FLAT: FEDERATED LIGHTWEIGHT AUTHENTICATION OF THINGS

MARIA LUIZA BULGARELLI ALVES DOS SANTOS

FLAT: FEDERATED LIGHTWEIGHT AUTHENTICATION OF THINGS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Leonardo Barbosa e Oliveira
Coorientador: Marco Aurélio Amaral Henriques

Belo Horizonte
Julho de 2018

MARIA LUIZA BULGARELLI ALVES DOS SANTOS

FLAT: FEDERATED LIGHTWEIGHT AUTHENTICATION OF THINGS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Leonardo Barbosa e Oliveira
Co-Advisor: Marco Aurélio Amaral Henriques

Belo Horizonte
July 2018

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Santos, Maria Luiza Bulgarelli Alves dos.

S237f FLAT: federated lightweight authentication of things. / Maria Luiza Bulgarelli Alves dos Santos. — Belo Horizonte, 2018.
xxix, 70 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Leonardo Barbosa e Oliveira.
Coorientador: Marco Aurélio Amaral Henriques.

1. Computação – Teses. 2. Internet das coisas – Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

FLAT: Federated Lightweight Authentication of Things

MARIA LUIZA BULGARELLI ALVES DOS SANTOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LEONARDO BARBOSA E OLIVEIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. MARCO AURELIO AMARAL HENRIQUES - Coorientador
Departamento de Engenharia Computação Automação Industrial - Unicamp

PROF. JEAN EVERSON MARTINA
Departamento de Informática de de Estatística - UFSC

PROF. JOÃO GUILHERME MAIA DE MENEZES
Departamento de Ciência da Computação - UFMG

PROF. DANIEL FERNANDES MACEDO
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 13 de julho de 2018.

To my beloved parents, Mary and Ronald, the greatest supporters in all my projects and dreams.

Acknowledgments

Although the language of this document is English, my acknowledgments will be written mostly in Portuguese, as I have to give my thanks to Portuguese and English speakers who supported and helped me during my Master studies.

Agradeço a Deus, por me proporcionar grandes oportunidades e momentos em minha vida e por sempre me guiar para os melhores caminhos. Agradeço e sempre agradecerei aos meus pais, Mary e Ronald, por todo carinho e amor dedicados a mim, e pelo grande incentivo em meus estudos e sonhos. A presença de vocês é um presente em minha vida. Agradeço à minha irmã Mariana pelo carinho, cuidado, incentivo e pelas conversas de sempre. I thank Viliam for giving me support and love, even from far.

Agradeço ao meu orientador Leo, por todo o apoio e generosidade durante o Mestrado. Aprendi muito convivendo e trabalhando com você Leo, e agradeço pelos numerosos conselhos e pelo seu imenso profissionalismo e empatia. Agradeço ao meu coorientador Marco pelas diversas sugestões, conselhos, contribuições, revisões de artigos e pelo apoio na conclusão deste trabalho. Agradeço ao Teixeira pelos diversos conselhos, conversas, contribuições e por todo o apoio durante o projeto. Agradeço aos meus parceiros de GT-CoFee e grandes contribuidores neste trabalho, Jéssica e Franco, por todas as contribuições, conversas e conselhos no decorrer de todo o projeto. Ao Pedro, Artur e Tonin pelas contribuições, conversas, sugestões e pelos produtivos projetos em que trabalhamos juntos.

A todo o pessoal do Wisemap, Sybers e aos funcionários do DCC por toda a ajuda e apoio durante estes anos do Mestrado. À Roberta, Gabriel e Lucas, pelas conversas e almoços juntos. Agradeço também a todos os amigos que fiz durante meus estudos na UFMG.

Gostaria de agradecer à RNP e ao CT-GId, pelas diversas sugestões e contribuições no decorrer deste trabalho e pelo apoio no andamento do projeto. Ao Emerson, Michelle, Clayton e André pelo apoio, críticas e sugestões valiosas. Agradeço novamente à RNP pelo apoio financeiro ao projeto. O presente trabalho foi realizado

com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Finalmente, agradeço a todos os meus amigos e família pelo apoio e a todos aqueles que de alguma maneira contribuíram para a conclusão deste trabalho.

Resumo

A Internet das Coisas (*Internet of Things* – IoT), por meio de novas aplicações e tecnologias, tem modificado a maneira como negócios e pessoas interagem. O crescimento da IoT, entretanto, vem acompanhado de diversos desafios. Dentre estes desafios, um aspecto crítico em IoT é a autenticação de dispositivos e o controle de acesso dos mesmos aos recursos disponíveis na rede. A Gestão de Identidades (*Identity Management* – IdM) provê meios para gerir as identidades de dispositivos e usuários, sendo também responsável pelas tarefas de autenticação e autorização. Neste sentido, é essencial que exista um modelo de IdM para IoT que contemple as características específicas deste contexto, especialmente quando consideramos as restrições computacionais e de armazenamento dos dispositivos e sua potencial mobilidade entre diferentes domínios. Atualmente, soluções amplamente utilizadas em IdM são baseadas em criptografia assimétrica, e, portanto, requerem maior processamento e armazenamento dos dispositivos, o que não é uma abordagem desejável em dispositivos restritos, comumente encontrados em cenários IoT. Como solução para este problema, é proposto o FLAT, um protocolo de autenticação federada para IoT. FLAT associa: (i) o uso de apenas criptossistemas simétricos no Cliente, (ii) a substituição de criptossistemas tradicionais como o RSA/DSA por criptossistemas equivalentes baseados em curvas elípticas, e (iii) o uso de certificados implícitos, fornecendo uma alternativa leve para autenticação de dispositivos restritos. FLAT é comparado com um protocolo de referência baseado em soluções tradicionais para IdM (solução Baseline). Os resultados mostram que FLAT é capaz de reduzir o total de dados transmitidos em cerca de 31% em relação à solução Baseline. O dispositivo Cliente no FLAT também é mais eficiente do que a solução Baseline em termos de dados transmitidos, dados recebidos, total de dados trocados e tempo de computação. FLAT oferece uma alternativa para cenários IoT em que seja necessária a autenticação de dispositivos de diferentes domínios, podendo ser executado mesmo em dispositivos com restrições de processamento e armazenamento.

Palavras-chave: Internet das Coisas, Autenticação, Gestão de Identidades Federada.

Abstract

The Internet of Things (IoT) applications and technologies have been modifying the way people and businesses interact. The IoT growth, however, is followed by several challenges. Among them, a critical aspect in IoT is the authentication of devices and its access control to the available network resources. The Identity Management (IdM) provide means to manage the identities of devices and users, being also responsible for the authentication and authorization tasks. In this sense, it is essential to IoT the development of an IdM model that contemplates the specific characteristics of this context, especially considering the computational and storage restrictions of devices and their potential mobility between different domains. Nowadays, widely used IdM solutions are based on asymmetric cryptography, and thus, require more computation and storage from devices, which is not a desirable approach in restricted devices, commonly found in IoT scenarios. As a solution to this problem, it is proposed FLAT, a federated authentication protocol for IoT. FLAT associates: (i) the use of only symmetric cryptosystems in the Client side, (ii) the replacement of traditional cryptosystems such as RSA/DSA by equivalent cryptosystems based on elliptic curves, and (iii) the use of implicit certificates, providing a lightweight solution for authenticating restricted devices. FLAT is compared to a reference protocol based on traditional IdM approaches (Baseline solution). The results show that FLAT can reduce the data exchange overhead in around 31% when compared to the Baseline solution. FLAT's Client is also more efficient than the Baseline solution in terms of data transmitted, data received, total data exchange, and computation time. FLAT offers an alternative to IoT scenarios where the device authentication between different domains is necessary and can be executed even in devices with computational and storage restrictions.

Keywords: Internet of Things, Authentication, Federated Identity Management.

List of Figures

List of Tables

List of Protocols

Acronyms

ABAC Attribute Based Access Control.

AES Advanced Encryption Standard.

ANSI American National Standards Institute.

AS Authentication Server – Kerberos Server.

Baseline Reference protocol based on a traditional FIdM approach.

BLS Boneh-Lynn-Shacham.

BNDES *Banco Nacional de Desenvolvimento Econômico e Social*.

CA Certificate Authority.

CSRF Cross-Site Request Forgery attack.

DS Discovery Service.

DSA Digital Signature Algorithm.

ECC Elliptic Curve Cryptography.

ECDH Elliptic-curve Diffie–Hellman.

ECDSA Elliptic Curve Digital Signature Algorithm.

ECIES Elliptic Curve Integrated Encryption Scheme.

ECQV Elliptic Curve Qu-Vanstone.

EEPROM Electrically-Erasable Programmable Read-Only Memory.

FIdM Federated Identity Management.

FLAT Federated Lightweight Authentication of Things.

HMAC Hashed Message Authentication Code.

HTTP HyperText Transfer Protocol.

IBC Identity-Based Cryptography.

ICEx Instituto de Ciências Exatas.

IdM Identity Management.

IdP Identity Provider.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoT Internet of Things.

ISO International Organization for Standardization.

JSON JavaScript Object Notation.

JWE JSON Web Encryption.

JWS JSON Web Signature.

KC Kerberos Client.

KDC Key Distribution Center.

MAC Message Authentication Code.

NIST National Institute of Standards and Technology.

PDP Policy Decision Point.

PEP Policy Enforcement Point.

PUFs Physical Unclonable Functions.

RBAC Role-Based Access Control.

RFC Request for Comments.

RSA Rivest, Shamir, & Adleman Algorithm.

SAML Security Assertion Markup Language.

SECG Standards for Efficient Cryptography Group.

SP Service Provider.

SRAM Static Random Access Memory.

SSL Secure Sockets Layer.

SSO Single-Sign-On.

TA Trusted Authorities.

TGS Ticket Granting Server.

TLS Transport Layer Security.

UFMG Universidade Federal de Minas Gerais.

UFSJ Universidade Federal de São João Del-Rei.

WAYF Where Are You From.

XACML Extensible Access Control Markup Language.

XML Extensible Markup Language.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) [Atzori et al., 2010] is a research field that has grown significantly in the last years and has been causing a great impact on our daily lives. We are already surrounded by billions of connected devices (Fig. 1.1), more precisely around 11 billion in 2018, and studies show we will have around 20 billion IoT devices in 2020 [van der Meulen, 2017]. These devices come in several different applications, such as heartbeat monitoring, glasses that show information along the day, temperature and humidity monitoring in agriculture and even smartwatches that show notifications received in the user's smartphone.

Particularly, efforts have been made to develop and deploy IoT technologies in Brazil. The Brazilian Development Bank – *Banco Nacional de Desenvolvimento Econômico e Social* (BNDES) has released in 2017 a support to a study regarding the establishment of a strategic plan to promote the development of IoT applications in the country [BNDES, 2017]. With many possible uses, IoT remains a relevant topic in both academia and industry.

As IoT rapidly grows, however, it also brings several challenges regarding security and privacy. One of these challenges is the Identity Management (IdM) of users and devices in the IoT environment. Precisely, IdM deals with managing and controlling the identification (the process of authenticating the identity of a user [Menezes et al., 2001]) of users/devices in a given system. IdM, ideally, controls the creation, management, use and exclusion of identities (that identify people and things, in general), in order to allow/authorize certain actions [Windley, 2005]. IdM is important and essential to the secure use of a system because it identifies the users and makes it possible to control their access to the available resources.

Given the mobile nature of many IoT scenarios, there is a need to interact with different domains. In such situations, Federated Identity Management (FIdM) comes

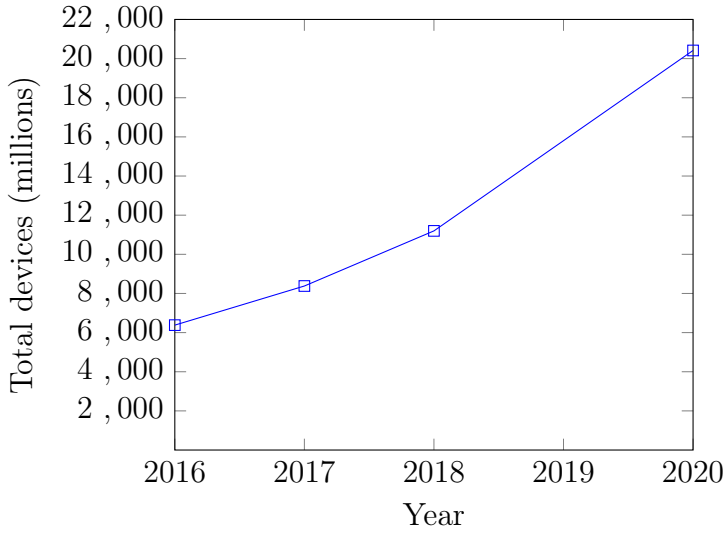

Figure 1.1. Expected growth of IoT devices. Based on van der Meulen [2017].

as a handy solution. FIdM allows a user to authenticate in an external server without creating new credentials or identities, using the identification information from the user's local domain. In FIdM, the authentication of the user in the Service Provider (SP) is delegated to the user's home domain Identity Provider (IdP). Thus, the use of FIdM allows the access control of external users [Shim et al., 2005]; provides a better user experience since it is not necessary to create new credentials for the user in each SP she would like to request a service; enhances privacy by limiting the number of entities that have access to the user's identity information; and makes Single-Sign-On (SSO) possible.

There are (F)IdM solutions widely used in the traditional Internet. However, these solutions are inadequate to the IoT environment. The first reason for the unsuitability of such solutions is that the devices make use of the user's credentials to authenticate and access services. This is an insecure practice since (whenever possible) devices should not have the same clearance level of human users and many times it is not possible to assign an identity of a single user to the device (e.g., devices that are used by multiple users). Furthermore, the IoT scenario usually involves a higher level of scalability and interoperability between domains when compared to traditional network elements [Fremantle et al., 2014]. Finally, traditional approaches are usually built on top of computationally expensive cryptosystems such as Rivest, Shamir, & Adleman Algorithm (RSA) or Digital Signature Algorithm (DSA) and thus, are not adequate to the resource-constrained devices present in several IoT scenarios.

As the traditional approaches cannot fully undertake the IoT demands, this work

presents a solution to such a problem. Federated Lightweight Authentication of Things (FLAT) proposes a lightweight approach that combines implicit certificates and symmetric cryptography in replacement of computationally expensive strategies and cryptosystems.

1.1 Research Questions

This work aims to answer the following research questions:

- Which FIdM and cryptographic techniques are adequate for authentication in IoT?
- What is the feasibility of an authentication protocol based on these specific FIdM and cryptographic techniques?
- What are the gains in terms of space and time when using these specific FIdM and cryptographic techniques?
- What are the possible applications for the designed authentication protocol for IoT?

1.2 Goals

The main goal of this work is to provide an authentication protocol for IoT that can be executed by resource-constrained devices. More specifically, the goals of this project are:

- design an authentication protocol specially tailored to IoT;
- implement a prototype of the solution;
- evaluate the impact of the solution in terms of communication, computation, and storage;
- provide application scenarios where the solution can be applied.

1.3 Contributions

This work presents the design, development, and evaluation of FLAT, as well as a prototype of the authentication protocol built on top of restricted devices and applied

to an automatic parking system. Following is a list of publications based on the solution being presented in this document:

- Full paper published in the 2018 Brazilian Symposium on Computer Networks and Distributed Systems (*Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC'18*) under the title "FLAT: Um Protocolo de Autenticação Federada para a Internet das Coisas". Authors: Maria L. B. A. Santos, Jéssica C. Carneiro, Antônio M. R. Franco, Fernando A. Teixeira, Marco A. A. Henriques, Leonardo B. Oliveira.

- Paper published in the Demo Session of the 2018 Brazilian Symposium on Computer Networks and Distributed Systems (*Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC'18*) under the title "Autenticação Federada para IoT Aplicada a um Sistema Automático de Estacionamento". Authors: Maria L. B. A. Santos, Jéssica C. Carneiro, Antônio M. R. Franco, Fernando A. Teixeira, Marco A. A. Henriques, Leonardo B. Oliveira. This work was awarded the best paper of the Demo Session in SBRC'18.

- Paper published in the Demo Session of the 2018 International Symposium on Information Processing in Sensor Networks (IPSN'18) under the title "Federated Authentication of Things: demo abstract". Authors: Maria L. B. A. Santos, Jéssica C. Carneiro, Fernando A. Teixeira, Antônio M. R. Franco, Marco A. A. Henriques, Leonardo B. Oliveira.

- Demonstration presented in the 2018 Brazilian Research and Educational Network Workshop (*Workshop da Rede Nacional de Ensino e Pesquisa – WRNP'18*) under the title "GT-CoFee: Um Esquema de Gestão de Identidade Federada para IoT". Authors: Maria L. B. A. Santos, Jéssica C. Carneiro, Fernando A. Teixeira, Antônio M. R. Franco, Marco A. A. Henriques, Leonardo B. Oliveira.

- Poster presented in the 2017 Brazilian Research and Educational Network Workshop (*Workshop da Rede Nacional de Ensino e Pesquisa – WRNP'17*) under the title "GT-CoFee: Um Esquema de Gestão de Identidade Federada para IoT". Authors: Maria L. B. A. Santos, Jéssica C. Carneiro, Marco A. A. Henriques, Leonardo B. Oliveira.

1.4 Organization

The remainder of this work is organized as follows. Chapter 2 presents the background concepts necessary to the authentication solution. Chapter 3 presents the previous works related to FLAT. Chapter 4 presents the authentication protocol and the assumptions necessary to its operation. Chapter 5 presents the development of FLAT's prototype and Chapter 6 shows an evaluation of the proposed solution. Finally, Chapter 7 summarizes the results.

Chapter 2

Background

This chapter will introduce some necessary concepts related to FLAT, including the characterization of IoT devices, fundamental concepts in FIdM and some cryptosystems used in the solution.

2.1 IoT

Society is experiencing an increasing number of connected devices: smartphones, sensors, and computers can communicate and cooperate to perform certain tasks [Atzori et al., 2010]. In this sense, IoT is a network of everyday objects with certain capabilities communicating to reach a specific goal [Whitmore et al., 2015].

There are numerous possibilities in IoT, including applications in several different domains such as healthcare, transportation, smart cities, smart homes, agriculture and traffic management [Gubbi et al., 2013]. Undoubtedly, IoT can bring a series of advantages and enhancements to businesses and also increase the comfort and quality of services. The possibilities brought by IoT have caught the attention of the scientific community, hence making IoT a research topic of general interest.

Among several open research topics in IoT, there are privacy, standardization and authentication [Atzori et al., 2010]. FLAT approaches exactly the authentication issue in IoT, considering the lack of computational resources and the potential mobility of devices, as well as other inherent aspects of IoT that are not present in the traditional Internet.

2.2 IoT Devices

As the IoT devices contemplate environments with very different needs, these devices also have diverse characteristics, varying in size, type, and computational capabilities.

There are devices that have very low computational and storage capabilities, so protocols and processes executed by these devices must be lightweight. Here, this kind of IoT devices will be referred to as restricted devices. As Table 2.1 shows, devices such as Memsic IRIS[1], Arduino Mega 2560[2] and Arduino Due[3] are in this category.

There are also devices that do not have major resource constraints, with flash memory in gigabytes and higher CPU speeds. These IoT devices will be referred here as intermediary devices. Raspberry Pi Zero W[4] and BeagleBone Black[5], shown in Table 2.1, are in this category, both with 512 MB of SRAM. The flash memory in Raspberry Pi Zero W is variable, but there is support for up to 32 GB, while the BeagleBone Black has a flash memory of 4 GB.

There are also devices in IoT networks that have superior computational power when compared to the devices shown in Table 2.1. These devices are equivalent to robust servers and equipment found in the traditional Internet.

Table 2.1. Examples of IoT devices.

Class	**Device**	**CPU(GHz)**	**SRAM(MB)**	**Flash(GB)**
restricted	Memsic IRIS	8.0×10^{-3}	7.8×10^{-3}	1.2×10^{-4}
	Arduino Mega 2560	1.6×10^{-2}	7.8×10^{-3}	2.4×10^{-4}
	Arduino Due	8.4×10^{-2}	9.4×10^{-2}	4.9×10^{-4}
intermediary	Raspberry Pi Zero W	1.0	512.0	until 32.0
	BeagleBone Black	1.0	512.0	4.0

In FLAT, the Client device is a restricted IoT device with a high level of mobility. On the other hand, the SP is represented by an intermediary IoT device.

In some scenarios, the restricted devices might be connected to the network through gateways, which can take care of more complex network and security operations, as gateways are more powerful devices. It is important to note that this work considers the direct connection of restricted devices to the network, and thus, does not make use of gateways to execute authentication and authorization tasks or to connect to the network.

[1] `http://www.aceinna.com//wireless-sensor-networks/index.cfm`
[2] `https://store.arduino.cc/usa/arduino-mega-2560-rev3`
[3] `https://store.arduino.cc/usa/arduino-due`
[4] `https://www.raspberrypi.org/products/raspberry-pi-zero-w/`
[5] `https://beagleboard.org/black`

In the authentication protocol proposed here, the only device considered to have a higher computational power, equivalent to servers in the traditional Internet, is the IdP.

2.3 IdM and FIdM

When it comes to security in IoT, IdM has a fundamental role in protecting the network resources, since it controls the access of user and devices to these resources and manages the entire lifecycle of users' (and devices') digital identities [Windley, 2005]. Digital identity, in turn, is a set of attributes or any other data that together can uniquely identify one entity (user, device, institution), usually proven through the use of credentials [Windley, 2005].

According to Nogueira et al. the main operations of an IdM system are: (i) identification, (ii) authentication, (iii) authorization and (iv) auditing [Nogueira et al., 2011b]. Identification provides an identity to the system. Authentication, in turn, occurs when the system verifies the identity using the credentials. The process of authorization involves granting privileges to an entity after authentication. Auditing is registering the actions of an entity in the system.

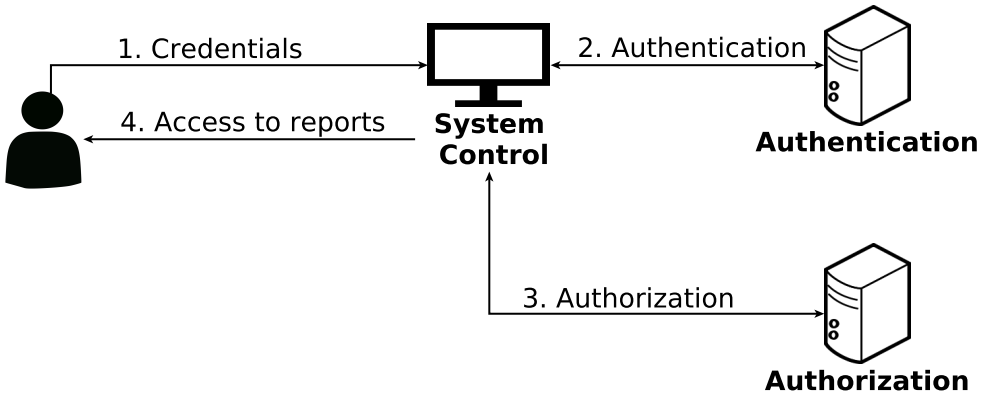

Figure 2.1. IdM main operations.

For instance, imagine a user would like to access a technical report provided by a technology company (Fig. 2.1). This user would present her credentials to the company's system to be identified (step 1, Fig. 2.1). These credentials will be sent to the authentication process (step 2, Fig. 2.1), where it will verify if the user is legitimate, i.e., if the user is really whom she claims she is. After the authentication process, the system will check the access privileges of the user (authorization process, step 3,

Fig. 2.1) to grant access to the technical report or not (step 4, Fig. 2.1). The auditing process takes place registering all the details related to these steps in a log file kept by the system.

Windley defines the specific steps of authorizing the execution of a requested action in an IdM system (Fig. 2.2) [Windley, 2005]. The user shows her credentials to a Policy Enforcement Point (PEP) and makes a request to the system (step 1, Fig. 2.2). Then, the PEP will authenticate the user using her credentials in the authentication server (step 2, Fig. 2.2). The PEP gives the authenticated credentials to the Policy Decision Point (PDP) (step 3, Fig. 2.2), that using information about the security policy (step 4, Fig. 2.2) and the identities (step 5, Fig. 2.2), decides which access permission the user has in the system. This decision is sent to the PEP, denying or allowing the user's request (step 6, Fig. 2.2).

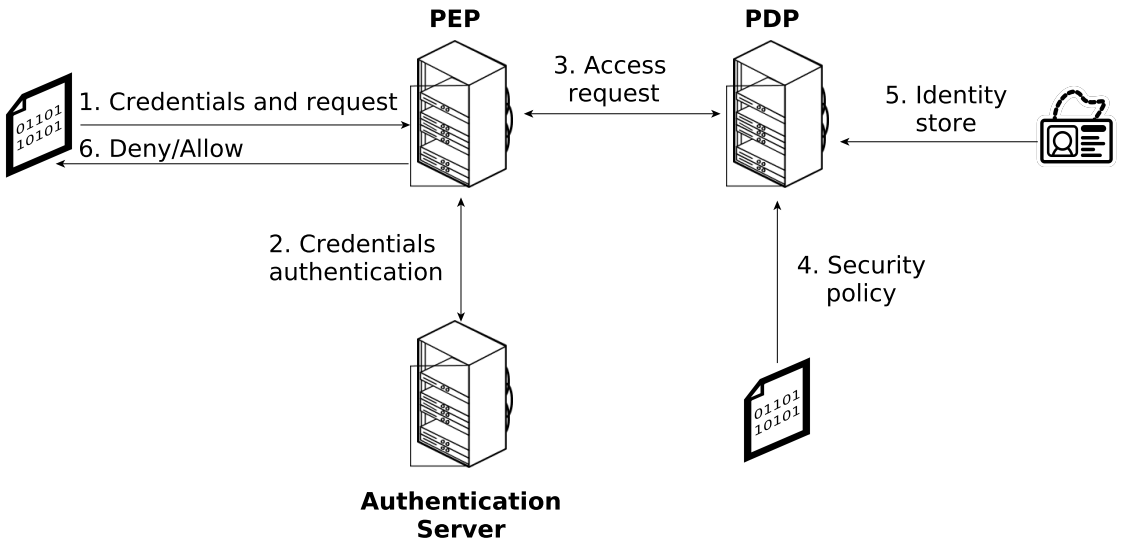

Figure 2.2. Authentication and authorization steps. Adapted from [Windley, 2005].

FLAT focus on the authentication part of the IdM system, defining necessary assumptions and implementing the mechanisms to authenticate restricted devices in IoT. When describing FLAT the term Client will be used to refer to the system user/device.

Traditionally, there are three different models used in IdM systems [Cao and Yang, 2010]: isolated, centralized and federated.

In the isolated model, a single server is responsible for providing the service and also for storing identity information [Cao and Yang, 2010]. In practice, an SP would not only provide the service requested by the user but also performs authentication

and authorization tasks. As all the information is stored in each SP, the user would need different credentials for each SP it would like to access, and thus, this solution does not provide good usability (as the user would have to manage several different credentials) and it is also not storage efficient, as the identity information of the same user will be stored in different SPs. This model is represented in Fig. 2.3.

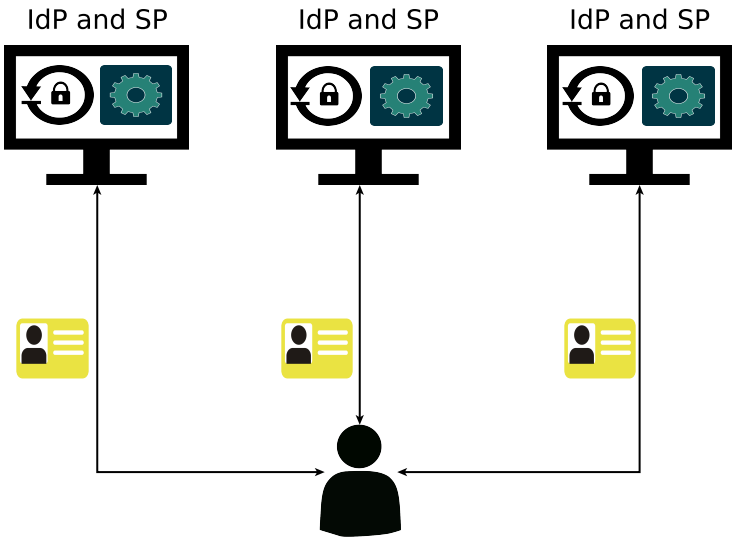

Figure 2.3. Isolated IdM model. Based on [Cao and Yang, 2010].

The centralized model (Fig. 2.4), on the other hand, delegates to a central IdP the task of storing the identity information of all users from all SPs [Cao and Yang, 2010]. Differently from the isolated model, there is a clear definition of different entities for SP and IdP, but the identity information that was stored in each SP in the isolated model, are now all placed in a central server, the IdP. As the number of users grows, this model can present scalability problems, since only one entity is responsible for every single user. This model can also be problematic in the occurrence of a security flaw, as all the identity information is centralized: once the IdP is compromised, all user's information is also compromised.

In the federated model (FIdM), the IdPs and SPs are in separate servers, as in the centralized model. However, there is more than one IdP. The federated model enables the use of services from different domains by external users in a transparent way: the user credentials are global identities in the federation and the SPs from different domains are a single global SP [Cao and Yang, 2010]. Fig. 2.5 shows the federated IdM model.

Considering the need for a solution that takes into account the potential mobility of IoT devices between different domains, FLAT adopts a federated model. According

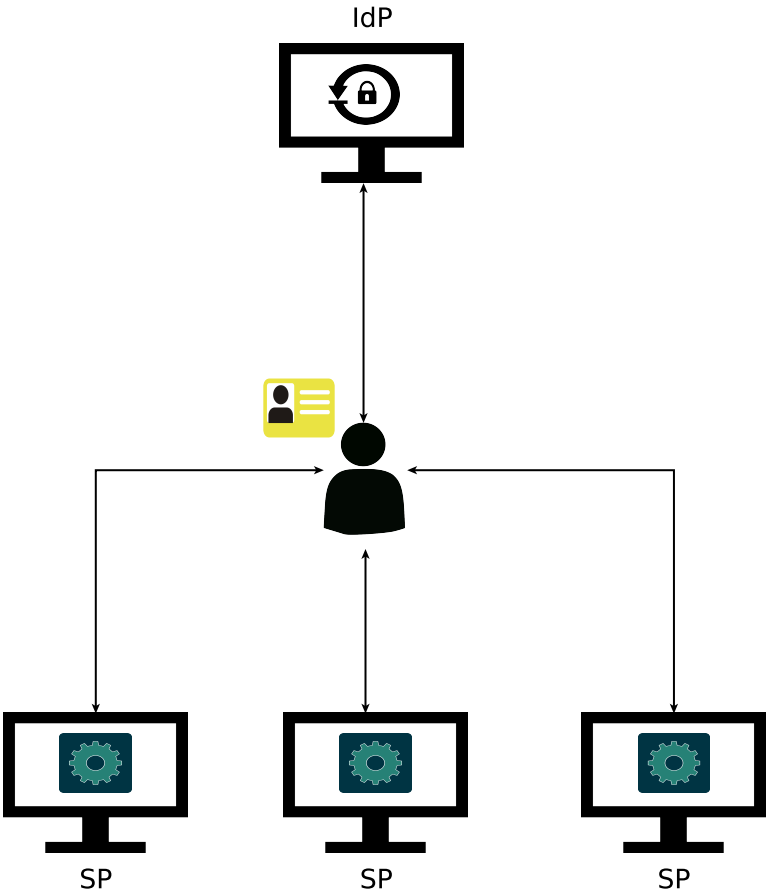

Figure 2.4. Centralized IdM model. Based on [Cao and Yang, 2010].

to Maler and Reed, FIdM is technologies and processes that allow the distribution of identity information and the delegation of activities related to these identities between different domains [Maler and Reed, 2008]. The FIdM model allows a domain to manage the identities of external users transparently [Jøsang and Pope, 2005]. It also makes possible the use of SSO, where a user can authenticate a single time to gain access to several services, avoiding frequent login tasks and remembering different authentication data [Maler and Reed, 2008]. However, as it involves different domains, FIdM also imposes challenges related to security and system architecture, such as deployment of secure communication, strong authentication mechanisms, user information management to preserve privacy, management of identifiers and IdP discovery [Maler and Reed, 2008].

In FIdM, the SPs and IdPs exchange authentication and authorization messages, so the users from a local system can access resources of an external system that participates in the federation [Chadwick, 2009]. The information flow in SSO applications in

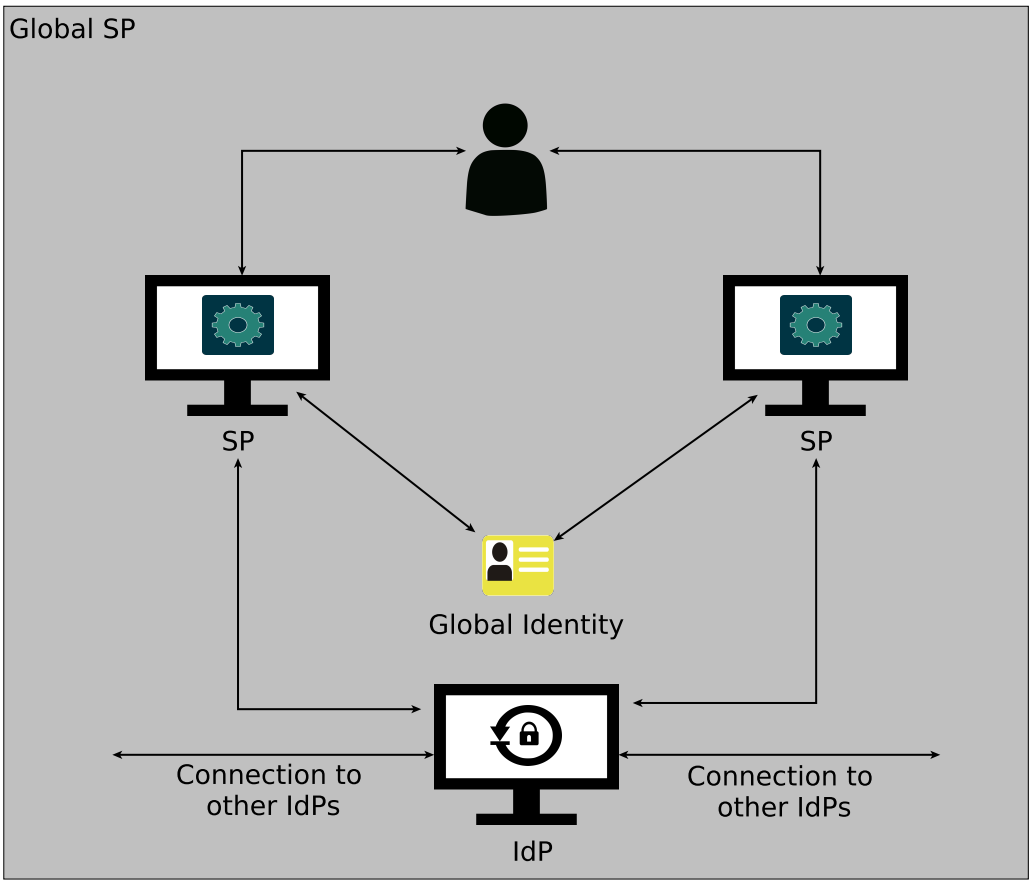

Figure 2.5. Federated IdM model. Based on [Cao and Yang, 2010].

a FIdM model can have different variants, with two main standards being identified: (i) SP-initiated, where the SP sends an explicit access request to the IdP; and (ii) IdP-initiated, where the IdP typically is an access portal so the user can utilize services from several SPs [Maler and Reed, 2008]. In the SP-initiated approach, there is the problem of finding the IdP of the user or the so-called Where Are You From (WAYF), that can be solved by a smart client (that knows the correct IdP) or even asking the user to choose from a list of IdPs [Maler and Reed, 2008]. IdP-initiated approaches, on the other hand, might be susceptible to Cross-Site Request Forgery attack (CSRF) [Armando et al., 2011].

FLAT adopts the IdP-initiated approach, as the Client needs to contact its IdP before requesting the service to the SP. This choice was made since the IdP works as a Key Distribution Center (KDC) for the symmetric key used in $Client \leftrightarrow SP$ communication, and thus, needs to previously distribute the key before Client and SP can exchange messages.

Fig. 2.6 shows how a traditional FIdM message flow works. Client will make a

service request to SP (step 1, Fig. 2.6). The SP, in turn, will redirect the Client to her IdP (step 2, Fig. 2.6). Then, Client asks for an assertion so she can prove her identity to the SP (step 3, Fig. 2.6). Next, the IdP requests and receives the Client's credentials (steps 4 and 5, Fig. 2.6), and sends the assertion to the Client (step 6, Fig. 2.6). The Client forwards the assertion to the SP (step 7, Fig. 2.6) to get access to the service (step 8, Fig. 2.6).

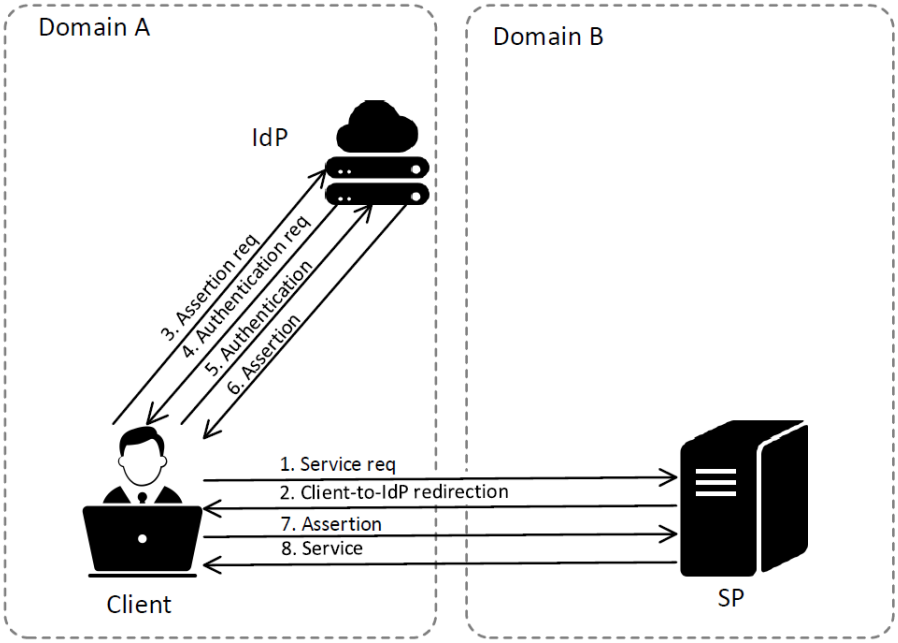

Figure 2.6. Traditional FIdM message flow. Based on [Birrell and Schneider, 2013].

There are well-established standards that define the identity information exchange between different domains [Shim et al., 2005]. Security Assertion Markup Language (SAML), Shibboleth, OAuth, and OpenID Connect are widely used FIdM solutions.

2.3.1 SAML

SAML[6] is an ITU (X.1141) and OASIS standard framework based on Extensible Markup Language (XML) created to exchange security information and used to enable SSO and identity federation[7]. The basic components of SAML are assertions, profiles, bindings and protocols [Ragouzis et al., 2008]. Assertions are a description of attributes, authentication or authorization of an entity. These assertions are sent in a

[6] `https://www.oasis-open.org/standards#samlv2.0`

[7] `http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html`

certain format determined by the SAML protocols, which define how the request and response messages should be sent. SAML bindings establish how the integration between SAML messages and communication protocols will be executed. SAML profiles, in turn, are responsible for deciding how all the other components will be described in order to fit the needs of a specific application.

As a protocol made to provide security, it implements not only a standard way of exchanging authentication and authorization messages but also defines security components that should be present, such as XML-signatures, Transport Layer Security (TLS) and HyperText Transfer Protocol (HTTP) over Secure Sockets Layer (SSL). SAML also makes use of XML-encryption. Regarding privacy, SAML supports the use of pseudonyms, transient identifiers, and assurance levels [Ragouzis et al., 2008].

Like in other FIdM systems, a simple authentication flow using SAML starts with the user trying to access a resource in an SP. The user will then receive a SAML request for authentication that will be passed to its IdP. After the user provides her credentials to the IdP, it will build a SAML response regarding the authentication of the user, which is signed by the IdP. The response is passed to the SP, that checks the information in the response and can provide access or not to the requested resource.

SAML is a highly configurable solution, being able to adapt to different application needs. Although SAML is already a consolidated solution and can be integrated into other frameworks, it requires long XML messages and high computational power, as it uses asymmetric cryptography. Thus, it is not adequate for computationally restricted devices and it is not used in FLAT.

2.3.2 Shibboleth

Shibboleth[8] is an open source federated SSO solution based on SAML 2.0. There are mainly three components in Shibboleth: the IdP, the SP and the Discovery Service (DS). As in other FIdM solutions, the SP provides services to the users and the IdP is responsible for the identities and authentication. The DS is responsible for finding out which IdP holds the identity information about a specific user[9]. Shibboleth also carries the advantages of FIdM systems already mentioned in this chapter, such as increased user privacy and usability.

The typical flow in Shibboleth is similar to other FIdM solutions, starting with the user trying to access a service in the SP [Shibboleth Project, 2018]. After the IdP discovery process, the user will receive an authentication request from the SP and will

[8] `https://www.shibboleth.net/`

[9] `https://wiki.shibboleth.net/confluence/display/CONCEPT`

redirect it to the IdP. Depending on the SAML binding and profile determined by the SP, the authentication response provided by the IdP will contain more or less attributes regarding the identity of the user in the SAML assertion. With this authentication response, the SP can then decide if it will authorize or not the access to the resource. Note that in this context, the IdP is responsible for authentication, while the SP is responsible for authorization. It relies in a previously established trust between the SPs and IdPs in the federation, as in other FIdM solutions.

As mentioned before, Shibboleth is a solution based on SAML, and thus, also incurs in the costs related to the use of the framework, as explained in the previous subsection. FLAT, on the other hand, does not incorporate SAML and makes use of only symmetric cryptography in the Client side, in order to be a more suitable and lightweight solution for IoT.

2.3.3 OAuth

OAuth[10] is an open authorization framework and an Internet Engineering Task Force (IETF) standard – Request for Comments (RFC) 6749. OAuth aims to be a secure authorization solution for accessing HTTP services/resources [Hardt, 2012]. OAuth provides a layer of secure authorization in order to enable limited access to protected resources through access tokens, that contains access attributes restricting the time and scope of the permission [Hardt, 2012]. Although OAuth is an authorization solution, it is usually integrated into other tools to provide authentication and authorization, enabling the user to log in and enjoy services in third-party websites using her credentials from an existing account.

The participating entities of an OAuth flow are the resource owner, the resource server, the Client and the authorization server. The resource owner, as the name says, owns a resource that should be protected when being accessed by a Client. The resource is hosted in the resource server. The Client is an entity that would like to access the protected resource. The authorization server, under authorization provided by the resource owner, provides access tokens to the Client, so she can access the resource [Hardt, 2012].

In a typical scenario, a Client makes an authorization request to the resource owner, that then provides an authorization grant (credential granting resource owner authorization) to the Client. Client forwards the authorization grant to the authorization server, in order to get an access token. With the access token, Client can contact the resource server again and get access to the protected resource [Hardt, 2012].

[10] `https://oauth.net/`

OAuth is used with HTTP and TLS and is highly configurable, making it possible to interoperate with other solutions to enable authorization in several different applications. Differently from OAuth, FLAT is an authentication solution, not an authorization framework.

2.3.4 OpenID Connect

OpenID Connect[11] is a solution built on top of OAuth 2.0, providing an identity layer able to verify identities in the authentication process. OpenID Connect integrates an authentication solution to the authorization defined in OAuth 2.0, using JavaScript Object Notation (JSON) Tokens to communicate authentication information [Sakimura et al., 2014]. JSON Web Signature (JWS) and JSON Web Encryption (JWE) are used to protect the integrity and confidentiality of the exchanged messages.

In OpenID Connect basic flow, the Client sends a request to the OpenID Provider (OAuth 2.0 authorization server with authentication capabilities). Using the terms of OAuth, the resource owner will then be authenticated in the OpenID Provider. With the authorization of the resource owner, the OpenID Provider is able to provide an ID Token together with an access token to the Client [Sakimura et al., 2014]. This ID token is one of the most important extensions of OpenID to OAuth, since it is responsible to provide information about the authentication of the resource owner, including the intention of the authentication and its expiration. The Client sends the access token to the protected resource, being returned an authorization grant.

OpenID Connect also allows the use of levels of assurance and is implemented using a REST-like approach [Sakimura et al., 2014]. OpenID, however, is also based on asymmetric cryptography, and thus, not adequate to the IoT environment where restricted devices will also make authentication requests. FLAT only uses symmetric cryptography on the Client side, allowing even resource-constrained devices to run the protocol and authenticate.

2.4 Cryptosystems

This section presents a description of the cryptosystems used in FLAT and the reasons why they were chosen as lightweight alternatives to the IoT context.

[11] `http://openid.net/connect/`

2.4.1 ECDSA and ECIES

The discrete logarithm problem for elliptic curves, in which the Elliptic Curve Cryptography (ECC) is based, does not have a known solution in subexponential time, and thus, in cryptosystems based on ECC is possible to obtain the same security level with significantly smaller keys, when compared to algorithms based on the discrete logarithm problem for finite sets or even the problem of integer factorization [Johnson et al., 2001]. The use of shorter keys provides a set of advantages to ECC-based algorithms such as lower requirements for storage and memory, less bandwidth to transmit keys over the network, and less energy consumption in restricted devices [Martínez et al., 2010]. Elliptic Curve Integrated Encryption Scheme (ECIES) is an asymmetric encryption algorithm based on ECC, and thus, is able to provide the same security level of traditional algorithms (such as RSA) with the use of smaller keys. For example, considering a security level of 128 bits, using RSA the key length would be around 3072 bits while using ECC the key length can be as short as 256 bits [Martínez et al., 2010]. The operation of ECIES is described using three procedures [Certicom, 2009]:

- key deployment: after defining a set of parameters to be used in the scheme (MAC function, symmetric encryption scheme, elliptic curve parameters, point compression), the entity V (the recipient of the encrypted message) generates an elliptic curve key pair (d_V, Q_V) and ensures that U (the sender of the encrypted message) receives the public key Q_V.

- encryption: to encrypt a message M, entity U generates an elliptic curve key pair (k, R), where $R = (x_R, y_R)$. U derives a shared secret z using the secret key k and the public key Q_V generated by V. Next, U uses a key derivation function to generate a symmetric key K from z. U parses the a leftmost bits to form an encryption key EK and the b rightmost bits to form a MAC key MK, where a is the size of the key used in the symmetric encryption scheme chosen and b is the size of the key used in the MAC algorithm chosen. U then uses the key EK to encrypt the message M, generating ciphertext EM. Finally, U computes the tag D on the ciphertext EM using the MAC scheme and the key MK. The final ciphertext output by ECIES is the triple $C = (R, EM, D)$.

- decryption: Upon the reception of ciphertext C, V first needs to parse C to recover R, EM and D. V derives a shared secret z using her secret key d_V and the public key R. V then uses a key derivation function to obtain the key K from z. Next, V parses the a leftmost bits to form an encryption key EK and the b rightmost bits to form a MAC key MK, where a is the size of the key used in

the symmetric encryption scheme chosen and b is the size of the key used in the MAC algorithm chosen. V uses the MAC algorithm chosen to check the tag D and verify it is the tag on EM. The last step in the decryption procedure is to decrypt EM using EK to obtain the original message M.

ECIES is also an ANSI (ANSI X9.63), IEEE (IEEE 1363a), ISO (ISO 18033-2) and SECG (SECG SEC1) standard. The use of smaller keys and its consequent advantages make ECIES a good asymmetric encryption solution to IoT, and thus, was chosen as the asymmetric encryption scheme used in FLAT.

The Elliptic Curve Digital Signature Algorithm (ECDSA), as the name indicates, is a digital signature algorithm equivalent to the Digital Signature Algorithm (DSA) but based on ECC. ECDSA provides faster signature generation when compared to DSA, since it is possible to achieve the same security level with smaller parameters. There are three different procedures necessary for the functioning of ECDSA [Certicom, 2009]:

- key deployment: after defining a set of parameters to be used in the scheme (hash function, elliptic curve parameters), entity U (the subject of the signature, who signs) generates an elliptic curve key pair (d_U, Q_U) and entity V (the recipient of the signature) receives the public key Q_U.

- signing: U generates an elliptic curve key pair (k,R), where $R = (x_R, y_R)$. U then sets $r = x_R$ and computes the hash e of the message M to be signed: $e = H(M)$. Next, U computes $s = k^{-1}(e + rd_U)$ $(mod\ n)$, where n is the order of the base point generator. The signature is the pair $S = (r, s)$.

- verifying: to verify the signature of U over M, V needs to compute the hash of M: $e = H(M)$. Then V computes u_1 and u_2 using the two parts of the signature (r and s): $u_1 = es^{-1}\ mod\ n$ and $u_2 = rs^{-1}\ mod\ n$. Next, V computes $R = (x_R, y_R) = u_1G + u_2Q_U$, where G is the base point generator. V then sets $v = x_R$ and compares v and r: if v is equal to r the signature is valid.

Using point compression, ECDSA is able to produce public key certificates around 25% smaller than other algorithms based on asymmetric cryptography [Johnson and Menezes, 1998]. ECDSA is an ISO (ISO 14888-3), ANSI (ANSI X9.62), IEEE (IEEE 1363-2000), FIPS (FIPS 186-2) and SECG (SEC 1) standard.

ECDSA is used as the digital signature algorithm in FLAT, given the characteristics and advantages presented by ECC in general and by the ECDSA itself.

2.4.2 ECQV

The Elliptic Curve Qu-Vanstone (ECQV) certificate [Brown et al., 2001] is an implicit certificate scheme, where the public key is reconstructed from public data [Brown et al., 2001]. The public key is not stored explicitly in the certificate, eliminating the need for separate public key and signature fields (as in traditional certificates) and thus, provides a smaller certificate, when compared to traditional solutions such as RSA or ECDSA certificates. For instance, considering a security level of 128 bits, while an implicit certificate size is around 257 bits of length (plus identification data), an ECDSA certificate is around 769 bits and an RSA certificate is around 6144 bits [Certicom, 2018].

The main operations of the implicit certificate scheme (ECQV) are [Certicom, 2013]:

- setup: the Certificate Authority (CA) establishes the elliptic curve domain parameters and the hash function and generates a key pair. Formally, the CA generates an elliptic curve key pair (d_{CA}, Q_{CA}), where d_{CA} is the private key and Q_{CA} is the public key. Q_{CA} is also distributed to U, the subject requesting a certificate, and to V, the one that will receive the certificate.

- certificate request: the subject who is requesting a certificate to the CA generates a key pair and sends a certificate request to the CA using its public key. Formally, U generates an elliptic curve key pair (k_U, R_U), where k_U is a private value necessary to compute the private key, and R_U is a public value. U also generates a string U, that represents her identity. The pair (U, R_U) is the certificate request that will be sent to the CA.

- certificate generation: CA generates and sends an implicit certificate to the subject after confirming her identity and receiving the certificate request. Formally, the CA generates an elliptic curve key pair (k, kG), where the value k is private and the value kG is public. The CA then computes $P_U = R_U + kG$, an elliptic curve point that is the public key reconstruction data. The certificate $Cert_U = (P_U \mid U)$ is the concatenation of the public key reconstruction data and U, the value representing the identity of U. Next, the CA computes $e = H(Cert_U)$ as the hash of the certificate $Cert_U$. Finally, the CA computes the private key contribution data $r = ek + d_{CA}(mod\ n)$, where n is the order of the base point generator.

- public key extraction: extracts the public key of the subject using CA's public key, the implicit certificate and the elliptic curve domain parameters. When the certificate receiver V receives the certificate, it has to extract the public key. Formally, V computes $e = H(Cert_U)$ as the hash of the certificate received. Then, V computes $Q_U = eP_U + Q_{CA}$, using the public key reconstruction data from the certificate and the CA's public key. Q_U is U's public key extracted from the certificate.

- certificate reception: when the subject receives the certificate, she checks that the implicit certificate she received is valid. Formally, U extracts the public key from the certificate, computing the hash of the certificate $e = H(Cert_U)$ and then calculating $Q_U = eP_U + Q_{CA}$ to obtain the public key. Then U computes $d_U = r + ek_U(mod\ n)$ to obtain the private key. Next, U computes $Q'_U = d_U G$, where G is the base point generator. If Q'_U is equal to Q_U, then the certificate is valid.

As explained above, the only operation necessary when the receptor of the certificate (relying party) receives the implicit certificate, is the public key extraction. This makes implicit certificates also more computationally efficient than traditional certificates since the process of extracting the public key is faster than signature verification (of the CA's explicit signature) in traditional certificates [Certicom, 2018]. Differently from traditional certificates, in implicit certificates, after being extracted, the public key will only be validated when it will be used in some communication process, with the same being applied to the proof that the subject of the certificate really has knowledge of the private key [Brown et al., 2001]. This means that the association of the public key to its owner and the ownership of the private key are not distinct processes [Brown et al., 2001].

FLAT uses implicit certificates in the communication between SP and IdP, in order to reduce communication, storage, and computational costs.

2.5 Summary

In this chapter was presented a characterization of the IoT devices considered in this project as well as existing FIdM technologies and their differences when compared to FLAT. Selected cryptosystems were also described, followed by an explanation of why they were chosen to integrate FLAT. The next chapter will cover previous works related to FLAT.

Chapter 3

Related Work

In order to set FLAT apart from other existing approaches, this chapter presents works related to the federated authentication solution presented in this document. The previous works are divided into two categories: authentication and (F)IdM for the traditional Internet; authentication and (F)IdM for IoT.

3.1 Authentication and (F)IdM for the Internet

This section covers works related to authentication and FIdM for the traditional Internet, where there are already several known standards and solutions, such as Shibboleth, OpenID and OAuth.

There are various works addressing (F)IdM, its main characteristics and existing solutions [Chadwick, 2009; Shim et al., 2005]. These works also provide evaluations and suggestions of which solution is more appropriate to different business needs [Maler and Reed, 2008; Birrell and Schneider, 2013].

The work of Chadwick gives a detailed view of FIdM, starting by the definition of a digital identity and finishing with FIdM standards, widely used FIdM systems and related research topics [Chadwick, 2009]. The paper presents some good practices that should be considered when designing an IdM system and privacy concerns regarding FIdM models.

Shim et al. presents main concepts of FIdM, describing existing standards and architectures [Shim et al., 2005]. This paper gives a broad view of FIdM and a big picture of how SAML, Liberty Alliance FIdM architecture and WS-Federation work. It also reinforces the advantages of using FIdM, such as SSO and a higher control by the user of which information will be shared across domains.

The work of Maler and Reed also addresses main concepts and configurations of FIdM models [Maler and Reed, 2008]. It presents widely used FIdM standards such as SAML, OpenID and InfoCard and a comparison between them in terms of security and privacy, user-centric identity support, SSO and how lightweight they are. They offer a brief interoperability analysis between the described standards. This paper also presents the main security and privacy issues related to FIdM (user's tracking by IdP, phishing attacks to username/password, etc.) and some approaches to mitigate them.

Birrell and Schneider makes an analysis of existing FIdM systems, focusing on privacy-related issues [Birrell and Schneider, 2013]. It evaluates widely used FIdM architectures under the privacy properties of undetectability, unlinkability, and confidentiality. Based on the design choices used in these architectures, the authors perform an analysis of the benefits and disadvantages present in each approach. The authors claim that the taxonomy presented in their work can be used as a guide to designers of identity management systems in order to build solutions that are able to preserve user's privacy.

Some recent works, however, aim to provide improvements or changes in widely used solutions to achieve enhanced privacy, attribute aggregation or to increase efficiency [Chadwick and Inman, 2009; Isaakidis et al., 2016; Nogueira et al., 2011a].

Chadwick and Inman proposes a linking service to aggregate attributes of different IdPs [Chadwick and Inman, 2009]. The authors argue that for many existing applications in the Web, it is not enough to receive identity attributes from only one IdP per session established with a selected SP. When the user would like to access a service, the SP will redirect it to its IdP. In turn, the IdP will send a response to the SP, asking it to contact the linking service, which is able to provide attribute aggregation from the IdPs selected by the user. In this model, the user has to previously authenticate into the IdPs for the linking service to create a link entry assigning the user to its local identifier. With attribute aggregation, the user is able to select the information it would like to share with each SP and also to use attribute information from different IdPs in the same SP session.

Isaakidis et al. also focus on the privacy aspect of FIdM systems [Isaakidis et al., 2016]. In their work, however, instead of performing an analysis and providing a taxonomy for design choices, the authors propose a new approach based on algebraic Message Authentication Codes (aMACs) to enhance privacy in existing FIdM systems. The solution, UnlimitID, uses attribute-based pseudo-identities to provide unlinkability between the SPs and IdPs, and thus, avoid the creation of user behavior profiles or logs. UnlimitID is designed to work with OAuth, without any changes in the SPs (some changes are necessary for Client and IdP).

Nogueira et al. propose modifications to the Shibboleth authentication process in order to improve the efficiency of the authentication through certificates in a federated environment, especially in academic federations [Nogueira et al., 2011a]. Their solution involves the use of Notary Based Public Key Infrastructure (NBPKI) that uses self-signed certificates and the X.509 standard. The authors claim that the proposed changes based on NBPKI are able to provide a simpler and less redundant authentication process.

Still in the approaches for the traditional Internet, Kerberos [Steiner et al., 1988] is an authentication protocol that shares some characteristics with FLAT, although it is not a solution developed for IoT. Kerberos was initially proposed by Steiner et al. as an authorization and authentication system, which is still being used in its versions 4 and 5 [Stallings, 2006]. The specification of version 5 can be found on RFC 4120 [Neuman et al., 2005]. Although there are similarities between the solutions (FLAT and Kerberos), such as the fact that both allow the authentication of users and/or devices from different domains, there are several factors that make them different. One of the main differences between FLAT and Kerberos is that FLAT uses both symmetric and asymmetric cryptography, while Kerberos uses only symmetric cryptography in all its steps.

Furthermore, there are four entities in Kerberos, the Application Server (AP) equivalent to the SP in FLAT, the Authentication Server – Kerberos Server (AS) equivalent to FLAT's IdP, the Kerberos Client (KC) equivalent to the Client in FLAT, and the Ticket Granting Server (TGS). In FLAT there is no TGS since Client's authentication and the emission of the assertion to access the service are both done by the IdP.

Another aspect that set them apart is that in FLAT there is a previous distribution of a symmetric key between Client and IdP, while in Kerberos it is necessary the previous distribution of a symmetric key between the AP and the AS and the key used to encrypt the messages exchanged between AS and KC is a symmetric key derived from the user's password.

Finally, Kerberos generates two session keys, with two entities working as a KDC: the AS and the TGS, while FLAT generates only one key ($Client \leftrightarrow SP$ communication) with the IdP being the only entity working as KDC.

3.2 Authentication and (F)IdM for IoT

In the IoT scenario, there is no consolidated solution or consensus on how to provide IdM for smart devices with very different computational power and characteristics. In this sense, several works have tried to solve the identity management issue in IoT [B. Oliveira et al., 2009; Horrow and Sardana, 2012; Liu et al., 2012; Hummen et al., 2013; Yavuz, 2013; Porambage et al., 2014; Fremantle et al., 2014; Fremantle and Aziz, 2016; Markmann et al., 2015; Cirani et al., 2015; Witkovski et al., 2015; Hong et al., 2016; Domenech et al., 2016; Neto et al., 2016; Silva and Silva, 2017].

The work of B. Oliveira et al. presents a solution for authentication of wireless sensor networks devices to multiple users using digital signatures [B. Oliveira et al., 2009]. The authors claim that the use of digital signatures is the most adequate method of authentication in this scenario, mentioning aspects such as the elimination of shared keys stored in the devices or the possibility of forwarding the authenticated message to multiple users. Their solution, Secure Tiny Web Service (Secure-TWS), was implemented in resource-restricted devices, considering different signature schemes: ECDSA, and Boneh-Lynn-Shacham (BLS). Based on their analysis in terms of computational time, memory usage, and energy consumption, the authors show that ECDSA is the best alternative to the considered scenario. Despite using ECDSA as part of the authentication protocol, FLAT does not use digital signatures on the Client side, abiding by symmetric cryptography in the resource-constrained device.

Horrow and Sardana propose an IdM framework for IoT using cloud-based technologies [Horrow and Sardana, 2012]. Instead of having computing nodes to process the information sent by sensors, this computation will be performed in the cloud. The cloud will also be responsible for managing the devices' life cycle, including the registration of sensor and receiver nodes, authentication, deletion and relocation of devices. Their work is a general architecture to approach the identity management issue in IoT, but they do not describe in details how would be the protocols to implement this architecture. The authors also do not present an implementation or evaluation of the proposed framework. Differently from this solution, FLAT do not transfer the computation to an external entity (cloud), developing a protocol that can be executed even by restricted devices, providing a proof-of-concept and an evaluation of the authentication protocol.

The work of Liu et al. presents a framework for authentication and authorization in IoT [Liu et al., 2012]. Their approach uses OpenID to provide authentication and Role-Based Access Control (RBAC) to deal with the authorization. The work also describes a protocol to establish session keys between two entities and provides a brief

security evaluation. The key establishment is based on ECC and both key request procedure to access a device and key establishment procedure use an intermediary entity, similar to a gateway. The authors do not provide an implementation/prototype or a performance evaluation of the proposed solution. As mentioned before, FLAT does not use gateways to provide authentication, relying on symmetric cryptography to be lightweight enough to run in Client devices. FLAT also do not address authorization and is not based on OpenID or other known FIdM technologies for the traditional Internet, in order to deliver a more tailored and lightweight solution for IoT.

In Hummen et al. [2013] the authors claim the existing solutions for authentication and information transmission in the traditional Internet use protocols based on certificate exchange that are not feasible in low power smart objects commonly used in IoT scenarios. They evaluate the DTLS handshake protocol based on certificates, showing that the overhead was indeed too high for constrained devices. As a solution, they propose some changes/adaptations to DTLS in order to use certificates in IoT devices, such as pre-validation of certificates in a gateway and a delegation procedure that allows another entity to validate the certificate instead of the smart object itself. However, the authors only present a brief computational evaluation of the proposed solution, without addressing issues such as communication and storage overhead. This solution, as other (F)IdM approaches to IoT, makes use of gateways and traditional asymmetric cryptography, while FLAT provides a lightweight symmetric solution in the Client side without delegation of tasks to gateways. Furthermore, when it comes to certificates (used between the IdP and SP), FLAT adopts implicit certificates to reduce communication and computational costs.

The work of Yavuz addresses the authentication of resource-constrained devices through the use of digital signatures, and thus, it is a solution also based on asymmetric cryptography [Yavuz, 2013]. However, the author claims that the existing digital signature schemes are not adequate to devices with limited computational resources, as they incur in high computational and storage costs. Yavuz addresses this issue proposing a signature scheme called Efficient and Tiny Authentication (ETA). The author presents a security analysis and an evaluation of the solution in terms of public/private key size, signature size, and the analytical costs to generate/verify a signature. The results presented by Yavuz show that ETA has smaller public/private key sizes as well as a smaller signature size, when compared to other signature schemes, such as ECDSA. ETA differs from FLAT as it is based on asymmetric cryptography and do not make use of the FIdM paradigm.

Porambage et al. also propose the use of asymmetric cryptography as an authentication solution for IoT [Porambage et al., 2014]. However, as there are several

devices with restricted computational resources in IoT scenarios, Porambage et al. adopt implicit certificates as a strategy to reduce computational costs. They approach the authentication problem in IoT through a two-step process: a registration step and an authentication step, taking as inspiration the ECQV and the Elliptic-curve Diffie–Hellman (ECDH) schemes. Their solution can be used in a sensor to sensor authentication as well as end-user to sensor authentication. The authors present a security analysis and also an implementation of the proposed solution to show its feasibility. Although FLAT also makes use of implicit certificates, they are not used in the Client side, as the restricted device does not process asymmetric cryptographic primitives.

In Fremantle et al. [2014] the authors propose a federated identity model to provide access control to IoT devices. This paper evaluates the possibility of an authentication and authorization model for IoT based on OAuth 2.0 with the MQTT protocol. The proposed solution was implemented in IoT devices and some implementation challenges were analyzed. Although the use of a standardized solution such as OAuth has its benefits, it also has a high computational cost when considering constrained devices. In order to diminish the costs to the constrained devices, the strategy proposed by FLAT is to avoid asymmetric cryptography in such nodes.

In another work, the OAuth 2.0 protocol is extended to provide a FIdM solution for IoT devices, as well as enhance the privacy of users [Fremantle and Aziz, 2016]. The proposed solution, called OAuthing, is a model that enables users and devices to be registered automatically and provides each user/device with anonymous identities. The authors present a prototype of the proposed model with a mapping of OAuth 2.0 to the MQTT protocol, claiming it is able to reduce the costs when compared with the traditional OAuth 2.0. They present a brief evaluation of the protocol in terms of communication costs, but it lacks a discussion about the security systems/techniques used to allow their security/privacy features. On the other hand, FLAT comes with a tailored authentication solution to IoT, based only on symmetric cryptosystems in the Client side.

Another work by Markmann et al. addresses the end-to-end authentication issue in IoT [Markmann et al., 2015]. They propose the use of Identity-Based Cryptography (IBC) in order to replace traditional PKI approaches. The solution presented in this work makes use of gateways as Trusted Authorities (TA), that are able to manage subdomains and the private keys of the nodes present in such subdomain. Authentication between different subdomains is performed by establishing a federation between the TAs. The authors present a brief performance analysis of the solution in terms of CPU time to demonstrate its feasibility. Differently from the work of Markmann et al.,

FLAT does not make use of gateways to provide authentication, adopting a different strategy, where the restricted device is able to run the authentication protocol (since it is based on lightweight symmetric cryptography in the Client device) without the use of external devices.

Cirani et al. proposes an authorization architecture for IoT using an external authorization service based on OAuth [Cirani et al., 2015]. Their solution, called IoT-OAS transfers the authorization logic from the constrained device to the external service, allowing the device or smart object to protect its resources by only making a request to the authorization framework. The authors claim that implementing OAuth authorization in the smart object is infeasible due to computational and energy restrictions, and thus, delegating this to an external architecture is a solution to provide authorization services in IoT scenarios. The focus of this paper is to provide an authorization solution, while FLAT is an authentication protocol. Furthermore, it delegates the authorization to a gateway, which is based in the OAuth traditional FIdM solution, while FLAT does not make use of gateways and it is exclusively tailored to IoT.

Witkovski et al. propose the use of a gateway as a way to integrate the traditional Internet and IoT scenarios [Witkovski et al., 2015]. The authors also designed a solution for authenticating IoT devices in a smart home context, based on symmetric cryptography and session keys. Using the proposed solution, technicians could perform maintenance tasks in the IoT devices in an authenticated way. Their solution is based on standards (such as CoAP, DTLS, OpenID Connect, OAuth 2.0), and the authors present a prototype with an evaluation of the feasibility of the solution, analyzing the overall response time per request. However, it lacks a complete evaluation of the solution impact in a real constrained device, such as storage costs. The solution proposed by Witkovski et al. addresses a very specific scenario, though. FLAT, on the other hand, aims to provide a more comprehensive solution that can be applied to different IoT scenarios.

Furthermore, while in FLAT the KDC role (to generate the session keys) is delegated to the IdP, in the work of Witkovski et al. this task is executed by the device (Appliance).

This solution also presents two distinct authentication points: one for the device, able to correlate the devices' serial number to the symmetric key (Customer Service); and an IdP responsible for both authentication and authorization of the appliance technicians (Access Authorization Server and Authentication Server), that works with asymmetric encryption. In addition, while FLAT does not have a gateway entity, Witkovski et al.'s solution proposes a gateway that works similarly to a parser, breaking and reconstructing messages from the devices to the Customer Service (and also

technician) and vice-versa.

In Hong et al. [2016] Bluetooth Low Energy (BLE) technology is used to provide access control in IoT, more specifically for smart homes and personal appliances. The authors developed a prototype of the solution based on BLE and applied it to different scenarios, such as door locking and a remote home monitoring system. This solution, however, focuses on access control in a very specific scenario, while FLAT is designed to be a general solution for IoT authentication.

The work of Domenech et al. describes an authentication and authorization model for IoT [Domenech et al., 2016]. Their solution, AAI4WoT, is a FIdM architecture based on SAML and XACML standards, allowing any access control model and authentication of users and devices. AAI4WoT is also available in the GIdLab [Wangham et al., 2013] IdM testbed. The authors present a prototype of the solution and provide an evaluation in terms of storage, communication, CPU usage, and power consumption. The proposed solution, however, incurs in significant overhead for constrained devices and can be executed by devices with intermediate computational capabilities, such as a BeagleBone Black. FLAT, on the other hand, aims to be a solution for restricted devices with very low computational and storage capacity and was specially tailored to the IoT environment.

Neto et al. presents the Authentication of Things (AoT), a solution to provide authentication and access control to IoT devices, making use of IBC and Attribute Based Access Control (ABAC) [Neto et al., 2016]. The authors designed protocols considering all the steps in the life cycle of an IoT device, going from its pre-deployment until its retirement. The work of Neto et al. also includes an evaluation of the developed solution in both resource-restricted devices (such as Arduino Due) and more powerful devices (such as a smartphone LG G4). FLAT, on the other hand, focus only on the authentication of resource-constrained devices, and thus, adopts the strategy of using only symmetric cryptography on the Client side.

Silva and Silva provide a federated authorization solution for IoT based on SAML and RBAC or ABAC in order to provide physical access [Silva and Silva, 2017]. The authors claim that existing solutions do not properly address issues such as the multi-factor authentication for IoT or the authentication and authorization in non-Web IoT environments. They present an architecture where the authorization is based on Extensible Access Control Markup Language (XACML) and RBAC or, with a multi-factor authentication process through FIDO[1] using SAML. This work also presents a description of an implementation/prototype. FLAT addresses the authentication problem in

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

a different way, proposing a protocol that replaces asymmetric cryptography and traditional FIdM technologies (such as SAML) by lightweight cryptographic primitives in order to be able to run in constrained devices, while Silva and Silva's is suitable for more powerful devices, such as smartphones, and focused on solving a specific use case scenario (physical access control) using multi-factor authentication.

Although there are FIdM solutions being proposed for IoT, the existing approaches are either designed to very specific scenarios [Witkovski et al., 2015; Hong et al., 2016] or are based in expensive cryptographic primitives that can only be processed by devices with intermediate computational capabilities [Domenech et al., 2016]. Several solutions, to address the lack of computational resources in IoT devices, make use of gateways, that process the data for the restricted device [Hummen et al., 2013; Cirani et al., 2015].

Differently from these approaches, FLAT aims to be a general solution for authentication in IoT, that can be applied to different scenarios. The authentication protocol proposed in this work uses only symmetric cryptographic primitives in the Client side, so even restricted devices can process the protocol steps without the use of external gateways. FLAT was exclusively designed for IoT, considering its inherent characteristics and the computational restriction of devices.

Table 3.1 compares FLAT with the existing IdM solutions for IoT described in this chapter. Note that most of the solutions presented in this section make use of gateways to provide authentication and authorization for IoT devices, and only Witkovski et al. [2015] considers the use of symmetric cryptography in the computationally constrained devices, although applied to a different context when compared to FLAT.

3.3 Summary

In this chapter were presented several works related to (F)IdM and explained the main differences between them and FLAT. Differently from the previous works presented here, FLAT is a solution designed to provide authentication in general IoT scenarios, being able to be executed even in resource restricted devices without the use of gateways. The next chapter will provide more details about FLAT, describing the authentication protocol and its related assumptions.

Table 3.1. IdM for IoT related work comparison.

Work	Symmetric Crypto on Client	Gateway	Client device	Context	Authorization	Authentication
B. Oliveira et al. [2009]	—	—	Restricted	General	—	Yes
Horrow and Sardana [2012]	—	Yes	Restricted	General	Yes	Yes
Liu et al. [2012]	—	Yes	—	General	Yes	Yes
Hummen et al. [2013]	—	Yes	Restricted	General	—	Yes
Yavuz [2013]	—	—	Restricted	General	—	Yes
Porambage et al. [2014]	—	—	Restricted	General	—	Yes
Fremantle et al. [2014]	—	Yes	Restricted	General	Yes	Yes
Markmann et al. [2015]	—	Yes	Restricted	General	—	Yes
Cirani et al. [2015]	—	Yes	Restricted	General	Yes	—
Witkovski et al. [2015]	Yes	Yes	Restricted	Smart homes	Yes	Yes
Fremantle and Aziz [2016]	—	Yes	Restricted	General	Yes	Yes
Domenech et al. [2016]	—	Yes	Intermediary	General	Yes	Yes
Hong et al. [2016]	—	Yes	Restricted	Smart homes	Yes	—
Neto et al. [2016]	—	—	Restricted Powerful	General	Yes	Yes
Silva and Silva [2017]	—	—	Intermediary	Physical access	Yes	Yes
FLAT	Yes	—	Restricted	General	—	Yes

Chapter 4

FLAT: Federated Lightweight Authentication of Things

This chapter presents FLAT, a federated authentication solution for IoT, providing a detailed description of the assumptions, technical aspects, and steps necessary in the protocol.

4.1 Assumptions

This section provides assumptions regarding the authentication protocol, i.e., describes procedures that take place before the protocol is running in order to enable the correct operation of FLAT.

4.1.1 Pre-deployment

In order to minimize the computational and communication costs, the Client device only performs symmetric cryptographic primitives. To enable this, it is necessary to provide a symmetric key to the Client device prior to the operation of the authentication protocol, so it can communicate with its IdP.

In an IoT environment, like other high mobility environments, mechanisms to protect memory resources against physical attacks are necessary. This is usually obtained using Electrically-Erasable Programmable Read-Only Memory (EEPROM) or Static Random Access Memory (SRAM) equipped with continually powered tampering detection and prevention systems [Herder et al., 2014].

An alternative to avoid physical attacks is through the use of Physical Unclonable Functions (PUFs), a physical system partially disordered which when presented to an

external impulse or input, generates a determined output [Rührmair et al., 2010]. This output does not depend only on the input, but also on device's physical characteristics (hardware) and environmental variations, so even being presented to the same input, it can produce a slightly different output [Katzenbeisser et al., 2012]. An error correction module is able to correct the errors (small differences) of the generated outputs when the same input is presented to a PUF multiple times [Katzenbeisser et al., 2012].

It is assumed that prior to FLAT's execution, PUFs are used to generate the device's cryptographic key, as described in the work of Suh and Devadas [Suh and Devadas, 2007]. The PUF circuit generates an output that passes through an error correction process. After the correction, a hash function is applied in the output and can be used as a symmetric key. The Client device generates the symmetric key and sends it to its IdP through a secure channel when the Client device is passing through its initial configuration (pre-deployment stage). This key will be used in Client $\leftrightarrow$ IdP communication.

4.1.2 Service Discovery

Prior to the execution of FLAT, the Client device needs to know about the SP and its services. The service discovery is the process that lets the Client device obtain this information. Service discovery allows entities that participate in a network to advertise and consult services, select the most appropriate ones according to their needs, as well as use these services [Ververidis and Polyzos, 2008]. In an IoT scenario, it is reasonable to assume that devices make use of services provided by other devices physically close to them.

A method of transmitting service information is through the use of beacons, broadcast messages sent periodically in a specific range.

It is assumed that prior to FLAT's execution, beacons are used to transmit information about a service being offered. An SP sends beacons informing the available services at the moment. The Client device receives the information from the SP, so it can request a service through the authentication in its IdP using FLAT's protocol.

4.1.3 Trust Establishment

Before FLAT's execution, it is necessary to establish trust between Client and its IdP, as well as between SPs and IdPs. As for the Client–IdP trust establishment, this is performed in the pre-deployment stage, where Client and IdP exchange a symmetric key for their further communication.

As IdPs and SPs are usually from different domains, the trust establishment between them is based on certificates issued by a common CA. The public keys of these entities will be guaranteed by certificates, that can be easily validated by each of them when necessary. Thus, each SP and IdP must have received the certificate of the CA they have in common (and that signed their certificates) in a secure manner.

4.1.4 Interfederation

In some cases, Clients can request a service being offered by a different federation. In such cases, to enable the trust establishment between different federations, FLAT assumes a model based on **G**ÉANT **A**uthorisation **IN**frastructure for the research and **edu**cation community (eduGAIN)[1]. eduGAIN is an interfederation service developed within the project GÉANT-GN2[2], to allow the connection between academic identity federations and its respective authentication and authorization infrastructures. To establish trust between federations, they have their components (IdPs and SPs) registered in the eduGAIN's trust store.

Based on eduGAIN's trust model, it is considered that previously to the execution of FLAT there is a PKI with a root CA and one CA for each of the participating federations. This root CA signs the CA's certificates of each participating federation. Thus, instead of establishing pairwise trust between the CAs of each federation, there is a trust establishment between the CAs of each federation and the root CA.

In this interfederation stage (assumed to happen before FLAT's execution), there are two steps: the trust establishment between the federations, with the root CA signing the certificates (Interfederation 1) and the distribution of the new certificates (now also signed by the root CA) to the components of the federation (Interfederation 2). Interfederation 1 is shown in Protocol 4.1 and Interfederation 2 is shown in Protocol 4.2.

In Interfederation 1, CA of federation A presents its self-signed certificate to the root CA (step 1, Protocol 4.1). This certificate contains CA's public key together with relevant identity information. Upon receipt of the self-signed certificate, the root CA can then sign it (step 2, Protocol 4.1). The certificate is now signed by CA of federation A itself and also by the root CA.

It is important to note that this process of trust establishment described in Interfederation 1 usually involves a formal contract and it is not done in an automatic manner with the transmission of information through a network.

[1] `https://technical.edugain.org/`

[2] `https://www.geant.org/Projects/Trust_and_Identity/Pages/Home.aspx`

Interfederation 1($federation\ A's\ CA\ CA_A$, $root\ CA\ CA_{root}$)

1. $CA_A \rightarrow CA_{root}$: $\text{cert}_{CA_A}^{CA_A}$
2. $CA_{root} \rightarrow CA_A$: $\text{Sign}(S_{CA_{root}}, \text{cert}_{CA_A}^{CA_A})$

The symbols denote:

$\rightarrow$: unicast transmission
$\text{cert}_{CA_X}^{CA_{X,Y}}$: CA_X certificate signed by CA_X and CA_Y
S_X : X's private key

Protocol 4.1. Interfederation 1.

As before the interfederation stage the CAs do not know each other, they cannot trust the information sent through the network by one another, including self-signed certificates. Thus, this process is done in such a way that the entities can ensure and verify the authenticity of the certificates, and it is usually a manual process or performed through a secure channel pre-established between the entities involved.

Interfederation 2 occurs after the federation's CA has received the certificate signed by the root CA. In this process, the CA will distribute this new certificate to the entities of the federation (SPs and IdPs) so IdPs and SPs from other federations can trust them based on the signature of a common root CA. For example, in Protocol 4.2, CA of federation A sends the certificate to an IdP (IdP A) of the federation. First, CA of federation A sends a certificate signed by itself and the root CA to IdP A, together with a signed nonce (step 1, Protocol 4.2). To acknowledge the receipt of the certificate, IdP A sends the nonce back to the CA (step 2, Protocol 4.2).

Interfederation 2($federation\ A's\ CA\ CA_A$, $federation\ A's\ IdP\ IdP_A$)

1. $CA_A \rightarrow IdP_A$: $\text{Sign}(S_{CA_A}, \text{cert}_{CA_A}^{CA_{A,root}} \mid \text{n}_A)$
2. $IdP_A \rightarrow CA_A$: $\text{Sign}(S_{IdP_A}, \text{n}_A)$

The new symbols denote:

$\mid$: concatenation
n_X : nonce generated by X

Protocol 4.2. Interfederation 2.

4.2 Overview

FLAT is a federated authentication protocol specially designed for IoT that can be executed by devices with computational and storage restrictions. The lightweight au-

thentication solution presented here is based on the following strategies:

- use of only symmetric cryptographic primitives in the Client device, instead of adopting asymmetric cryptosystems;
- use of ECC cryptosystems, instead of computationally expensive cryptosystems like RSA/DSA;
- reduction of the message exchange load in the Client device.

These strategies were taken to provide efficiency to the protocol, especially when considering the Client device, the most restricted entity in the solution.

Before describing FLAT's operation, this section shows an overview of the protocol, explaining its main steps in high-level using as examples a cashless toll system (Fig. 4.1) and a mining truck maintenance system (Fig. 4.2). These examples shown in Fig. 4.1 and Fig. 4.2 show two of the numerous possibilities of application of FLAT.

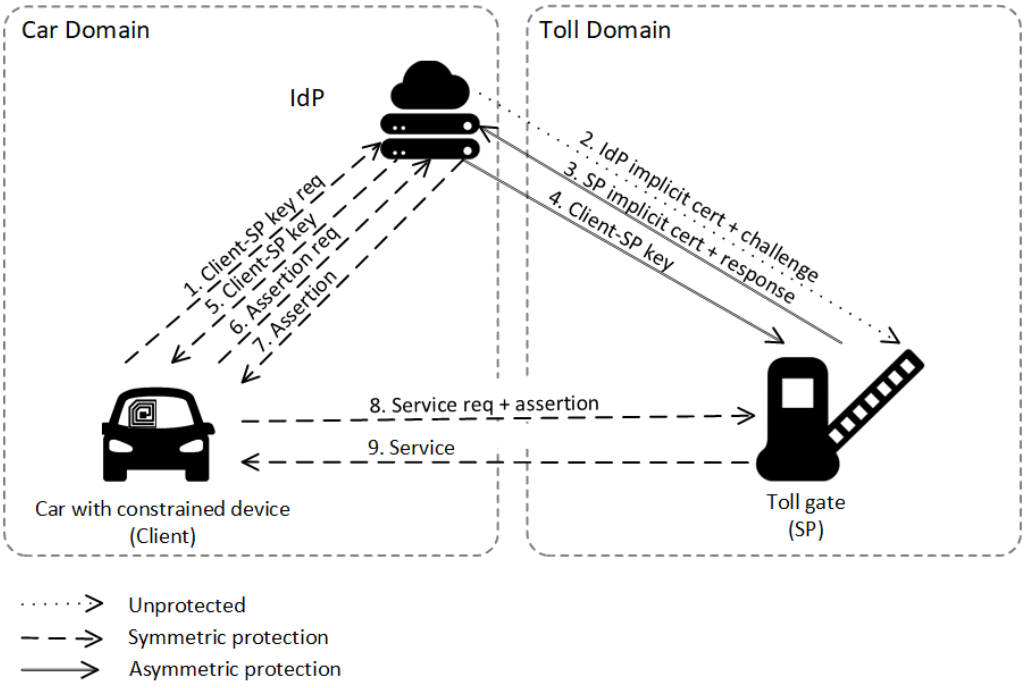

Figure 4.1. Example of use of FIdM: cashless toll system.

In Fig. 4.1, the car is equipped with a restricted device (Client) that communicates with a toll gate (SP) and with its IdP in order to open the gate and be able to pass. The vehicle needs to cross different states with different toll companies.

Remember that prior to the authentication protocol, Client has knowledge of the SP and its services. In step 1 of Fig. 4.1, the Client device sends a key request

to the IdP to communicate with the SP. After establishing a secure channel with the SP, through the exchange of certificates (steps 2 and 3, Fig. 4.1), the IdP sends the Client-SP key to the SP (step 4, Fig. 4.1). Next, the Client also receives the Client-SP key (step 5, Fig. 4.1). After requesting and receiving an assertion from the IdP (steps 6 and 7, Fig. 4.1), the Client can request and access the service provided by the SP (steps 8 and 9, Fig. 4.1).

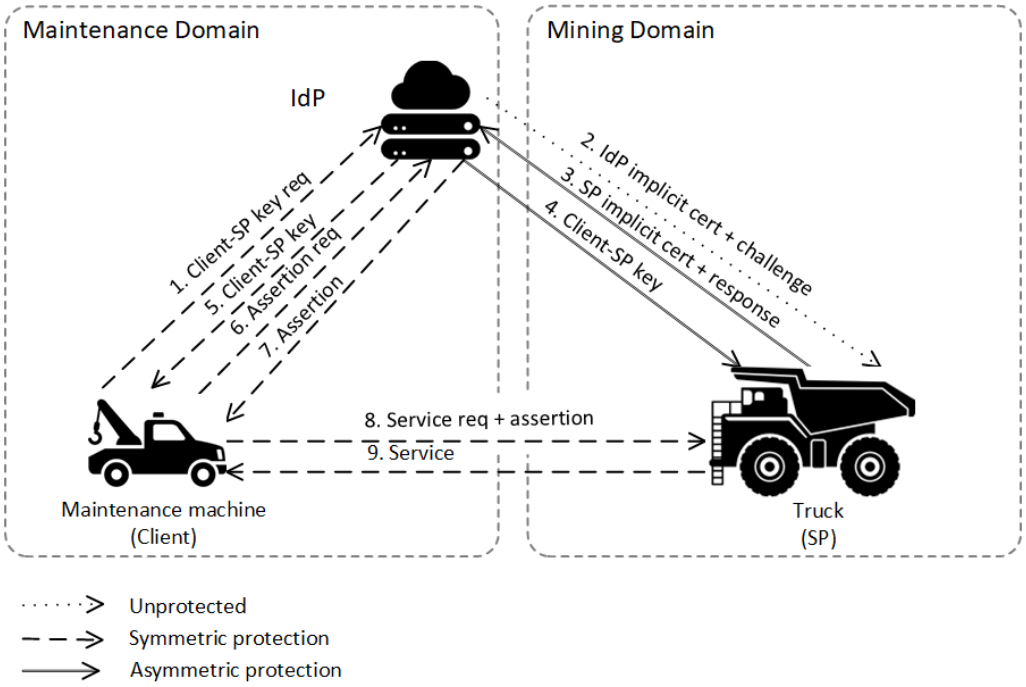

Figure 4.2. Example of use of FIdM: mining.

Considering the Brazilian IoT scenario, it is possible to apply a FIdM model to improve the services in mining in a similar way as the cashless toll system. Consider a situation where the engine of a driverless mining truck has gone inoperative inside a mine (Fig. 4.2). The maintenance machine or technical equipment operated by a maintenance company needs to communicate with the truck's engine, so they can fix it. In such a scenario, devices (truck's engine and technical equipment) from different domains (mining company and maintenance company) need to communicate securely to provide assistance to the truck. In a similar way to the cashless toll system, the same authentication process could be applied to this scenario.

The cashless toll system and the mining truck maintenance system are two possibilities of application of FLAT, that can be used in different scenarios involving authentication between different domains.

4.3 Operation

FLAT's operation is comprised of four stages:

1. Client requests and receives a symmetric key to communicate with the SP;
2. SP receives the symmetric key to communicate with the Client;
3. Client requests and receives the assertion;
4. Client requests and access the service provided by the SP.

Protocol 4.3 shows FLAT's protocol operation. First, the Client device sends a key request to the IdP using the symmetric key (k_IdP/C) previously shared in the pre-deployment stage (step 1.1, Protocol 4.3).

Next, the IdP sends its certificate to the SP (step 2.1, Protocol 4.3) and receives the SP's certificate (step 2.2, Protocol 4.3). After IdP and SP have exchanged certificates, the IdP sends to the SP the symmetric key (k_SP/C), so the SP can communicate with the Client in a secure manner (step 2.3, Protocol 4.3). To finish the stage 2 of FLAT's operation, the SP sends a signed nonce to the IdP to indicate that it received the key (step 2.4, Protocol 4.3).

The next step of FLAT's operation finishes the stage 1 of the protocol when the IdP also sends to the Client the symmetric key (k_SP/C) to communicate with the SP (step 1.2, Protocol 4.3).

Then, the Client requests an assertion to the IdP, indicating the service it would like to access in the SP (step 3.1, Protocol 4.3). The IdP replies to the Client's request, sending the assertion: a signed message containing the service the Client would like to access and a nonce received in step 2.2, indicating this assertion belongs to the same communication initiated before (step 3.2, Protocol 4.3).

With the assertion received, the Client forwards it to the SP using the key k_SP/C (step 4.1, Protocol 4.3). The SP then replies to the Client's request, providing access to the service (step 4.2, Protocol 4.3).

4.4 Message Format

Fig. 4.3 shows the message format used in FLAT's authentication protocol. The first byte is used to determine the message type. As there are nine types of messages in FLAT (key request, Client key, certificate and challenge, key acknowledgment, certificate and response, SP key, assertion request, assertion, service request, and service),

Operation(*Client C*, *Identity Provider IdP*, *Service Provider SP*)

1.1. $C \rightarrow IdP$: $\text{Mac}(k_{IdP,C}, \text{n}_C \mid \text{SP} \mid \mathsf{key\_req})$
2.1. $IdP \rightarrow SP$: $\text{n}_{IdP} \mid \text{cert}_{IdP}$
2.2. $SP \rightarrow IdP$: $\text{Sign}(S_{SP}, \text{cert}_{SP} \mid \text{n}_{IdP} \mid \text{n}_{SP} \mid \text{n'}_{SP})$
2.3. $IdP \rightarrow SP$: $\text{Sign}(S_{IdP}, \text{Enc}(P_{SP}, k_{SP,C}) \mid \text{n}_{SP} \mid \text{n'}_{IdP})$
2.4. $SP \rightarrow IdP$: $\text{Sign}(S_{SP}, \text{n'}_{IdP})$
1.2. $IdP \rightarrow C$: $\text{Mac}(k_{IdP,C}, \text{Enc}(k_{IdP,C}, k_{SP,C}) \mid \text{`serv[1...n]'} \mid \text{n}_C \mid \text{n"}_{IdP})$
3.1. $C \rightarrow IdP$: $\text{Mac}(k_{IdP,C}, \text{Enc}(k_{IdP,C}, \mathsf{assert\_req}(\text{`serv[i]'})) \mid \text{n"}_{IdP} \mid \text{n'}_C)$
3.2. $IdP \rightarrow C$: $\text{Mac}(k_{IdP,C}, \text{Enc}(k_{IdP,C}, \text{assert} = \text{Sign}(S_{IdP}, \text{`serv[i]'} \mid \text{n'}_{SP}) \mid \text{n'}_C)$
4.1. $C \rightarrow SP$: $\text{Mac}(k_{SP,C}, \text{Enc}(k_{SP,C}, \text{assert}) \mid \text{n"}_C \mid \text{n'}_{SP})$
4.2. $SP \rightarrow C$: $\text{Mac}(k_{SP,C}, \text{Enc}(k_{SP,C}, \text{serv[i]}) \mid \text{n"}_C)$

The symbols denote:

$\text{Mac}(k, \text{m})$:	MAC over m calculated using key k
$k_{X,Y}$:	symmetric key shared by X and Y
$\mathsf{key\_req}$:	key request label
$\mathsf{assert\_req}$:	assert request label
cert_X :	X's certificate
P_X :	X's public key
$\text{Sign}(k, \text{m})$:	signature over message m using key k
$\text{Enc}(k, \text{m})$:	encryption over message m using key k
'serv[1...n]':	list of services
'serv[i]':	description of service i
serv[i]:	service i
assert:	assertion

Protocol 4.3. FLAT's operation description.

one byte is enough for this field. The next byte is used for keeping the message sequence number. The following two fields of three bytes each are for the destination and source identification values in the application layer, being able to support around 16 million devices. The next two bytes are used for the payload size. In order to avoid padding in the payload to save bandwidth, the recipients are informed of how much data is being sent in the payload. The last field is the payload, that has a maximum size of 280 bytes.

Note that a sequence number was used to keep track of the message sequence as the implementation of FLAT used UDP as the transport layer protocol. As for the source and destination identification, these bytes are used to identify the devices in the application layer in a specific subdomain.

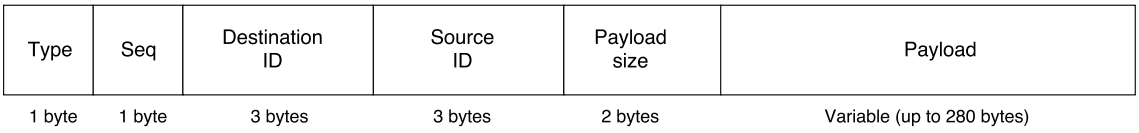

Figure 4.3. FLAT's message format.

4.5 Attack Model

As mentioned in Section 4.1, there is a pre-distribution of the symmetric key used in the $Client \leftrightarrow IdP$ communication. It is assumed the presence of a secure channel in order to share the key before the execution of the authentication protocol.

The adversary may also have physical access to the IoT devices. FLAT considers the use of PUFs to mitigate the risks in such a scenario.

FLAT also assumes that the adversary has access to the messages being transmitted in the protocol. This means that the adversary can eavesdrop and modify messages. The adversary is also able to replay messages previously sent in the protocol. It is assumed that the adversary has knowledge about the authentication protocol and its steps. FLAT adopts a set of countermeasures to secure the messages transmitted in the protocol (see Section 6.3 for an analysis of the countermeasures utilized in FLAT in order to protect the integrity, confidentiality, authenticity, liveness, availability, and privacy).

4.6 Security Level

According to Barker [Barker, 2016], a security level of 128 bits is considered acceptable for current use and also for 2030 and beyond, i.e., this security level is not known to be insecure for the mentioned period. Another document from the National Institute of Standards and Technology (NIST) [Barker and Roginsky, 2018] also considers AES-128, SHA-256 and HMAC (with a key length of at least 112 bits) as safe to use, as well as a security level of 112 bits for elliptic curve signatures.

The security level implemented in the protocol is approximately 128 bits. For the symmetric encryption was used AES-128, that gives a security level of 128 bits. For the HMAC was used the cryptographic hash function SHA-256. Regarding the ECC cryptographic functions (ECIES, ECDSA, implicit certificates), was used the curve BN-254, which was thought to provide a security level of 128 bits. However, a recent work shows that it actually offers a security level of around 100 bits [Barbulescu and Duquesne, 2017].

4.7 Summary

This chapter showed an overview of FLAT, describing some possible applications of the authentication protocol. The assumptions necessary to the execution of FLAT were also explained and a complete description of the protocol was presented. Other aspects of FLAT were explained, such as message format, attack model and security level, giving a complete view of the authentication protocol. The next chapter will describe a prototype of the proposed solution and its implementation details.

Chapter 5

Development and Prototype

This chapter describes FLAT's prototype, its implementation details, and some design choices.

5.1 Prototype Overview

The prototype was built based on a use case scenario where a car from a domain would like to access a parking lot of another domain. For instance, consider that Bob is a researcher from Universidade Federal de São João Del-Rei (UFSJ) and that he has a partnership in a project with Universidade Federal de Minas Gerais (UFMG). Occasionally, Bob will have to go to UFMG to have meetings with other research members, pick up new equipment or even verify the status of some experiments being performed using UFMG's laboratory infrastructure.

Considering that UFMG and UFSJ have already established trust between them, Bob would like to access services in UFMG using his UFSJ's credentials, through an authentication process in his local IdP in São João Del-Rei. Bob has access to an official UFSJ's car and would like to use this car for the route São João Del-Rei $\leftrightarrow$ Belo Horizonte when it would be necessary to visit UFMG. When arriving at UFMG, Bob would like to have access to the parking lot of the building he is going to visit (for instance, the Instituto de Ciências Exatas – ICEx), only available to accredited and authorized personnel and controlled by a parking barrier.

FLAT can be used as a solution to allow Bob to successfully authenticate in his local UFSJ's IdP and access the service provided by the SP at UFMG, opening the parking barrier and being able to park the official car at ICEx.

Based on this idea, a prototype was built using an Arduino Due as a Client device, a Raspberry Pi Zero W as an SP device and a laptop Dell Inspiron as the IdP. Table 5.1

shows the technical specification of the devices related to the protocol entities.

It is important to note that other devices could have been chosen. For instance, the SP could have been represented by an Arduino Due device, similarly to the Client. However, the current design choice regarding the devices was made in order to show the heterogeneity of devices and to evidence that the Client is the most resource-constrained device in the protocol.

Table 5.1. Devices' specification.

Attribute/Device	Client	SP	IdP
Clock	84.0 MHz	1.0 GHz	2.9 GHz
Memory	96.0 kB	512.0 MB	8.0 GB
Storage	512.0 kB	8.0 GB (SD Card)	500.0 GB

A model consisting of the ICEx building at UFMG and its parking lot was built to represent the scenario previously described (Fig. 5.1). A car toy with an Arduino Due approaches the parking barrier (SP). The Arduino will try to authenticate to its IdP. If the authentication is successful, a green LED lights and the car is allowed to pass the parking barrier and enter the ICEx parking lot. If the authentication is not successful, a red LED lights and the parking barrier does not move, and thus, the car is not allowed to pass.

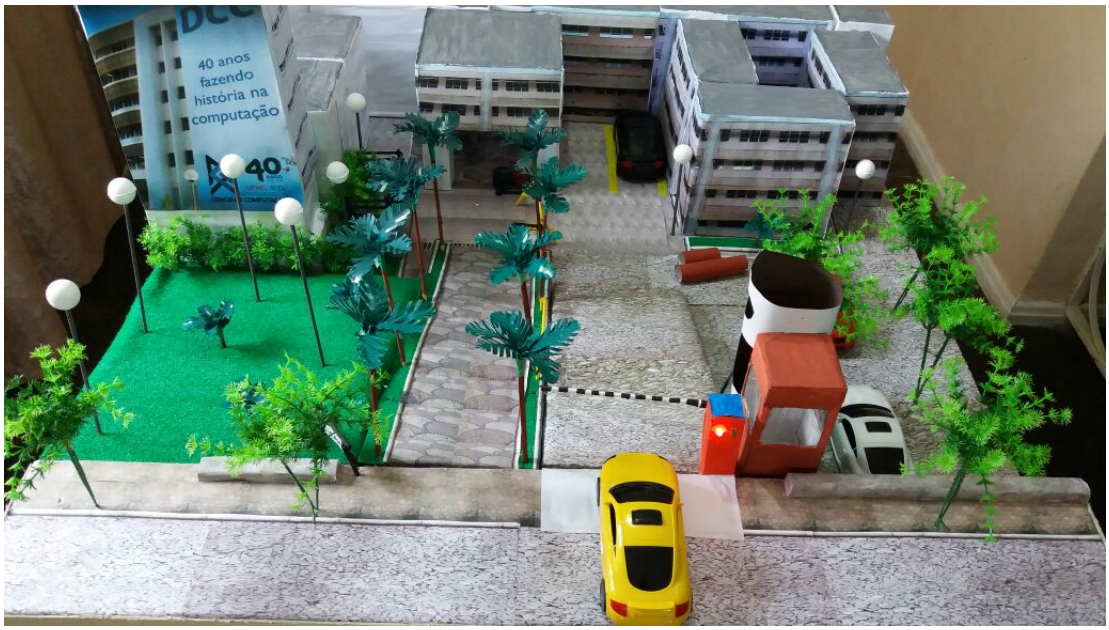

Figure 5.1. FLAT's model: automatic parking system.

The parking barrier is composed of a Raspberry Pi Zero W connected to a servo motor Tower Pro 9G SG90 that controls the movement of the parking barrier. The

Raspberry Pi also controls the red and green LED lights, lighting the green one when the car is authenticated, and lighting the red one when the car is not successfully authenticated. The car toy with the Arduino Due is equipped with an Arduino Wi-Fi shield to enable wireless communication between Arduino (Client) ↔ Raspberry Pi (SP) and Arduino (Client) ↔ Laptop (IdP). The Raspberry Pi Zero W (SP) and the laptop (IdP) are already able to connect to wireless networks without additional equipment. The wireless network is provided by a MikroTik hAP ac lite access point, enabling the communication between the devices in a local network (802.11 b/g). The Arduino is powered by a 9V battery, while the laptop and Raspberry Pi are powered through an adapter/charger connected to an energy outlet.

5.2 Architecture

Fig. 5.2 shows the architecture used in FLAT. The solution incorporates modules already implemented in existing libraries and also new implementations in order to come up with an authentication protocol for IoT.

FLAT's architecture is composed of three parts: the Client, the SP, and the IdP. The Client part has two cryptographic modules: Advanced Encryption Standard (AES) and Hashed Message Authentication Code (HMAC). The AES is used for symmetric encryption and the HMAC is used for MACs present in the protocol description. As the Client only uses symmetric cryptographic primitives, AES and HMAC are the only cryptographic functions implemented in the Client part.

The SP has more cryptographic modules when compared to the Client device. This happens because it implements both asymmetric and symmetric cryptographic primitives. Thus, the SP has five cryptographic modules: AES, HMAC, ECDSA, ECIES, and Implicit Certificates. As in the Client part, AES and HMAC are used for symmetric encryption and MACs, respectively. ECDSA is implemented to allow the use of digital signatures and ECIES is used for asymmetric encryption. The Implicit Certificates implement the ECQV scheme described in Chapter 2, used for the certificate exchange between SP and IdP.

The IdP part implements the same cryptographic modules present in the SP part.

All entities (Client, SP, and IdP) have the FLAT Crypto module. This module is used to adapt the cryptographic modules already implemented in an existing library (and that compose the solution) to FLAT's needs and specifications. Furthermore, all entities have the Authentication module, which actually implements the protocol steps in each of the entities, according to their role in the solution. Other modules present in

all entities (protocol stack necessary for the communication) were already implemented and used in the solution.

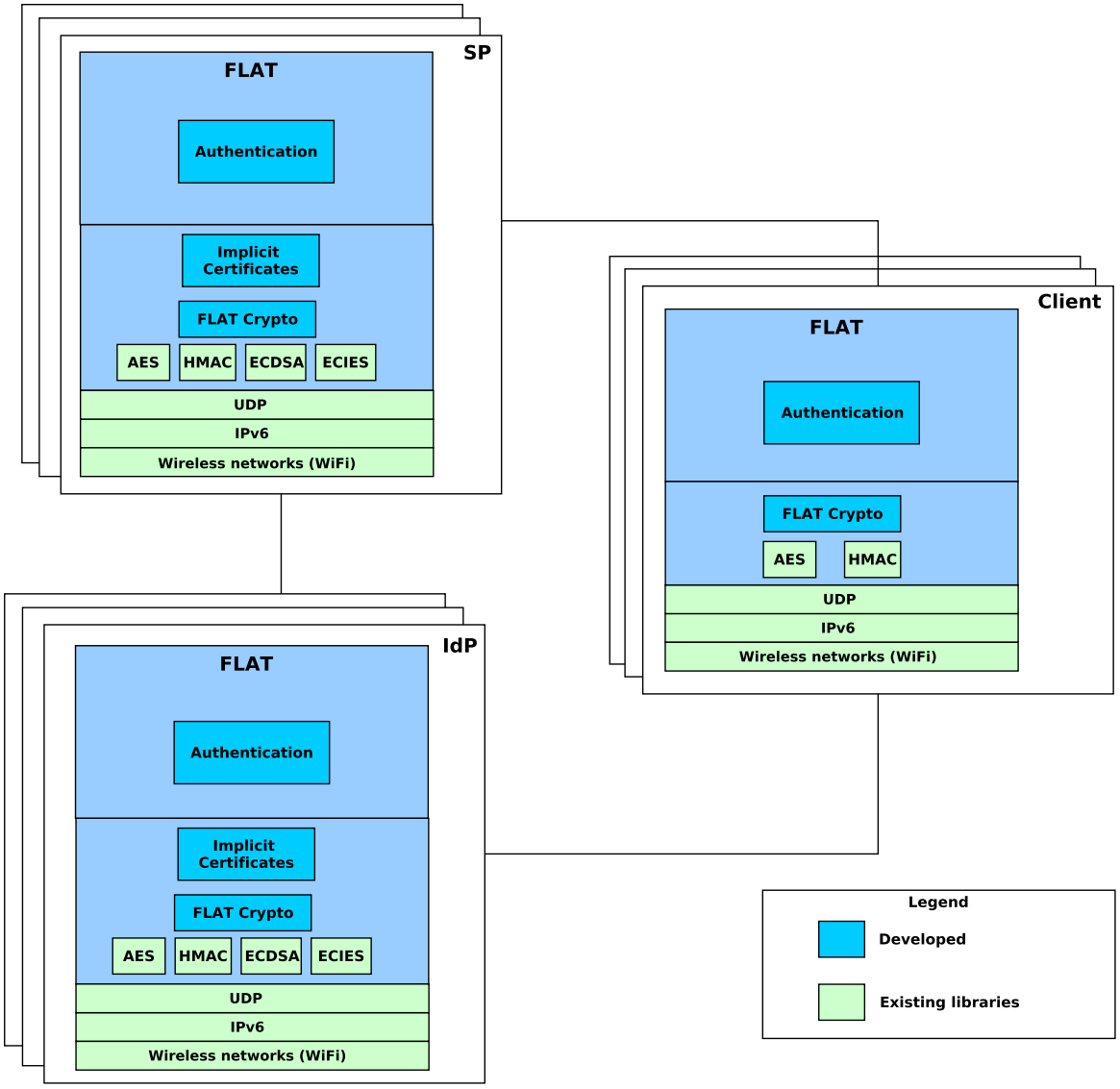

Figure 5.2. FLAT's architecture.

The cryptographic library RELIC [Aranha and Gouvêa, 2018] was used to implement the cryptographic modules necessary to the protocol. The modules AES, HMAC, ECDSA, and ECIES, were already implemented in RELIC and used in the solution with the integration provided by the module FLAT Crypto. The implicit certificates module was not already implemented in RELIC but was built using RELIC's elliptic curve implementation and features. RELIC was chosen for being lightweight and customizable, two features that were really necessary for this project, given the resource

restriction of the Arduino.

The communication implementation in Arduino used the Arduino Wi-Fi library. It allows message transmission using the Arduino Wi-Fi shield coupled to the Arduino Due and offers support to TCP and UDP. In FLAT, the solution was built on top of UDP. This transport protocol was preferred since it is lightweight (i.e., incurs in less computational and bandwidth costs when compared to TCP), and thus, it is more suitable to the IoT scenario.

The integration of the Arduino Wi-Fi library and RELIC was performed using a customized Makefile for Arduino Due[1]. Other libraries used in the solution were RPi.GPIO (to control the GPIO on Raspberry Pi Zero W), C libraries for socket programming in the laptop and Raspberry Pi Zero W and the core Arduino library.

The Arduino code is developed in C/C++. Regarding the libraries used, RELIC is a C library, and the Arduino Wi-Fi library is written in C/C++. The Raspberry Pi Zero W code is all written in C, with a few scripts in Python. For the IdP part, all code is in C. The authentication protocol was developed as a finite-state machine that goes through each step of the protocol. Fig. 5.3 shows a code snippet of the Client's finite-state machine implementation.

```
if (state == 0) // send key to IdP
    if (run_state_0() == EXIT_SUCCESS)
        state = 1;
else if (state == 1) // process key response from IdP
    if (run_state_1() == EXIT_SUCCESS)
        state = 2;
else if (state == 2) // send assert to IdP
    if (run_state_2() == EXIT_SUCCESS)
        state = 3;
else if (state == 3) // process assert response from IdP
    if (run_state_3() == EXIT_SUCCESS)
        state = 4;
else if (state == 4) // send service request to SP
    if (run_state_4() == EXIT_SUCCESS)
        state = 5;
else if (state == 5) // process service response from SP
    if (run_state_5() == EXIT_SUCCESS)
        state = 0;
```

Figure 5.3. Client's finite-state machine code snippet.

Demonstration videos in the parking lot scenario previously described and a cashless toll gate scenario; manuals and other information about the project can be found

[1] https://github.com/pauldreik/arduino-due-makefile

at `https://sites.google.com/view/flat`.

The implementation of an authentication protocol involving different platforms and hardware communication is a challenging task. This is mainly because they have different response times and clocks, especially when dealing with a resource-constrained device such as an Arduino. For instance, the shield's communication buffer only accepts 90 bytes at a time, and thus, it was also necessary to implement fragmentation of messages in the application layer. Fig. 5.4 shows a code snippet referring to message 2.4 (SP to IdP) of FLAT's operation. It is important to note that the SP uses sockets to send the messages (in the same way as the IdP), while the Client makes use of the Arduino Wi-Fi library.

```
// copy IdP's nonce
memcpy(payload, nonce_idp, NONCE_SZ);

// sign message
memcpy(msg, payload, NONCE_SZ);
sig_msg_ecdsa(private_key, first, second, msg);

// Copy signature to payload
memcpy(&payload[NONCE_SZ], sign, SIGN_SZ);

// generating and sending packet
make_pkt(buffer, TYPE_MSG_KEY_SHARE_CONF, IDP_ID, size, payload);
send_msg(sock, buffer, size + MSG_HEADER_SZ, addr, addr_size)
```

Figure 5.4. Code snippet of message 2.4 of FLAT's operation.

5.3 Summary

This chapter described details of a prototype implementing FLAT. This prototype was based on a scenario of authenticating a car equipped with a resource-restricted device in order to access a parking lot of a university. The next chapter shows an analysis in terms of memory usage, computation time, communication and total run-time of the developed authentication protocol.

Chapter 6

Evaluation

This chapter evaluates FLAT under different aspects. Initially, it is conducted an evaluation of the constrained Client device, considering its storage and SRAM usage. Then, FLAT is compared to a Baseline solution in terms of communication, total run-time, and computational costs. This chapter also presents a security evaluation of FLAT.

The results shown in this chapter reflects the average of a total of 100 runs for each of the experiments. For the computation and total run-time evaluation, confidence intervals are presented. Note that the communication results (analytical) and storage results do not vary.

6.1 Client Device

The first analysis of FLAT considers the Client device. This approach was taken because the Client is the most restricted entity in the solution, and thus, the memory and storage usage in such a device is a critical part of the authentication protocol. As the device used to represent the Client in FLAT's prototype is an Arduino Due, the SRAM and storage tests were run on this device.

6.1.1 SRAM Usage

To measure the SRAM usage, the approach was to use a function to measure the free SRAM available in the Arduino when executing a sketch. Two different scenarios were considered: the SRAM used by the Arduino's firmware and the SRAM used when the Arduino Due is running FLAT.

To measure the SRAM used by the Arduino's firmware, an Arduino sketch, whose only goal was to calculate the SRAM in use, was loaded to the Arduino Due board. This application did not execute any other function other than the one to measure the free SRAM in order to calculate the SRAM in use. This first scenario provided a result of 4.8 kB of SRAM in use by Arduino's firmware, equivalent to about 5% of the total available in Arduino Due (Firmware, Fig. 6.1).

The second measurement was taken running FLAT on the Arduino Due board and using the same function that measures the free SRAM to calculate the SRAM usage by FLAT. This scenario did not consider the Arduino's firmware memory usage. The value obtained for this scenario was 10.9 kB, equivalent to 11% of the total SRAM (FLAT, Fig. 6.1). As the Arduino Due has 96 kB of SRAM available, there is 80.3 kB of free SRAM, equivalent to 84% of the total memory available (Free, Fig. 6.1). The usage caused by Arduino's firmware together with FLAT corresponds to 16.4% of the available SRAM.

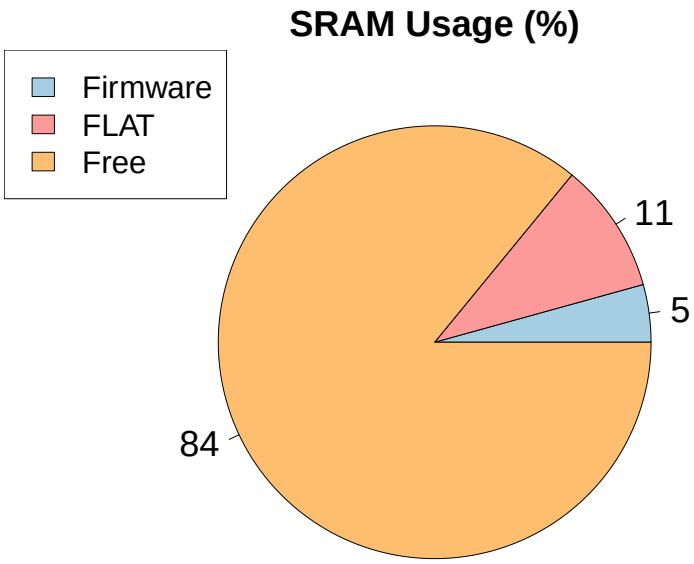

Figure 6.1. SRAM usage in the Client device.

6.1.2 Storage

Similar to the SRAM usage, four different scenarios were considered to measure the storage usage in Arduino Due board: the size occupied by Arduino's native libraries, the size occupied by FLAT's communication functions, the size of FLAT's cryptographic

functions and the total size occupied by FLAT, including Arduino's native libraries. The separation between the communication functions and cryptographic functions was adopted in order to measure which of them has more impact in the storage usage of the Client device.

To measure the size of Arduino's native libraries, an empty sketch that did not execute any specific function was loaded into the Arduino board. Its size in bytes was 73.5 kB (Arduino, Fig. 6.2).

Next, a version of FLAT's Client application without the cryptographic functions was uploaded to the Arduino Due in order to measure the size of the communication functions. The size in bytes occupied by this application (not considering the native libraries) was 6.8 kB (Communication, Fig. 6.2).

FLAT was then uploaded to the Arduino, including the cryptographic library RELIC and all communication and cryptographic functions. The size of the application was measured and then the size of the communication functions and Arduino's native libraries was not considered, in order to calculate the storage usage by the cryptographic library and functions alone (45.5 kB, Crypto, Fig. 6.2). The value measured when FLAT's complete Client application (including RELIC, cryptographic functions and communication functions) is running on Arduino is the total size occupied by FLAT, including the storage usage by Arduino's native libraries (125.8 kB, Total, Fig. 6.2).

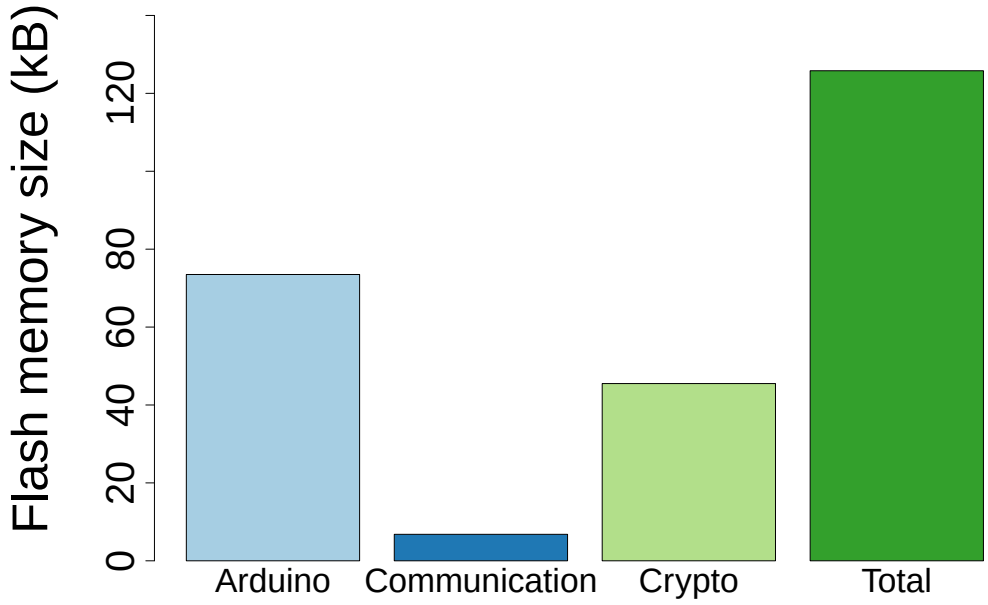

Figure 6.2. Flash memory usage in the Client device.

Recall that the total flash memory available in Arduino Due is 512 kB. Considering this value, FLAT uses 24.6 % of the available flash memory in Arduino Due. The numbers also show that the cryptographic functions and library (not considering Arduino's native libraries) are responsible for the majority of the flash memory used by FLAT.

6.2 FLAT Costs: Communication, computation and total run-time

In order to provide an evaluation of the communication, computational and total run-time costs of the protocol, it was established a generic baseline protocol, based on a modified version of a traditional FIdM authentication protocol (for instance, Shibboleth). The traditional FIdM approach was modified in order to provide an adapted lightweight version. This strategy was adopted since it would not be adequate to compare a solution specially designed for IoT with a traditional FIdM solution without modifications. Throughout this chapter, this baseline protocol will be called Baseline and is depicted in Protocol 6.1. The naming convention used in Baseline's description is the same as defined in the symbols used for Protocol 4.3.

OPERATION(*Client C*, *Identity Provider IdP*, *Service Provider SP*)

1. $C \rightarrow SP$: n_C |$cert_C$
2. $SP \rightarrow C$: SIGN(S_{SP}, $cert_{SP}$ | n_C | n_{SP})
3. $C \rightarrow SP$: SIGN(S_C, ENC(P_{SP}, $k_{SP,C}$) | n_{SP} | n'$_C$)
4. $SP \rightarrow C$: MAC($k_{SP,C}$, n'$_C$ | n'$_{SP}$)
5. $C \rightarrow IdP$: n"$_C$ |'serv[i]'
6. $IdP \rightarrow C$: SIGN(S_{IdP}, n"$_C$ | n_{IdP} | $cert_{IdP}$)
7. $C \rightarrow IdP$: SIGN(S_C, n_{IdP} | n"$_C$+1 | $cert_C$)
8. $IdP \rightarrow C$: SIGN(S_C, n_{IdP} | n"$_C$+1 | ENC(P_C, assert))
9. $C \rightarrow SP$: ENC($k_{SP,C}$, MAC($k_{SP,C}$, n'$_{SP}$ | assert))
10. $SP \rightarrow C$: ENC($k_{SP,C}$, MAC($k_{SP,C}$, serv[i]))

Protocol 6.1. Baseline's description.

Baseline starts with the Client sending its certificate to the SP (step 1, Protocol 6.1). The SP will reply by sending its certificate followed by nonces (step 2, Protocol 6.1). Then the Client generates and sends a symmetric key k_SP/C to the SP and encrypts it using SP's public key (step 3, Protocol 6.1). The SP replies sending back to the Client a MAC using the symmetric key it just received to show that it was able to recover the symmetric key (step 4, Protocol 6.1).

Next, the Client will send to its IdP the service it would like to access (step 5, Protocol 6.1) and the IdP will reply sending its certificate to the Client (step 6, Protocol 6.1). The Client will then send its certificate to the IdP (step 7, Protocol 6.1), that after checking Client's certificate will send back an assertion (step 8, Protocol 6.1).

Client will forward the assertion to the SP (step 9, Protocol 6.1) in order to receive the access to the service (step 10, Protocol 6.1).

Note that at the beginning of each interaction of Baseline (Client $\leftrightarrow$ SP and Client $\leftrightarrow$ IdP) there is a certificate exchange, where both parties have to present their certificates to each other, establishing a secure channel for further communication. This kind of interaction only happens in FLAT in IdP $\leftrightarrow$ SP communication, which are entities with more computational resources than the Client. Furthermore, FLAT uses implicit certificates, while Baseline uses ECDSA certificates.

Another difference is that in FLAT the IdP works as a KDC, generating the keys for the Client $\leftrightarrow$ SP communication. In Baseline, the symmetric key used between Client and SP is generated by the Client. FLAT also reduces the load of message exchange on the Client side, while in Baseline the Client is responsible for starting all the interactions (steps 1, 3, 5, 7 and 9, Protocol 6.1) that are followed by replies from either IdP or SP.

6.2.1 Communication

A comparison of the communication costs of FLAT and Baseline, considering each entity (Client, SP, and IdP) in the authentication protocols is shown in Fig. 6.3. This chart depicts the amount of data received (Rx) and transmitted (Tx) by each entity. For each message transmitted in the protocol, the corresponding sizes of each cryptographic primitive in the message were calculated and summed. This procedure was taken for all the messages of FLAT as well as Baseline. This analytical approach was preferred since the data types used in the implementation might send some extra data. For instance, if we evaluate step 2.4, Protocol 4.3, we will consider the size of the ECDSA signature (64 bytes) together with the size of the nonce (2 bytes). The communication costs only contemplate application layer data, i.e., the sizes considered for evaluation were the messages defined in the protocol description (payload) and the size of the defined message format (Fig. 4.3), i.e., the header. For instance, taking step 2.4, Protocol 4.3 as an example, the total size considered would be 76 bytes (payload of 66 bytes plus a header of 10 bytes). Step 2.4, Protocol 4.3 is sent from the SP to the IdP, so in this case, the evaluation will consider more 76 bytes as transmitted by the SP and more 76 bytes as received by the IdP. UDP/IP headers were not contemplated in this analysis

since it is applied the same underlined technologies for both FLAT and Baseline.

FLAT's Client is around 65% more efficient in Tx, Rx and in total data exchanged (Tx + Rx). FLAT's SP is also more efficient than Baseline's SP: around 9% regarding Tx, 14% in Rx, and 12% in total data exchanged. For the IdP case, however, Baseline's IdP is around 20% more efficient than FLAT.

In order to reduce the communication costs in Client and SP, FLAT puts more load on the IdP side. Even adopting implicit certificates, FLAT's IdP was less efficient than the Baseline's IdP, as it pays the price of the Client's and SP's reduced communication costs. Still, FLAT's total data exchanged (including Client, SP, and IdP) is around 31% more efficient than Baseline.

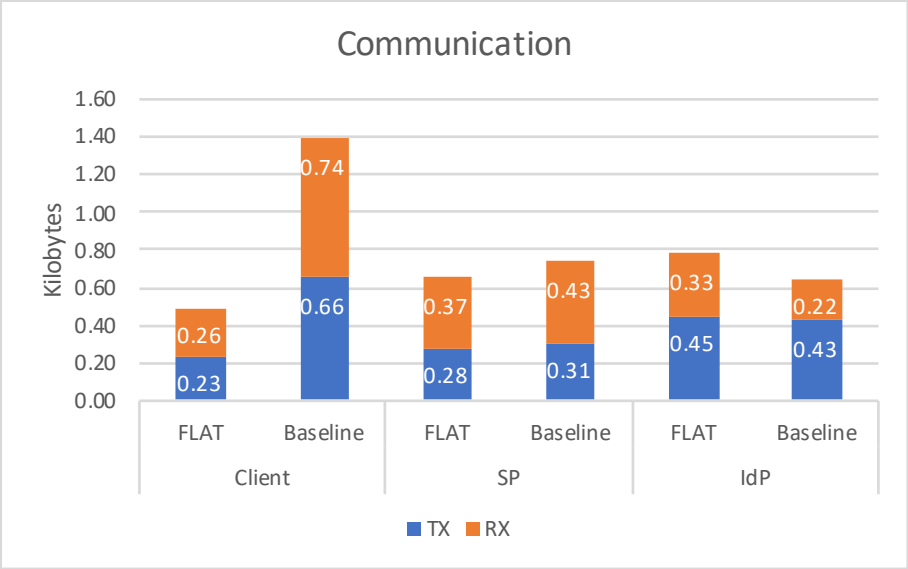

Figure 6.3. Communication costs.

6.2.2 Computation

Fig. 6.4 shows the computational costs of FLAT and Baseline considering Client, SP and IdP instances. The computational costs were calculated considering the time to execute each cryptographic function in each message sent or received by the entities involved in the protocol. First, the time to execute the cryptographic functions (ECIES encryption, ECDSA signature verification, etc.) was measured in each device used in the protocol (Arduino Due, Raspberry Pi, and laptop). Then, considering each message exchanged in the protocol, the times were summed to obtain the total time for the Client, SP, and IdP. For instance, taking step 2.4, Protocol 4.3 as an example, it would be computed the time to generate a signature (around 5 ms) in the SP side and the

time to verify the signature in the IdP side (around 0.5 ms). The time to generate or verify nonces was considered negligible. The time necessary to execute symmetric cryptography in the SP and IdP side was also considered negligible. For instance, in the IdP side, the time necessary to encrypt a message using AES is less than 1 microsecond, with the microsecond being used as the scale for the measurements. The computational costs calculation only included the cryptographic functions necessary to the operation stage of FLAT and Baseline, i.e., did not consider computational costs involved in the steps executed before the protocol, such as certificate generation by a CA and generation of public/private key pairs.

FLAT's Client is around 530 times faster than Baseline's Client (For a 90% confidence interval FLAT's Client computation time is 2.94 ms while Baseline's Client is 1558.97 ± 0.01 ms). However, Baseline's IdP is around 2% faster than FLAT's IdP, and Baseline's SP is around 4% faster than FLAT's SP. For a 90% confidence interval, FLAT's IdP computation time is 2.30 ± 0.06 ms while the Baseline's IdP computation time is 2.27 ± 0.06 ms. In the case of the SP, for a 90% confidence interval, FLAT computation time is 59.41 ± 0.32 ms, while the computation time for Baseline is 57.25 ± 0.05 ms.

FLAT delegates several tasks to its IdP and to the SP, and thus, they carry the load of reducing the computational tasks in the Client side. This explains the fact that FLAT is less efficient than Baseline when comparing the computational costs from IdP and SP. Furthermore, one of the main strategies proposed by FLAT is exactly the use of only symmetric cryptographic primitives in the Client side, in order make

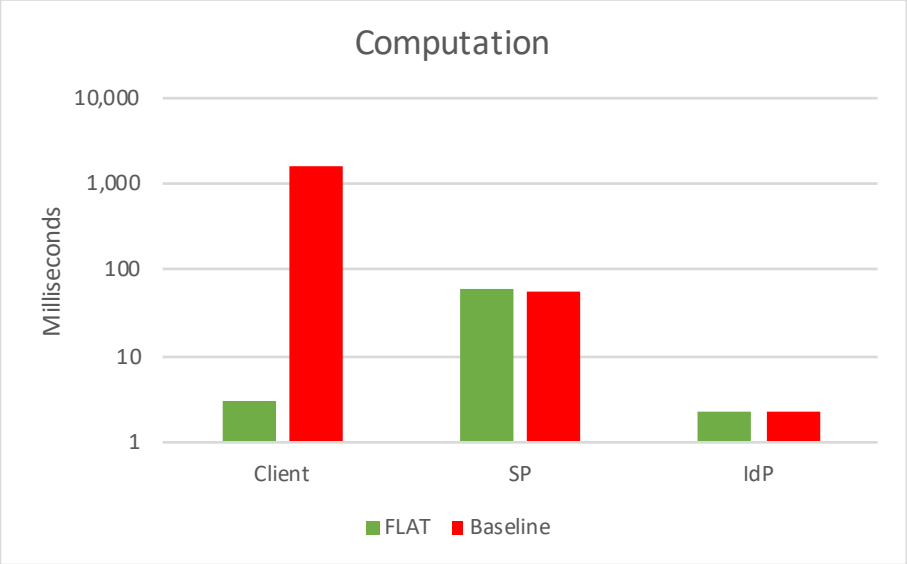

Figure 6.4. Computation costs.

it a lightweight solution for constrained devices. While FLAT adopts this approach, Baseline uses several asymmetric cryptographic primitives in the Client side (including at least five ECDSA signature verifications), which are known to be inefficient. As a result of these different approaches, FLAT's Client is more efficient than Baseline. It is also important to note that although the computation time of FLAT's IdP and SP is higher than the Baseline, the use of implicit certificates indeed reduces the solution's computational costs: note that in Fig. 6.5, signature verification time (ECDSA VER SIG) is higher than the time to extract the public key from the implicit certificate (ECQV PK EXTRACT). For a 90% confidence interval, the SP is able to extract the public key from the implicit certificate in 11.02 ± 0.32 ms, while the traditional ECDSA signature verification takes 13.91 ± 0.03 ms. It should also be observed that among the most costly asymmetric operations are the ECIES encryption (ECIES ENC) and decryption (ECIES DEC), and thus, reducing the number of messages using asymmetric encryption/decryption, also reduces the protocol cost in each entity. In the SP, the ECIES encryption runs in 14.94 ± 0.04 ms, and the ECIES decryption takes 10.5 ± 0.02 ms to run (for a 90% confidence interval). The ECDSA signature generation (ECDSA SIG) is the asymmetric cryptographic operation that takes less time to be executed by SP and IdP (5.04 ± 0.02 ms for the SP with a 90% confidence interval).

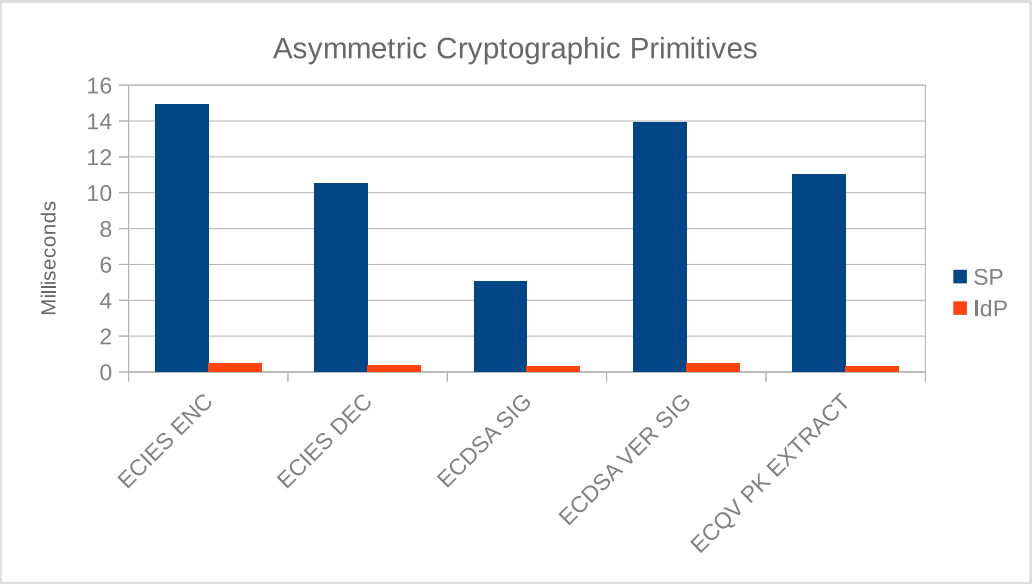

Figure 6.5. Asymmetric cryptographic primitives run-time in SP and IdP.

6.2.3 Total Run-Time

FLAT's and Baseline's total run-time are compared in Fig. 6.6. This evaluation contemplated the time necessary to run all the cryptographic operations executed by all the entities of the protocols (Client, SP, and IdP). The total time is the sum of the computational costs calculated for each device participating in the protocol. This means that, as in the computational costs evaluation, the total run-time also considered only the time to execute the cryptographic functions.

FLAT takes around 65 ms to run (for a 90% confidence interval FLAT runs in 64.65 ± 0.33 ms), which corresponds to around 4% of the total time required to run Baseline (for a 90% confidence interval Baseline runs in 1618.49 ± 0.08 ms).

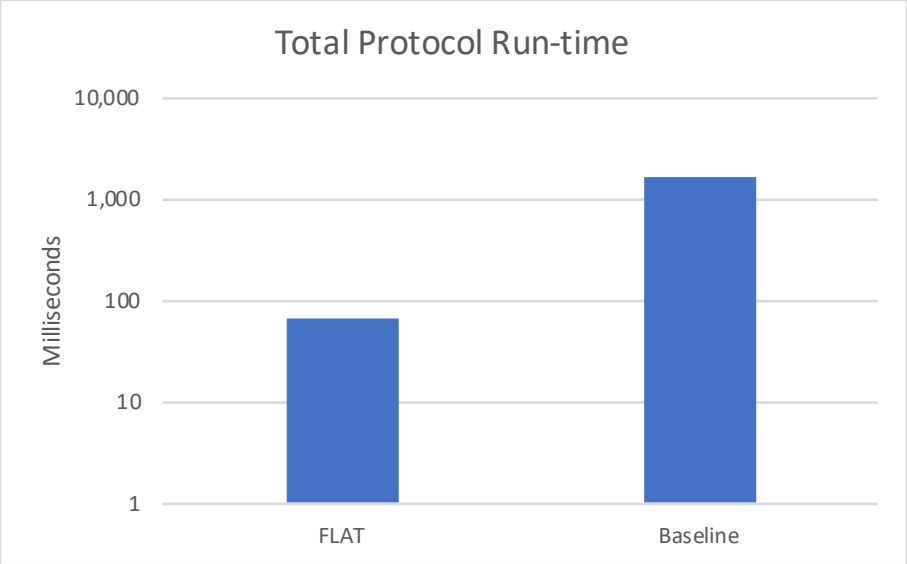

Figure 6.6. Total run-time.

Recalling the previous numbers from the computational and communication costs gives an idea of why FLAT has this advantage over Baseline. In the communication, FLAT is more efficient in terms of total data exchange. In the computation, FLAT's Client is more efficient than Baseline, and only slightly less efficient when considering the IdP and SP. In fact, FLAT reduces the computational costs in the bottleneck of the solution, which is the Client device. With the significant reduction in Client's computation time (remember FLAT's Client is 530 times faster than Baseline's Client), FLAT is able to also reduce the total time required to run the protocol, when compared to the Baseline solution.

6.3 Security Evaluation

This section discusses FLAT under the security properties of authenticity, confidentiality, liveness, integrity, availability, and privacy.

6.3.1 Authenticity

In order to ensure authenticity in $Client \leftrightarrow SP$, $IdP \leftrightarrow SP$ and $Client \leftrightarrow IdP$ communication, FLAT uses MACs and digital signatures.

Regarding $Client \leftrightarrow IdP$ communication, FLAT relies on a pre-shared symmetric key ($k_{IdP,C}$). When the IdP sends a message to Client using this key (steps 1.2 and 3.2, Protocol 4.3), Client knows it is communicating with its IdP, since only the IdP has access to the symmetric key shared through a secure channel in the pre-deployment stage. The same applies to the case when the Client is sending a message to its IdP (steps 1.1 and 3.1, Protocol 4.3). This symmetric key is used in MACs that protect the messages being exchanged, ensuring its authenticity, i.e., that it was really the IdP (or Client) who sent it.

$IdP \leftrightarrow SP$ communication starts with a certificate exchange (steps 2.1 and 2.2, Protocol 4.3). Both SP and IdP certificates are signed by a common CA, and thus, can be verified by both entities. Once the certificate is verified, when the SP receives a message signed by the IdP and verifies it using the certificate's public key, the SP knows it was really the IdP who sent the message (steps 2.3 and 3.2, Protocol 4.3). Similarly, if the IdP receives a message signed by the SP, it can verify its authenticity (steps 2.2 and 2.4, Protocol 4.3).

Communication between Client and SP also makes use of a symmetric key. After the IdP makes sure it is really interacting with the SP (through certificate exchange) and the Client (through the pre-shared symmetric key), it distributes to both SP and Client a symmetric key ($k_{SP,C}$). The authenticity of messages sent between Client and SP is ensured by MACs (steps 4.1 and 4.2, Protocol 4.3).

Note that SP's certificate sent in step 2.2, Protocol 4.3 is included within the signature. As it bears the CA's signature over the certificate's contents, this signature is not necessary. This certificate was included in the signature for the sake of readability.

FLAT makes use of implicit certificates (ECQV) to authenticate $IdP \leftrightarrow SP$ communication. The implicit certificate scheme is considered secure[1], i.e., an attacker is not able to successfully falsify an implicit certificate and a private key that was not issued by a legitimate CA [Brown et al., 2009]. The composition of ECQV implicit

[1]Proof available in Brown et al. [2001].

certificate scheme and ECDSA (as used in FLAT) is also known to be secure [Brown et al., 2009].

6.3.2 Confidentiality

FLAT protects the confidentiality of the messages containing sensitive information. For that, it uses symmetric encryption in $Client \leftrightarrow IdP$ (steps 1.2, 3.1 and 3.2, Protocol 4.3) and $Client \leftrightarrow SP$ (steps 4.1 and 4.2, Protocol 4.3) communication, and also asymmetric encryption in $IdP \leftrightarrow SP$ (step 2.3, Protocol 4.3) communication. Messages not carrying sensitive information, such as nonces and certificates (steps 1.1, 2.1, 2.2, and 2.4, Protocol 4.3), are not encrypted.

A known attack on confidentiality is the eavesdropping of messages, i.e., when an attacker successfully intercepts a protocol message during its execution. As pointed before, messages containing sensitive information are encrypted, and thus, even if an attacker is able to intercept one of FLAT's message, she will not be able to decrypt its contents.

The Man-In-The-Middle attack (MITM) can also affect the confidentiality of the information being sent in FLAT. In such attack, the attacker stays in the middle of the communication between two parties, intercepting and passing on the messages they are sending to each other, while the communicating parties believe they are talking to each other without any interference. A countermeasure to this attack is to enable authentication between the parties, requiring proof that they are really whom they claim to be during the communication. In FLAT's operation, the $IdP \leftrightarrow SP$ communication is authenticated through the use of digital signatures, with certificate exchange before the shared key is sent to the SP. Furthermore, Client's identity is confirmed by the IdP and vice-versa, since their previously shared symmetric key is used in their communication. The communication between Client and SP is also authenticated since their identities have already been verified by the IdP before the shared symmetric key was sent to each of them. This symmetric key is then used by both SP and Client in their communication. Thus, FLAT enables authentication between the communicating parties in order to avoid MITM attacks.

6.3.3 Liveness

An attack on liveness is the replay attack. In this scenario, an attacker intercepts messages from the protocol and can use them later out of the original session or in another moment of the same session. FLAT uses in its operation randomly generated

nonces to ensure the liveness of the messages being sent. Thus, even in case an attacker is able to intercept a message and replays it, the attack will not be successful since the liveness of the message can be verified by the nonces.

Furthermore, FLAT uses a nonce to identify the session with the SP. Besides ensuring the liveness of the message, the nonce n'_SP (received by the IdP in step 2.2, Protocol 4.3) is sent to the Client (step 3.2, Protocol 4.3), so when the SP will receive this message (step 4.1, Protocol 4.3) it will know the message refers to the same service request made in the previous steps.

Another attack that can affect liveness and confidentiality is the oracle attack. In this kind of attack, the adversary uses the system as an oracle, analyzing the answer to a known message in order to obtain information. This kind of attack can be performed using different sessions when the reply related to a specific message sent in a previous session is used by the attacker in another session [Hamid et al., 2010]. As a countermeasure, FLAT uses a pseudo-random generator to generate nonces in order to avoid nonce repetition, and thus, leakage of information. FLAT also uses HMAC to authenticate the symmetrically encrypted messages and signatures for the asymmetrically encrypted messages in order to mitigate the risks of padding oracle attacks.

To increase security the nonces could be encrypted as well. Although it is an unlikely possibility, if an attacker has access to a large number of messages and one of the nonces is repeated at some time, she might be able to perform an attack using an old message. For the sake of readability, nonce encryption is suppressed from the protocol description.

6.3.4 Integrity

To ensure integrity in protocol messages, FLAT uses MACs and digital signatures. In $Client \leftrightarrow IdP$ and $Client \leftrightarrow SP$ communication MACs are used to provide authenticity and integrity to the messages, since it makes use of symmetric cryptography (steps 1.1, 1.2, 3.1, 3.2, 4.1, 4.2, Protocol 4.3). $IdP \leftrightarrow SP$ communication is protected using digital signatures to guarantee non-repudiation, authenticity, and integrity to the messages being exchanged (steps 2.2, 2.3 and 2.4, Protocol 4.3). Step 2.1, Protocol 4.3 is the only one that does not include MACs or signatures (except for the signature in the certificate), sending a nonce and a certificate. Step 3.2, Protocol 4.3, besides the MAC also contains the assertion, which is a signature ensuring the assertion was really issued by the IdP. Although this message is being sent to the Client, it will be forwarded and interpreted by the SP (step 4.1, Protocol 4.3), as the Client can not

process asymmetric cryptosystems.

One attack to integrity is tampering, which is basically making changes to messages being transmitted. MITM attacks can also involve the alteration of messages, i.e., instead of only collecting and passing on the information being sent by each of the communicating parties, the attacker can also change the contents of the messages. In this case, FLAT's adopted countermeasure is to protect the integrity through the use of MACs or digital signatures, as explained before.

Physical tampering is another attack on integrity that can possibly also affects confidentiality. In such a scenario, the attacker will have physical access to the device, getting access to the stored keys or another information. If the attacker can successfully extract the key from the device, the entire authentication process can be compromised. FLAT uses PUFs to generate keys, which are considered robust and of evident tampering [Katzenbeisser et al., 2012]. Thus, even if a physical tampering will occur in the IoT device, there will be changes in its behavior and the key generation will be compromised. In this case, it will not be possible to communicate with the IdP and the protocol will not run.

6.3.5 Availability

Availability is another security property evaluated in the context of FLAT. This security property ensures that the system will be available when the users will need it.

In FLAT, it can be considered an attack where a fake SP sends fake messages to a legitimate Client. At the time the Client receives the advertisement message, it cannot know that it is a fake SP. However, since FLAT incorporates a certificate exchange step between SP and IdP (steps 2.1 and 2.2, Protocol 4.3), the SP will be identified as fake by the IdP and the protocol will not be concluded.

Another possibility would be fake Clients sending series of requests to legitimate SPs in order to achieve a denial of service attack. In this case, Clients would need to communicate with their respective IdP first, since Clients only communicate directly with the SPs in the two last steps of the protocol (steps 4.1 and 4.2, Protocol 4.3). For the protocol to run, the IdP will need to communicate with the SP, confirming its identity through the use of certificates. It is unlikely that a fake Client will be bounded to a legitimate IdP since the symmetric key exchange that guarantees the communication between Client and IdP is done through a secure channel in the pre-deployment stage. The SP, after receiving the certificate (step 2.1, Protocol 4.3), will interrupt the protocol execution, after verifying that the certificate is not valid. Even in the case of a fake Client impersonating an IdP, the result would be the same, as

the certificate will not be valid. If the number of Clients making a request to the SP (which is an intermediary IoT device) is large enough it might be too many requests for it to process. However, in either case, the authentication will not be successful.

Table 6.1. Countermeasures adopted by FLAT.

Property	**Countermeasure**
Authenticity	MACs and digital signatures
Confidentiality	Symmetric and asymmetric encryption
Liveness	Nonces
Integrity	MACs, digital signatures and PUFs
Availability	Authentication between the parties
Privacy	FIdM model: limit the access to identity information

6.3.6 Privacy

As FLAT adopts the federated model, it increases privacy by limiting the number of entities that hold the device's identity information by only its own IdP. The federated model also increases privacy as it allows the entities (user/device) to limit the identity information that they would like to share with the SPs. For instance, the IdP can share with the SP only the necessary data to provide access to the service, avoiding sending all the information it has about a specific user/device.

Other strategies not addressed here (as they are out of the scope of this project) include the improvement of the privacy in federated models by the use of pseudonyms [Birrell and Schneider, 2013].

Table 6.1 summarizes the countermeasures adopted by FLAT in order to reduce the possibilities of attacks related to each of the security properties.

6.4 Summary

This chapter presented a security analysis of FLAT, as well as a performance evaluation of the proposed authentication solution, showing that FLAT is able to run in a resource-constrained device and comparing the results with a reference authentication protocol based on traditional FIdM models (Baseline). The results show that FLAT is a feasible solution, with the Client device being more efficient than the Baseline protocol in terms of computation time and data exchange. The final chapter of this document will summarize the solution and results presented here, as well as possible future works.

Chapter 7

Conclusion

This document described FLAT, an authentication solution specially conceived for the IoT environment. FLAT combines a set of strategies to guarantee efficiency and security in the authentication process. Namely, FLAT approaches the problem of authenticating constrained devices in IoT by: (i) using only symmetric cryptographic functions in the restricted Client device; (ii) using implicit certificates in the IdP and SP; (iii) replacing inefficient cryptosystems like RSA/DSA for others more suitable to IoT; and (iv) reducing the message load in the Client device.

A prototype of the proposed solution was also presented, including the implementation details and the development of a use case scenario where a car should authenticate in a parking barrier to have access to a university's parking lot. All the assumptions based on existing solutions, necessary to the correct operation of FLAT's protocol were also described. Additionally, this work presented an evaluation of the proposed authentication solution in terms of storage, SRAM, communication, computation, total run-time, and security.

FLAT uses around 16% of Client's SRAM and 25% of Client's flash memory when running in an Arduino Due device. FLAT computation in the Client side is 520 times more efficient than the Baseline protocol. Furthermore, FLAT total run-time represents 4% of the total time necessary to run Baseline. FLAT is also 31% more efficient than Baseline when considering the total data exchange.

As IoT gains more visibility and IoT solutions are being incorporated in the most diverse fields (agriculture, transports, smart homes, smart cities, etc.), there is a real need for an IdM that can take care of all the life cycle of an IoT device. The work presented in this document is a step in this direction, providing an authentication solution that is lightweight, contemplates the mobility aspect of devices considering different security domains and has an operational prototype.

The natural path to future works from FLAT is the development of a complete IdM system. By now, FLAT is an authentication solution for IoT. The next steps are completing the IdM solution with the implementations of the authorization, service discovery process, and tear-down.

Once the IdM system is implemented, one possible future work is to evaluate the costs of the service discovery, not covered in this project.

Another possible work includes the proposal and evaluation of different communication possibilities for the Client device. For instance, Client could use the SP as an intermediate to communicate with the IdP, taking advantage of SP's permanent Internet connection and enabling the Client to only have a Bluetooth or low power Wi-Fi connection. There is also another possibility that includes the SP as an intermediate, where the Client will not need a network connection or to communicate directly with the IdP, with the SP resending the messages between Client and IdP. In this case, however, the authentication protocol would be modified.

Bibliography

Aranha, D. F. and Gouvêa, C. P. L. (2018). RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

Armando, A., Carbone, R., Compagna, L., Cuellar, J., Pellegrino, G., and Sorniotti, A. (2011). From Multiple Credentials to Browser-based Single Sign-on: Are We More Secure? In *IFIP International Information Security Conference (IFIP SEC'11)*, pages 68--79. Springer.

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805. ISSN 1389-1286.

B. Oliveira, L., Kansal, A., Priyantha, B., Goraczko, M., and Zhao, F. (2009). Secure-TWS: Authenticating Node to Multi-user Communication in Shared Sensor Networks. In *International Conference on Information Processing in Sensor Networks (IPSN'09)*, pages 289–300. ACM.

Barbulescu, R. and Duquesne, S. (2017). Updating Key Size Estimations for Pairings. *Journal of Cryptology*, pages 1–39.

Barker, E. (2016). Recommendation for Key Management, Part 1: General. In *NIST Special Publication 800–57 Part 1, Revision 4*.

Barker, E. and Roginsky, A. (2018). Transitioning the Use of Cryptographic Algorithms and Key Lengths. In *SP 800-131A Rev. 2 (DRAFT)*.

Birrell, E. and Schneider, F. B. (2013). Federated Identity Management Systems: A Privacy-based Characterization. *IEEE Security & Privacy (IEEE S & P)*, 11(5):36--48.

BNDES (2017). Relatório do Plano de Ação – Iniciativas e Projetos Mobilizadores. In *Internet das Coisas: um plano de ação para o Brasil*.

Brown, D. R., Campagna, M. J., and Vanstone, S. A. (2009). Security of ecqv-certified ecdsa against passive adversaries. *IACR Cryptology ePrint Archive*, 2009:620.

Brown, D. R. L., Gallant, R. P., and Vanstone, S. A. (2001). Provably Secure Implicit Certificate Schemes. In *International Conference on Financial Cryptography (FC'02)*, pages 156--165. Springer.

Cao, Y. and Yang, L. (2010). A Survey of Identity Management Technology. In *International Conference on Information Theory and Information Security (ICITIS'10)*, pages 287--293. IEEE.

Certicom (2009). Standards for Efficient Cryptography 1 (SEC 1) Version 2.0: Elliptic Curve Cryptography. Technical report, Standards for Efficient Cryptography. `http://www.secg.org/sec1-v2.pdf`.

Certicom (2013). Standards for Efficient Cryptography 4 (SEC4): Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV). Technical report, Standards for Efficient Cryptography. `http://www.secg.org/sec4-1.0.pdf`.

Certicom (2018). Explaining Implicit Certificate. `https://www.certicom.com/content/certicom/en/code-and-cipher/explaining-implicit-certificate.html`.

Chadwick, D. (2009). Federated Identity Management. *Foundations of Security Analysis and Design V*, pages 96--120.

Chadwick, D. W. and Inman, G. (2009). Attribute Aggregation in Federated Identity Management. *IEEE Computer*, 42(5):33--40.

Cirani, S., Picone, M., Gonizzi, P., Veltri, L., and Ferrari, G. (2015). IoT-OAS: An OAuth-based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sensors Journal*, 15(2):1224--1234.

Domenech, M. C., Boukerche, A., and Wangham, M. S. (2016). An Authentication and Authorization Infrastructure for the Web of Things. In *Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet'16)*, pages 39--46. ACM.

Fremantle, P. and Aziz, B. (2016). OAuthing: Privacy-enhancing Federation for the Internet of Things. In *Cloudification of the Internet of Things (CIoT'16)*, pages 1--6. IEEE.

Fremantle, P., Aziz, B., Kopeckỳ, J., and Scott, P. (2014). Federated Identity and Access Management for the Internet of Things. In *International Workshop on Secure Internet of Things (SIoT'14)*, pages 10--17. IEEE.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29(7):1645--1660.

Hamid, J., Gianluigi, M., and Lilburn, W. D. (2010). *Handbook of electronic security and digital forensics*. World Scientific.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor. `http://www.rfc-editor.org/rfc/rfc6749.txt`.

Herder, C., Yu, M.-D., Koushanfar, F., and Devadas, S. (2014). Physical Unclonable Functions and Applications: A Tutorial. *Proceedings of the IEEE*, 102(8):1126--1141.

Hong, J., Levy, A., and Levis, P. (2016). Demo: Building Comprehensible Access Control for the Internet of Things using Beetle. In *International Conference on Mobile Systems, Applications, and Services Companion (MobiSys'16)*, pages 102--102. ACM.

Horrow, S. and Sardana, A. (2012). Identity Management Framework for Cloud Based Internet of Things. In *Security of Internet of Things (SecurIT'12)*, pages 200--203. ACM.

Hummen, R., Ziegeldorf, J. H., Shafagh, H., Raza, S., and Wehrle, K. (2013). Towards Viable Certificate-based Authentication for the Internet of Things. In *Workshop on Hot Topics on Wireless Network Security and Privacy (HotWiSec'13)*, pages 37--42. ACM.

Isaakidis, M., Halpin, H., and Danezis, G. (2016). UnlimitID: Privacy-preserving Federated Identity Management Using Algebraic MACs. In *Workshop on Privacy in the Electronic Society (WPES'16)*, pages 139--142. ACM.

Johnson, D., Menezes, A., and Vanstone, S. (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36--63.

Johnson, D. B. and Menezes, A. J. (1998). Elliptic Curve DSA (ECDSA): An Enhanced DSA. In *USENIX Security Symposium (Security'18)*, volume 7, pages 13--23. USENIX.

Jøsang, A. and Pope, S. (2005). User Centric Identity Management. In *AusCERT Asia Pacific Information Technology Security Conference*, page 77. Citeseer.

Katzenbeisser, S., Kocabaş, Ü., Rožić, V., Sadeghi, A.-R., Verbauwhede, I., and Wachsmann, C. (2012). PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon. *Cryptographic Hardware and Embedded Systems*, pages 283--301.

Liu, J., Xiao, Y., and Chen, C. P. (2012). Authentication and Access Control in the Internet of Things. In *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 588--592. IEEE.

Maler, E. and Reed, D. (2008). The Venn of Identity: Options and Issues in Federated Identity Management. *IEEE Security & Privacy (IEEE S & P)*, 6(2).

Markmann, T., Schmidt, T. C., and Wählisch, M. (2015). Federated End-to-end Authentication for the Constrained Internet of Things Using IBC and ECC. *SIGCOMM Computer Communication Review*, 45(4):603--604.

Martínez, V. G., Encinas, L. H., and Ávila, C. S. (2010). A Survey of the Elliptic Curve Integrated Encryption Scheme. *Journal of Computer Science and Engineering*, 2:7–13.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (2001). *Handbook of Applied Cryptography*. CRC Press.

Neto, A. L. M., Souza, A. L., Cunha, I., Nogueira, M., Nunes, I. O., Cotta, L., Gentille, N., Loureiro, A. A., Aranha, D. F., Patil, H. K., et al. (2016). AoT: Authentication and Access Control for the Entire IoT Device Life-cycle. In *Conference on Embedded Network Sensor Systems (Sensys'16)*, pages 1--15. ACM.

Neuman, C., Yu, T., Hartman, S., and Raeburn, K. (2005). The Kerberos Network Authentication Service (V5). RFC 4120, RFC Editor. `http://www.rfc-editor.org/rfc/rfc4120.txt`.

Nogueira, H., Custódio, R., Moecke, C., and Wangham, M. (2011a). Using Notary Based Public Key Infrastructure in Shibboleth Federation. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais – Workshop de Gestão de Identidades (SBSeg'11 – WGID)*. SBC.

Nogueira, M., Santos, A., Torres, J., Zanella, A., and Danielewicz, Y. (2011b). Gerência de Identidade na Internet do Futuro. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – Minicurso (SBRC'11)*, pages 1--6. SBC.

Porambage, P., Schmitt, C., Kumar, P., Gurtov, A., and Ylianttila, M. (2014). Two-phase Authentication Protocol for Wireless Sensor Networks in Distributed IoT Applications. In *Wireless Communications and Networking Conference (WCNC'14)*, pages 2728--2733. IEEE.

Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., and Scavo, T. (2008). Security Assertion Markup Language (SAML) V2.0 Technical Overview. Technical report, OASIS. `http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf`.

Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., and Schmidhuber, J. (2010). Modeling Attacks on Physical Unclonable Functions. In *Conference on Computer and Communications Security*, pages 237--249. ACM.

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C. (2014). OpenID Connect Core 1.0 Incorporating Errata Set 1. Technical report, OpenID Foundation. `http://openid.net/specs/openid-connect-core-1_0.html`.

Shibboleth Project (2018). Shibboleth System Flow – Wiki webpage. `https://wiki.shibboleth.net/confluence/display/CONCEPT/FlowsAndConfig`.

Shim, S. S., Bhalla, G., and Pendyala, V. (2005). Federated Identity Management. *IEEE Computer*, 38(12):120--122.

Silva, C. E. and Silva, G. C. (2017). Uma Proposta de Arquitetura para Autorização Federada com Internet das Coisas. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais – Workshop de Trabalhos de Iniciação Científica (SBSeg'17 – WTICG)*, pages 783--766. SBC.

Stallings, W. (2006). *Cryptography and Network Security: Principles and Practices*. Pearson Education.

Steiner, J. G., Neuman, B. C., and Schiller, J. I. (1988). Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter Conference*, pages 191--202. USENIX.

Suh, G. E. and Devadas, S. (2007). Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Design Automation Conference (DAC'07)*, pages 9--14. ACM.

van der Meulen, R. (2017). Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. `https://www.gartner.com/newsroom/id/3598917`.

Ververidis, C. N. and Polyzos, G. C. (2008). Service Discovery for Mobile Ad Hoc Networks: a Aurvey of Issues and Techniques. *IEEE Communications Surveys & Tutorials*, 10(3).

Wangham, M., Mello, E., Souza, M., and Coelho, H. (2013). Gidlab: Laboratório de Experimentação em Gestão de Identidades. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg'13)*, pages 481--486. SBC.

Whitmore, A., Agarwal, A., and Da Xu, L. (2015). The Internet of Things – A Survey of Topics and Trends. *Information Systems Frontiers*, 17(2):261--274.

Windley, P. J. (2005). *Digital Identity: Unmasking Identity Management Architecture (IMA)*. O'Reilly Media, Inc.

Witkovski, A., Santin, A., Abreu, V., and Marynowski, J. (2015). An IdM and Key-based Authentication Method for Providing Single Sign-on in IoT. In *Global Communications Conference (GLOBECOM'15)*, pages 1--6. IEEE.

Yavuz, A. A. (2013). ETA: Efficient and Tiny and Authentication for Heterogeneous Wireless Systems. In *Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'13)*, pages 67--72. ACM.