.4.2. Actuator

$$ACTUATOR = rc?c \rightarrow act!c \rightarrow ACTUATOR$$

The notation $command?c$ indicates that this event is receiving the value c. The sensor and actuator have no need to synchronise and can operate without recourse to each other and therefore are *interleaved*. A device is modeled as follows.

Definition 6.4.3. Device

$$DEVICE = SENSOR \;|||\; ACTUATOR$$

From a set theoretic viewpoint, of course there can be multiple sensors and/or actuators per device. The CSP view of the device only defines that it can produce data or consume commands. In this model, each sensor or actuator belongs to exactly one device. This is modeled as a function *device* that maps from sensors and actuators to the device they belong to:

$$\forall\, s \in Sensors, \exists\, d \in Devices : device(s) = d$$

$$\forall\, a \in Actuators, \exists\, d \in Devices : device(a) = d$$

The function $device$ is a surjection:

$$\forall\, d \in Devices, \exists\, x \in Sensors \cup Actuators \;\Big|\; device(x) = d$$

and the domain of the function $device$ is equal to all devices:

$$\text{dom}(device) = Devices$$

In other words, a device must have at least one sensor or actuator.

This is expressed in UML in Figure 6.6.

The model will be built up in stages. As will be seen, this creates a natural lifecycle for a device. This lifecycle is also shown in UML in Figure 6.4 above. At the high level, the device lifecycle is:

$$claim \rightarrow consent \rightarrow connect \rightarrow DEVICE$$

In other words, a device is claimed by a user, who consents to its operating on their

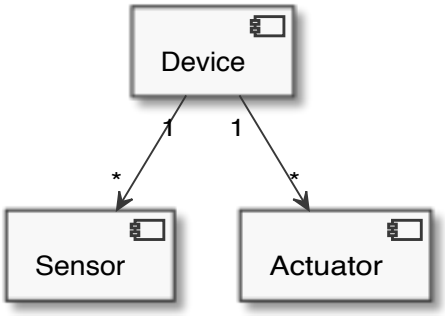

Figure 6.6: UML for a Device

behalf. The device then connects to a data-sharing system and then shares data and accepts commands. In this model, the specifics of the OAuth2 claim, login and consent flows, as well as the data sharing model are deliberately modeled in detail. This refinement is important because the specifics of the device lifecycle, and the usability of the system when devices have no user interface or input mechanism are key requirements of the model and system.

A device is initially a *fresh device* (*FD*). This means that it has just been manufactured and does not have a well-defined identity. This model does not rely on any implicit identity such as a Media Access Control (MAC) address: instead the system injects an identity and credential into the device at manufacturing time. A *Registered Device* (*RD*) is a device that has a credential injected into it during the manufacturing process. Because aspects of this model are closely based on existing OAuth2 work, OAuth2 *ClientIDs* and *ClientSecrets* are explicitly modeled as the credentials. Each device is modeled as a unique OAuth2 Client. As discussed above, this mapping of OAuth2 to the IoT device world was chosen because it is important that every device has a unique credential. This is called out in the requirements in Chapter 2 and also in the IGNITE work above in Chapter 5. This removes a significant security flaw which was present in FIOT, which was identified during the creation of this model.

Definition 6.4.4. Device Registration

$$FD = manufacture \rightarrow updatedevice?cid.cs \rightarrow RD(cid, cs)$$

A registered device must then be "owned". This process consists of a User consenting to allow the device to publish data and/or receive commands on the user's behalf. In this system, the OAuth2 approach is extended to support this. The extension is based on the requirement to support devices with almost no user interface. The only hard requirement on a UI that this system must have is the ability for a user to see if

the device is switched on. When the device is switched on, it notifies the DIdP that it is *present*. The user then initiates a *claim* process for the device. This involves the *ClientID* of the device. For example, in the prototype, the device has a QR code printed onto it, which embeds the ClientID into a URL. When the QR code is scanned, it takes the user to a claim page specific to that device.

The specific OAuth2 flow that is used for this work is the *Authorisation Code* flow. This is a two-part flow whereby the client is first issued an Authorisation Code (AuthCode), which is then "swapped" for a *Refresh Token* and *Bearer Token*. The second step is used to ensure that the client authenticates. As discussed in Chapter 4 this is known as a three-legged flow as the user and the client both authenticate to the DIdP.

Definition 6.4.5. Ownership

$$RD(cid, cs) = presence!cid \rightarrow authcode?ac \rightarrow RD_A(cid, cs, ac)$$

$$RD_A(cid, cs, ac) = token\_ac\_req!cid.cs.ac \rightarrow token\_ac\_resp?r.a \rightarrow OD(cid, cs, r)$$

This produces an *Owned Device*. Every device is either owned or unowned. In other words, there is a function *owns*, such that:

$$\forall\, d \in OwnedDevices, \exists\, u \in Users : owns(d) = u$$

$$\forall\, d \in UnownedDevices, \nexists\, u \in Users : owns(d) = u$$

$$Devices = UnownedDevices \cup OwnedDevices$$

Fresh devices and registered devices are not owned:

$$FreshDevices \subseteq UnownedDevices$$

$$RegisteredDevices \subseteq UnownedDevices$$

This definition of ownership is very high-level. In the model, a user must *claim* a device. In the implementation, claiming is done by being the first person to scan a QR code attached to the device, but it could also be done by an NFC chip or simply typing a URL or code into a browser. The claiming process requires the device to be switched on and connected, and therefore *present*. Once the user initiates the claim, they must *login* and then *consent* to the device acting for them.

As a simplification of the model (and, in fact, the prototype as well), the device only needs to remember the Refresh Token. In the OAuth2 specification, a client has both a refresh token and a bearer token. The bearer token is inherently insecure as anyone

who has a copy of it can act as the client. Therefore, it expires regularly and once it has expired, the client must refresh it, which requires the client to present its credentials to the OAuth2 server.

Definition 6.4.6. Refresh Flow

$$OD(cid, cs, r) = \\ token\_refresh\_req.cid.cs.r \rightarrow \\ token\_refresh\_resp?b \rightarrow SD(b, cid, cs, r)$$

When a device was modeled above, there was nothing that addressed whether a device can publish events without those events being accepted by the Gateway. In the prototype, the MQTT protocol, supports queueing at the Gateway and backpressure on the client. Modeling queueing adds an unnecessary complexity to the model, as well as being specific to a protocol. On the other hand, if there is nothing addressing queueing or backpressure, the model also becomes unmanageable because of state explosion. Therefore, this model uses the simplest approach, which is that the Device must wait for an acknowledgement before publishing further data. This is a simple form of backpressure. This corresponds to protocols like TCP, where the server acknowledges the client messages, etc.

Therefore, the refined definition of a device is:

Definition 6.4.7. Acknowledging Device

$$SENS = sense\$d \rightarrow sd!d \rightarrow dackd \rightarrow SENS$$

$$ACT = rc?c \rightarrow act!c \rightarrow dackc \rightarrow ACT$$

$$DV = SENS \;|||\; ACT$$

Finally a Secure Device is one that connects using the bearer token and then publishes and subscribes:

Definition 6.4.8. Secure Device

$$SD(bearer, cid, cs, r) = connect!bearer \rightarrow connected \rightarrow DV$$

6.4.3 Manufacturer

In order to register a device there needs to be an interface that will create a ClientID and ClientSecret. This API is defined by the OAuth2 Dynamic Client Registration (DCR) API [231]. The details of the actual API are abstracted in the model. The manufacturer indicates that a device has been manufactured and requests a ClientID and ClientSecret which are stored in the device.

The device itself does not connect to the DCR API. Instead, there is a manufacturer that requests the credential and updates the device. The reason for this is that in order to call the DCR API on the DIdP, the requestor needs its own credential. Adding this credential to every device would be a significant security issue. In the model, the trust relationship between the Manufacturer and the DIdP is not explicitly modeled but assumed. In an implementation, this is enabled by a specific token for the manufacturer with scope allowing DCR access.

Definition 6.4.9. Manufacturer

$$MAN = manufacture \rightarrow dcrrequest \rightarrow credential?cid.cs \rightarrow updatedevice!cid.cs \rightarrow \checkmark$$

This is the only process involving the manufacturer.

6.4.4 User Identity Provider

The modelling of federated login into the User Identity Provider (UIdP) is purposely minimal. The only requirement is that the UIdP validates the user and provides a unique identifier for the user. In addition, it needs to be clear in the model that the User's credentials are only shared with the UIdP. In the prototype, multiple approaches for UIdP login are supported, mainly based around OAuth2. As in other aspects of the model, it would be possible to refine this further to cover those alternative approaches. However, unlike the device authentication and authorisation flows, these flows take place in a normal browser and are well understood, so there is no benefit in further refinement of the model in this area.

The login process starts with a request for login (*login*). The user is either successful in which case a user identifier ($lu.u$) is returned, or it fails (*failure*).

Definition 6.4.10. Login

$$LOGIN(u) = login \rightarrow (success \rightarrow lu.u \rightarrow SKIP \,\Box\, failure \rightarrow STOP)$$

This is refined to include a *federated login* (*fedlogin*), which takes a credential (*fc*).

Definition 6.4.11. User Identity Provider

$$UIDP = \\ login \rightarrow fedlogin?fc \rightarrow (success \rightarrow lu.u \rightarrow SKIP \,\Box\, failure \rightarrow STOP)$$

The credential is unimportant. The requirement is that there is a bijective function from federated credentials to users:

$$fu \in Fedcred \times Users \wedge \mathrm{dom}\, fu = Users \wedge fu(c_1) = fu(c_2) \Leftrightarrow c_1 = c_2$$

6.4.5 Device Identity Provider

The *Device Identity Provider* (DIdP) implements all the identity and policy model for the device. It defers user logins to the User Identity Provider (UIdP), using a pattern known as the *Identity Broker* pattern. The DIdP is modeled as a collection of stateful processes that evolve, based on their interactions with existing devices, manufacturers and users. These processes start with the issuance of a credential.

Definition 6.4.12. Device Registration API

$$DCR(cid, cs) = dcrrequest \rightarrow credential.cid.cs \rightarrow UR(cid, cs)$$

Once a credential is issued, the system must support *User Registration*. User registration for a device (*URD*) is as follows:

Definition 6.4.13. User Registration of Devices

$$URD(cid, cs) = presence.cid \rightarrow claim.cid \rightarrow login \rightarrow URD_A(cid, cs)$$

$$URD_A(cid, cs) = \\ success \rightarrow lu?u \rightarrow URD_B(cid, cs, u) \,\Box\, failure \rightarrow error \rightarrow STOP$$

$$URD_B(cid, cs, u) = \\ devconsent \rightarrow authcode\$ac \rightarrow TOKEN\_AC(p(u), cid, cs, ac, PdRc) \\ \Box\, noconsent \rightarrow error \rightarrow STOP$$

Firstly, the device must be present. There must be a claim by a user, which initiates a user login, which either succeeds or fails. If it fails then the process ends with an error. Otherwise, the federated login returns a user identifier. Then there must be user consent for a device. This consent is by definition consent for the device to use scope *PdRc*, which means the device can publish data and receive commands. If the consent is granted, the device receives an AuthCode. At this stage the DIdP is now ready to handle the second half of the authorisation code flow (swapping the AuthCode for the Refresh Token).

The scopes defined in this model are deliberately simple. In the previous approach described in Chapter 5 an extensible model of scopes coded in JSON was proposed. This resulted in complexity for the user. This approach therefore simplifies that considerably:

Definition 6.4.14. Scopes

$$Scope = PdRc \mid RdPc \mid Pd \mid Rc \mid Rd \mid Pc$$

The meanings of these are described in Table 6.2.

Pd	Publish Data
Pc	Publish Commands
Rd	Receive Data
Rc	Receive Commands
PdRc	Publish Data and Receive Commands
RdPc	Read Data and Publish Commands

Table 6.2: Scopes and their Meanings

Note that there are other scopes in the overall system (e.g. a *dcr* scope which allows manufacturers to call the DCR system, and an *introspection* scope that allows a gateway to call the introspection API. These are not formally modeled as they don't affect the core model or the proofs of data sharing.

At this stage, the DIdP applies a *pseudonymisation* function $p(u)$, which replaces the username with a random pseudonym. Note that the definition of pseudonymisation in this work is somewhat different to other privacy-enhancing systems. In many systems, the user may choose to use multiple pseudonyms and these are shared with third-parties to reduce identifiability. In this system, the pseudonym is automatically generated and remains hidden from third-parties. It is used to make the attack tree more complex by requiring an attacker to attack two different systems to identify a user.

Definition 6.4.15. Pseudonymisation

$$\forall\, u \in \mathit{Users}, p(u) \in Pseud$$

$$\forall\, u_1, u_2 \in \mathit{Users},\ p(u_1) = p(u_2) \Leftrightarrow u_1 = u_2$$

Before continuing with the device flow of the DIdP, it is useful to examine the application approval process. A user must also authorise an application to be able to interact with the system. In this case, an application is expected to have the scope *RdPc*, which allows it to receive data and publish commands. A further refinement of the model could support apps that only send commands, or only receive data. Similarly the model can support devices that can only act and not sense. However, this refinement is not required to demonstrate the core properties of the model.

Definition 6.4.16. User Approval of Applications

$$\mathit{URA}(\mathit{cid}, \mathit{cs}) = \mathit{useraccess} \rightarrow \mathit{login} \rightarrow \mathit{URA}_A(cid, cs)$$

$$\mathit{URA}_A(\mathit{cid}, \mathit{cs}) = \mathit{success} \rightarrow \mathit{lu?u} \rightarrow URA_B(cid, cs, u) \\ \Box\ \mathit{failure} \rightarrow \mathit{error} \rightarrow \mathit{STOP}$$

$$\mathit{URA}_B(\mathit{cid}, \mathit{cs}, u) = \\ \mathit{appconsent} \rightarrow authcode\$ac \rightarrow \mathit{TOKEN\_AC}(p(u), \mathit{cid}, \mathit{cs}, \mathit{ac}, \mathit{RdPc}) \\ \Box\ \mathit{noconsent} \rightarrow \mathit{error} \rightarrow \mathit{STOP}$$

This definition is analogous to the device consent flow, except for two differences. Firstly, that instead of a claim and presence event, there is simply an event *useraccess*. This event signifies that a user has requested access to an application. Secondly, the user offers *appconsent* which gives the app the *RdPc* scope.

The overall user approval process is the combination of these two processes:

Definition 6.4.17. User Approval support in the DIdP

$$UR(cid, cs) = \mathit{URD}(\mathit{cid}, \mathit{cs}) \,\Box\, \mathit{URA}(\mathit{cid}, \mathit{cs})$$

This ties back to Definition 6.4.12.

The DIdP supports a Token interface (part of the OAuth2 specification) which supports the AuthCode flow and the Refresh flow.

Definition 6.4.18. Token Response to AuthCode

$$TOKEN\_AC(p, cid, cs, ac, scope) = \\ token\_ac\_req.cid.cs.ac \rightarrow token\_ac\_resp\$r\$a \\ \rightarrow TOKEN\_REF(p, cid, cs, r, scope)$$

The refresh flow allows a client to send a valid refresh token, together with the client's credentials, and receive a fresh, active bearer token. Once this is done, the DIdP can then support *introspection* of the bearer token.

Definition 6.4.19. Token Refresh Flow at the DIdP

$$TOKEN\_REF(p, cid, cs, r, s) = \\ token\_refresh\_req.cid.cs.r \rightarrow \\ token\_refresh\_resp\$bearer \rightarrow INTRO(bearer, p, s)$$

This introspection is defined by the OAuth2 Introspection API [220]. Effectively, the introspection looks at a bearer token and returns the validity, user and scope of the token. In the model, introspection only returns the pseudonym instead of the actual user information.

Definition 6.4.20. Introspection

$$INTRO(b, p, s) = \\ introspect.b \rightarrow valid!p!s \rightarrow INTRO(b, p, s) \\ \Box\ introspect?x : \left\{ x \;\middle|\; x \in Bearer, x \neq b \right\} \rightarrow invalid \rightarrow INTRO(b, p, s)$$

This definition says that the specific process that has previously initialised the client with a bearer b, pseudonym p, and scope s will respond to events "querying" a bearer token. If the bearer token queries matches, the scope and pseudonym will be returned, otherwise it will return *invalid*.

Note that a further refinement of this is possible using *Timed CSP* [68] which allows one to model time. This refinement would include the timing-out of bearer tokens, forcing a refresh. While this would be a nice enhancement to the model, it would not aid in proving any of the fundamental properties of the model and therefore it is left for further work.

The DIdP that supports a given credential is now defined, and this can be generalised:

Definition 6.4.21. Device Identity Provider

$$DIDP = \Box_{cid \in ClientID, cs \in ClientSecret} DCR(cid, cs)$$

6.4.6 Third Party Application

The DIdP and the Device, are now defined, so it is a good time to look at the system that interacts at the other end. This is called a *Third Party Application* (TPA) or simply an *App*. The reason that this is called a *third-party* application is that the model enforces that no system is inherently trusted to look at device data. Therefore, every app that wishes to access data from a device or send commands to a device must register with the DIdP and gain consent, just as a third-party would in any OAuth2 flow.

The *data consumer* (DC) is a component that receives data (rd), and then logs that data.

Definition 6.4.22. Data Consumer

$$DC = rd?d \rightarrow logdata.d \rightarrow DC$$

A *command publisher* first *demands* an action in the form of a command, and then *sends* that command (sc):

Definition 6.4.23. Command Publisher

$$CP = demand\$c \rightarrow sc!c \rightarrow CP$$

An application can support data consumption and command publication concurrently:

Definition 6.4.24. Application

$$APP = DC \;|||\; CP$$

Once again there is a refinement of this that supports acknowledgements.

Definition 6.4.25. Acknowledging Application

$$DC_{ack} = rd?d \rightarrow logdata!d \rightarrow aackd \rightarrow DC_{ack}$$

$$CP_{ack} = demand\$c \rightarrow sc!c \rightarrow aackc \rightarrow CP_{ack}$$

$$APP_{ack} = DC_{ack} \;|||\; CP_{ack}$$

The logic of an application is very similar to the logic of a device. Given the similarity, the logic can be presented more concisely before discussing the differences.

Definition 6.4.26. Application

$$APPCREATE = dcrrequest \rightarrow credential?cid?cs \rightarrow RA(cid, cs)$$

$$RA(cid, cs) = useraccess \rightarrow authcode?ac \rightarrow RA_A(cid, cs, ac)$$

$$RA_A(cid, cs, ac) = token\_ac\_req!cid.cs.ac \rightarrow token\_ac\_resp?r.b \rightarrow SA(cid, cs, r)$$

$$SA(cid, cs, r) = token\_refresh\_req!cid.cs.r \rightarrow token\_refresh\_resp?bearer \rightarrow TA(bearer, cid, cs, r)$$

$$TA(b, cid, cs, r) = connect!b \rightarrow connected \rightarrow APP_{ack}$$

The difference is minor: the application needs a user to request access (*useraccess*). Other than that, the consent flow is analogous as far as the app is concerned. However, the user sees a different flow, explored next.

6.4.7 User

The only roles that the users play in this (which are key roles admittedly) are:

- to login,
- to claim devices,
- to request access to apps, and
- to provide consent to the devices and applications.

The user participates in two flows, consenting to devices and applications.

The user can claim a device. In this, the User u specifically claims a device with identity cid. The user must login successfully and consent to the device, or the login may fail, or the user may not agree to the scope of sharing requested.

Definition 6.4.27. User Registration of a Device

$$\begin{aligned} USERCLAIM(fc, cid) = claim.cid \rightarrow login \rightarrow \\ (fedlogin!fc \rightarrow (success \rightarrow lu!u \rightarrow \\ (devconsent \rightarrow SKIP \,\Box\, noconsent \rightarrow STOP) \\ \Box\, failure \rightarrow STOP)) \end{aligned}$$

A given user u may potentially claim any device:

$$UC(fc) = \Box_{cid \in ClientID}\, USERCLAIM(fc, cid)$$

The second user flow is similar, where the user consents to an application seeing their data and sending commands to their devices.

Definition 6.4.28. User Approval of an Application

$$\begin{aligned} USERAPPROVE(fc) = useraccess \rightarrow login \rightarrow \\ (fedlogin!fc \rightarrow (success \rightarrow lu!u \rightarrow \\ (appconsent \rightarrow SKIP \,\Box\, noconsent \rightarrow STOP)) \\ \Box\, failure \rightarrow STOP) \end{aligned}$$

A user is simply the combination of these two processes:

Definition 6.4.29. User

$$USER(fc) = UC(fc) \;|||\; USERAPPROVE(fc)$$

An important aspect of a user's capability is the opportunity to revoke tokens and remove access. This applies to both devices and applications. A device that has a token revoked needs to be informed of token revocation by the DIdP. This is managed by an appropriate return code from the DIdP to the device on the *TOKEN_REF* interaction. This then changes the device from the *OD* stage back to the *RD* stage where it can then be claimed by the same user or another user.

Similarly an App that has been revoked will lose all access and will need to be re-authorised by a user.

Currently there is no modeling of token revocation. This has been identified as an area for further work, but is not expected to provide major insights as this is a well understood area.

6.4.8 Intelligent Gateway (IG)

The role of the IG is to validate the bearer token of either the device or the app using the introspection defined in 6.4.20. This returns a pseudonym and a scope. The IG should not share this pseudonym, and the DIdP permissions do not allow other parties to access introspection. Once the IG has a valid response, it passes the message to the Personal Cloud Middleware (PCM), which implements the scope sharing.

Definition 6.4.30. Gateway

$$IG = \mathit{connect}?b \rightarrow \mathit{introspect}!b \rightarrow (\mathit{valid}?p?s \rightarrow \mathit{connected} \rightarrow \mathit{PCM}(p,s) \,\Box\, \mathit{invalid} \rightarrow \mathit{error} \rightarrow \mathit{STOP})$$

6.4.9 Personal Cloud Middleware (PCM)

Each user has their own instance of PCM. If u is a user, then $\mathit{PCM}(u)$ is the user's PCM.

$$\forall\, u \in \mathit{Users}\, \exists\, p \in \mathit{PCM}, p = \mathit{PCM}(u)$$

The PCM is modeled in two halves: a sending component and a receiving component. This way, it can be ensured that a device that has permission to publish data is allowed to publish and an app that is allowed to recieve data can correctly subscribe. In order to model the PCM properly, once again there is need to support simple acknowledgements to prevent state explosion in the model. The acknowledgements are in two parts. Firstly, the PCM acknowledges that it has received or sent messages to the device or app (these acknowledgements were seen in Definition 6.4.7). The second type of acknowledgement ($\mathit{pcmacksd}, \mathit{pcmackrd}, \mathit{pcmacksc}, \mathit{pcmackrc}$) allows the two halves of the PCM to acknowledge delivery. For example, if the device is correctly connected to the PCM, but there is no authorised application, the PCM will not acknowledge that messages have been delivered.

The connection between the two halves is modeled using the linked parallel construct described above. For example, the "device half" of the PCM will $\mathit{senddata}.p!d$ which will be renamed to be received by the "app half" of the PCM as $\mathit{recdata}.p?d$. Notice that these messages are synchronised on the pseudonym. This models the privacy of the PCM: each user's PCM can only communicate with its other half. The PCM is defined using pattern matching on the different scopes:

Definition 6.4.31. Personal Cloud Middleware

The “device half” is defined:

$$PCM(p, Pd) = sd?d \rightarrow senddata.p!d \rightarrow pcmacksd \rightarrow dackd \rightarrow PCM(p, Pd)$$

$$PCM(p, Rc) = reccom.p?c \rightarrow rc!c \rightarrow pcmackrc \rightarrow dackc \rightarrow PCM(p, Rc)$$

$$PCM(p, PdRc) = PCM(p, Pd) \;|||\; PCM(p, Rc)$$

The “app half” is defined:

$$PCM(p, Pc) = sc?c \rightarrow sendcom.p!c \rightarrow pcmacksc \rightarrow aackc \rightarrow PCM(p, Pc)$$

$$PCM(p, Rd) = recdata.p?d \rightarrow rd!d \rightarrow pcmackrd \rightarrow aackd \rightarrow PCM(p, Rd)$$

$$PCM(p, RdPc) = PCM(p, Rd) \;|||\; PCM(p, Pc)$$

One of the concepts that is important in the PCM is that of summarisation and filtering. These were not modeled in the PCM for the following reasons. One of the aims of the CSP model is to prove specific properties of the model, such as end-to-end data flow. In the case where there is summarisation and filtering these proofs will become intractable. In addition, the complexity of the model increases making it much harder to analyse using FDR.

A summarisation or filtering is a function defined on a stream of data or commands:

$$stream =< d_1, d_2, d_3, d_4, \ldots d_n \ldots >$$

$$f(stream) =< d'_1, d'_2, \ldots d'_m \ldots >$$

where the number of output data elements can be the same, fewer or even more than the number of input data points.

Summarisation and filtering are — as identified in the literature review — key technologies to fight against fingerprinting. In addition, during the creation of this model I identified the possibility of filtering commands, which did not emerge in the literature review.

Filtering commands may initially seem counter-intuitive: if a device user wishes to turn off a light, that user does not want the light turned off “on average”. However, take the example of a connected car. Command filtering could be seen as an example of an application-level firewall for device commands. For example, the filter may allow commands to remotely switch on the engine when the device is parked, but disallow any events that effect the speed or direction of the car. A more complex filtering rule might allow a command to switch off the engine when the speed is zero and the parking brake is applied, but filter out any other attempts to switch off the engine.

While there are examples of specific firewalls, e.g, for cars, in the literature, there was no evidence of a more general filtering and summarising approach to commands.

That completes the definition of the individual components of the architecture. The next step is to combine the components into a full architecture and to demonstrate specific properties of the system.

6.4.10 Event Sharing Across The Overall System

Here is modeled a system in which one device is connected to one application. In this modelling, the credentials of the users who authorise each of these components are parameterised to allow testing with different users. This section builds up the composition of individual components and analyses the events that are shared between those components.

The first part is to connect a device to the manufacturer, DIdP, user and gateway.
Definition 6.4.32. System with Connected Device

Connecting the manufacturer and the DIdP:

$$DSYS_1 = MAN \;{}_{me}\|_{didpe}\; DIDP$$

where me is the set of events that the manufacturer interacts on, and $didpe$ is the set of events that the DIdP interacts on.

$$me = \{manufacture, dcrrequest, \\ credential.cid.cs, updatedevice.cid.cs \;\Big|\; cid \in ClientID, cs \in ClientSecret\}$$

The set of events that the DIdP interacts with is quite extensive. Firstly, the events between the device and the DIdP:

$$didp2d = \{presence.cid, authcode.ac, \\ token\_ac\_req.cid.cs.ac, token\_ac\_resp.r.b, \\ token\_refresh\_req.cid.cs.r, token\_refresh\_resp.b \;\Big|\; \\ cid \in ClientID, cs \in ClientSecret, ac \in AuthCode, \\ r \in Refresh, b \in Bearer\}$$

The events between the manufacturer and the DIdP:

$$didp2m = \{dcrrequest, credential.cid.cs \mid cid \in ClientID, cs \in ClientSecret\}$$

The events between the DIdP and the application creator are the same:

$$didp2ac = \{dcrrequest, credential.cid.cs \mid cid \in ClientID, cs \in ClientSecret\}$$

The events between the DIdP and the Gateway:

$$didp2gw = \{introspect.b, valid.p.s, invalid \mid b \in Bearer, p \in Pseud, s \in Scope\}$$

The events between the DIdP and the User:

$$didp2u = \{lu.u, claim.cid, useraccess, devconsent, appconsent, noconsent \\ login, success, failure \mid cid \in ClientID, u \in User\}$$

The error event:

$$didperr = \{error\}$$

All DIdP events:

$$didpe = didp2d \cup didp2m \cup didp2ac \cup didp2gw \cup didp2u \cup didperr$$

As discussed in the introduction to CSP above, the *alphabetised parallel* operator $({}_A\|_B)$ has the following rule:

Law 6.4.1. Alphabetised Parallel

$$P \ {}_A\|_B \ Q = (P \underset{\Sigma \setminus A}{\|} STOP) \underset{A \cap B}{\|} (Q \underset{\Sigma \setminus B}{\|} STOP)$$

where Σ is the set of all events in the system.

In other words, when the alphabetised parallel operator $P \ {}_A\|_B \ Q$ is used, all possible events that communicate between P and Q can be defined as the intersection $A \cap B$.

One of the key aspects of this system is where there are multiple parties conjoined using alphabetised parallel. Alphabetised parallel has both symmetry and associative laws.

Law 6.4.2. Alphabetised Parallel Symmetry

$$P \ {}_A\|_B \ Q = Q \ {}_B\|_A \ P$$

Law 6.4.3. Alphabetised Parallel Associativity

$$(P \ {}_A\|_B \ Q) \ {}_{A \cup B}\|_C \ R = P \ {}_A\|_{B \cup C} \ (Q \ {}_B\|_C \ R)$$

This gives the following theorem:

Theorem 6.4.1. Manufacturer to DIdP Events As this is the only place where the DIdP interacts with the manufacturer, it can be asserted that the Manufacturer only sees the events $didpe \cap me$:

$$\{dcrrequest, credential.cid.cs \mid cid \in ClientID, cs \in ClientSecret\}$$

The proof is an obvious instantiation of Law 6.4.1. In addition, this can be derived using the probe capabilities of FDR that allow this combined process to be explored.

Of course the manufacturer also interacts with the device, but in this case the device is simply receiving data that the manufacturer provides to the device. One thing to note is that the manufacturer can retain the ClientID and ClientSecret. A possible improvement to the model would be to enable a flow whereby the device directly contacts the DIdP to update the ClientSecret, which would ensure that the manufacturer is not in possession of the device credentials. This increases security at the cost of making it harder for the manufacturer to provide support. This is left for further work.

The next stage of constructing the system is to conjoin this with a fresh device:

$$DSYS_2 = FD \ {}_{deve}\|_{didpe \cup me} \ DSYS_1 \setminus me$$

where $deve$ is the set of events that the device communicates on. Note that the overall set of events that a device interacts on are specified here individually, but later the actual events that two parties interact on is formally derived using laws of CSP and the use of FDR.

The device to manufacturer events:

$$dme = \{manufacture, updatedevice.cid.cs \mid cid \in ClientID, cs \in ClientSecret\}$$

The device to DIdP events:

$$\begin{aligned} dde = \{&presence.cid, authcode.ac, \\ &token\_ac\_req.cid.cs.ac, token\_ac\_resp.r.b, \\ &token\_refresh\_req.cid.cs.r, token\_refresh\_resp.b \mid \\ &cid \in ClientID, cs \in ClientSecret, ac \in AuthCode, \\ &r \in Refresh, b \in Bearer\} \end{aligned}$$

Note that $dde = didp2d$.

The device to gateway events:

$$\begin{aligned} dge = \{&connect.b, connected, sd.d, rc.c, dackd, dackc \\ &\mid b \in Bearer, d \in Data, c \in Command\} \end{aligned}$$

The devices own events:

$$hde = \{act.c, sense.d \mid d \in Data, c \in Command\}$$

All device events are the union of these:

$$deve = dme \cup dde \cup dge \cup hde$$

The device is conjoined with both the DIdP and the Manufacturer. Once this happens, the the manufacturer's events are no longer visible, and therefore are hidden.

Theorem 6.4.2. Device to Manufacturer Events

The device to manufacturer events are $deve \cap me$. The intersection is calculated by FDR.

$$\{manufacture, updatedevice.cid.cs \mid cid \in ClientID, cs \in ClientSecret\}$$

This can be proven using FDR or by applying the alphabetised parallel rule twice.

Theorem 6.4.3. Device to DIdP Events The device to DIdP events are $didpe \cap deve$.

The intersection is calculated by FDR.

$$\{presence.cid, authcode.ac, \\ token\_ac\_req.cid.cs.ac, token\_ac\_resp.r.b, \\ token\_refresh\_req.cid.cs.r, token\_refresh\_resp.b \mid \\ cid \in ClientID, cs \in ClientSecret, ac \in AuthCode, \\ r \in Refresh, b \in Bearer\}$$

In order to authorise the system, the user must connect to the UIdP. Firstly, the events that the UIdP interacts on:

$$fedevents = \{fedlogin.fc, login, lu.u, success, failure \mid u \in User, fc \in Fedcred\}$$

The set of events that the federated user interacts on are:

$$fue = \{claim.cid, useraccess, fedlogin.fc, lu.u \\ login, success, failure, \\ appconsent, devconsent, noconsent \\ \mid cid \in ClientID, fc \in Fedcred, u \in User\}$$

Once the federated user is combined with the UIdP, this communicates with the DIdP on a set of events due:

$$due = \{claim.cid, useraccess, lu.u, login, success, failure \\ appconsent, devconsent, noconsent \\ \mid cid \in ClientID, u \in User\}$$

A user claiming a device:

$$FUSER(fc) = UC(fc) \ {}_{fue}\|_{fedevents} \ UIDP \upharpoonright due$$

Theorem 6.4.4. User to UIdP Events By application of the alphabetised parallel law[2], it can be ascertained that the user to UIdP events are $fue \cap fedevents$. These are calculated by FDR.

$$\{login, lu.u, success, failure, fedlogin.fc \mid u \in User, fc \in FedCred\}$$

[2]Note that the user is also present in the app approval process below. The law is applied to both conditions, but the theorem is presented here for reasons of explication.

Now this user can be conjoined with the existing system, hiding the $login$ and lu events from the rest of the system. $DSYS3(fc)$ indicates the user with credentials fc approving a fresh device (with the associated DIdP, UIdP and manufacturer).

$$DSYS_3(fc) = DSYS_2 \ {}_{didpe \cup deve}\|_{due} \ FUSER(fc) \setminus \{login, lu.u \mid u \in User\}$$

Theorem 6.4.5. UIdP to DIdP Events Using the Associative Law 6.4.3 it can be show that the DIdP to UIdP events are $(didpe \cup deve) \cap fedevents$, which (using FDR) evaluates to:

$$\left\{login, lu.u, success, failure \mid u \in User\right\}$$

Theorem 6.4.6. User to DIdP Events The user to DIdP events are $(didpe \cup deve) \cap ue$. These are calculated by FDR.

$$\{devconsent, appconsent, noconsent, login, lu.u, success, failure, claim.cid, useraccess \mid u \in User, cid \in ClientID\}$$

Definition 6.4.33. Device System

This is now joined with the Gateway (IG):

$$DSYS(fc) = IG \ {}_{gwe \cup pcme}\|_{didpe \cup gwe \cup hde} \ DSYS_3(fc)$$

where gwe are the events that the gateway interacts on, and $pcme$ are the events that the two halves of the PCM interact on:

$$gwe = \{connect.b, connected, introspect.b, valid.p.s, invalid, dackd, dackc, aackd, aackc, rd.d, sd.d, rc.c, sc.c \mid b \in Bearer, p \in Pseud, d \in Data, c \in Command, s \in Scope\}$$

$$pcme = \{senddata.p.d, sendcom.p.c, recdata.p.d, reccom.p.c, pcmackrd, pcmacksd, pcmacksc, pcmackrc \mid d \in Data, c \in Command, p \in Pseud\}$$

This process ($DSYS(fc)$) indicates a user with credential fc approving a device that is now fully connected to the rest of the system (IG, DIdP, UIdP, Manufacturer and User).

Theorem 6.4.7. Device to Gateway Events

Once again the associative law can be used to calculate all events that the device can communicate with the gateway on, which is $(gwe \cup pcme) \cap deve$:

$$\{sd.d, rc.c, connect.b, connected, dackd, dackc \mid d \in Data, c \in Command, b \in Bearer\}$$

Theorem 6.4.8. Gateway to DIdP Events The events shared between the DIdP and the IG are derived to be $(gwe \cup pcme) \cap (didpe \cup hde)$, which evaluates to:

$$\{introspect.b, valid.p.s, invalid \mid b \in Bearer, p \in Pseud, s \in Scope\}$$

Theorem 6.4.9. The Gateway Shares no Events with the UIdP, the User and the Manufacturer

By similar calculations of the alphabetised parallel laws, it is shown that the set of events shared between the gateway and the manufacturer, user, and UIdP are all empty.

The creation of the App half of the system is analagous, and therefore is presented without discussion.

Definition 6.4.34. Application Connected to the DIdP, UIdP, User and IG

$$FUA(fc) = USERAPPROVE(fc) \ {}_{fue}\|_{fedevents} \ UIDP \upharpoonright due$$

All the events an app can participate in are defined as ae, and the set of events that the DIdP communicates with apps as $appdidpe$. The user to App events are uae.

$$ASYS_1 = DIDP \ {}_{appdidpe}\|_{ae} \ APPCREATE$$

$$ASYS_2(fc) = (ASYS_1 \ {}_{didpe \cup ae}\|_{uae} \ FUA(fc)) \setminus \{login, lu.u \mid u \in User\}$$

$$ASYS(fc) = IG \ {}_{gwe \cup pcme}\|_{appdidpe \cup gwe \cup hae} \ ASYS_2(fc)$$

The process $ASYS(fc)$ indicates an app that has been approved by user with credential fc that is connected to the DIdP, UIdP, User, and IG.

6.4.11 The Complete System

Finally, these two systems (*DSYS* and *ASYS*) are connected together. They synchronise only at the PCM using the PCM events. This is modeled using the linked parallel

approach described above. The parameterisation of the users' credentials is retained so that the system can be tested with different users (with one credential being used to authorise the device and the other to authorise the app).

Definition 6.4.35. The Complete System

$$SYS(fc_1, fc_2) = \left(DSYS(fc_1) \left[\begin{array}{c} reccom \leftrightarrow sendcom \\ pcmacksd \leftrightarrow pcmackrd \\ pcmackrc \leftrightarrow pcmacksc \\ senddata \leftrightarrow recdata \end{array} \right] ASYS(fc_2) \right) \Theta_{\{error\}} STOP$$

Theorem 6.4.10. Application Events The same logic as above can be applied to calculate the events that the App shares with different components. The proofs can be derived by the use of the Associative Law, or more effectively through probing the model with FDR.

These are summarised in Table 6.3.

User	$\emptyset$
UIdP	$\emptyset$
Device	$\emptyset$
Manufacturer	$\emptyset$
DIdP	$\{presence.cid, authcode.ac, token\_ac\_req.cid.cs.ac, token\_ac\_resp.r.b, token\_refresh\_req.cid.cs.r, token\_refresh\_resp.b \mid cid \in ClientID, cs \in ClientSecret, ac \in AuthCode, r \in Refresh, b \in Bearer\}$
Gateway	$\{rd.d, sc.c, connect.b, connected, aackd, aackc \mid d \in Data, c \in Command, b \in Bearer\}$

Table 6.3: Events Shared Between the App and Other Components

6.5 Properties Of The System

Two sets of properties of the system are derived from the model: end-to-end properties, and data sharing properties.

6.5.1 End-To-End Analysis

The definition of the complete system allows reasoning about the whole system and not just the components. To do this, specifications can be defined in CSP that are either expected to pass or fail.

Theorem 6.5.1. Consent Cannot Follow Failed Login In order to demonstrate that consent must follow successful login, an "anti-specification" can be disproved:

$$LSPEC = failure \rightarrow appconsent \rightarrow STOP \,\Box\, failure \rightarrow devconsent \rightarrow STOP$$

The specification suggests that the model will support *appconsent* or *devconsent* after a failure. This must be disproven. The specification will only be evaluated on these events:

$$lspecevents = \{error, appconsent, devconsent, success, failure\}$$

Because this is an anti-specification, disproving means proving that the traces of this are not a subset of the traces of the system.

Therefore FDR is used to evaluate whether or not:

$$SYS(FC.0, FC.0) \upharpoonright lspecevents \sqsubseteq_{\mathrm{T}} LSPEC$$

where *FC.0* is a valid credential in the system.

FDR shows that this is not true with the following trace of events:

```
Counterexample (Trace Counterexample)
            Specification Debug:
                Trace: <failure>
                Available Events: {error, success}
            Implementation Debug:
                LSPEC (Trace Behaviour):
                    Trace: <failure>
                    Error Event: devconsent
```

In other words, the system does not accept *devconsent* after *failure*.

A positive specification is that when the system is properly consented on both halves, by the same user, then data that is sensed in the device will be logged in the app, and commands that are demanded in the app, will be acted on in the device.
Definition 6.5.1. Normal System Specification (NS)

This outlines the possibilities for consent. Either there are both consents (in either order), or consent on one side followed by an error, or an error. Internal choice properly identifies the specification because it does not consider whether the consent failed because of failed login, or the user refused to grant consent, and therefore there are

non-deterministic choices of how these event patterns emerge.

$$\begin{aligned} NSSUCC = appconsent \rightarrow devconsent \rightarrow NS2 \sqcap \\ devconsent \rightarrow appconsent \rightarrow NS2 \sqcap \\ error \rightarrow STOP \sqcap \\ appconsent \rightarrow error \rightarrow STOP \sqcap \\ devconsent \rightarrow error \rightarrow STOP \end{aligned}$$

A successful operation is defined as:

$$NS2 = sense\$d \rightarrow logdata.d \rightarrow NS2 \,\Box\, demand\$c \rightarrow act.c \rightarrow NS2$$

This specification only considers the following events:

$$specevents = \{error, appconsent, devconsent, logdata.d, act.c \mid d \in Data, c \in Command\}$$

The overall specification is then:

$$NS = NSSUCC \,\Theta_{\{error\}}\, STOP \upharpoonright specevents$$

The system is now evaluated using FDR's refinement checker.

Theorem 6.5.2. The System Meets the Specification if the Same User Authorises Device and App

$$\begin{aligned} NS \sqsubseteq_{\mathrm{T}} SYS(FC.0, FC.0) \upharpoonright specevents \wedge \\ NS \sqsubseteq_{\mathrm{F}} SYS(FC.0, FC.0) \upharpoonright specevents \wedge \\ NS \sqsubseteq_{\mathrm{FD}} SYS(FC.0, FC.0) \upharpoonright specevents \end{aligned}$$

This is proved by FDR:

```
NS [T= SYS(FC.0, FC.0) |\ specevents:
    Log:
        Result: Passed
        Visited States: 1,461,364
        Visited Transitions: 7,440,420
        Visited Plys: 68
        Estimated Total Storage: 268MB
```

```
NS [F= SYS(FC.0, FC.0) |\ specevents:
    Log:
        Result: Passed
        Visited States: 1,461,364
        Visited Transitions: 7,440,420
        Visited Plys: 68
        Estimated Total Storage: 268MB
NS [FD= SYS(FC.0, FC.0) |\ specevents:
    Log:
        Result: Passed
        Visited States: 1,461,364
        Visited Transitions: 7,440,420
        Visited Plys: 68
        Estimated Total Storage: 268MB
```

Theorem 6.5.3. The System Fails the Specification if Different Users Authorise Device and App

If user with credential *FC.0* authorises the device and user with credential *FC.1* authorises the app then there are no data or commands transferred:

$$SYS(FC.0, FC.1) \upharpoonright \mathit{specevents} \sqsubseteq_{\mathrm{T}} NS$$

In other words, this tests whether the correct traces are a subset of the traces of the incorrect system, and it is desired that this is not the case. This is once again proved by FDR, which postulates the following counter-example:

```
SYS(FC.0, FC.1) |\ specevents [T= NS:
    Log:
        Result: Failed
        Visited States: 27
        Visited Transitions: 42
        Visited Plys: 8
        Estimated Total Storage: 0B
        Counterexample (Trace Counterexample)
            Specification Debug:
                Trace: <appconsent, devconsent>
                Available Events: {}
            Implementation Debug:
                NS (Trace Behaviour):
                    Trace: < ,  ,  ,  , appconsent, devconsent,  , >
                    Error Event: logdata.D.0
```

This shows that even though *appconsent* and *devconsent* occur, no data is transferred as the *logdata* event cannot occur in the system where different users consent.

6.5.2 Data Flow Between Components

In the analysis of the model above, the messages that flow between systems are identified. Those findings are tabulated in Table 6.4, which captures all the data elements that are transferred in messages between each component. For example, in Theorem 6.4.8 it was identified that the events passed are:

$$\{introspect.b, valid.p.s, invalid \mid b \in Bearer, p \in Pseud, s \in Scope\}$$

From this, it can be identified that the data passed is a bearer token, pseudonym and scope.

	Man	**Dev**	**User**	**UIdP**	**DIdP**	**GW/PCM**	**App**
Man	—	{*ClientID*, *ClientSec*}	∅	∅	{*ClientID*, *ClientSec*}	∅	∅
Device	{*ClientID*, *ClientSec*}	—	∅	∅	{*ClientID*, *ClientSec*, *AuthCode*, *Bearer*}	{*Bearer*, *Data*, *Command*}	∅
User	∅	∅	—	{*FedCred*, *User*}	{*User*, *ClientID*}	∅	∅
UIdP	∅	∅	{*FedCred*, *User*}	—	{*User*}	∅	∅
DIdP	{*ClientID*, *ClientSec*}	{*ClientID*, *ClientSec*, *AuthCode*, *Bearer*}	{*User*, *ClientID*}	{*User*}	—	{*Bearer*, *Pseud*, *Scope*}	{*ClientID*, *ClientSec*, *AuthCode*, *Bearer*}
GW/PCM	∅	{*Bearer*, *Data*, *Command*}	∅	∅	{*Bearer*, *Pseud*, *Scope*}	—	{*Bearer*, *Data*, *Command*}
App	∅	∅	∅	∅	{*ClientID*, *ClientSec*, *AuthCode*, *Bearer*}	{*Bearer*, *Data*, *Command*}	—

Table 6.4: Component Data Sharing Matrix

This data sharing matrix captures a key property of the system. The distribution and sharing of data between components can be used to analyse the security and privacy properties of the system. This is done by threat modeling. Threat modeling allows the analysis of different attacks on the system. This matrix is directly used as the input to the threat modeling exercise in Chapter 7, where it is used to demonstrate enhancements to security and privacy.

6.6 Conclusions Of The Formal Modelling

This concludes the formal analysis of the system using CSP and FDR. In summary, the formal model of the system has shown that the system cannot transfer messages unless:

- A user has successfully logged in to consent
- The user has granted consent for applications to read data and publish commands
- The user has granted consent for devices to publish data and read commands
- The same user has granted consent for both the application and device

In addition, the model was used to derive the set of messages with which each component interacts with the other components. These findings form the basis of the threat modelling, which comes next.

In this chapter, both an informal and formal description of the model have been outlined. Using the formal model specific properties of the system have been proved, including data sharing properties and end-to-end properties. There is a clear proof that data is shared between devices and apps if and only if the same user has consented to both systems being authorised to do so. This analysis will be used in the next chapter to produce a threat model.

Chapter 7

Threat Modelling

This chapter proceeds as follows. In Section 7.1, the overall threat modeling approach is discussed; the proven properties from Chapter 6 are evaluated to create an threat model; and the assets of the system as identified in the model are evaluated. In Section 7.2, specific threats against security are evaluated. In Section 7.3, privacy properties are evaluated to understand threats against privacy. In Section 7.4 those properties and the threat model are compared to the requirements identified in Chapter 2, and then the overall results of threat modeling are discussed.

7.1 Threat Modeling

Threat models have emerged as a key approach to ensure the security of systems [259]. The approach taken to threat modeling is broadly based on [183], with further input from [246] and [71]. In these papers there is a high-level three step process:

- characterising the system,
- identifying assets and access points, and
- identifying threats.

The first two steps are the same for both privacy and security. For the third step — identifying the threats — two different sets of threats will be proposed, treating security and privacy separately.

In [246], a set of threats based on security properties is proposed, named *STRIDE*. In [71] a set of threats based on privacy properties is proposed, named *LINDDUN*. The LINDDUN properties were discussed in Section 2.20.1. The STRIDE properties are outlined in Section 7.2.

The usual approach in threat modeling is to use *Data Flow Diagrams* [152] to characterise the flows in a system. In this work, there is already a detailed model providing a characterisation of the system. In particular, there is a full characterisation of all the data flows throughout the network, which provides a strong basis for threat modeling. The fundamental input to the threat modeling is the *Data Sharing Matrix* presented in Table 6.4 in Chapter 6. This will be used instead of data flow diagrams.

As a consequence of this approach, it is clear that this threat modeling is not aimed to be comprehensive. A full deployment of this architecture would require further threat modeling to assess the risks inherent in such a deployment. Instead, this threat modeling aims to address the inherent risks of this model, as it is modeled in Chapter 6, and using the output of that model as the basis.

7.1.1 Assets And Access Points

The assets can also be extracted from the model, and these are summarised in Table 7.1. The access points of the system are well characterised by the components and associated process flows identified in the CSP model.

User information	Federated credential, User identifier, Pseudonym
Device Information	*IP address*, *Sensor fingerprints*, Bearer Token, Refresh Token, ClientID, ClientSecret
Data	Data Command
Temporary identifiers	AuthCode

Table 7.1: Assets of the System

Note that *IP Address* and *Sensor fingerprints* are not part of the formal model. One of the major concerns identified in the literature review is the possibilities of fingerprinting devices based on metadata or sensor anomalies. Therefore these are considered in the threat model.

It is assumed that the User Identity Provider is secure, as it is out of the control of this model. In addition, one of the key benefits of the architecture is *choice*: users can select a more secure UIdP over a less secure one. The federated credentials cannot be stolen except in cases where the UIdP is compromised as they are only passed between User and UIdP. Note that attacks where the user is persuaded to choose an insecure UIdP are ignored, as once again this is out of the control of the system.

Before assessing potential attacks and access points, let us address the potential value of each of the assets to an attacker.

- **Federated Credential**: as discussed, this is not considered this to be accessible as part of this analysis.
- **User Identifier**: in the system, the user identifier is an id sent back from the UIdP. Typically in federated web systems, this is an internal identifier used in the UIdP's system. For example, in Github, the login flow returns the userid (e.g., `pzfreo`) as well as a unique number (e.g., `2341892233`). In some cases, the UIdP returns more information than the DIdP needs due to the OAuth2 login models of some systems, and in this case the DIdP must discard this data immediately. Obviously one of the aims of the system is to protect metadata and therefore any attack that identifies users of the system may harm privacy. However, leakage of the userid itself does not, in most cases, inherently harm the user, unless this is also tied to devices, apps, data or pseudonyms. However, ee can imagine a system that was very restricted (e.g., if this model was applied to diabetic disease management devices), where the fact of using the system inherently ties the user to a specific type of device and or disease.
- **Pseudonym**: The pseudonym is a randomly generated identifier for a user. If the attacker only knows the pseudonym then there is little benefit. However, if the pseudonym can be tied to data and devices that increases the risk of privacy infringement. If the pseudonym is also tied to userid then the attacker has full visibility.
- **IP address**: If an attacker identifies the IP address of a device they can find an approximate geo-location of the device, as well as identifying the ISP. In addition, it is possible that they can fingerprint the IP address to a given user through other shared devices using the same network.
- **Sensor fingerprint**: If an attacker can collect enough raw data from multiple devices, they can fingerprint those devices and potentially tie them to users.
- **Bearer Token**: The bearer token is a powerful token in the OAuth2 flow and anyone stealing it can spoof the device or app that it belongs to. This is a well-known issue with the OAuth2 specification and there are possible approaches such as Proof of Possession (PoP) tokens [129] that address this. However, the use of those token approaches is inherently costly both in device bootstrap and in runtime code. This is examined in more detail below. Bearer tokens expire and this limits the exposure.
- **Refresh Token**: The refresh token is even more important. An attacker stealing a refresh token can permanently emulate a device or app, if they also have the

ClientID and ClientSecret. Without those, the Refresh Token is valueless.

- **ClientID**: The ClientID of the device is a random string that uniquely identifies the device. On its' own, this information has no specific value. An attacker needs to have a ClientID together with some other data (e.g., a User id, a Client Secret, or data/commands) to make use of the ClientID.
- **ClientSecret**: the ClientSecret is a random string that is designed to be unsusceptible to dictionary attacks. On its own, this has no value. The combination of ClientID and ClientSecret can be used *before a device has been claimed* to spoof that device and therefore potentially persuade a user they have claimed a device when in fact a different device is claimed. If a device is already claimed, then the claim flow will fail. If an attacker gains the ClientID, the Client Secret and the Refresh token, then they can spoof the device.
- **Data and Commands**: an attacker stealing data or commands may be able to fingerprint the device or may be able to tie the device to the user through aspects inherent in the data. For example, if the data contains GPS co-ordinates, then the attacker may be able to identify the driver of a car. However, inherently the data and commands are less valuable without tying them to a specific device or user.
- **AuthCode**: the AuthCode is valueless without a corresponding ClientID and ClientSecret.

The model has two categories of threat. There are threats against specific aspects of the system, such as the use of TLS, OAuth2 flows, etc. Secondly there are threats against the overall system. The threats against the overall system can be divided into security threats and privacy threats.

As discussed in Section 4.1 there is a published threat model for OAuth2, which makes recommendations for securing OAuth2. These are broadly applicable to this model, and specific issues are examined below. In addition, it is assumed that threats against TLS apply broadly to Internet systems and therefore these threats are not addressed in this work.

7.2 Security Threats

A common approach to identifying threats in threat modeling is the *STRIDE* system [246], which characterises the following threats:

- *Spoofing* — Using someone else's credentials to gain access to otherwise inaccessible assets.

- *Tampering* — Changing data to mount an attack.
- *Repudiation* Occurs when a user denies performing an action, but the target of the action has no way to prove otherwise.
- *Information disclosure* — The disclosure of information to a user who does not have permission to see it.
- *Denial of service* — Reducing the ability of valid users to access resources.
- *Elevation of privilege* — Occurs when an unprivileged user gains privileged status

The STRIDE approach will be used to identify threats. An important aspect with security and privacy is that there is always a cost/benefit analysis. For example, naturally a non-Internet connected device can infringe less on privacy than an IoT device. Hence, security and privacy must be enhanced at a reasonable cost to the user, manufacturer and service provider.

7.2.1 Spoofing

There are several systems that could be spoofed. Each of these is analysed in turn.

The device is protected from spoofing using the ClientID and ClientSecret. An attacker cannot create a new ClientID or ClientSecret as the DCR API is protected and only authorised manufacturers can call it. One possible attack is a physical attack on the device to retrieve the credentials. The benefit of this is restricted to certain attack cases (e.g., where the attacker wished to send bad data or acknowledge acting on commands which it did not act upon), because if the attacker has the specific device then they can perform other attacks on it. This could be mitigated by using a TPM and code attestation on the device, but as discussed this adds cost and complexity that most IoT manufacturers are unwilling to bear. A system stealing the bearer token can spoof the device for a given amount of time (until the bearer times out). This is protected against: the device receives the bearer over TLS from the DIdP and the device only sends the bearer over TLS to the gateway.

As discussed above, a device with stolen credentials (ClientID and ClientSecret) could be spoofed during the claim process. This can be protected against by having a positive confirmation of successful claim on the device (e.g., a distinct pattern of LED flashes).

The manufacturer uses a credential to call the DCR API. The connection between the manufacturer and the device is protected from spoofing using the manufacturers own process. In the prototype this was implemented using a private network and physical scanning presence.

The DIdP is protected from spoofing by the use of server-side TLS certificates. The

manufacturer and gateway use full certificate chains and are coded to validate the server name. The device can use the same approach or can have a TLS certificate fingerprint programmed in at manufacture time (which is the approach taken in the prototype).

The gateway is protected from spoofing in two ways. As a server, the gateway has a server-side TLS certificate. This can be coded into the device, or can be returned as metadata in the Refresh flow. As a client, the gateway uses a token of its own that gives it scope permission to call the introspection API on the DIdP.

The app is protected from spoofing by its ClientID and ClientSecret.

All the systems are protected from man-in-the-middle attacks by the use of TLS on all distributed links.

One overall aspect about spoofing where this model is potentially more risky than existing models is the complexity and number of connected systems. This is a balance between federation and the benefits of separation of data and concerns. The above mentioned authentication and authorisation measures protect against this, but there is inherently a risk to this approach. One question for this work is whether this risk is outweighed by the benefits. To answer that would require further validation of this system, especially with industrial partners, and this is called out in the further work in Chapter 11.

7.2.2 Tampering

Tampering attacks in the system would equate to either sending incorrect data or incorrect commands. This is protected against by the encryption and authentication of systems. Physical tampering of a device is a specific risk of any cyber-physical system. The protection against physical tampering is that every device has different credentials (preventing the stealing of one credential affecting other devices) and that affected devices and credentials can be remotely revoked through token revocation.

7.2.3 Repudiation

Repudiation is one area where this model potentially offers a worse option than a standard system. In a standard system, the raw metadata and raw data are available directly from the device to the service that consumes that data, and therefore it is harder for a repudiation to occur. In this model, devices are pseudonymously linked to service providers and therefore repudiation is actually easier. A protection against this can be proposed, which is not yet implemented, but forms a potential further

work. This approach involves running PCM systems under SCONE [18] or another system that supports remote attestation. This is discussed in Section 11.4.5.

7.2.4 Information Disclosure

The main aim of this system is to prevent information disclosure. The encryption model of the system provides general protection against attackers listening. Further avenues of information disclosure are based on the potential attacks to each of the systems and analysis of what information each system has available to it. Table 6.4 provides a key reference in assessing these possibilities of information disclosure.

An attacker of an IoT system might reasonably expect to target the data and commands of a particular device or the devices of a particular user. In order to do that, the attacker must identify the device or the user. To identify the device, an attacker must either tie it to a user or to a specific place or identity. The system and model do not use any inherent identity of the device such as a MAC address. An attacker who compromised a gateway or PCM would have the IP address of the device and could potentially use that, but would not have the user of the device, only the pseudonym. The attacker who compromises the DIdP has the users identity and their pseudonym but not any of the data or commands. Obviously, the controls against these are proper protections for the DIdP and Gateway, including normal secure development, operations and infrastructure processes.

An attacker who compromises a manufacturer only has access to the ClientID and ClientSecrets. They may be able to spoof unclaimed devices. To disable this one could implement a process for devices to update the ClientSecret immediately after manufacture.

An attacker who compromises a third-party application only has a subset of the data and neither the pseudonym nor the user identity.

An attacker who compromises a device does not have access to the userid or the pseudonym.

Specifically, this model provides significant protections against information disclosure. Firstly, by limiting data sharing to consented parties, and enabling summarisation and filtering, the model prevents a single party logging and storing all data by default, as is the case in many existing IoT systems. Secondly, by using pseudonyms and an intermediary that acts on behalf of the user, the system protects against sharing metadata and fingerprinting by default. Thirdly, by separating user identity, device identity and data/commands, an attacker must compromise multiple systems in order to compromise a user and their data.

7.2.5 Denial Of Service

All the communications are protected by encryption and authentication, thereby meaning that denial of service attacks can be minimised. In addition, any device or application that starts to misbehave can have its access revoked.

7.2.6 Elevation Of Privilege

There are no elevation of privilege attacks that were identified.

7.3 Privacy Threats

In Chapter 2, there was a summary of the LINDDUN model of privacy threats. The same approach as STRIDE can be taken to use the flow modeling of data across the system derived from the CSP model to analyse privacy threats.

7.3.1 Linkability

OAuthing allows users to share data without linking the specific user to a given device. However, OAuthing does allow the app to identify that multiple interactions come from the same user, since a user authorises the app with a given token. On the other hand, the app has only a random token. While it is possible to allow a more unlinkable approach, there would be a major usability challenge, requiring users to re-consent numerous times. The model does allow multiple devices to each be connected to the same app with different tokens, reducing linkability across a user's devices. Currently this is possible if the user separately authorises each device with the app. In addition, the hiding of the device's IP address, MAC address and other core properties is a useful addition in this space.

7.3.2 Identifiability

The model does not provide inherent identifiability as previously discussed. One potential improvement to OAuthing would be to offer different pseudonyms for the same user. However, it is not clear what advantage this would bring. Since the pseudonym is only shared with the Gateway, this logically only protects against attacks on the Gateway or initiated in the Gateway. In the OAuthing model, in order for sharing to occur, the device and app must share a PCM, which is based on the pseudonym.

One possibility is that the DIdP could wait until all tokens are expired (or potentially deliberately expire them) and then issue a new pseudonym.

7.3.3 Plausible Deniability

There is a wonderful tension between plausible deniability and non-repudiation. By allowing users to configure summarisation and filtering rules, and defining policies at the PCM, a user may genuinely gain plausible deniability by either refusing to share data, maintaining anonymity, or by hiding specific data. On the other hand, a user may genuinely wish to share data and the app may need guarantees about the data. This is discussed in Section 11.4.5.

7.3.4 Undetectability And Unobservability

The model addresses this through the ability to filter data, or to not share it in the first place.

7.3.5 Confidentiality

The model addresses this as discussed above via encryption of all external data channels. In addition, the PCM does not store data, except temporarily for summarisation and filtering. Therefore data is not held “at rest” in the system, making attacks on data harder.

7.3.6 Content Awareness

OAuthing does not inherently help with this aspect. The real challenge in this area is around creating user interfaces that offer users a clear view of the data — and the implications of that data — that they are sharing. However, it is certainly conceivable that OAuthing could offer an environment in which to offer such a UI. For example, the consent screen could utilise metadata about device types to show the implications of the data collected. The device type is captured by the DIdP as part of the DCR API, although this is currently not used.

7.3.7 Policy And Consent Noncompliance

This is an area where OAuthing offers a strong answer. Firstly, the system is designed to offer a consent management option independent of manufacturers and app

providers. Secondly, the system where each user has their own PCM implementing consent policies, summarisation and filtering policies offers an effective model for enforcing policies and consent management.

7.4 Comparing The Model Against The Requirements

In Chapter 2, a set of requirements for secure, private IoT middleware was identified. Here is a summary of how this model addresses or does not address those requirements.

- **REQ1 - Integrity and Confidentiality**
  This model provides a clear approach for integrity and confidentiality, through the use of encryption.
- **REQ2 - Access Control**
  The model ensures that all data and commands are protected with access control.
- **REQ2.1 - Consent**
  The model uses consent as the primary mechanism for access control, and it has been demonstrated formally that data or commands only flow between devices and apps when both systems have appropriate consent.
- **REQ2.2 - Policy-based access control** The policies in this model are deliberately simple. Extending these to a more powerful policy model is an area of further work. In doing this, it is clear that the content-awareness principle from LINDDUN will play an important role.
- **REQ3 - Authentication**
  The model requires authentication for all flows.
- **REQ3.1 - Federated Identity**
  The model supports federated identity, both for devices and users. In particular, the use of the identity broker pattern allows users to choose their own identity provider.
- **REQ3.2 - Secure Device Identity**
  The model uses OAuth2 ClientID and ClientSecret as secure device identities, and ensures these are only ever communicated with the DIdP after manufacturing is complete. In addition, an OAuth2 refresh token provides a secure identity that ties the device to the user.
- **REQ3.3 Anonymous Identities**
  The model supports pseudonymous identities, ensuring that the users identity

is only ever known by the DIdP, and the pseudonym is only shared with the gateway and PCM.

- **REQ4 - Attestation**
  The model does not support attestation at this point. This is an area for further work.
- **REQ5 - Summarisation and Filtering**
  The model supports summarisation and filtering through the use of the PCM. However, this has yet to be implemented and therefore implementation of this is seen as a significant area of further work.
- **REQ6 - Context-based security and Reputation**
  The model does not support context-based security or reputation at this point. This is an area for further work.

7.5 Conclusions Of The Threat Modeling

In this chapter, the properties and data flows derived in Chapter 6 were used to create a threat model. The threat model used the STRIDE and LINDDUN approaches to analyse the threats. In particular, the model provides clear benefits in a number of the security and privacy properties identified through the requirements analysis and threat modeling. The separation of concerns outlined in the data sharing matrix makes it harder for attackers to identify users, steal data, fingerprint devices, and forge commands.

Chapter 8

Implementation of a Prototype

8.1 Implementation

In order to validate the model, a prototype of the system was created. This was done to answer the following research questions:

- Is the proposed model implementable?
- Do any specific issues emerge during implementation?
- Does the implementation demonstrate *workability*: that this works in a cost-effective approach that could be used in practice?
- What performance does the prototype provide - in terms of memory usage, latency, transaction rates and cost per transaction?
- How does this performance compare to existing systems?
- What difference in power and energy consumption does this model create for the device?

The system was implemented primarily in the *Node.js*® [267] framework. This is a high-performance asynchronous runtime based on the JavaScript language. This was chosen because of previous experience coding in JavaScript, as well as the availability of a number of libraries that sped up development, including:

- **Oauth2orize** — a framework for implementing OAuth2 flows and APIs.
- **Dockerode** — a library for controlling Docker.
- **Aedes** — a lightweight implementation of an MQTT server.
- **Mqtt** — a library for interacting with MQTT packets.
- **Passport** — a library for working with third-party identity providers.

- **Express** — a Web framework for building dynamic web systems.

In addition to Node.js, a number of other open source technologies were used:

- **Docker** — a lightweight virtualisation solution for running systems in the cloud.
- **Cassandra** — a highly scalable multi-master NoSQL database.
- **Python** — for coding the registration server.
- **RSMB** — a lightweight MQTT server.
- **ArduinoIDE** — a coding system for IoT devices.

Docker [171] is a system that allows lightweight virtualised services (known as *containers*) to be built, stored, instantiated and managed in automated approaches. In particular, Docker containers have a significantly lower memory footprint than traditional Virtual Machine (VM) systems, allowing hundreds of Docker containers to run on a single machine. In addition, the automation of the Docker system allowed the whole prototype to be instantiated quickly and effectively as a network of cooperating systems in a cloud environment. The prototype uses a *Microservices* architecture [265], which has a strong fit with Docker. *Docker Compose* [75] is an approach for creating compositions of Docker containers that are interdependent, and the use of Docker Compose significantly simplified the configuration and setup of the system.

In particular, the PCM model is implemented as Docker containers. Each user has their own Docker container running a broker that works on their behalf, as discussed below.

The registration server (which is the code that the Manufacturer runs) was written to run on a Raspberry Pi small form factor computer. Using a specific distribution — *Hypriot* — even this was configure as a Docker system.

Figure 8.1 shows the overall prototyped system. The code for the prototype is Open Source under the Apache License and is available at `https://github.com/pzfreo/oauthing`.

A short video showing the prototype in action and the registration and data sharing process is available at `http://freo.me/oauthing-video`.

8.1.1 Protocol Mapping

The model has been designed to work independently of specific protocols. However, in order to implement a prototype there was a need to make specific choices on protocols. In particular, the aim was to enable a device to communicate using only one, IoT-optimised protocol. MQTT was chosen for this. In order to do make this work, it

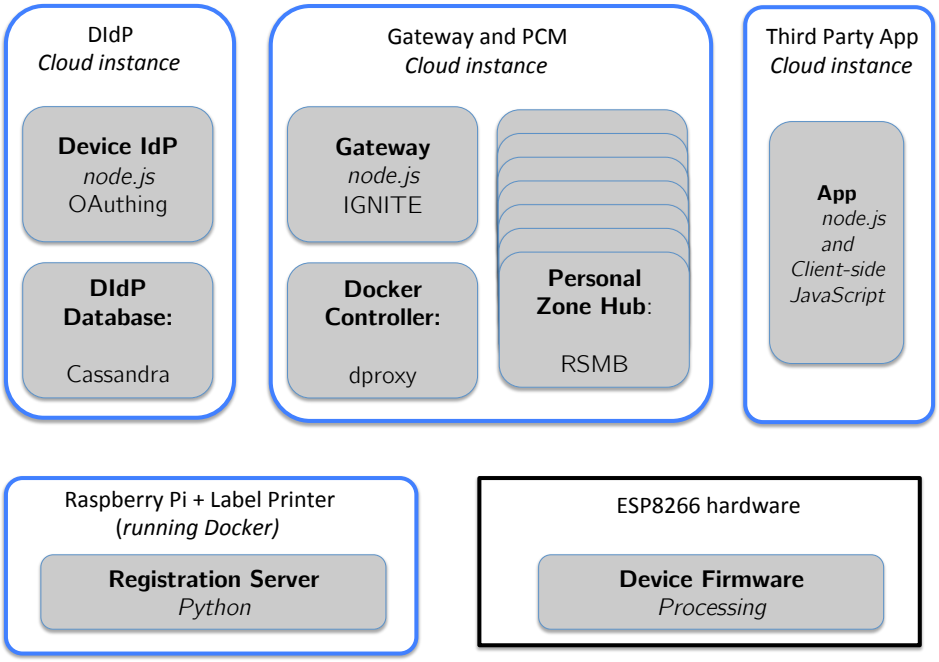

Figure 8.1: Prototype of the OAuthing System

was necessary to map certain parts of the OAuth2 specification into the MQTT protocol. In particular, the registration process normally requires the OAuth2 "client" to be an HTTP server. Figure 8.2 shows the HTTP flow and then Figure 8.3 shows the corresponding MQTT flow that was developed to support IoT devices.

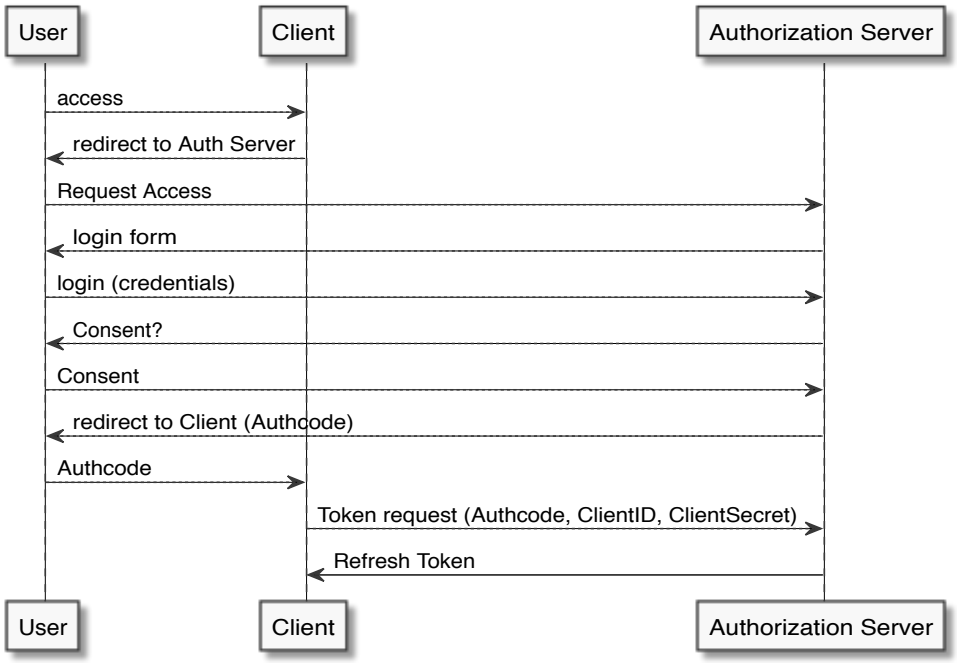

Figure 8.2: HTTP flow to Create OAuth2 Refresh Token

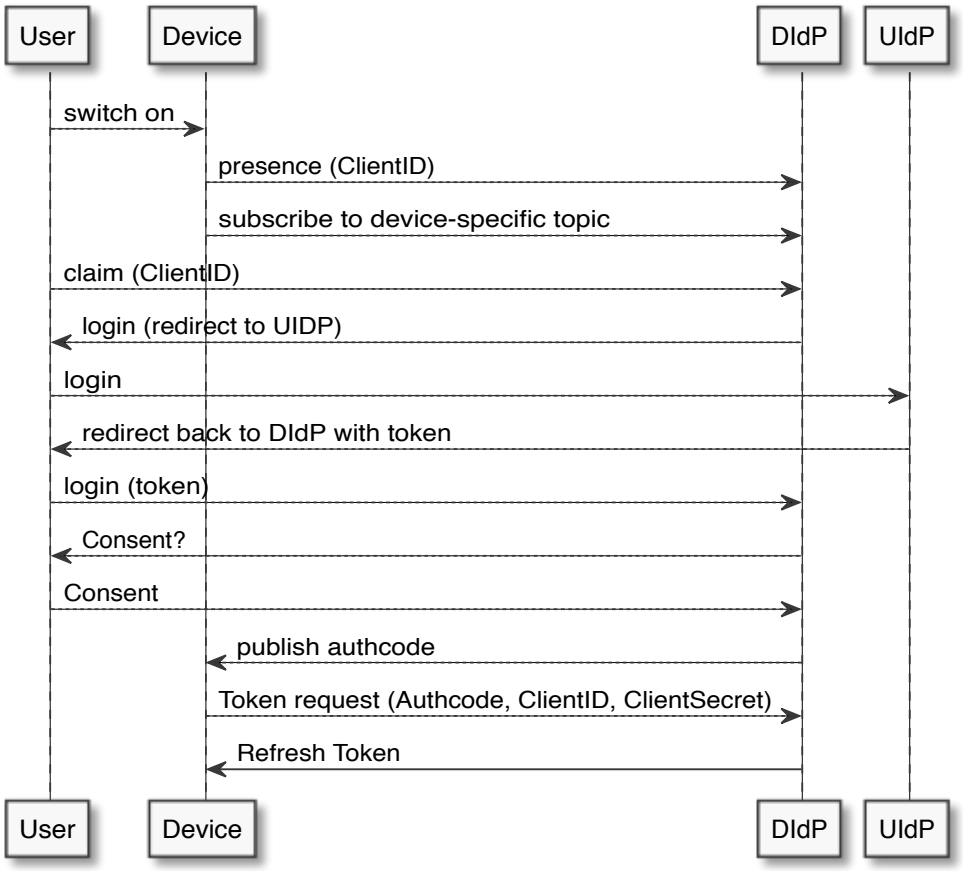

Figure 8.3: OAuthing MQTT flow to Create OAuth2 Refresh Token

8.1.2 Handling Refresh Flows In OAuthing

As well as the registration flow, it was necessary to map the refresh flow into MQTT. This ensures that the device does not need to implement multiple wire protocols. While the next version of MQTT is planned to support request-response flows [61], the current model does not inherently support request-response. In the previous prototypes, this was handled by using a subscription based on the MQTT client identifier and an access control rule based on that, which is supported by the Mosquitto broker. This is described in Chapter 5. However, after the creation of the prototype, I identified a significant security issue with this model. In MQTT, there is a rule that if a second connection is initiated with the same MQTT client identifier, the broker *must* disconnect the first connection and replace it with the second. This means that if an attacker can guess the MQTT client identifier of the device, they could potentially steal the refresh token.

In this OAuthing prototype, this was addressed with two aspects. Firstly, the Mosquitto broker was removed and the system used a new approach based on an *embedded broker*. Secondly, the OAuth2 ClientID was used as the basis for the subscription

instead of the MQTT client identifier.

The embedded broker pattern is where the MQTT broker used for the OAuth2 flows is embedded into another server, in this case the DIdP. A diagram comparing the embedded broker pattern to the FIOT prototype is shown in Figure 8.4.

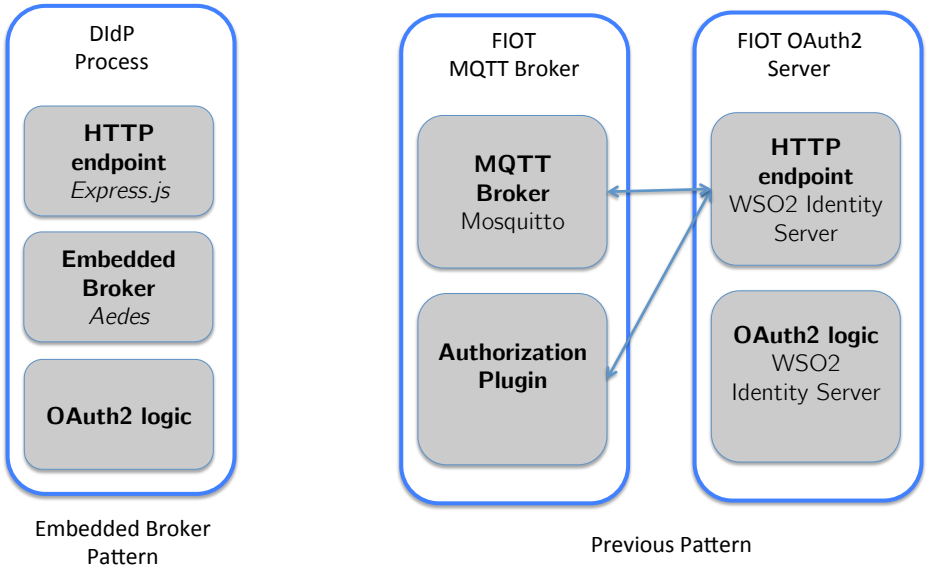

Figure 8.4: Comparing the Embedded Broker to the FIOT Approach

The embedded broker is only used for security flows, including device registration and token refresh. There are two main benefits to using the embedded broker for security flows. Firstly, the refresh token is only ever seen by the DIdP and the Device. In the previous pattern it was passed over the intermediate Mosquitto broker. Secondly, the broker can use the OAuth2 ClientID and ClientSecret as credentials and ensure that responses are only sent to clients that have those credentials. Effectively, although the system is using the MQTT protocol, the publish/subscribe approach is not used, ensuring that only the device that made the request can receive the response. This tight coupling between the MQTT handling and the authorisation code ensures that the security hole identified in FIOT is not present in OAuthing.

8.1.3 Data And Command Protocol

To support interoperability of devices and CSs, a minimalist API for communicating data and commands, was defined based on MQTT. This could be extended in future, with protocols such as HTTP and CoAP [244]. In the previous work in Chapter 5, there was an authorisation policy model that permitted complex rules. While this was more powerful, it was not easy to present users with a clear consent choice. In this model, there is a simplified approach that is based on a simple *topic* model:

- `/d/#`: any messages sent to the `/d` topic or subtopics thereof are considered *data* published by sensor devices.
- `/c/#`: any messages sent to the `/c` topic or subtopics therof are considered *commands* published to device actuators.

It is also specified in this system that all messages should be published in the SenML [9] format.

One view of the system is to consider it from the perspective of its APIs. Table 8.1 shows the various APIs and the protocols that each of the systems offers. It also shows the OAuth2 scopes that are required for each API.

API	Client	Provider	Scope	Proto	Description
Dynamic Client Registration	Manu-facturer	DIdP	dcr	HTTP	This allows the creation of OAuth2 clients.
OAuth2 Token API	Device CS	DIdP	NA	HTTP MQTT	This API is defined by the OAuth2 specification.
Introspection	IG	DIdP	intro	HTTP	IG to DIdP to introspect bearer tokens
Publish Data	Device	IG	Pd	MQTT	Publish Data from a device sensor
Read/Subscribe to Data	CS	IG	Rd	MQTT	CS subscribing to Data from a sensor
Publish Commands	CS	IG	Pc	MQTT	CS sending commands to an actuator
Read/Subscribe to Commands	Device	IG	Rc	MQTT	Actuator subscribing to commands from a CS

Table 8.1: APIs and their Associated Scopes

8.2 Components

The following sections outline the components that were implemented to demonstrate the system.

8.2.1 The OAuthing DIdP

The prototype DIdP is called the OAuthing DIdP. Although there were a number of capabilities already in the OAuth2orize framework, there were several capabilities that were needed or desirable that were not implemented. These included:

- The identity broker pattern where users are allowed to utilize their existing identities with UIdPs and broker new OAuth2 tokens based on the tokens passed from third-party systems;
- an implementation of the Dynamic Client Registration API [222];
- an implementation of the OAuth2 Introspection API [219];
- a Cassandra persistence layer; and
- an *embedded* MQTT broker that supports the mapping of the OAuth2 Token API into the MQTT protocol.

These were all implemented as part of this work.

The prototype supports a choice of UIdP from the following providers:

- Google
- Twitter
- Github
- Facebook

All of these are based on variants of OAuth2 flows, but the system can easily be extended to support other federated systems such as Shibboleth, OpenID Connect and SAML2.

The OAuthing DIdP is implemented as a set of containers running in the Docker container system, allowing it to be efficiently deployed and tested in a cloud environment. In this system the users profile information is not stored. Instead each user is issued with a new secure random pseudonym identifier. In order to associate this identity to the UIdP's identity, the name of the UIdP and the UIdPs unique identifier are hashed together and stored against the secure random pseudonym identifier.

8.2.2 IGNITE

The prototype of the gatewat is called IGNITE. This is a significant rewriting of the system described in Chapter 5. This version of IGNITE runs in a Docker cloud container environment and has access to control this Docker environment, using the Node.js *dockerode* library. When a device or TPA initiates an MQTT connection with

the `CONNECT` packet, IGNITE first validates the Bearer Token by calling the DIdP's introspection API. This either returns a random anonymous ID together with a set of scopes; or informs IGNITE that the token is invalid. IGNITE is then responsible for launching a new cloud instance to act as the PCM on behalf of the user, or routing the request to the existing PCM. IGNITE also implements the security policies defined in scopes by the DIdP, enforcing them before routing requests to the PCM.

8.2.3 Personal Cloud Middleware

The PCM was implemented using the open source RSMB MQTT broker [195]. This broker has a very low memory overhead, and enabled the system to run a significant number of PCM containers on standard hardware. This version of the prototype has not yet implemented summarisation and filtering on the PCM, which will potentially enlarge the memory footprint. At the same time, there was no attempt to optimise the Docker runtime of the PCMs or the underlying Operating System, and therefore there is some opportunity that the addition cost of summarisation and filtering logic can be offset by improved configuration and Docker tuning.

8.2.4 Device Hardware

Building a very simple device provides a baseline evaluation of whether the model is implementable on very small footprint devices. The commonly available ESP8266 platform was chosen for the reference device. This chip provides an embedded 32-bit processor, Wifi connectivity and a number of digital inputs and outputs for less than US$2.50 each (at the time of writing).

The device was coded using the ArduinoIDE which offers a C-like language — Processing — to support coding the device. The device code is available at `https://github.com/pzfreo/oauthing/blob/master/device/secure_thing/secure_thing.ino`. A picture of the sample device is shown in Figure 8.5.

The ESP8266 coded under ArduinoIDE supports TLS without full certificate authority chains[1]. Instead, it uses fingerprints of SSL certificates to validate the server certificate. Given the footprint requirements, this is a much more memory and storage efficient approach. The device is configured with the DIdP TLS fingerprint at manufacturing time and then the IG TLS fingerprint at User Registration time (using the DIdP as the trusted source for the IG TLS fingerprint).

[1] Since building the prototype there is now early support for certificate chains in the latest ESP8266 Arduino support

Figure 8.5: Sample Device

8.2.5 Manufacturing Process

Once the device is operational, it contacts a local manufacturing server via MQTT. The server may also have a physical means of identifying the device (e.g. scanning a bar code on the device). The manufacturing server calls the DCR API on the DIdP to register the device, and then sends the Client ID and Secret to the device over MQTT, where they are stored in EEPROM. As discussed in Chapter 6, this is because the DCR API needs to be secured, and it would be a problem for the device to have the credentials for the DCR API, as they might then be stolen by attacking the device. In the prototype, the device claiming is handled by creating a *User Registration URL*. This is a URL that takes the user to the DIdP and initiates the registration process for a given ClientID (the ClientID is encoded in the URL). This URL can be typed into a browser, but in this case it is encoded as a QR code. The QR Code is then printed and attached to the device. When this QR code is scanned, the DIdP first checks that the device is switched on (present) and connected to the DIdP. It then prompts the user to login via a UIdP (e.g. redirecting to Google's authentication). Once the user is logged

on the user is asked to authorise the device. This initiates a modified OAuth2 flow with the device, resulting in the refresh token being stored in the device's EEPROM. The device is now ready for use.

8.2.6 Sample Third Party Application

In order to demonstrate this system a simple web-based cloud service was also created. This first connects to the OAuthing DIdP using a standard OAuth2 HTTP flow to request access to IGNITE. The user logs in using the same UIdP that they registered their device with. After the user authorises the Sample TPA to access IGNITE the sample app is loaded. This uses MQTT over WebSockets to communicate, and presents a simple UI allowing users to interact with the device.

8.3 Conclusions And Further Work

The aim of building this prototype was not to create a full production quality version of the OAuthing model. In particular, a number of key aspects were not implemented:

- Summarisation and Filtering
- Device revocation
- Listing devices, apps and consents
- Clustering
- Docker memory optimisation

However, at the same time there was an aim to build a system that had reasonable performance and used effective, scalable technologies such as Node.js and Cassandra.

One technology approach that offers a significant potential improvement over Docker for the PCM is that of *unikernels* [45]. Unikernels use low-level hypervisors to run dedicated VMs that only include the code needed to run a single service. Based on existing data comparing unikernels and Docker [167], it is plausible that rebuilding the prototype around a unikernel model of PCMs would allow significantly more PCMs to be run on a given hardware system than using Docker, and potentially reduce initialisation time as well.

In conclusion, a working instantiation of the main flows of the OAuthing model was built. The prototype demonstrates that the system is implementable, and the next chapter will explore the performance and costs of this model.

Chapter 9

Test framework, methodology and results

In this chapter, the experimental testing of the prototype system will be presented. The test methodology and test harnesses will be explored in Section 9.1. The results of the tests will be presented in Section 9.2. Finally, the significance of these results will be examined in Section 9.3.

9.1 Test Methodology And Frameworks

The test methodology was influenced by both academic papers on middleware performance testing [163, 134] as well as commercial testing [5, 62].

9.1.1 Measures

Four key measures were identified for evaluating the prototype.

- **Transactions per second**
  The total number of transactions per second (TPS) that a system can handle is the most common measurement of distributed systems. The *maximum TPS* of a system can provide a cost analysis by analysing the cost per transaction. Measurement of TPS under varying workloads (e.g. number of concurrent clients) can provide insight into how a system copes under load, and provide evidence of thread-contention or other inefficiencies. Measurement of how TPS increases as a system is scaled up onto nodes in a cluster provides a measure of scalability known as the Karp-Flatt metric [210]. Since this system is only a prototype,

there was no evaluation of scalability across multiple nodes. However, there are measures of the maximum TPS and TPS under varying workloads.

- **Latency**
  The latency of a system measures how quickly the system reacts. For example, the latency of a web-based system measures the time taken for a server to respond to the client with a web page. In publish-subscribe messaging systems, there is no "response" to a message. A commonly accepted measure of latency in such systems is the end-to-end latency. This is the time taken from the start of a publication process on the publisher to the time that the message is received at the subscriber. The latency of a system is a key measure for users, because it captures the responsiveness of a system: as an example, the time taken between a switch being pressed in a smart home and the corresponding light switching on. In a well-designed distributed system, one sees a clear relationship between latency and load [210]. The latency of the system starts out at a baseline, when the system is less than fully loaded, as each new client can be accommodated in spare CPU and network capacity. Once either the network or CPU becomes fully loaded, each new client should be allocated a fair share of these contended resources and, as a result, there should be a linear relationship between the number of concurrent clients and the latency each client observes. Two measures are used: the end-to-end latency of data publication, and the latency of a newly registered device connecting. In addition the latency of web-based APIs (such as manufacturing and introspection) under load is tested.

- **Device Memory**
  Devices often have constrained memory, and therefore a key measure identified was to understand the memory requirements of the created firmware and therefore the remaining memory available to device designers to implement their own logic. Many IoT devices operate under the *Harvard Architecture*, where instruction memory and variable memory are separated. The ESP8266 device used for the prototype uses this model. The impact of the prototype firmware on both instruction and variable memory was therefore examined.

- **Power Consumption and Energy Usage**
  A key aspect for IoT devices is power consumption. Many IoT devices (including the prototype system) run off of a battery with a limited capacity and therefore rely on using low power. In particular, this was a common request for enhancement from other researchers when the initial results of this work were presented at conferences. Therefore a test harness was created to accurately measure power usage and energy consumed by the sample device. Further details of the power measurement system are described below.

9.1.2 Testing Across Multiple Clients

There are some commercial and Open Source test tools for MQTT, but none of them met the requirements:

- The ability to simulate the OAuth2 token process in order to populate the bearer token.
- The ability to simulate multiple clients, each with their own unique credentials
- The ability to measure end-to-end latency from publication to delivery.

As a result of this it was necessary to create a new test framework for simulating IoT devices communicating over MQTT.

In order to simulate multiple clients for the latency and TPS tests a test harness was created that was designed to run across multiple cloud servers. The test harness is, like the prototype, based on the Docker container system. A key aim in testing approaches is to have repeatable results. The use of Docker and Docker Compose for both the *testing* system and the *tested* system meant that it was simple and repeatable to launch the infrastructure in a cloud environment, spin up test systems, scale up the test, and record the results.

The test harness consists of multiple Test Load Drivers (TLDs). Each TLD can simulate one or more clients, emulating the network behaviour of the IoT device. In the system the TLD runs each client as a separate process, with each TLD running up to 50 procesesses.

The TLDs then report the performance and latency data to a Test Manager, using a separate MQTT system from the system under test. The Test Manager subscribes to the results from multiple TLDs and accumulates the overall results from across all the TLDs and all the virtual IoT clients. All the test systems were written in Node.js. There was no attempt to simulate the actual timing of devices over the network or low-level TCP behaviour of clients. This could potentially form a further work, although it is not clear if this would provide any further insight.

Figure 9.1 shows a simplified diagram of the test environment, which is running in a public cloud environment.

The Test Manager is running on a separate instance, and this service collates the results. The TLDs implemented three different workloads:

- **One Second Client**
  The *One Second Client* emulates a device that sends one message per second. This is designed to test the system under moderate load.

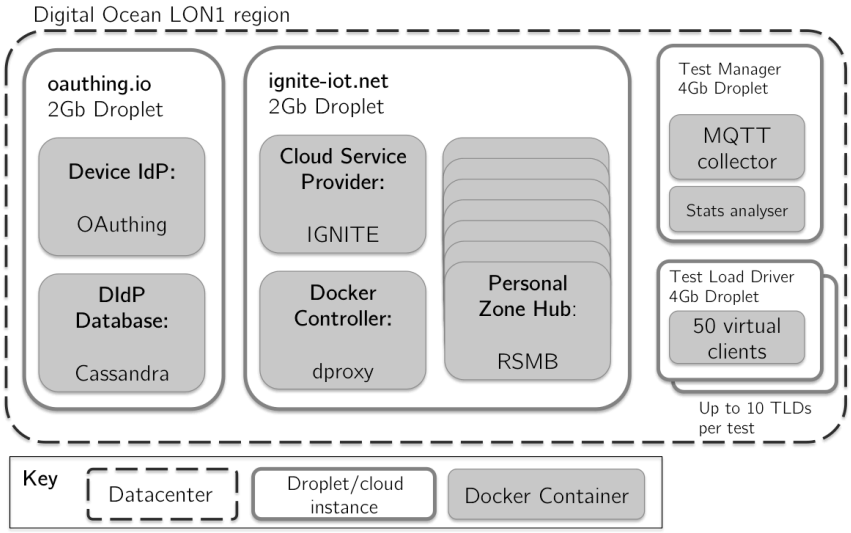

Figure 9.1: Test Environment

- **Stress Client**
  The *Stress Client* emulates a device that sends messages continuously. This is designed to test the system under heavy load.

- **Connection Time**
  The *Connection Time* test first populates a brand new user at the DIdP, ensuring that when the test connects, the IG will need to create a new PCM instance. The test then disconnects and reconnects, timing the time it takes to connect when a PCM already exists for that user.

In all the tests described, it was ensured that the systems were warmed up and running and that all the results were repeatable across multiple tests. In addition a *baseline* system was created for comparison. The baseline system does not use the OAuthing backend, but instead uses the commonly used Mosquitto MQTT server. The baseline system does not implement OAuth flows or PCM, and it does not implement authorisation or authentication on the Mosquitto server. Therefore it is to be expected that the comparison will underestimate the costs of a comparable system, and therefore overestimate the additional cost of OAuthing.

All the tests were carried out using standard cloud server instances with fixed sizes from the Digital Ocean cloud provider [1].

[1] `https://www.digitalocean.com`

9.1.3 Energy And Power Measurement

In order to test the additional power burden of using the OAuthing approach, a power measurement harness was created to evaluate the power usage of the system. Traditional power measurement systems are not optimised for measuring sub-watt power usage. No effective ready-built system for measuring low power usage was identified and therefore it was not possible to use an "off-the-shelf" system. Figure 9.2 shows a logical diagram of the power management test system that was created.

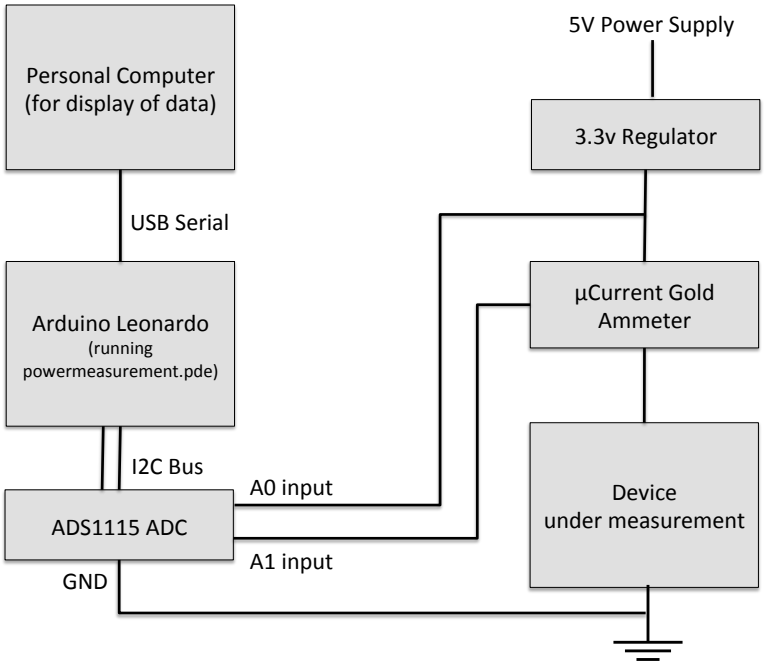

Figure 9.2: Power Management Test System

Simple physics defines that:

$$P = IV$$

where P is power, I is current, and V is voltage.

Furthermore

$$E = \int P \, \mathrm{d}t$$

where E is energy and t is time.

The created system measures milliwatt power usage with better than 1% accuracy. The power management test system is based on an Arduino, together with a high-resolution analogue-to-digital converter (ADC) for measuring voltages (based on the

TI ADS1115 chip[2]). This device has typical accuracy of $\pm 0.2\%$ according to the data sheet. In the circuit used in this test, it was configured so full scale was 4.096V, giving a maximum precision of 0.125mV per bit. To measure the current flowing through the sample device a high-accuracy low-burden ammeter[3] was used. This device acts as a current-to-voltage converter with stated accuracy better than $\pm 0.1\%$. The ADC was used to measure both the input voltage to the device and the current flowing through the device. Given that the ammeter converts 1mA to 1mV, the maximum precision of current reading is 0.125mA.

The maximum current of the measured device is under 300mA. Given the range of accuracy of the ADC converter, the overall accuracy of this power measurement is better than $\pm 1\%$.

A short program[4] was written to capture the power data and integrate it over time to provide energy usage. This runs on a separate Arduino Leonardo system and is therefore completely independent of the ESP8266. The power measurement harness takes 313 samples per second, giving a sample approximately every 3ms.

Two different scenarios were measured, comparing the OAuthing device to the same device configured to talk to the baseline system without using the OAuthing model

- The first scenario was measuring the total energy usage from initial power-on until the first message is sent to the server. This measures the bootstrap power phase, especially capturing the overhead of the refresh flow and the credential based MQTT CONNECT message. This is measured in milliwatt-hours. Each test was run 20 times.
- The second scenario was the on-going power usage over the next 15 minutes after startup. To ensure both systems were comparable, the test waited until the device was fully warmed up and then took 900 seconds of samples (approximately 280,000 samples).

Once again it was desirable to compare the OAuthing usage to the baseline system. In this case, a version of the device firmware was created that connects to the baseline system instead of OAuthing. This device firmware did not use a refresh flow or any credentials. However, it did use TLS to encrypt the communication.

[2] `http://www.ti.com/lit/ds/symlink/ads1115.pdf`
[3] `http://alternatezone.com/electronics/ucurrent/uCurrentArticle.pdf`
[4] Available at `https://freo.me/powercode`

9.2 Test Results

Figure 9.3 shows the latency comparison of IGNITE compared to the baseline system (Mosquitto) using the *one second client*. Figure 9.4 shows the mean, 95% and 99% percentiles for the IGNITE latency responses in the same test. The graph demonstrates that IGNITE shows consistently low latencies across all workloads: the additional latency added to message interactions compared to Mosquitto was around 1ms. The percentiles show that 99% of requests had latency of less than 11ms even when the system was loaded with 400 test clients, and 95% of requests had latency less than 6ms.

The next data point collected was the maximum number of PCMs that could be run on a single cloud server. The tests were run on a server with 2Gb of memory and no swap configured, costing US$20/month. This environment was able support at least 400 PCM instances, with the server running out of memory beyond 415 containers. Simply adding swap will increase this number at the cost of some latency, but this has not yet been evaluated.

Figure 9.5 shows the average connect time for three different scenarios. The fastest is the Mosquitto broker, with an average connect time of 24.5ms. The slowest is the IGNITE when the user has not previously connected. In this scenario, the system needs to introspect the token and then wait until the new container is launched and ready. This takes on average 1294ms (1.3 seconds).

The third scenario is the connect time when there is already a user container running. The average time here was 35.9ms. The extra latency compared with Mosquitto (35.9ms vs 24.5ms) is well within acceptable ranges.

Figure 9.6 shows the performance of the IGNITE system under stress. This shows that the server was handling more than 4500 TPS at all levels and the average latency rose to 83.3ms when the system had 400 concurrent clients. This test demonstrates that the system as deployed in the test environment can support each user owning 600 devices each interacting once a minute, even when the system is fully loaded with 400 concurrent PCM containers. The latency line shows that as new clients are added the latency increases in direct proportion, demonstrating fair allocation of resources.

Figure 9.7 shows the performance of the OAuthing IdP while issuing new Client IDs during manufacturing. This is the least well-performing part of the system because the system uses a secure hashing algorithm (PBKDF2 [139]) to ensure that the password database is resistant to dictionary attacks. The result is that adding a client incurs considerable CPU time. In fact, the use of this system is mostly unnecessary, as the "passwords" that are hashed are in fact the ClientSecrets. These are 30-byte random identifiers that are created by the system and therefore not susceptible to dictionary

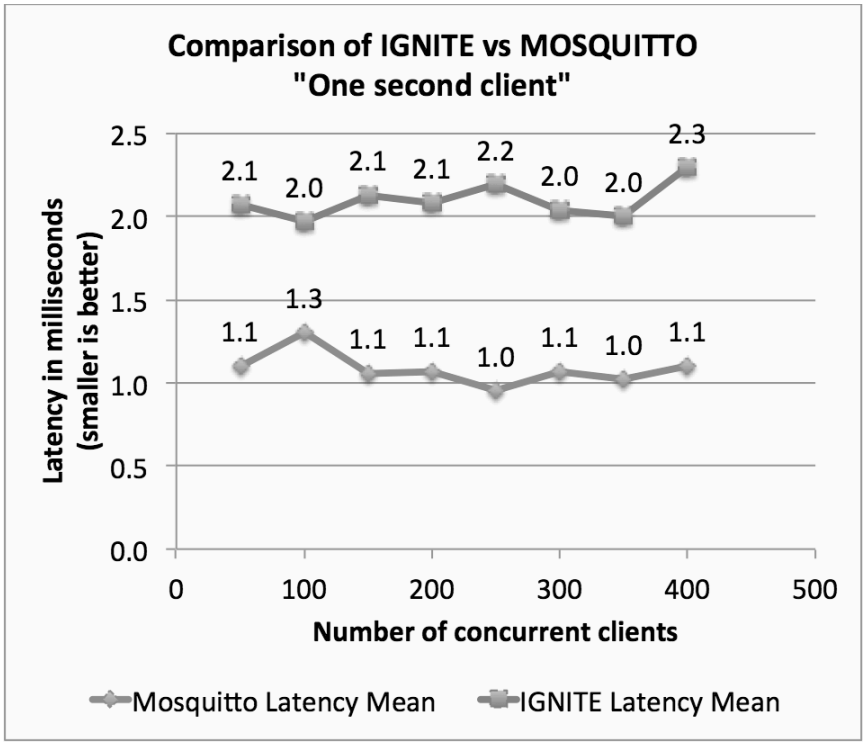

Figure 9.3: One Second Client IGNITE vs Mosquitto

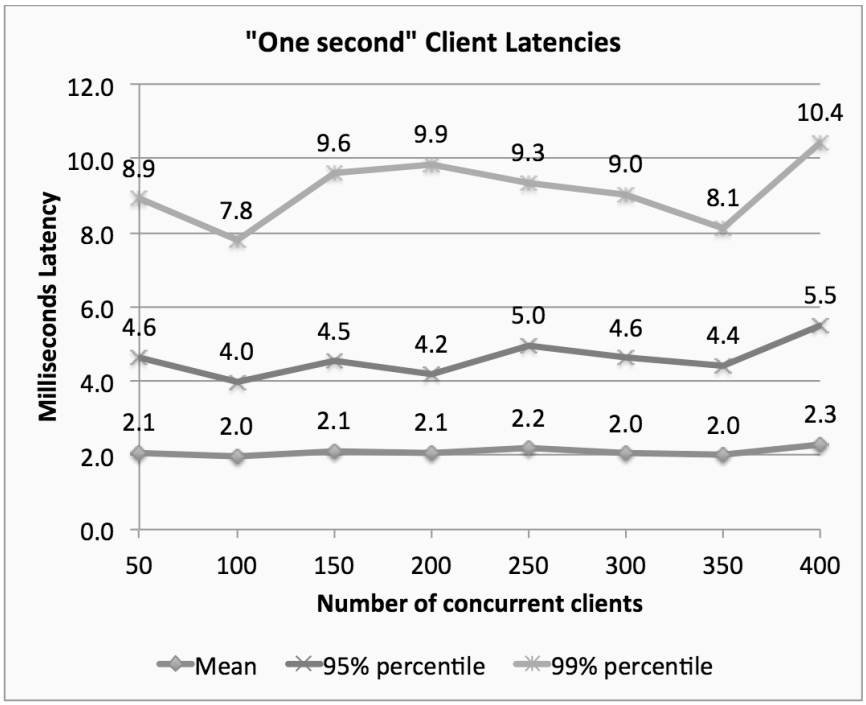

Figure 9.4: One Second Client IGNITE Percentiles

attacks in reasonable time.

Performance tests measuring the latency and throughput of the DIdP under introspection were performed. This test the performance when the IGNITE gateway asks

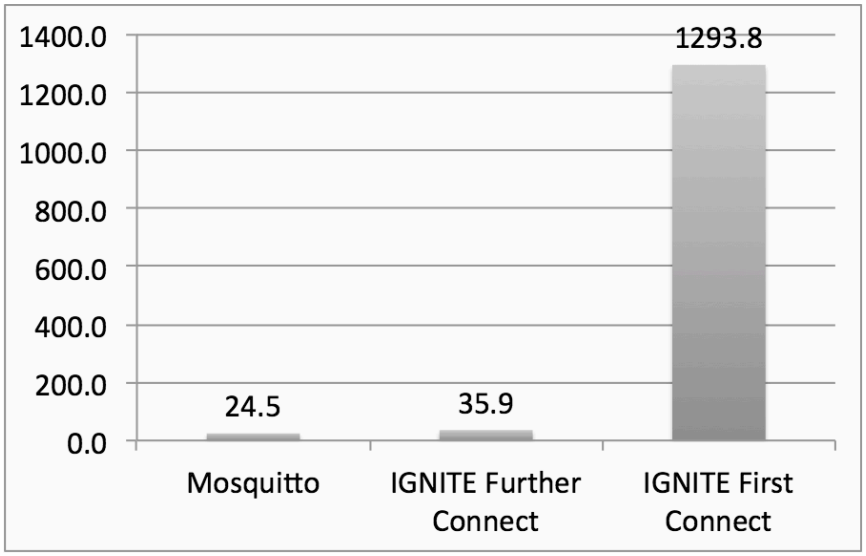

Figure 9.5: Device Connect Latency

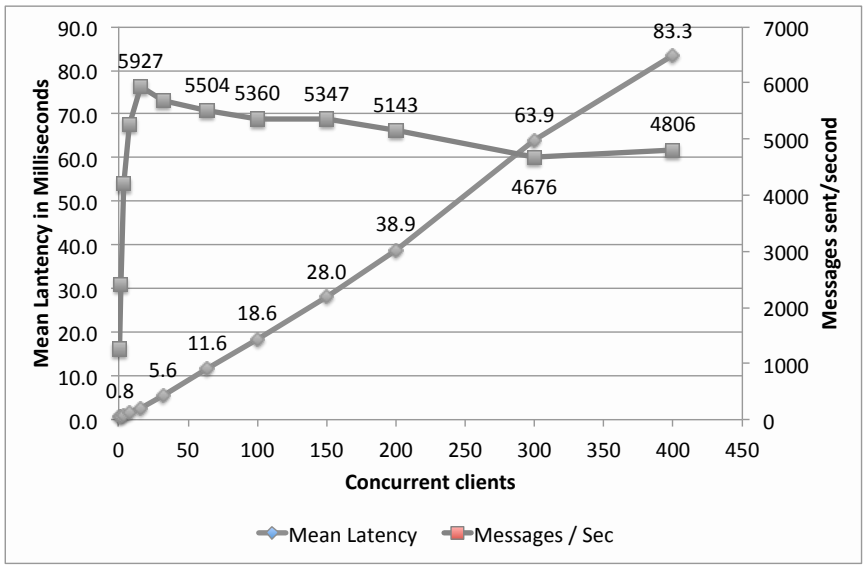

Figure 9.6: Stress Client IGNITE Performance

the DIdP for the anonymous identity and authorisation policies. This is presented in Figure 9.8.

9.2.1 Device Memory Usage

The Arduino development environment that was used to create the device firmware provides statistics on the program and variable memory usage. Figure 9.9 shows the program and variable memory usage of the ESP8266 when loaded with the OAuthing sample device loader code. The graph captures the usage of different components of the loader. Each column includes the previous column. The largest component is

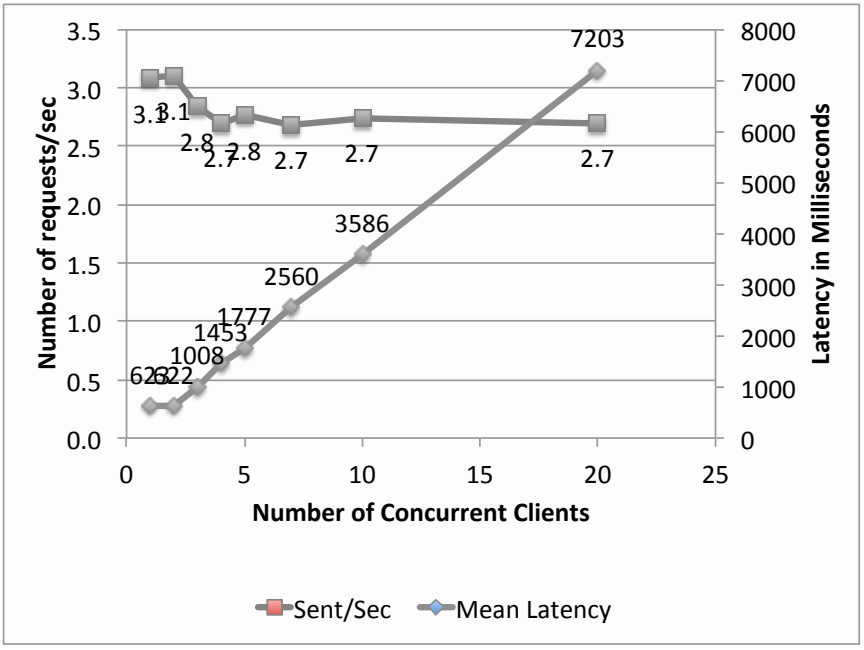

Figure 9.7: Dynamic Client Registration Latency and Throughput

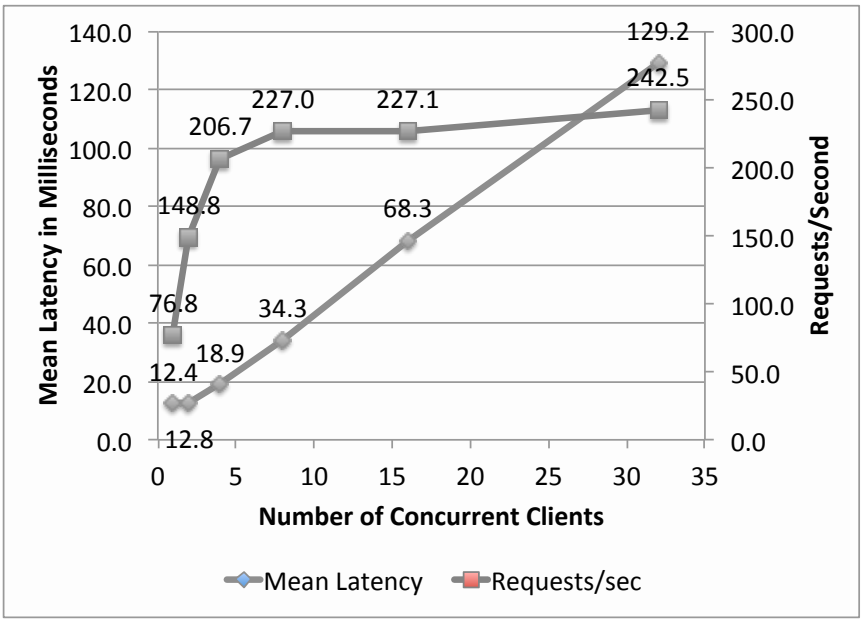

Figure 9.8: Throughput and Latency of the Introspection API on the DIdP

the base C libraries needed by the Arduino system, which take up 40% of variable memory and 24% of program memory. The next largest aspect is the TLS support which incrementally takes 5.5% of the program memory and 7.7% of the variable memory. Overall, the loader leaves over 38k of variable memory and over 700k of program memory for the developers of any device sensor and actuator logic, which is sufficient to build complex device applications and support a variety of sensors and actuators. There was no code optimisation after the working prototype was created,

and therefore there may be room for improvement.

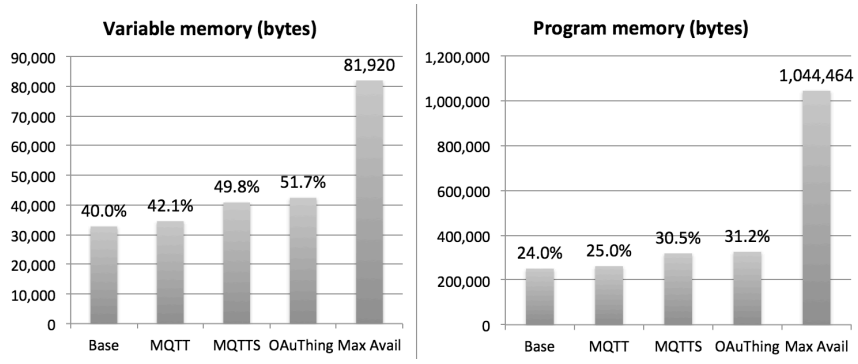

Figure 9.9: ESP8266 Memory Utilisation

9.2.2 Power And Energy Measurement

Figure 9.10 shows the time-to-first-message results. The data shows that the OAuthing device took 0.68s (5.7%) longer to send its first message, using 7.5% more energy (0.195mWH) than when connecting to Mosquitto. This incorporates the extra time and energy required to connect to the DIdP to refresh the token before connecting to the IG.

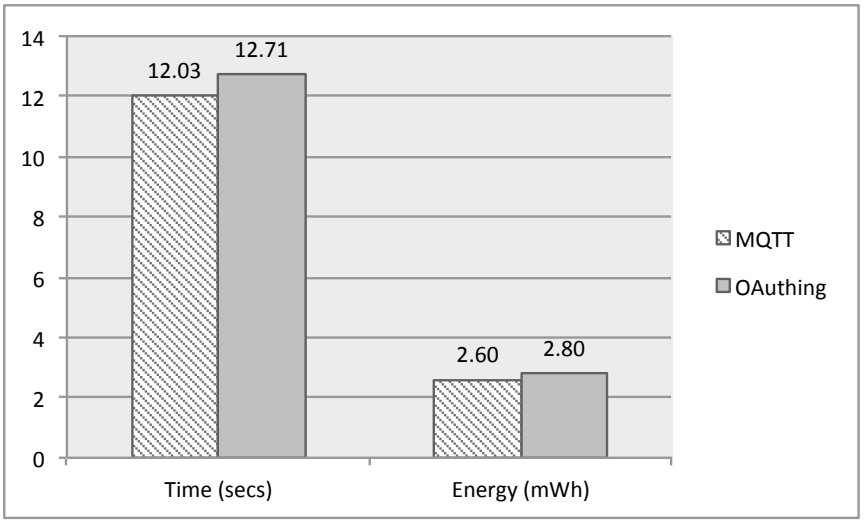

Figure 9.10: Time and Energy to Bootstrap

Figure 9.11 shows the on-going power requirement is 26mW higher for the OAuthing device, using 11.5% more power than the device connecting to Mosquitto.

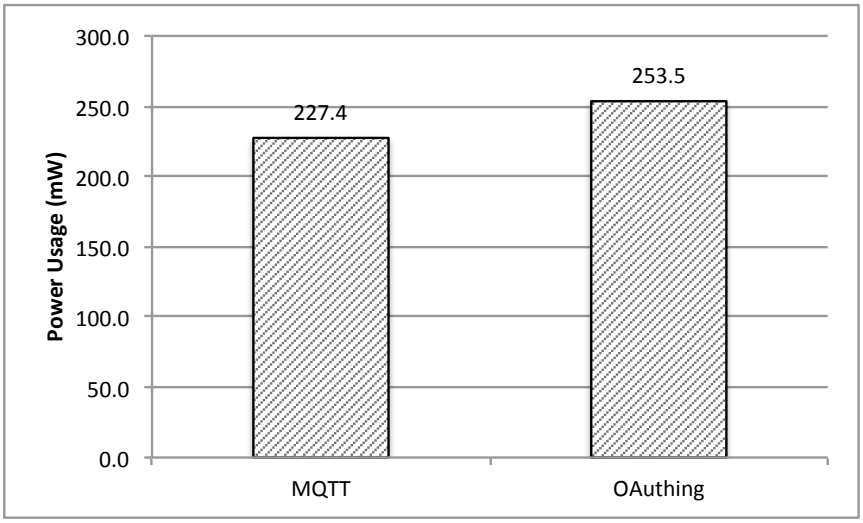

Figure 9.11: Power Usage

9.3 Analysis Of Results

There is a high degree of confidence in the overall accuracy of these results. The tests involved multiple iterations and broadly similar results were observed with each run. There are small discrepancies in latencies as one increases the number of simulated devices (for example between 50 and 100 clients on the "one second" client where there is a small drop in latency as one adds users). However, these are well within the normal experimental errors in testing distributed systems. The coefficient of variation (CV)[5] of the power and energy usage numbers was calculated, and this found that the CV for the boot energy usage was less than 6%, which indicates a very low variability of results. The CV for the long running power usage tests was less than 23%, showing a greater variability but not excessively so.

The analysis of what these results mean in practice is more difficult. Unsurprisingly, the increased federation and use of tokens, together with the extra network and logical hops introduced by the PCM model adds time and cost to the system compared to a system that has a single central user model and shared middleware. The question of whether the increased data separation, pseudonymity, consent and federation are *worth the extra costs* is the challenge here.

In order to evaluate these results the additional time, cost and usage will be compared to industry norms in each case.

The additional latency of the system compared to Mosquitto is one measure where it can be definitively stated that the extra cost of the system is negligible. The most obvious comparison point here is the round trip time of a message on the Internet.

[5]The standard deviation divided by the mean

Given that average round trip ping times over the Internet are in the 20-80ms range, these results demonstrate that the overhead of the proposed approach is insignificant to users. Even when the system is heavily loaded with hundreds of users, the additional latency is under 100ms, and for many IoT scenarios this is acceptable. An example might be switching on a light, where a delay of $\frac{1}{10}$ is acceptable. In addition, many IoT use cases involve logging data (e.g. fitness, health, pollution, etc) where this additional latency is irrelevant.

When comparing the additional latency in the first connect time, one might compare with the standard time taken by an IoT device to register with its Wifi hub, which is in the range 1-10 seconds. The additional latency in the first connect time (1.3 seconds) is therefore not a serious concern. If there was a significant issue, this could be ameliorated by pre-loading unused containers and associating them with users at connect time. The actual additional time taken by the device to perform a token refresh, and then connect to the server and publish when it was already registered and the PCM is running was on average 0.68s, which is again of no consideration in most use cases.

The DIdP results show that the introspection service can successfully scale to support many IGs. Given that CSs and Devices only connect intermittently (due to the persistent TCP session model of MQTT), even a single DIdP server could handle significant numbers of devices and third-party clients. The throughput of the DIdP in handling introspection demonstrates supporting up to 30 gateways per DIdP, handling more than 200 introspections per second. If the bearer token timeout of a device is 30 minutes, then assuming the devices timeouts occur with an even distribution throughout the time, a single DIdP could handle more than 300,000 devices each refreshing every 30 minutes.

The device creation performance is less attractive and a better balance between hashing and performance would need to be achieved to support the system in production.

The costs of running a PCM for each user are harder to evaluate. As a cost per user, this is a low amount compared to the cost of an annual mobile or fixed-line data package. On the other hand, if this cost was absorbed into the device cost there is a wide variety of device costs, ranging from $5 for an Amazon Dash button to hundreds of dollars or more for high-end connected medical devices. Re-implementing the system using unikernels might offer an order of magnitude more users per server and therefore bring down the cost per user by a factor of ten, which would make the ongoing cost of a PCM achievable for almost any type of device.

The final aspect to analyse is the extra power and energy consumption. The additional energy cost (0.195mWH) for boot up can be considered reasonable. For example, an average rechargeable AAA battery has around 1000mWH capacity, so the extra energy

usage for a boot up is less than 0.02% of that.

However, the additional power usage during operation of 26.5mWh is unexpected and requires further investigation. As a percentage, an 11% drop in battery life would be considered a problem in almost any IoT use case where battery lifetime is a consideration. The ESP8266 offers three different low-power modes that offer improved battery life over normal usage. None of these three were implemented, and therefore one can expect the magnitude of this power usage to be reduced in a production device.

Overall, the additional costs of OAuthing are within the bounds of reasonableness. Given that the system is a prototype that has not yet been optimised, it can be asserted that the results of this testing show that there is considerable scope for this model in real environments.

9.4 Conclusions

This chapter has covered the test methodology, the creation of two test harnesses, the results of the tests and an analysis of the test results. The test methodology was designed to measure a set of measures that was derived from looking at other middleware and IoT studies. On these measures the extra costs of using the system compared to a baseline were identifed. These extra costs were then compared against the normal costs of IoT systems (both in time, power and money).

Part IV

Part IV - Conclusions

Chapter 10

Comparison with related work

In this chapter, the OAuthing approach is compared to related work.

10.1 Comparison With FIOT And IGNITE

First, the model presented in Chapter 6.2 is compared with the preliminary investigations, presented in Chapter 5. The initial work — *FIOT* — on using OAuth2 tokens with IoT devices had three main concerns:

- A potential security flaw in the refresh flow, whereby a device that spoofed the MQTT client identifier could use a timing attack to steal a bearer token.
- No use or demonstration of TLS or encryption.
- Lack of unique credentials per device apart from the refresh token, resulting from treating the broker as the OAuth2 client instead of the device.
- No defined device and user registration processes — the device needed to be coded with the token manually.

In the second preliminary work, *IGNITE*, the concern of individual identifiers for devices was addressed. This was addressed by treating each device as a unique OAuth2 client and populating the client identifiers using the DCR API.

In both previous works, identifiers needed to be manually added to the device, which is unrealistic in manufacturing processes. Existing public IoT middleware such as *IBM Watson IoT* and *AWS IoT* also currently have this concern.

The OAuthing model and prototype address these issues:

- By treating each device as an OAuth2 client, it is ensured that stealing a single device only compromises that single device. There is no requirement to trust

any other party apart from the DIdP with client identifiers or secrets.

- As each device has a ClientID and ClientSecret, the refresh flow can be secured. In addition, the use of the *embedded broker* approach, described in Chapter 8, resolves wider concerns about using MQTT for request-reply semantics with authentication flows.
- By taking advantage of improvements in available IoT hardware, it was possible to use a device that supports TLS, while remaining at a similar cost level to the previous prototype.
- The device and user registration flows outlined in Chapter 6 and coded in Chapter 8 demonstrate that there is a usable approach to configuring devices with secure tokens.

In addition, the OAuthing model also defines the use of PCM and pseudonyms. This extends the privacy and security controls beyond those in the previous two systems.

10.2 Comparison With Others

IOT-OAS [59] addresses the use of OAuth2 for IoT devices with the CoAP protocol. Compared to OAuthing, IOT-OAS lacks well-defined processes around device and user registration. Because IOT-OAS is simply an authorisation service it doesn't offer the additional benefits of the PCM model, such as device masking, summarisation and filtering. In addition, there is no pseudonymisation.

The mapping of the OAuth2 Token API to support IoT devices using the CoAP protocol is being formalised in ACE [130], and is described in [271]. ACE is a draft protocol and set of associated protocols that are currently an active work in progress. Unlike OAuthing, which uses simple bearer tokens, ACE requires the use of PoP tokens. PoP tokens are more secure than bearer tokens, but there is an additional burden on the IoT developer to support them. For example, it was not possible at the time of writing to identify any available code to support PoP tokens in an Arduino environment. In general, the mapping of OAuth2 into CoAP is simpler than into MQTT, because CoAP has semantics based on HTTP. However, while ACE does map the *Client to AS* flows into CoAP, there is no well-defined user or device registration process that compares to the OAuthing device and user registration processes, especially when it comes to supporting devices with no UI. There is an even newer working paper on supporting ACE with MQTT [240], which does not, as yet, address the mapping of *client to AS* flows into MQTT.

In [192] there is a demonstration of the OAuth1 protocol with MQTT, favouring OAuth1 over OAuth2 for IoT devices. This is based around concerns with the security

of bearer tokens. The reasons for choosing the older OAuth protocol are obviated by the mapping of the refresh flow into MQTT which OAuthing offers. While this model also uses the concept of OAuth2 Client ID for each device, the work does not define any clear device registration flow, requiring those identifiers to be manually inserted onto devices (as in the FIOT model). In addition, every device must support HTTP and MQTT, which is a significant burden on development.

In [212] and in [85] there are platforms that support OAuth2 for IoT devices that communicate via HTTP and WebSockets. None of these works address automated registration processes, and none provide the privacy controls of anonymous identifiers or isolated personal cloud instances.

In [226] a capability-based access system is described that allows anonymous identities to be used. [38] provides an Architecture Reference Model for an approach that supports anonymous identities. Neither of these systems separate the provision of anonymous identities from the data-sharing middleware.

The concept of a Personal Zone Hub (PZH) is described in the Webinos [72] system: this has similarites to the PCM, in that both aim to be a central place where a user's devices and data sharing can be in the control of the user. The difference between PCM model and PZH model is that the PZH model is primarily seen as a local, non-cloud based system that manages devices and communications on behalf of a user. In Webinos users must instantiate the PZH themselves. While this is an attractive concept, Webinos does not address many of the configuration challenges of that model. By comparison, the PCM model in OAuthing is a system that is based in the cloud, but created automatically upon registration of a user into the system. In addition, this work provides a cost model for PCMs that addresses the additional cost of providing the per-user infrastructure. Webinos does not address federated device identities and does not provide a registration process.

Chapter 11

Discussion, Further Work and Conclusions

In this chapter, the work is assessed against the requirements identified in Chapter 2, the gaps identified in Chapter 3, as well as the research questions in Chapter 1. The strengths and weaknesses of this work are reflected upon and further areas for research are identified.

11.1 Addressing The Research Questions

In Chapter 1, three major research questions were proposed:

RQ1
: What are the privacy and security requirements, especially those on identity and access control, for the Internet of Things? How do they differ from the existing requirements on the classical Internet.

RQ2
: What is a model and architecture for IoT systems that can meet the requirements identified in RQ1? What are the threats and risks of this compared to the risks and threats identified in RQ1.

RQ3
: Is there a practical instantiation of this architecture, and if so, what are the increased costs in complexity, performance, latency and resources compared to existing systems.

11.1.1 Research Question 1

This question is fundamentally answered in Chapters 2 and 3, where a a set of requirements, gaps and challenges for IoT were identified. An improved ontology for addressing threats and challenges for IoT — the matrix model — was presented to elucidate the differences in security and privacy challenges compared to classical Internet systems. This identified a number of issues that are specific to IoT systems, especially around low-power, constrained environments; hardware attacks; secure identities and registration processes; fingerprinting; and others.

In Chapter 3, a structured search strategy identified a number of existing middleware systems targeting IoT scenarions. Overall this approach identified 55 middleware systems of which 20 systems demonstrated a security architecture that was detailed enough to be considered for review. In a literature review of these 20 systems, a significant set of gaps was identified, which are documented in Chapter 3.

11.1.2 Research Question 2

This question is fundamentally answered in Chapter 6, where a model is outlined that addresses a significant proportion of these requirements. In Section 7.4 the match between the model and the requirements is documented. In summary, the model addresses many of the key requirements, but there is further work to do on summarisation and filtering, attestation and context-based security. These gaps are addressed in the discussion on further work in Section 11.4. A threat model for the system is presented, identifying the ways in which this model protects against attacks, enhances privacy and implements the above requirements.

11.1.3 Research Question 3

This question is answered in Chapters 8 and 9, where a prototype is demonstrated that implements the model defined in Chapter 6. The prototype system, while not an optimised or production system, demonstrates that the additional costs of the system work are reasonable. In answering RQ3, two test harnesses were created for measuring the prototype both with a simulated client and directly understanding the power consumption of the sample device.

11.2 Contributions And Impact

11.2.1 Contributions

The main contributions of this work are as follows.

- In Chapter 2, two approaches to evaluating threats and challenges for IoT: firstly, a matrix ontology extending the CIA+ ontology, and secondly, a mapping of IoT privacy threats into Spiekermann and Cranor's three-layer privacy model.
- In Chapter 3, there is a structured literature review of secure middleware for IoT, that provides the most detailed available analysis of security in IoT middleware systems.
- Early published work into the use of Federated Identity and Access Management in IoT that has had impact on other academic and industrial systems, described in Chapter 5.
- A novel model for IoT middleware in Chapter 6, that provides:
  - A clear mapping of user and device registration into the OAuth2 specification, providing a model that does not require any UI on the device to successfully register and take ownership of devices.
  - An approach that decouples different roles and components, that ensures consent for all data and command sharing, and inherently reduces identifiability and linkability of devices and device data.
  - An extension of the concept of a PZH into a cloud environment to create *Personal Cloud Middleware*, that provides users with a single central point of control for devices and device interactions in the cloud that prevents fingerprinting and identification. The PCM model includes an approach for instantiating and managing these runtimes on behalf of users.
  - A formal model of the approach that proves end-to-end properties of the system and data flows (in Chapter 6).
  - A threat model — based on the STRIDE and LINDDUN threat modeling techniques — that demonstrates the enhancement of privacy and security in this model (in Chapter 7).
- A cloud-based prototype of the model (in Chapter 8)
- A test harness and methodology designed to produce repeatable test results (in Chapter 9).

- Experimental results, including performance, capacity and latency measures of the DIdP, Gateway and PCM, together with power and energy consumption of the device. These results are compared to a baseline approach to demonstrate the workability of the model, and identify an upper bound for the extra costs of the system.

11.2.2 Impact

The impact of this work is evolving.

- The early work on OAuth2 with MQTT published in [103] has 19 existing citations excluding self-citations, which demonstrates some impact upon literature and other research.
- A pre-print of the literature review presented in Chapters 2 and 3 has 9 existing citations excluding self-citation.
- OAuth2 support in MQTT has been incorporated into a commercial product — *WSO2 IoT Server* — and the product approach was inspired by this work.
- The paper [103] has directly influenced the IETF work on MQTT with ACE [240], as evidenced by citation in the first draft.

11.3 Strengths And Limitation

The strengths of this work are:

- A clear logical thesis from structured review, requirements identification, modeling, prototype through to results.
- Clear structured approaches to literature review, including the threat matrix, and the application of Spiekermann and Cranor's three-layer privacy model.
- Robustness of the model based on formal proof of properties.
- Use of well-defined threat modeling methodologies.
- A repeatable test methodology and test harnesses.
- Peer-reviewed publication of the results and incorporation of the feedback into the ongoing research.

The limitations of this work are:

- The work does not address attestation of devices, middleware or third-party applications.

- The technical nature of this work fails to address many of the important human aspects. For example, there is no usability study of the system.
- There has been dissemination of this work at academic and industry conferences, as well as some limited presentation to industrial partners. However, a more structured approach to gaining feedback from industrial partners has not been attempted.
- The policy model for consent scopes is deliberately simple and therefore only addresses the most basic scenarios.
- There is no proof of concept around summarisation and filtering.
- The prototype does not implement device or application revocation.
- There needs to be an approach to allowing users to choose which DIdP they use and which cloud provider for the PCMs, allowing full choice.

These limitations are addressed in the proposed future work.

11.4 Further Work

There remain a number of unexplored aspects of this model.

11.4.1 Provider Choice

One significant aspect is ability to allow users to choose the DIdP and cloud provider for the system. There are two ways in which this could happen.

The first is to enhance the existing model to support *DIdP migration*. In this approach the model would be extended to allow users to request a migration between DIdPs and provide a mechanism to transfer devices and apps from one DIdP to another. As a secondary aspect it is also easily conceivable to allow the DIdP to enable cloud provider choice. This can be done by passing the IG address, or even the PCM address directly, and TLS fingerprint back in an extended response with the bearer token in the refresh flow.

11.4.2 Distributed Ledger Design

The other approach to allowing users to choose the DIdP is to have a shared governance model across multiple instances of DIdPs. This can be acheived by using an *Distributed Ledger* or *Blockchain* model. I explored this concept in [100], which lays out a proposed approach of further work whereby the OAuthing model could be backed

by a distributed ledger. This does create a concern on how constrained IoT devices are to trust a given DIdP to correctly represent the state of the ledger, which is also addressed by a proposal in that paper. Overall, this is probably the most important area of further research arising out of this work. The OAuthing model and prototype provide an excellent basis for further research on how blockchains can improve the privacy and security of IoT systems, because they provide a clearly defined and well modeled identity interface for the device, which could be extended to support a blockchain as the back-end.

11.4.3 Improved Scopes, Summarisation And Filtering

A clear area of further research is to find common ways that IoT data can be summarised and filtered. This would require a technical ontology of the data that would allow the correct summarisation of data types. For example, understanding that an average of blood sugar levels is useful whereas an average of alerts is not. It also requires a policy model that can be usefully understood by users. For example, the user needs to understand the implications of consenting to raw data sharing versus summarised data sharing, and there needs to be a usable approach to allowing that. Finally, there is considerable interesting work to be done on filtering and even summarising commands. Once again an ontology and intelligent approach is required. For example, certain types of commands are more amenable to filtering than others, as discussed in the work. The creation of standard filters and summarisers needs a corresponding improvement in the policy language of scopes. For example, a user needs an effective and usable way to consent to share a rolling average of blood sugar over a day and to understand the outcomes of that.

11.4.4 Device And Application Revocation

A more detailed model and prototype could model revocation of tokens, allowing users to shut down devices and apps remotely and at any time. This is unlikely to uncover any major new research, but would make a more complete model and prototype.

11.4.5 Attestation Of PCMs

One aspect of this model is that it shifts the control point and trust from a direct connection to an intermediated system with the PCM as the main intermediary. This has a number of benefits as discussed above. However, there is a clear challenge that both parties need to trust the PCM. For example, the device owner and device need

to trust that the PCM correctly handles data and commands. Similarly the third-party application must trust that the PCM is correctly representing the device, especially if summarisation or filtering is happening. As an example, suppose a driver is sharing data from their car to an insurance company, and is lawfully using summarisation to only share certain aggregate data. The insurance company needs to know that this data has been correctly summarised. A solution to this can be proposed based on using attestation and secure enclave technology such as Intel Secure Guard Extensions (SGX). This would enable the PCM to run in an *enclave*, which assures that the code runs as intended and can be remotely attested, validating that a particular codebase is running. For example, SCONE [18] is a system that can run Docker containers on SGX and this is an area where further research could be profitable.

11.4.6 Unikernel Approach For PCMs

Although the cost per user is not unreasonable, the overall footprint of running the PCM system (RSMB) in Docker is still significant. RSMB can be run in just 200k of memory. Tuning of Docker will likely improve the number of containers, but another option would be to explore *unikernel* technology [45] to improve this, which would also potentially improve security concerns around Docker for the PCM.

11.4.7 Validation With Commercial And Other Organisations

One clear area for further study would be to assess the response of industrial, commercial and other organisations to this model. The approach has been informally presented to Daimler-Benz and Zühlke Engineering in October 2016, providing useful feedback and some expression of interest from Zühlke. A clear concern is that the business models of many IoT device manufacturers are actually based on harvesting data, so there may be push back on this approach which minimises the manufacturer's access to data. However, there are certainly domains such as health informatics where manufacturers are aware of the dangers of collecting user data. One other potential use case raised in discussions with industrial parties is in factory and manufacturing environments, which could involve further research. In addition, the introduction of the European Union's General Data Protection Regulations (GDPR) may also change the attitude of manufacturers and service providers.

11.5 Discussion

The OAuthing model provides enhanced security and privacy, as shown by the results of the threat modeling. Compared to both the previous prototypes FIOT and IGNITE

as well as others work, OAuthing provides a strong foundation for IoT privacy and security, that extends beyond the initial work on federated identity for IoT into a wider model of middleware for IoT. Data and identity are not shared without consent, and data can be shared anonymously. Device and user registration are automated, and the PCM model can prevent fingerprinting and sharing of IP addresses, protecting user and device identity. Formal proofs of the properties of OAuthing provide a strong basis for threat modeling as well as proving the end-to-end properties that are requured from the work. A prototype implementation provides experimental performance, cost and energy usage data to demonstrate that the system is workable.

One interesting way of looking at OAuthing is through the lens of the three-layer privacy model from Spiekermann and Cranor. The federated identity model of OAuthing aims to enhance the controls within the User Sphere and Joint Sphere, by enabling users to have clear ownership models for devices, removing the control and access that manufacturers have, and by enabling strong consent for all data sharing and commands. The PCM approach aims to move data sharing from the Recipient Sphere back into the Joint Sphere, so that users have more control and more visibility over the system that enforces policies. Finally, the consent model of OAuthing and summarisation and filtering aim to reduce the impact of the Recipient Sphere by enabling users to reduce the data that is shared to third-parties.

A key concern around the PCM model is that the cost per user might be too high. The prototype demonstrates that PCMs can be automatically deployed on behalf of the user with acceptable times. The experimental results demonstrate that a US$20/month cloud server can support 400 users, resulting in a cost per user of just $0.05 per month. Further optimisation could reduce this cost.

The OAuthing model and prototype demonstrate that devices can be connected to TPAs without inherently leaking the user's identity to either system. User's may choose to provide TPAs with their identity, but that becomes a positive consent of the user rather than the default. In addition users can bring pre-existing identities to the system rather than being required to create new credentials, which reduces the chances of password theft and gives users a choice of identity provider.

The work involved in this thesis is wide ranging. It encompasses literature research, modeling, implementation and testing approaches, including server-side, client-side and device programming, as well as some electronics. This work starts from addressing one very constrained problem — the identity of devices — and then used a variety of approaches, iteration, modeling and prototyping, to address that issue. The benefit of this wide approach is that it gives a more holistic view of the problem, and helps assess the overall strengths and weaknesses of the approach.

The original aim of this research was to examine security and privacy of IoT systems,

which, even 4 years ago when this research started, seemed to have concerns. In the meantime, an increasing number of attacks have been seen that use IoT systems as both the target and as an attack vector. This work is more relevant now than ever. The studies into identity of IoT devices and preliminary research have taken this research into the direction of creating a secure, private, federated middleware for IoT, with a foundation based on token-based, federated identity systems.

No single solution is going to address the security and privacy concerns of IoT, and there is much work to be done in this area — not just in technical approaches, but in economic, social, education and policy. This work is a stepping-stone in that direction, providing technical contributions into a far-reaching area.

Bibliography

[1] A. Abdul-Rahman and S. Hailes. “Supporting Trust in Virtual Communities”. In: *HICSS ’00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6*. Washington, DC, USA: IEEE Computer Society, 2000. ISBN: 0-7695-0493-0.

[2] Karl Aberer, Manfred Hauswirth, and Ali Salehi. “Middleware support for the Internet of Things”. In: *Proceedings of 5. GI/ITG KuVS Fachgespraech—Drahtlose Sensornetze* (2006), pp. 15–19.

[3] Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[4] Adedayo O. Adetoye and Atta Badii. “Foundations and Applications of Security Analysis”. In: ed. by Pierpaolo Degano and Luca Viganò. Berlin, Heidelberg: Springer-Verlag, 2009. Chap. A Policy Model for Secure Information Flow, pp. 1–17. ISBN: 978-3-642-03458-9. DOI: 10.1007/978-3-642-03459-6_1. URL: http://dx.doi.org/10.1007/978-3-642-03459-6_1.

[5] AdroitLogic. *ESB Performance*. http://esbperformance.org/. (Accessed on 03/20/2017). 2016.

[7] Gul A Agha. *Actors: A model of concurrent computation in distributed systems*. Tech. rep. Massachusetts Institute of Technology Cambridge Artificial Intelligence Lab, 1985.

[8] Aitor Agirre et al. “QoS Aware Middleware Support for Dynamically Reconfigurable Component Based IoT Applications”. In: *International Journal of Distributed Sensor Networks* 12.4 (2016). DOI: 10.1155/2016/2702789. URL: http://dx.doi.org/10.1155/2016/2702789.

[9] Jennings et al. *draft-jennings-core-senml-06 - Media Types for Sensor Markup Language (SenML)*. https://tools.ietf.org/html/draft-jennings-core-senml-06. (Accessed on 30th August 2016).

[10] M Alessi et al. “A web based virtual environment as a connection platform between people and IoT”. In: *Computer and Energy Science (SpliTech), International Multidisciplinary Conference on*. IEEE. 2016, pp. 1–6.

[11] Muneeb Ali et al. “Blockstack: A global naming and storage system secured by blockchains”. In: *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association. 2016, pp. 181–194.

[12] *An Open Source MQTT v3.1 Broker*. (Visited on 11/13/2013). URL: `http://mosquitto.org/`.

[13] Prateek Anand. “Enabling Context-Aware Computing in Internet of Things Using M2M”. In: *Nanoelectronic and Information Systems (iNIS), 2015 IEEE International Symposium on*. IEEE. 2015, pp. 219–224.

[14] Kim Andersson and Patryck Szewczyk. “Insecurity by obscurity continues: are ADSL router manuals putting end-users at risk”. In: *Proceedings of the 9th Australian Information Security Management Conference* (2011).

[15] Jean-Paul Arcangeli et al. “INCOME–multi-scale context management for the internet of things”. In: *International Joint Conference on Ambient Intelligence*. Springer. 2012, pp. 338–347.

[16] Arduino. *Arduino*. `http://arduino.cc/`. 2015.

[17] Pandarasamy Arjunan et al. “SensorAct: A Decentralized and Scriptable Middleware for Smart Energy Buildings”. In: *Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 IEEE 12th Intl Conf on*. IEEE. 2015, pp. 11–19.

[18] Sergei Arnautov et al. “SCONE: Secure linux containers with Intel SGX”. In: *12th USENIX Symp. Operating Systems Design and Implementation*. 2016.

[19] Kevin Ashton. “That ‘Internet of Things’ Thing”. In: *RFiD Journal* 22 (2009), pp. 97–114.

[20] Atmel. *Atmel AT97SC3204 Datasheet*. http://www.atmel.com/devices/AT97SC3204.aspx. 2015.

[21] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The internet of things: A survey”. In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[22] Francois Audet and Cullen Jennings. *Network address translation (NAT) behavioral requirements for unicast UDP*. Tech. rep. IETF, 2007.

[23] Alexandre B Augusto and Manuel E Correia. “An xmpp messaging infrastructure for a mobile held security identity wallet of personal and private dynamic identity attributes”. In: *Proceedings of the XATA* (2011).

[24] Jacek Augustyn, Paweł Maślanka, and Grzegorz Hamuda. "Hi-Speed USB Based Middleware for Integration of Real-Time Systems with the Cloud". In: *International Journal of Distributed Sensor Networks* 12.3 (2016). DOI: 10.1155/2016/2415016. URL: http://dx.doi.org/10.1155/2016/2415016.

[25] Benjamin Aziz. "A Formal Model and Analysis of the MQ Telemetry Transport Protocol". In: *9th International Conference on Availability, Reliability and Security (ARES 2014)*. IEEE. 2014.

[26] Benjamin Aziz, Paul Fremantle, and Alvaro Arenas. "A reputation model for the Internet of Things". In: *Engineering Secure Internet of Things Systems*. IET, 2016. Chap. 10.

[27] Benjamin Aziz et al. "A utility-based reputation model for the Internet of Things". In: *IFIP International Information Security and Privacy Conference*. Springer. 2016, pp. 261–275.

[28] Senthil Murugan Balakrishnan and Arun Kumar Sangaiah. "MIFIM — Middleware solution for service centric anomaly in future internet models". In: *Future Generation Computer Systems* (Aug. 2016). ISSN: 0167-739X. DOI: http://doi.org/10.1016/j.future.2016.08.006.

[29] James Ball. *NSA stores metadata of millions of web users for up to a year, secret files show*. http://www.theguardian.com/world/2013/sep/30/nsa-americans-metadata-year-documents. (Visited on 06/08/2015). 2013.

[30] Soma Bandyopadhyay et al. "A Survey of Middleware for Internet of Things". In: *Recent Trends in Wireless and Mobile Networks: Third International Conferences, WiMo 2011 and CoNeCo 2011, Ankara, Turkey, June 26-28, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 288–296. ISBN: 978-3-642-21937-5. DOI: 10.1007/978-3-642-21937-5_27. URL: http://dx.doi.org/10.1007/978-3-642-21937-5_27.

[31] Soma Bandyopadhyay et al. "Role of middleware for internet of things: A study". In: *International Journal of Computer Science & Engineering Survey (IJCSES)* 2.3 (2011), pp. 94–105.

[32] Andrew Banks and Rahul Gupta. "MQTT Version 3.1.1". In: *OASIS standard* (2014).

[33] Yassine Banouar et al. "Monitoring solution for autonomic Middleware-level QoS management within IoT systems". In: *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*. IEEE. 2015, pp. 1–8.

[34] Chetan Bansal et al. "Discovering concrete attacks on website authorization by formal analysis1". In: *Journal of Computer Security* 22.4 (2014), pp. 601–657.

[35] Chris Baraniuk. *Ashley Madison:'Suicides' over website hack*. http://www.bbc.co.uk/news/technology-34044506. (Accessed on 04/21/2017). Aug. 2015.

[36] Gianluca Barbon et al. "Taking Arduino to the Internet of Things: the ASIP programming model". In: *Computer Communications* 89 (2016), pp. 128–140.

[37] Rafael J CARO BENITO et al. "Smepp: A secure middleware for embedded p2p". In: *Proceedings of ICT-MobileSummit* 9 (2009).

[38] Jorge Bernal Bernabe et al. "Privacy-preserving security framework for a social-aware internet of things". In: *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer. 2014, pp. 408–415.

[39] Benjamin Billet and Valérie Issarny. "Dioptase: a distributed data streaming middleware for the future web of things". In: *Journal of Internet Services and Applications* 5.1 (2014), pp. 1–19.

[40] Manuel Binna. *Feasibility and Performance Evaluation of Canonical XML*. http://www.w3.org/2008/xmlsec/papers/C14N2_Performance_Evaluation_Thesis.pdf. (Visited on 06/09/2015). Oct. 2010.

[41] Bitcoin. *System Requirements*. https://bitcoin.org/en/bitcoin-core/features/requirements. (Accessed on 02/26/2017). Jan. 2017.

[42] Hendrik Bohn, Andreas Bobek, and Frank Golatowski. "SIRENA-Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains". In: *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*. IEEE. 2006, pp. 43–43.

[43] Hristo Bojinov et al. "Mobile device identification via sensor fingerprinting". In: *arXiv preprint arXiv:1408.1416* (2014).

[44] Bogdan Botezatu. *25 Percent of Wireless Networks are Highly Vulnerable to Hacking Attacks, Wi-Fi Security Survey Reveals | HOTforSecurity*. http://www.hotforsecurity.com/blog/25-percent-of-wireless-networks-are-highly-vulnerable-to-hacking-attacks-wi-fi-security-survey-reveals-1174.html. (Visited on 07/14/2015). 2011.

[45] Alfred Bratterud et al. "IncludeOS: A minimal, resource efficient unikernel for cloud services". In: *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE. 2015, pp. 250–257.

[46] Ernie Brickell, Jan Camenisch, and Liqun Chen. "Direct anonymous attestation". In: *Proceedings of the 11th ACM conference on Computer and communications security*. ACM. 2004, pp. 132–145.

[47] Geoff Brown. *MQTT and the NIST Cybersecurity Framework Version 1.0*. Tech. rep. OASIS, May 2014.

[48] *Building Facebook Messenger*. URL: https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger.

[49] Albert John Camilleri. "Reasoning in CSP via the HOL Theorem Prover". In: *Information Technology, 1990.'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No. 90TH0326-9)*. IEEE. 1990, pp. 173–183.

[50] Nancy Cam-Winget et al. "Security flaws in 802.11 data link protocols". In: *Communications of the ACM* 46.5 (2003), pp. 35–39.

[51] Mauro Caporuscio, Pierre-Guillaume Raverdy, and Valerie Issarny. "ubiSOAP: A service-oriented middleware for ubiquitous networking". In: *Services Computing, IEEE Transactions on* 5.1 (2012), pp. 86–98.

[52] Jon Card. *Anonymity is the internet's next big battleground*. http://www.theguardian.com/media-network/2015/jun/22/anonymity-internet-battleground-data-advertisers-marketers. (Visited on 07/13/2015). 2015.

[53] Miguel Almeida Carvalho and João Nuno Silva. "Poster: Unified remoteui for mobile environments". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM. 2015, pp. 245–247.

[54] Ann Cavoukian. "Privacy in the clouds". In: *Identity in the Information Society* 1.1 (2008), pp. 89–108.

[55] Rajiv Chakravorty, Joel Cartwright, and Ian Pratt. "Practical experience with TCP over GPRS". In: *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*. Vol. 2. IEEE. 2002, pp. 1678–1682.

[56] Moumena Chaqfeh and Nader Mohamed. "Challenges in middleware solutions for the internet of things". In: *Collaboration Technologies and Systems (CTS), 2012 International Conference on*. IEEE. 2012, pp. 21–26.

[57] Dong Chen et al. "TRM-IoT: A trust management model based on fuzzy reputation for internet of things". In: *Computer Science and Information Systems* 8.4 (2011), pp. 1207–1228.

[58] Hao Chen, Xueqin Jia, and Heng Li. "A brief introduction to IoT gateway". In: *IET International Conference on Communication Technology and Application (ICCTA 2011)*. 2011, pp. 610–613.

[59] Simone Cirani et al. "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios". In: *IEEE sensors journal* 15.2 (2015), pp. 1224–1234.

[60] Davide Conzon et al. "The virtus middleware: An xmpp based architecture for secure iot communications". In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE. 2012, pp. 1–6.

[61] R Coppen and B Raymor. *MQTT Version 5.0*. https://www.oasis-open.org/committees/download.php/59482/mqtt-v5.0-wd09.pdf. (Accessed on 05/23/2017). Nov. 2016.

[62] Transaction Processing Performance Council. *TPC*. http://www.tpc.org/. (Accessed on 03/27/2017). 2017.

[63] Thaddeus Czauski et al. "NERD – middleware for IoT human machine interfaces". In: *Annals of Telecommunications* 71.3-4 (2016), pp. 109–119.

[64] Li Dan and Li Danning. "An approach to formalize uml sequence diagrams in csp". In: *proceedings of 2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010 no. 2)*. 2012.

[65] George Danezis. *Introduction to Privacy Technology*. http://www0.cs.ucl.ac.uk/staff/G.Danezis/talks/Privacy_Technology_cosic.pdf. (Accessed on 05/29/2017). July 2011.

[66] National Vulnerabilities Database. *CVE-2014-9222*. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-9222. (Accessed on 02/02/2017). Dec. 2014.

[67] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. "An IoT gateway centric architecture to provide novel M2M services". In: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE. 2014, pp. 514–519.

[68] Jim Davies and Steve Schneider. "A brief history of Timed CSP". In: *Theoretical Computer Science* 138.2 (1995), pp. 243–271.

[69] Luciana Moreira Sá De Souza et al. "Socrades: A web service based shop floor integration infrastructure". In: *The internet of things* (2008), pp. 50–67.

[70] Harvey M Deitel. *An introduction to operating systems*. Vol. 3. Addison-Wesley Reading, Massachusetts, 1984.

[71] Mina Deng et al. "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements". In: *Requirements Engineering* 16.1 (2011), pp. 3–32.

[72] Heiko Desruelle et al. "On the challenges of building a web-based ubiquitous application platform". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM. 2012, pp. 733–736.

[73] Oxford English Dictionary. *Oxford English Dictionary Online*. Dec. 2017.

[74] Tim Dierks. *The transport layer security (TLS) protocol version 1.2*. Tech. rep. IETF, 2008.

[75] Docker. *Docker Compose - Docker Documentation*. https://docs.docker.com/compose/. (Accessed on 05/19/2017). May 2017.

[76] John R Douceur. "The sybil attack". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 251–260.

[77] Blake Dournaee and Blake Dournee. *XML security*. Mcgraw-hill, 2002.

[78] Clément Duhart, Pierre Sauvage, and Cyrille Bertelle. "A resource oriented framework for service choreography over wireless sensor and actor networks". In: *International Journal of Wireless Information Networks* 23.3 (2016), pp. 173–186.

[79] Adam Dunkels and Dogan Yazar. "Efficient application integration in IP-based sensor networks". In: *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM. 2009, pp. 43–48.

[80] Ed. E. Hammer-Lahav. *The OAuth 1.0 Protocol*. RFC 5849. Available at `http://tools.ietf.org/html/rfc5849`. IETF, Apr. 2010.

[81] D. Hardt (ed). *The OAuth 2.0 Authorization Framework*. RFC 6749. Available at `http://www.rfc-editor.org/rfc/rfc6749.txt`. IETF, Oct. 2012.

[82] Markus Eisenhauer, Peter Rosengren, and Pablo Antolin. "A development platform for integrating wireless devices and sensors into ambient intelligence systems". In: *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops' 09. 6th Annual IEEE Communications Society Conference on*. IEEE. 2009, pp. 1–3.

[83] George Eleftherakis et al. "Architecting the IoT paradigm: a middleware for autonomous distributed sensor networks". In: *International Journal of Distributed Sensor Networks* 11.12 (2015), p. 139735.

[84] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. "A middleware for the Internet of Things". In: *arXiv preprint arXiv:1604.04823* (2016).

[85] Shamini Emerson et al. "An OAuth based authentication mechanism for IoT networks". In: *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*. IEEE. 2015, pp. 1072–1074.

[86] ETSI. *ETSI - M2M*. `http://www.etsi.org/technologies-clusters/technologies/m2m`. (Visited on 07/08/2015). 2015.

[87] Patrick Th Eugster et al. "The many faces of publish/subscribe". In: *ACM Computing Surveys (CSUR)* 35.2 (2003), pp. 114–131.

[88] Dave Evans. "The internet of things". In: *How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG)* (2011).

[89] *Facebook Login*. URL: `https://developers.facebook.com/docs/facebook-login/`.

[90] Fitbit. *Fitbit Official Site for Activity Trackers & More*. http://www.fitbit.com/. (Visited on 07/09/2015). 2015.

[91] P. Fremantle and B. Aziz. "OAuthing: Privacy-enhancing federation for the Internet of Things". In: *2016 Cloudification of the Internet of Things (CIoT)*. Nov. 2016, pp. 1–6. DOI: 10.1109/CIOT.2016.7872911.

[92] Paul Fremantle. *A Reference Architecture for the Internet of Things*. Tech. rep. WSO2, 2014.

[96] Paul Fremantle. *Using OAuth 2.0 with MQTT*. http://pzf.fremantle.org/2013/11/using-oauth-20-with-mqtt.html. (Accessed on 02/13/2017). Nov. 2013.

[97] Paul Fremantle and Benjamin Aziz. "OAuthing: privacy-enhancing federation for the Internet of Things". In: *Proceedings of the 2nd International Conference on the Cloudification of the Internet of Things*. IEEE, 2016.

[99] Paul Fremantle, Benjamin Aziz, and Tom Kirkham. "Enhancing IoT Security and Privacy with Distributed Ledgers - A Position Paper". In: *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,* INSTICC. ScitePress, 2017, pp. 344–349. ISBN: 978-989-758-245-5. DOI: 10.5220/0006353903440349.

[100] Paul Fremantle, Benjamin Aziz, and Tom Kirkham. "Enhancing IoT Security and Privacy with Distributed Ledgers — a Position Paper". In: *IoTBDS 2017: 2nd International Conference on Internet of Things, Big Data and Security*. SCITEPRESS–Science and Technology Publications. 2017.

[101] Paul Fremantle, Jacek Kopecký, and Benjamin Aziz. "Web API Management Meets the Internet of Things". In: *The Semantic Web: ESWC 2015 Satellite Events: ESWC 2015 Satellite Events, Portorož, Slovenia, May 31 – June 4, 2015, Revised Selected Papers*. Ed. by Fabien Gandon et al. Cham: Springer International Publishing, 2015, pp. 367–375. ISBN: 978-3-319-25639-9. DOI: 10.1007/978-3-319-25639-9_49. URL: http://dx.doi.org/10.1007/978-3-319-25639-9_49.

[102] Paul Fremantle and Philip Scott. "A survey of secure middleware for the Internet of Things". In: *PeerJ Computer Science* 3 (2017), e114.

[103] Paul Fremantle et al. "Federated Identity and Access Management for the Internet of Things". In: *3rd International Workshop on the Secure IoT*. 2014.

[104] Theodoros Fronimos et al. "Unified Service-Oriented Access for WSNs and Dynamically Deployed Application Tasks". In: *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*. IEEE. 2016, pp. 247–252.

[105] K. Fullam and K.S. Barber. “Learning trust strategies in reputation exchange networks”. In: *AAMAS ’06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. Hakodate, Japan: ACM Press, 2006, pp. 1241–1248. ISBN: 1-59593-303-4.

[106] Steve B Furber. *ARM system Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996. ISBN: 0201403528.

[107] Flavio D Garcia et al. “Dismantling MIFARE classic”. In: *European Symposium on Research in Computer Security*. Springer. 2008, pp. 97–114.

[108] T Gibson-Robinson et al. “FDR3 — A Modern Refinement Checker for CSP”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Erika Ábrahám and Klaus Havelund. Vol. 8413. Lecture Notes in Computer Science. 2014, pp. 187–201.

[109] Len Gilman and Richard Schreiber. *Distributed computing with IBM MQSeries*. John Wiley & Sons, Inc., 1996.

[110] Daniel Giusto et al. *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010.

[111] Nenad Gligorić, Igor Dejanović, and Srđan Krčo. “Performance evaluation of compact binary XML representation for constrained devices”. In: *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE. 2011, pp. 1–5.

[112] Alex Glikson. “Fi-ware: Core platform for future internet applications”. In: *Proceedings of the 4th Annual International Conference on Systems and Storage*. 2011.

[113] Simon Godik et al. *OASIS eXtensible access control 2 markup language (XACML) 3*. Tech. rep. Tech. rep., OASIS, 2002.

[114] Berto Gomes et al. “A comprehensive cloud-based IoT software infrastructure for ambient assisted living”. In: *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*. IEEE. 2015, pp. 1–8.

[115] D Goodin. *New linux worm targets routers, cameras, internet of things devices*. 2013.

[116] Nils Gura et al. “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs”. In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 119–132. ISBN: 978-3-540-22666-6. DOI: 10.1007/978-3-540-28632-5_9. URL: http://dx.doi.org/10.1007/978-3-540-28632-5_9.

[117] DE Hammer-Lahav and D Hardt. *The OAuth2.0 authorization protocol. 2011*. Tech. rep. IETF Internet Draft, 2011.

[118] Eran Hammer-Lahav. *Hueniverse: Explaining OAuth*. https://web.archive.org/web/20071020052421/http://www.hueniverse.com:80/hueniverse/2007/09/explaining-oaut.html. (Accessed on 05/19/2017). Sept. 2007.

[119] Patrick Hanks. "Collins dictionary of the English language". In: *London: Collins,| c1986, 2nd ed., edited by Hanks, Patrick* 1 (1986).

[120] Thomas Hardjono, Ned Smith, and Alex Sandy Pentland. *Anonymous Identities for Permissioned Blockchains*. Tech. rep. MIT, 2014.

[121] Souleiman Hasan and Edward Curry. "Thingsonomy: Tackling Variety in Internet of Things Events". In: *Internet Computing, IEEE* 19.2 (2015), pp. 10–18.

[122] Randy Heffner. "The Forrester Wave™: API Management Solutions, Q3 2014". In: (2014).

[123] Marco E Pérez Hernández and Stephan Reiff-Marganiec. "Autonomous and self controlling smart objects for the future internet". In: *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE. 2015, pp. 301–308.

[124] José L Hernández-Ramos et al. "Distributed capability-based access control for the internet of things". In: *Journal of Internet Services and Information Security (JISIS)* 3.3/4 (2013), pp. 1–16.

[125] Kashmir Hill. *When 'Smart Homes' Get Hacked: I Haunted A Complete Stranger's House Via The Internet - Forbes*. http://www.forbes.com/sites/kashmirhill/2013/07/26/smart-homes-hack/. (Visited on 07/09/2015). 2013.

[126] C.A.R. Hoare. "Communicating sequential processes". In: *Commun. ACM* 21.8 (1978), pp. 666–677. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/359576.359585.

[127] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[128] Charles Antony Richard Hoare. "Communicating sequential processes". In: *The origin of concurrent programming*. Springer, 1978, pp. 413–443.

[129] P Hunt et al. *OAuth 2.0 Proof-of-Possession (PoP) Security Architecture*. Tech. rep. IETF, Jan. 2017.

[130] IETF. *Authentication and Authorization for Constrained Environments (ace) - Documents*. https://datatracker.ietf.org/wg/ace/documents/. (Accessed on 30th August 2016).

[131] Antti Iivari et al. "Harnessing XMPP for Machine-to-Machine Communications & Pervasive Applications." In: *Journal of Communications Software & Systems* 10.3 (2014).

[132] Yoshinao Isobe and Markus Roggenbach. "A generic theorem prover of CSP refinement". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2005, pp. 108–123.

[133] Daniel Jackson. "Alloy 3.0 reference manual". In: *Software Design Group* (2004).

[134] KR Jayaram, Patrick Eugster, and Chamikara Jayalath. "Parametric content-based publish/subscribe". In: *ACM Transactions on Computer Systems (TOCS)* 31.2 (2013), p. 4.

[135] Zhanlin Ji et al. "A cloud-based car parking middleware for IoT-based smart cities: design and implementation". In: *Sensors* 14.12 (2014), pp. 22372–22393.

[136] Casey Johnston. *What Google can really do with Nest, or really, Nest's data*. `https://arstechnica.com/business/2014/01/what-google-can-really-do-with-nest-or-really-nests-data/`. (Accessed on 05/28/2017). Jan. 2014.

[137] A. Jøsang, R. Ismail, and C. Boyd. "A Survey of Trust and Reputation Systems for Online Service Provision". In: *Decision Support Systems* 43.2 (Mar. 2007), pp. 618–644.

[138] *The application/json Media Type for JavaScript Object Notation (JSON)*. RFC 4627. Available at `http://www.rfc-editor.org/rfc/rfc4627.txt`. IETF, July 2006.

[139] Burt Kaliski. *RFC 2898; PKCS# 5: Password-Based Cryptography Specification Version 2.0*. 2000.

[140] Kantara Initiative. *User managed access (UMA)*. `https://kantarainitiative.org/confluence/display/uma/Home`. 2013.

[141] S Keoh, S Kumar, and O Garcia-Morchon. *Securing the IP-based Internet of Things with DTLS*. Tech. rep. draft-keoh-lwig-dtls-iot-02. Fremont, CA, USA: IETF, 2013.

[142] Himanshu Khurana et al. "Smart-grid security issues". In: *Security & Privacy, IEEE* 8.1 (2010), pp. 81–85.

[144] Andreas Kliem. "Cooperative Device Cloud". PhD thesis. Technischen Universitat Berlin, 2015.

[145] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[146] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. "Remote physical device fingerprinting". In: *IEEE Transactions on Dependable and Secure Computing* 2.2 (2005), pp. 93–108.

[147] Jacek Kopecký, Paul Fremantle, and Rich Boakes. "A history and future of Web APIs". In: *Information Technology* (2014).

[148] Manuel Koschuch, Matthias Hudler, and Michael Krüger. "Performance evaluation of the TLS handshake in the context of embedded devices". In: *Data Communication Networking (DCNET), Proceedings of the 2010 International Conference on*. IEEE. 2010, pp. 1–10.

[149] Peter Kostelnik, Martin Sarnovsk, and Karol Furdik. "The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain". In: *Scalable Computing: Practice and Experience* 12.3 (2011).

[150] Lina Lan et al. "An Event-driven Service-oriented Architecture for Internet of Things Service Execution". In: *International Journal of Online Engineering (iJOE)* 11.2 (2015), pp–4.

[151] Davy Landman. *DavyLandman/AESLib*. `https://github.com/DavyLandman/AESLib`. (Visited on 07/09/2015). 2015.

[152] Peter Gorm Larsen, Nico Plat, and Hans Toetenel. "A formal semantics of data flow diagrams". In: *Formal aspects of Computing* 6.6 (1994), pp. 586–606.

[153] Jeff Larson, Nicole Perlroth, and Scott Shane. *The NSA's Secret Campaign to Crack, Undermine Internet Encryption - ProPublica*. `http://www.propublica.org/article/the-nsas-secret-campaign-to-crack-undermine-internet-encryption`. (Visited on 06/08/2015). 2013.

[154] Thinh Le Vinh et al. "Middleware to integrate mobile devices, sensors and cloud computing". In: *Procedia Computer Science* 52 (2015), pp. 234–243.

[155] Rodger Lea. "HyperCat: an IoT interoperability specification". In: (2013).

[156] Shinho Lee et al. "Correlation analysis of MQTT loss and delay according to QoS level". In: *Information Networking (ICOIN), 2013 International Conference on*. IEEE. 2013, pp. 714–717.

[157] Tapio Levä, Oleksiy Mazhelis, and Henna Suomi. "Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications". In: *Decision Support Systems* 63 (2014), pp. 23–38.

[158] Tihamer Levendovszky et al. "Distributed real-time managed systems: A model-driven distributed secure information architecture platform for managed embedded systems". In: *Software, IEEE* 31.2 (2014), pp. 62–69.

[159] Ladar Levinson. *Secrets, lies and Snowden's email: why I was forced to shut down Lavabit*. `http://www.theguardian.com/commentisfree/2014/may/20/why-did-lavabit-shut-down-snowden-email`. (Visited on 06/08/2015). 2014.

[160] Léon Lim et al. "Enhancing context data distribution for the internet of things using qoc-awareness and attribute-based access control". In: *Annals of Telecommunications* 71.3-4 (2016), pp. 121–132.

[161] Linksmart. *LinkSmart Open Source Middleware*. `https://linksmart.eu/redmine/projects/linksmart-opensource/wiki/`. (Visited on 06/09/2015). 2015.

[162] Chi Harold Liu, Bo Yang, and Tiancheng Liu. "Efficient naming, addressing and profile services in Internet-of-Things sensory environments". In: *Ad Hoc Networks* 18 (2014), pp. 85–101.

[163] Yan Liu et al. "Designing a test suite for empirically-based middleware performance prediction". In: *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*. Australian Computer Society, Inc. 2002, pp. 123–130.

[164] Torsten Lodderstedt, Mark McGloin, and Phil Hunt. "OAuth 2.0 threat model and security considerations". In: (2013).

[165] Victor Lomne et al. "Side Channel Attacks". In: *Security Trends for FPGAS*. Springer, 2011, pp. 47–72.

[166] Thomas Luckenbach et al. "TinyREST-a protocol for integrating sensor networks into the internet". In: *Proceedings of the First REALWSN 2005 Workshop on Real-World Wireless Sensor Networks*. Stockholm, Sweden, 2005, pp. 101–105.

[167] Anil Madhavapeddy et al. "Jitsu: Just-In-Time Summoning of Unikernels." In: *NSDI*. 2015, pp. 559–573.

[168] Parikshit N Mahalle et al. "Identity establishment and capability based access control (IECAC) scheme for Internet of Things". In: *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*. IEEE. 2012, pp. 187–191.

[169] Patrick McDaniel and Stephen McLaughlin. "Security and privacy challenges in the smart grid". In: *Security & Privacy, IEEE* 7.3 (2009), pp. 75–77.

[170] Erhan Mengusoglu and Brian Pickering. "Automated management and service provisioning model for distributed devices". In: *Proceedings of the 2007 workshop on Automating service quality: Held at the International Conference on Automated Software Engineering (ASE)*. ACM. 2007, pp. 38–41.

[171] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2.

[172] Admire Mhlaba and Muthoni Masinde. "Implementation of middleware for Internet of Things in asset tracking applications: In-lining approach". In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE. 2015, pp. 460–469.

[173] Pietro Michiardi and Refik Molva. "Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks". In: *Advanced Communications and Multimedia Security*. New York, NY, USA: Springer, 2002, pp. 107–121.

[174] Victor S Miller. "Use of elliptic curves in cryptography". In: *Advances in Cryptology—CRYPTO'85 Proceedings*. Springer. 1986, pp. 417–426.

[175] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press, 1999.

[176] Rebecca Montanari, Alessandra Toninelli, and Jeffrey M Bradshaw. "Context-based security management for multi-agent systems". In: *Multi-Agent Security and Survivability, 2005 IEEE 2nd Symposium on*. IEEE. 2005, pp. 75–84.

[177] Thomas Morris. "Trusted platform module". In: *Encyclopedia of Cryptography and Security*. New York, NY, USA: Springer, 2011, pp. 1332–1335.

[178] Robert Moskowitz. *HIP Diet EXchange (DEX)*. Tech. rep. draft-ietf-hip-dex-05. Fremont, CA, USA: IETF, 2012.

[179] Robert Moskowitz. *Host identity protocol architecture*. Tech. rep. RFC 5201. Fremont, CA, USA: IETF, 2012.

[180] Aristides Mpitziopoulos et al. "A survey on jamming attacks and countermeasures in WSNs". In: *IEEE Communications Surveys & Tutorials* 11.4 (2009).

[181] *MQTT History*. URL: `http://mqtt.org/wiki/doku.php/history`.

[182] Chris Murphy. *Internet Of Things: Who Gets The Data? - InformationWeek*. `http://www.informationweek.com/strategic-cio/executive-insights-and-innovation/internet-of-things-who-gets-the-data/a/d-id/1252701`. (Visited on 06/08/2015). 2014.

[183] Suvda Myagmar, Adam J Lee, and William Yurcik. "Threat modeling as a basis for security requirements". In: *Symposium on requirements engineering for information security (SREIS)*. Vol. 2005. Citeseer. 2005, pp. 1–8.

[184] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system, 2008". In: *URL: http://www. bitcoin. org/bitcoin.pdf* (2012).

[185] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE. 2008, pp. 111–125.

[186] Anton Naumenko, Artem Katasonov, and Vagan Terziyan. "A security framework for smart ubiquitous industrial resources". In: *Enterprise Interoperability II*. New York, NY, USA: Springer, 2007, pp. 183–194.

[187] Google Nest. *How to keep your Nest products and the Nest app up to date*. https://nest.com/support/article/How-to-keep-your-Nest-products-and-the-Nest-app-up-to-date. (Accessed on 02/13/2017). Feb. 2017.

[188] James Newsome et al. "The sybil attack in sensor networks: analysis & defenses". In: *Proceedings of the 3rd international symposium on Information processing in sensor networks*. ACM. 2004, pp. 259–268.

[189] Muan Yong Ng and Michael Butler. "Tool support for visualizing CSP in UML". In: *International Conference on Formal Engineering Methods*. Springer. 2002, pp. 287–298.

[190] Muan Yong Ng and Michael Butler. "Towards formalizing UML state diagrams in CSP". In: *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*. IEEE. 2003, pp. 138–147.

[191] S Nicholas. *Power Profiling: HTTPS Long Polling vs. MQTT with SSL on Android (2012)*. http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https. 2012.

[192] Aimaschana Niruntasukrat et al. "Authorization mechanism for MQTT-based Internet of Things". In: *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE. 2016, pp. 290–295.

[193] et al O. Garcia-Morchon. *Security Considerations in the IP-based Internet of Things*. Internet Draft. Available at http://tools.ietf.org/html/draft-garcia-core-security-06. IETF, Sept. 2013.

[194] Zooko Wilcox O'Hearn. *Names: Distributed, Secure, Human-Readable: Choose Two*. https://web.archive.org/web/20011020191610/http://zooko.com/distnames.html. (Accessed on 02/13/2017). Oct. 2001.

[195] Nicholas O'Leary. *Really Small Message Broker*. https://github.com/mqtt/mqtt.github.io/wiki/Really-Small-Message-Broker. (Accessed on 05/19/2017). May 2017.

[196] Suhas Pai et al. "Formal verification of OAuth 2.0 using Alloy framework". In: *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*. IEEE. 2011, pp. 655–659.

[197] Namje Park and Namhi Kang. "Mutual Authentication Scheme in Secure Internet of Things Technology for Comfortable Lifestyle". In: *Sensors* 16.1 (2015). ISSN: 1424-8220. URL: http://www.mdpi.com/1424-8220/16/1/20.

[198] Ki-Woong Park, Hyunchul Seok, and Kyu-Ho Park. "pKASSO: towards seamless authentication providing non-repudiation on resource-constrained devices". In: *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*. Vol. 2. IEEE. 2007, pp. 105–112.

[199] Edoardo Patti et al. “Event-driven user-centric middleware for energy-efficient buildings and public spaces”. In: *IEEE Systems Journal* 10.3 (2016), pp. 1137–1146.

[200] Andrew Paverd and Andrew Martin. “Hardware Security for Device Authentication in the Smart Grid”. In: *First Open EIT ICT Labs Workshop on Smart Grid Security - SmartGridSec12*. Berlin, Germany, 2012. URL: `http://link.springer.com/chapter/10.1007/978-3-642-38030-3_5`.

[201] Vladislav Perelman and Mehmet Ersue. *TLS with PSK for Constrained Devices*. Tech. rep. Citeseer, 2012.

[202] Charith Perera and Athanasios V Vasilakos. “A knowledge-based resource discovery for Internet of Things”. In: *Knowledge-Based Systems* 109 (2016), pp. 122–136.

[203] Charith Perera et al. “Mosden: An internet of things middleware for resource constrained mobile devices”. In: *System Sciences (HICSS), 2014 47th Hawaii International Conference on*. IEEE. 2014, pp. 1053–1062.

[204] Julio Perez. “MQTT Performance Analysis with OMNeT++”. In: *Master’s Thesis, September* (2005).

[205] Adrian Perrig et al. “SPINS: Security protocols for sensor networks”. In: *Wireless networks* 8.5 (2002), pp. 521–534.

[206] Charles P Pfleeger and Shari Lawrence Pfleeger. *Security in computing*. Upper Saddle River, NJ, USA: Prentice Hall Professional Technical Reference, 2002.

[207] Linh Manh Pham et al. “CIRUS: an elastic cloud-based framework for Ubilytics”. In: *Annals of Telecommunications* 71.3-4 (2016), pp. 133–140.

[208] Check Point. *CPAI-2014-2294 Misfortune Cookie*. `https://www.checkpoint.com/defense/advisories/public/2014/cpai-2014-2294.html`. (Accessed on 02/13/2017). Dec. 2014.

[209] Neeli Prasad. *Aspire Project*. `http://www.fp7-aspire.eu/`. (Accessed on 04/21/2017). 2008.

[210] MJ Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education, 2004.

[211] Saša Radomirovic. “Towards a Model for Security and Privacy in the Internet of Things”. In: *Proc. First Int’l Workshop on Security of the Internet of Things*. 2010.

[212] Dave Raggett. “COMPOSE: An Open Source Cloud-Based Scalable IoT Services Platform”. In: *ERCIM News* 101 (2015), pp. 30–31.

[213] Yrjö Raivio, Sakari Luukkainen, and Saku Seppala. “Towards Open Telco-Business models of API management providers”. In: *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE. 2011, pp. 1–11.

[214] Gowri Sankar Ramachandran et al. “Hitch hiker: A remote binding model with priority based data aggregation for wireless sensor networks”. In: *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*. ACM. 2015, pp. 43–48.

[215] Mohammad Abdur Razzaque et al. “Middleware for internet of things: a survey”. In: *IEEE Internet of Things Journal* 3.1 (2016), pp. 70–95.

[216] Adam Rendle. *Who owns the data in the Internet of Things?* `http://www.taylorwessing.com/download/article\_data\_lot.html`. (Visited on 06/08/2015). 2014.

[217] Thomas Renner, Andreas Kliem, and Odej Kao. “The device cloud-applying cloud computing concepts to the internet of things”. In: *Ubiquitous Intelligence and Computing, 2014 IEEE 11th Intl Conf on and IEEE 11th Intl Conf on and Autonomic and Trusted Computing, and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UTC-ATC-ScalCom)*. IEEE. 2014, pp. 396–401.

[218] Eric Rescorla and Nagendra Modadugu. *Datagram transport layer security*. Tech. rep. IETF, 2006.

[219] J Richer. *OAuth 2.0 Token Introspection*. Tech. rep. 2015.

[220] Justin Richer. “OAuth Token Introspection”. In: (2013).

[221] Justin Richer, Dazza Greenwood, and Bruce Bakis. “Componentization of Security Principles”. In: *Symposium on Usable Privacy and Security (SOUPS)*. 2014.

[222] J Richer et al. *OAuth 2.0 Dynamic Client Registration Protocol*. Tech. rep. 2015.

[223] Ronald L Rivest, Adi Shamir, and Len Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[224] Andrew William Roscoe. *Understanding concurrent systems*. Springer Science & Business Media, 2010.

[225] AW Roscoe. “CSP is expressive enough for π”. In: *Reflections on the Work of CAR Hoare*. Springer, 2010, pp. 371–404.

[226] Domenico Rotondi, Cristoforo Seccia, and Salvatore Piccione. “Access control & iot: Capability based authorization access control system”. In: *1st IoT International Forum*. Berlin, Germany, 2011.

[227] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[228] Mike Ryan. "Bluetooth: With Low Energy Comes Low Security." In: *WOOT: Proceedings of the 7th USENIX Workshop of Offensive Technologies*. USENIX. 2013.

[229] Ahmad-Reza Sadeghi and Christian Stüble. "Property-based attestation for computing platforms: caring about properties, not mechanisms". In: *Proceedings of the 2004 workshop on New security paradigms*. ACM. 2004, pp. 67–77.

[230] Peter Saint-Andre. *Extensible messaging and presence protocol (XMPP): Core*. Tech. rep. RFC 6120. Fremont, CA, USA: IETF, 2011.

[231] N Sakimura, J Bradley, and M Jones. *Final: OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1*. 2015. URL: `http://openid.net/specs/openid-connect-registration-1_0.html` (visited on 03/20/2015).

[232] N Sakimura, J Bradley, and M Jones. *OpenID Connect Dynamic Client Registration 1.0-Draft 14*. 2013.

[233] Samsung. *Mobile Enterprise Security | Samsung KNOX*. `https://www.samsungknox.com/en`. (Visited on 03/24/2015). 2015.

[234] Ravi S Sandhu and Pierangela Samarati. "Access control: principle and practice". In: *Communications Magazine, IEEE* 32.9 (1994), pp. 40–48.

[235] Steve Schneider, Helen Treharne, and Heike Wehrheim. "A CSP account of Event-B refinement". In: *arXiv preprint arXiv:1106.4098* (2011).

[236] Bruce Schneier. "iPhone encryption and the return of the crypto wars". In: *Schneier on Security* 6 (2014), p. 2014.

[237] Bruce Schneier. *Tracking Vehicles through Tire Pressure Monitors*. `https://www.schneier.com/blog/archives/2008/04/tracking_vehicl.html`. (Accessed on 02/13/2017). Apr. 2008.

[238] Vasile-Marian Scuturici et al. "UbiWare: Web-based dynamic data & service management platform for AmI". In: *Proceedings of the Posters and Demo Track*. ACM. 2012, p. 11.

[239] Ronny Seiger, Steffen Huber, and Thomas Schlegel. "Toward an execution system for self-healing workflows in cyber-physical systems". In: *Software & Systems Modeling* (2016), pp. 1–22.

[240] Sengul and Kirby. *MQTT-TLS profile of ACE*. `https://datatracker.ietf.org/doc/draft-sengul-ace-mqtt-tls-profile/`. (Accessed on 02/13/2017). Jan. 2017.

[241] Arvind Seshadri et al. “Swatt: Software-based attestation for embedded devices”. In: *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE. 2004, pp. 272–282.

[242] Mohit Sethi. “Security in Smart Object Networks”. MA thesis. Aalto University, School of Science, 2012.

[243] Mohit Sethi, Jari Arkko, and Ari Keranen. “End-to-end security for sleepy smart object networks”. In: *Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on*. IEEE. 2012, pp. 964–972.

[244] Zach Shelby, Klaus Hartke, and Carsten Bormann. *Constrained Application Protocol (CoAP) draft-ietf-core-coap-18*. `https://tools.ietf.org/html/draft-ietf-core-coap-18`. Accessed: 2015-03-13.

[245] Peter W Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332.

[246] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[247] Sabrina Sicari et al. “A secure and quality-aware prototypical architecture for the Internet of Things”. In: *Information Systems* 58 (2016), pp. 43–55.

[248] Sabrina Sicari et al. “Security policy enforcement for networked smart objects”. In: *Computer Networks* 108 (2016), pp. 133–147.

[249] Gheorghe Cosmin Silaghi, Alvaro Arenas, and Luis Moura Silva. *Reputation-based trust management systems and their applicability to grids*. Tech. rep. TR-0064. Institutes on Knowledge, Data Management, and System Architecture, CoreGRID - Network of Excellence, Feb. 2007.

[250] Andrew Simmonds, Peter Sandilands, and Louis Van Ekert. “An ontology for network security attacks”. In: *Applied Computing*. New York, NY, USA: Springer, 2004, pp. 317–323.

[251] Sergei Skorobogatov. “The bumpy road towards iPhone 5c NAND mirroring”. In: *arXiv preprint arXiv:1609.04327* (2016).

[252] Sergei Petrovich Skorobogatov. “Semi-invasive attacks: a new approach to hardware security analysis”. PhD thesis. Citeseer, 2005.

[253] Martin et al Gudgin. *SOAP Version 1.2 Part 1: Messaging Framework*. Recommendation. Available at `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`. W3C, June 2003.

[254] Sergios Soursos et al. “Towards the cross-domain interoperability of IoT platforms”. In: *Networks and Communications (EuCNC), 2016 European Conference on*. IEEE. 2016, pp. 398–402.

[255] Sarah Spiekermann and Lorrie Faith Cranor. “Engineering privacy”. In: *IEEE Transactions on software engineering* 35.1 (2009), pp. 67–82.

[256] Andy J Stanford-Clark and Glenn R Wightwick. “The application of publish/subscribe messaging to environmental, monitoring, and control systems”. In: *IBM Journal of Research and Development* 54.4 (2010), pp. 1–7.

[257] Michael Steil. “17 mistakes Microsoft made in the Xbox security system”. In: *Proceedings of the 22nd Chaos Communication Congress*. Chaos Communication Club. 2005.

[258] David Svensson Fors et al. “Ad-hoc composition of pervasive services in the PalCom architecture”. In: *Proceedings of the 2009 international conference on Pervasive services*. ACM. 2009, pp. 83–92.

[259] Frank Swiderski and Window Snyder. *Threat modeling*. Microsoft Press, 2004.

[260] Tomasz Tajmajer et al. “Node/Proxy portability: Designing for the two lives of your next WSAN middleware”. In: *Journal of Systems and Software* 117 (2016), pp. 366–383.

[261] Luis E Talavera et al. “The mobile hub concept: Enabling applications for the internet of mobile things”. In: *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*. IEEE. 2015, pp. 123–128.

[262] TCG. *Trusted Computing Group - Home*. `http://www.trustedcomputinggroup.org/`. (Visited on 06/08/2015). 2015.

[263] Vagan Terziyan, Olena Kaykova, and Dmytro Zhovtobryukh. “Ubiroad: Semantic middleware for context-aware smart road environments”. In: *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. IEEE. 2010, pp. 295–302.

[264] Dinesh Thangavel et al. “Performance evaluation of MQTT and CoAP via a common middleware”. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE. 2014, pp. 1–6.

[265] Johannes Thönes. “Microservices”. In: *IEEE Software* 32.1 (2015), pp. 116–116.

[266] Lynn Thorndike. “Copyists’ Final Jingles in Mediaeval Manuscripts”. In: *Speculum* 12.2 (1937), pp. 268–268. DOI: `10.2307/2849582`. eprint: `http://dx.doi.org/10.2307/2849582`. URL: `http://dx.doi.org/10.2307/2849582`.

[267] Stefan Tilkov and Steve Vinoski. “Node. js: Using JavaScript to build high-performance network programs”. In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83.

[268] Ken Tindall. *How Bitcoin might fix the broken Internet of Things*. `https://freo.me/2jNZRBm`. (Accessed on 01/20/2017). Mar. 2015.

[269] Jan Tretmans. "Model based testing with labelled transition systems". In: *Formal methods and testing* (2008), pp. 1–38.

[270] H Tschofenig and T Fossati. *A TLS/DTLS 1.2 Profile for the Internet of Things*. Tech. rep. RFC 7925. Fremont, CA, USA: IETF, 2016.

[271] Hannes Tschofenig. "Fixing User Authentication for the Internet of Things (IoT)". In: *Datenschutz und Datensicherheit-DuD* 40.4 (2016), pp. 222–224.

[272] H Tschofenig et al. *Authentication and Authorization for Constrained Environments Using OAuth and UMA*. Tech. rep. IETF, 2015.

[273] Fatih Turkmen and Bruno Crispo. "Performance evaluation of XACML PDP implementations". In: *Proceedings of the 2008 ACM workshop on Secure web services*. ACM. 2008, pp. 37–44.

[274] Nikos Tziritas et al. "Middleware mechanisms for agent mobility in wireless sensor and actuator networks". In: *International Conference on Sensor Systems and Software*. Springer. 2012, pp. 30–44.

[275] Ioan Ungurean, Nicoleta Cristina Gaitan, and Vasile Gheorghita Gaitan. "A Middleware Based Architecture for the Industrial Internet of Things". In: *KSII Transactions on Internet and Information Systems (TIIS)* 10.7 (2016), pp. 2874–2891.

[276] European Union. *Reform of EU data protection rules - European Commission*. `http://ec.europa.eu/justice/data-protection/reform/index_en.htm`. Apr. 2016.

[277] University of Portsmouth Library. *Discovery Service*. `http://www.port.ac.uk/library/infores/discovery/filetodownload,170883,en.xls`. (Visited on 07/14/2015). 2015.

[278] Rafael Oliveira Vasconcelos et al. "An adaptive middleware for opportunistic mobile sensing". In: *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*. IEEE. 2015, pp. 1–10.

[279] Antonio Garćıa Vázquez et al. "FI-WARE security: future internet security core". In: *European Conference on a Service-Based Internet*. Springer. 2011, pp. 144–152.

[280] Marcos Augusto M Vieira et al. "Survey on wireless sensor network devices". In: *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*. Vol. 1. IEEE. 2003, pp. 537–544.

[281] James Vincent. *London's bins are tracking your smartphone*. http://www.independent.co.uk/life-style/gadgets-and-tech/news/updated-londons-bins-are-tracking-your-smartphone-8754924.html. (Accessed on 02/13/2017). Aug. 2013.

[282] Ronald Watro et al. "TinyPK: securing sensor networks with public key technology". In: *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. ACM. 2004, pp. 59–64.

[283] Jialu Wei. *DDoS on Internet of Things–a big alarm for the future*. http://www.cs.tufts.edu/comp/116/archive/fall2016/jwei.pdf. 2016.

[284] Jenifer S Winter. "Privacy and the emerging internet of things: using the framework of contextual integrity to inform policy". In: *Pacific Telecommunication Council Conference Proceedings 2012*. 2012.

[285] Jim Woodcock and Jim Davies. *Using Z: specification, refinement, and proof*. Vol. 39. Prentice Hall Englewood Cliffs, 1996.

[286] WSO2. *WSO2 ESB Performance Round 7.5*. http://wso2.com/library/articles/2014/02/esb-performance-round-7.5/. (Accessed on 02/13/2017). Feb. 2014.

[287] *WSO2 API Manager - 100% Open Source API Management Platform | WSO2 Inc*. URL: http://wso2.com/products/api-manager/ (visited on 03/20/2015).

[288] *WSO2 Identity Server | WSO2 Inc*. (Visited on 11/13/2013). URL: http://wso2.com/products/identity-server/.

[289] *Xively by LogMeIn – Business Solutions for the Internet of Things*. URL: https://xively.com/ (visited on 03/20/2015).

[290] Tim et al Bray. *Extensible Markup Language (XML) 1.0*. Recommendation. Available at http://www.w3.org/TR/REC-xml. W3C, Feb. 2004.

[291] Dong Xu et al. "Towards formalizing UML activity diagrams in CSP". In: *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*. Vol. 2. IEEE. 2008, pp. 450–453.

[292] Xiaobin Xu, Minbo Li, and Jia Liang. "A Middleware for Environmental Monitoring and Control". In: *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE. 2016, pp. 697–704.

[293] Song Y Yan. "Side-Channel Attacks". In: *Cryptanalytic Attacks on RSA*. New York, NY, USA: Springer, 2008, pp. 207–222.

[294] Jamie Yap. *450,000 user passwords leaked in Yahoo breach*. 2012.

[295] Seung Yi and Robin Kravets. "Key management for heterogeneous ad hoc wireless networks". In: *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*. IEEE. 2002, pp. 202–203.

[296] Jaeseok Yun et al. "A device software platform for consumer electronics based on the Internet of Things". In: *IEEE Transactions on Consumer Electronics* 61.4 (2015), pp. 564–571.

[297] Zee. *Fitbit users are unwittingly sharing details of their sex lives with the world*. (Visited on 06/04/2013). 2011. URL: `http://thenextweb.com/insider/2011/07/03/fitbit-users-are-inadvertently-sharing-details-of-their-sex-lives-with-the-world/`.

[298] *Zetta - An API-First Internet of Things (IoT) Platform - Free and Open Source Software*. URL: `http://www.zettajs.org/` (visited on 03/20/2015).

[299] Wang Zhiliang et al. "A SOA based IOT communication middleware". In: *Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on*. IEEE. 2011, pp. 2555–2558.

[300] Qian Zhu et al. "IoT gateway: Bridging wireless sensor networks into internet of things". In: *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE. 2010, pp. 347–352.

[301] Charikleia Zouridaki et al. "E-Hermes: A robust cooperative trust establishment scheme for mobile ad hoc networks". In: *Ad Hoc Networks* 7.6 (2009), pp. 1156–1168.

[302] Charikleia Zouridaki et al. "Hermes: A quantitative trust establishment framework for reliable data packet delivery in MANETs". In: *Journal of Computer Security* 15.1 (2007), pp. 3–38.

Explicit hoc totem, pro Christo
da mihi potem[266]

Appendix — Ethics Review Forms

FORM UPR16

Research Ethics Review Checklist

University of Portsmouth

Please include this completed form as an appendix to your thesis (see the Postgraduate Research Student Handbook for more information

Postgraduate Research Student (PGRS) Information				**Student ID:**	29713831
PGRS Name:	Paul Zachary Fremantle				
Department:	Computing	**First Supervisor:**	Dr. Benjamin Aziz		
Start Date: (or progression date for Prof Doc students)	**October 2013**				
Study Mode and Route:	Part-time ☒ Full-time ☐	MPhil ☐ PhD ☒	MD ☐ Professional Doctorate ☐		

Title of Thesis:	An Approach to Enhancing Security and Privacy of the Internet of Things with Federated Identity
Thesis Word Count: (excluding ancillary data)	46,548

If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study

Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).

UKRIO Finished Research Checklist: (If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: http://www.ukrio.org/what-we-do/code-of-practice-for-research/)	
a) Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES ☒ NO ☐
b) Have all contributions to knowledge been acknowledged?	YES ☒ NO ☐
c) Have you complied with all agreements relating to intellectual property, publication and authorship?	YES ☒ NO ☐
d) Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES ☒ NO ☐
e) Does your research comply with all legal, ethical, and contractual requirements?	YES ☒ NO ☐

Candidate Statement:	
I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)	
Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):	3BBD-77DF-5631-D442-58A9-DB61-2699-8AAF
If you have *not* submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:	
N/A	
Signed *(PGRS)*: Paul Zachary Fremantle	**Date:** 1/6/2017

University of Portsmouth

Certificate of Ethics Review

Project Title:	PhD thesis - a secure private middleware for IoT
User ID:	528399
Name:	Paul Fremantle
Application Date:	19/01/2016 14:27:33

You must download your certificate, print a copy and keep it as a record of this review.

It is your responsibility to adhere to the University Ethics Policy and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers and University Health and Safety Policy.

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

You are reminded that as a University of Portsmouth Researcher you are bound by the UKRIO Code of Practice for Research; any breach of this code could lead to action being taken following the University's Procedure for the Investigation of Allegations of Misconduct in Research.

Any changes in the answers to the questions reflecting the design, management or conduct of the research over the course of the project must be notified to the Faculty Ethics Committee. **Any changes that affect the answers given in the questionnaire, not reported to the Faculty Ethics Committee, will invalidate this certificate.**

This ethical review should not be used to infer any comment on the academic merits or methodology of the project. If you have not already done so, you are advised to develop a clear protocol/proposal and ensure that it is independently reviewed by peers or others of appropriate standing. A favourable ethical opinion should not be perceived as permission to proceed with the research; there might be other matters of governance which require further consideration including the agreement of any organisation hosting the research.

GovernanceChecklist
A1-BriefDescriptionOfProject: I am researching secure private middleware for the Internet of Things. This involves creating abstract models of security and privacy properties and ways to manage them.
A2-Faculty: Technology
A3-VoluntarilyReferToFEC: No
A5-AlreadyExternallyReviewed: No
B1-HumanParticipants: No
HumanParticipantsDefinition
B2-HumanParticipantsConfirmation: Yes

Certificate Code: 3BBD-77DF-5631-D442-58A9-DB61-2699-8AAF Page 1