GEOVANE FEDRECHESKI

Protecting interactions in IoT Swarms: a self-sovereign and attribute-based approach

Corrected Version

São Paulo
2022

GEOVANE FEDRECHESKI

Protecting interactions in IoT Swarms: a self-sovereign and attribute-based approach

Corrected Version

Doctoral dissertation presented to the Escola Politécnica da Universidade de São Paulo to obtain the title of Doctor in Sciences.

São Paulo
2022

GEOVANE FEDRECHESKI

Protecting interactions in IoT Swarms: a self-sovereign and attribute-based approach

Corrected Version

Doctoral dissertation presented to the Escola Politécnica da Universidade de São Paulo to obtain the title of Doctor in Sciences.

Program:

Electronic Systems Engineering

Advisor:

Marcelo K. Zuffo

Co-advisor:

Laisa C. C. De Biase

São Paulo
2022

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Fedrecheski, Geovane
Protecting interactions in IoT Swarms: a self-sovereign and attribute based approach / G. Fedrecheski -- versão corr. -- São Paulo, 2022.
108 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Internet das Coisas 2.Computação em Enxames 3.Controle de Acesso Baseado em Atributos 4.Identidade Auto-Soberana 5.Segurança Descentralizada I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Aos meus pais.

ACKNOWLEDGMENTS

Many are the persons who deserve my gratitude for helping me during the development of this work.

I would like to begin by acknowledging my parents, Elcio Fedrecheski and Lucia J. Fedrecheski. Even though my career choice implied being physically distant and working on things they were hardly familiar with, they always encouraged me to aim high and work towards my goals. For being there, and for being examples of work ethic to follow, I thank my parents.

My acknowledgements surely cannot miss my advisors, Marcelo K. Zuffo and Laisa C. C. De Biase. I still remember my first talk with prof. Marcelo, and his advice to seek the path of highest return, as work is work anyway. Similarly, it is impossible to forget the countless times when prof. Laisa sat by my side to help me develop the manuscript of my very first research article. For providing a thought-provoking environment and for teaching me resilience, I thank my advisors.

My gratitude is also extended to my advisor during the period I was abroad, professor Jan Rabaey, who received me at his lab (BWRC) and was available to discuss new ideas during a turning point of my research.

I would also like to thank all family members, friends, and colleagues who supported me during the development of this dissertation: Alexandre Yip, Arthur A. Miyazaki, Augusto R. Machado, Fábio do Vale, Jorge Jucoski, Leandro Bueno, Lucas R. Mata, Pablo C. C. Ccori, Rafael Costa, Samira Afzal and William Takeshi.

Finally, my acknowledgements to the Swarm Team and all who I have interacted with at Universidade de São Paulo and UC Berkeley.

I also thank LSI-TEC for supporting funding for the development of this research, and CAPES for providing the funding for my visit scholarship at Berkeley.

"O futuro está ali, é só ir pegar."

ABSTRACT

FEDRECHESKI, G. Protecting interactions in IoT Swarms: a self-sovereign and attribute-based approach. 101p. Dissertation (Doctor in Sciences) - Escola Politécnica. Universidade de São Paulo. São Paulo. 2022.

This dissertation presents contributions to enable autonomous agents in the Internet of Things (IoT) to securely interact with minimal support from third parties. While existing works present security solutions for IoT scenarios, they either use centralized architectures or present only partial solutions for decentralized identification, authentication, confidentiality, and authorization. To solve this gap, we integrate techniques of decentralized identity to authenticate IoT agents, improve a transport-independent protocol for secure communications, and develop a new attribute-based access control model that is suitable for embedded devices. We further consolidate these techniques into a reference architecture and implementation that is integrated within the SwarmOS, a framework for creation of decentralized IoT systems.

Results show that the framework achieves the goal of enabling fully self-protecting agents, while being able to run on embedded devices. Specifically, we tested the whole framework running on a single board computer, demonstrated that the attribute-based access control model can run on a 32MHz microprocessor, and showed that devices can use self-sovereign primitives even on highly constrained networks.

Thus, the key contribution of this dissertation is demonstrating that IoT devices can provide controlled resource sharing while also being fully self-protecting, thus satisfying two crucial requirements (collaboration and autonomy) for the establishment of IoT Swarms in the age of privacy and cybersecurity risks.

Keywords – Internet of Things, Swarm Computing, Attribute-Based Access Control, Self-Sovereign Identity, Decentralized Security.

RESUMO

FEDRECHESKI, G. Protegendo interações em IoT Swarms: uma abordagem auto-soberana e baseada em atributos. 101p. Tese (Doutor em Ciências) - Escola Politécnica. Universidade de São Paulo. São Paulo. 2022.

Esta tese apresenta contribuições para permitir que agentes autônomos na Internet das Coisas (IoT) interajam de forma segura com o mínimo de suporte de terceiros. Embora os trabalhos existentes apresentem soluções de segurança para cenários de IoT, eles usam arquiteturas centralizadas ou apresentam apenas soluções parciais para identificação, autenticação, confidencialidade e autorização descentralizadas. Para resolver essa lacuna, integramos técnicas de identidade descentralizada para autenticar agentes de IoT, otimizamos um protocolo para comunicações seguras independente da camada de transporte e desenvolvemos um novo modelo de controle de acesso baseado em atributos adequado para dispositivos embarcados. Consolidamos essas técnicas em uma arquitetura de referência e implementação integrada ao SwarmOS, um *framework* para criação de sistemas IoT descentralizados.

Os resultados mostram que o *framework* atinge o objetivo de habilitar agentes totalmente auto-soberanos, ao mesmo tempo em que pode ser executado em dispositivos embarcados. Especificamente, testamos todo o *framework* rodando em um computador de placa única, demonstramos que o modelo de controle de acesso baseado em atributos pode rodar em um microprocessador de 32MHz e mostramos que os dispositivos podem usar primitivas auto-soberanas mesmo em redes altamente restritas.

Assim, a principal contribuição desta tese é demonstrar que os dispositivos IoT podem fornecer compartilhamento de recursos de forma controlada ao mesmo tempo em que são totalmente auto-soberanos, satisfazendo assim dois requisitos cruciais (colaboração e autonomia) para o estabelecimento de enxames IoT na era dos riscos de privacidade e da segurança cibernética.

Palavras-chave – Internet das Coisas, Computação em Enxames, Controle de Acesso Baseado em Atributos, Identidade Auto-Soberana, Segurança Descentralizada.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

As the Internet of Things becomes pervasive, the increasing number of computational agents that operate independently of human assistance opens way to two possible scenarios. On one side, a large cohort of devices that are always connected to (and dependent on) management servers, collecting data and receiving commands under centralized architectures. On the other, a distributed collection of heterogeneous things that interact spontaneously, sharing data and services on demand, and self-organizing to better serve their human controllers. In this dissertation, we adopt the latter as a likely future scenario, and investigate what are the necessary means to implement security in such a dynamic environment.

1.1 Problem Definition

An important but easy to overlook difference between the Internet of Things and the Internet is the set of requirements to implement security. This is natural since the concepts have an overlap: both consider the existence of a global communication system, which must be protected from threats.

On the other hand, the most obvious difference between the IoT and the Internet is that the former includes computer-equipped physical objects that are capable of sensing, actuating, and processing information. These objects, or things, often feature resource constraints and network limitations, thus imposing certain requirements on the design of security mechanisms.

Furthermore, internet security mechanisms were conceived considering one user per computer interacting with other users, a model we call the 1:1 model, where security procedures can be fully configured and controlled by users and third parties (system administrators, for example). The advent of IoT brought as a legacy these security mechanisms configured by humans, however, unlike few personal computers on the internet, in the case of IoT users and system administrators face the problem of configuring tens,

hundreds or even thousands of devices. This context presents several challenges related to the possibility of autonomous or semi-autonomous security configuration mechanisms.

The IETF Internet Security Glossary (SHIREY, 2007) defines security as "a system condition that results from the establishment and maintenance of measures to protect the system". While Internet protection measures can be mainly derived from security principles, such as bit-level protection (DAEMEN; RIJMEN, 2001), IoT security mechanisms must also consider the trade-offs imposed by the limitations of its devices and networks. For example, some IoT networking protocols have severely limited bandwidth, which imposes a lean design on the security layer. Another difference is that the Internet is composed of IP-based networks, while the IoT tends to use heterogeneous networks, i.e., a mix of IP and non-IP environments. Implementing end-to-end security across these heterogeneous IoT networks is a considerable challenge.

Even if the challenges of device capabilities and heterogeneous networks were removed, commonly used measures to protect the Internet are not enough to protect more advanced IoT approaches, such the Swarm (LEE et al., 2014). Swarm is a distributed collection of cooperating IoT devices (COSTA et al., 2015), which act together to achieve a common goal. In its fullest form, a Swarm is expected to feature the emergence of a collective intelligence. In order to enable the creation of IoT Swarms, Costa et al. (2015) proposed an architecture for the SwarmOS control plane. It proposes that each IoT device runs one or more agents, where one of these agents, called the Broker, has the special role of providing common resources to regular agents, such as registration, discovery, and contracting (COSTA et al., 2015; CALCINA-CCORI et al., 2018; BIASE et al., 2018b). For example, a smartphone-based agent might use its local Broker to discover a nearby lamp, and then send the lamp a message to turn it on.

Current solutions to protect the Internet and the IoT do not suffice to protect the Swarm. For example, Transport Layer Security (TLS) (RESCORLA, 2018), and its variant DTLS (RESCORLA; MODADUGU, 2012), used to protect packets at the transport layer, rely on a centralized Public Key Infrastructure (PKI) (FEDRECHESKI et al., 2020), which is incompatible with the decentralization of swarm-based systems. In addition, typical IoT systems are managed by cloud-based frameworks that orchestrates fleets, manages identities, and specifies access control policies (KOVATSCH et al., 2022; OCF..., 2022; CHEHAB; MOURAD, 2020; GABILLON; GALLIER; BRUNO, 2020).

In this context, this work investigates and proposes solutions to the problem of **how to enable secure interactions in IoT Swarms**. That is, considering that the Swarm

is a distributed collection of cooperative IoT agents (COSTA et al., 2015), our challenge is to investigate a security framework that (1) protects Swarms according to known threat models and (2) does not violate properties of the Swarm architecture.

1.2 Approach

Since Swarms can neither be centrally managed, nor can rely on centralized roots of trust, a significant amount of common IoT and Internet security infrastructure must be rethought. This approach is not isolated, though, as evidenced by the movements to (re-)decentralize the web (BARABAS; NARULA; ZUCKERMAN, 2017) and present decentralized solutions for identity management.

Two emerging approaches that could be employed to solve this problem are Self-sovereign Identity (SSI) and Attribute-Based Access Control (ABAC). The SSI approach proposes that entities should be in full control of their digital identities (ALLEN, 2016), which improves ownership, privacy, and fosters decentralization (FEDRECHESKI et al., 2020). Emerging standards such as Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) (SPORNY et al., 2019a; SPORNY et al., 2019b) are being proposed to enable creation of self-sovereign digital entities. Although SSI still faces challenges in the IoT space, such as communication and processing overhead, its alignment with the Swarm decentralization makes it an attractive option to securely identify Swarm agents.

Conversely, the Attribute-Based Access Control approach can enable flexible authorization in the Swarm, which is needed to enable cooperation between Swarm agents. ABAC is a system in which attributes of interacting agents (including their cyber-physical context) are used to compute an access decision (HU et al., 2013). The use of generic attributes in ABAC provide flexibility at the cost of complexity for policy evaluation. Thus, the main challenge to adopt ABAC in IoT Swarms consists in leveraging its flexibility while keeping compatibility with constrained devices. In fact, existing solutions employing ABAC consider its use only in resource-abundant servers and gateways (CHEHAB; MOURAD, 2020; GABILLON; GALLIER; BRUNO, 2020; OUADDAH et al., 2017; RAVIDAS et al., 2019).

1.3 Hypothesis

The hypothesis of this work is:

Interactions among very large fleets of computer devices, even the constrained ones, can be secured without directly assigned interactions and without centralized management.

1.4 Contributions

The main contribution in this dissertation is the research, development, and evaluation of a framework that enables Swarm agents to protect their own interactions. The framework enables confidential, authenticated, and authorized interactions, without violating key principles of the Swarm, such as autonomy, decentralization, heterogeneity, and scalability.

Additional contributions include:

- An attribute-based access control architecture incorporated into the Swarm.
- An attribute-based access control model that supports expressive and lightweight policies, i.e., can be evaluated on constrained devices.
- A lightweight method for self-sovereign identification of IoT agents, which satisfies the requirements of Swarms.
- A network-agnostic protocol for end-to-end secure communications that can be authenticated by decentralized identifiers.

Work towards this dissertation has also directly generated two journal articles, and two conference papers, as listed below:

- FEDRECHESKI, G. et al. Attribute-based access control for the swarm with distributed policy management. *IEEE Transactions on Consumer Electronics*, IEEE, v. 65, n. 1, p. 90–98, 2019.
- FEDRECHESKI, G. et al. Self-sovereign identity for iot environments: a perspective. In: IEEE. *2020 Global Internet of Things Summit (GIoTS)*. [S.l.], 2020. p. 1–6.
- FEDRECHESKI, G. et al. SmartABAC: enabling constrained iot devices to make complex policy-based access control decisions. *IEEE Internet of Things Journal*, IEEE, 2021.

- FEDRECHESKI, G. et al. A low-overhead approach for self-sovereign identity in iot. In: IEEE. *2022 Global Internet of Things Summit (GIoTS)*. [S.l.], 2022. p. (in press).

1.5 Document structure

The text of this doctoral dissertation is organized into six chapters. Chapter 1, the current one, presented the problem to be solved, as well as the hypothesis and the contributions that were achieved. Next, Chapter 2 presents a literature review regarding protection of IoT device interactions, which is further divided into data models for digital identity and models for attribute-based access control. After having identified the gaps in the literature, Chapter 3 presents our proposed framework to address the problem of secure interactions in IoT Swarms. The framework is presented using a layered and top-down approach, starting with authorization, going through authentication, confidentiality, and identity; at the end, a combined operational flow is described. Chapter 4 presents relevant details of how the proposed framework was implemented, and Chapter 5 evaluates the framework and discuss its results. Again, the evaluation starts top-down by layers, then ends with a overall evaluation. Finally, Chapter 5 concludes the manuscript, presenting the key findings of this work as well as recommendations for continuing the research.

2 LITERATURE REVIEW

Several works have addressed the issue of security in IoT environments. The main gap lies in attempting to create completely self-sovereign IoT devices capable of interacting in an unsupervised manner. The following paragraphs provide an overview of related works, some of which partially fulfill this gap.

Note: Parts of the review presented in this section have been extracted from previous works. Specifically, Section 2.2 is derived from Fedrecheski et al. (2020), and Section 2.3 uses tables and text from Fedrecheski et al. (2021).

2.1 Security in IoT environments

This section presents and discusses the literature for IoT security from the point of view of trying to provide fully autonomous secure devices. Table 1 consolidates how related works compare to the solution proposed in this dissertation.

A number of standards have proposed security solutions for interactions of IoT devices. The Web of Things (KOVATSCH et al., 2022) proposes specifications and recommendations for interoperable and secure IoT platforms compatible with Web standards. To uniquely identify things, WoT relies on Universal Resource Names (URNs). For authentication of things interactions, WoT recommends using one or a combination of symmetric keys, raw public keys, or certificates. For encryption, it recommends different schemes based on the target layer, such as DTLS for transport layer and COSE for application layer. In cases where more complex authorization is required, it recommends using OAuth 2.0, which uses the CapBAC approach, where an authorization server issues capability tokens to things. The WoT does not cover the specification of authorization policies nor the use of attributes for authentication and authorization. Furthermore, concepts of decentralized identity are also not explored in the WoT architecture.

OneM2M (oneM2M..., 2022) proposes a set of specifications for development of IoT

and M2M solutions. It uses a custom identifier format named Application Entity Identifier (AE-ID) which may be locally or globally unique. Authentication in OneM2M is provided either via raw symmetric or public keys, or via certificates. For authorization, it defines a format of access control policies that must be enforced during each request. In addition, it supports generation of an access token that can be serialized in JWT notation. Communication security is provided by either TLS or DTLS. OneM2M differs from our work in that it does not consider decentralized identity mechanisms, nor attribute-based authentication and authorization. Finally, transport-independent secure communications, an important requirement in heterogeneous Swarms, is not considered as well.

The Open Connectivity Foundation (OCF. . . , 2022) proposes a set of specifications with the aim of enabling interoperability in the Internet of Things. To identify devices, it uses UUIDs. To authenticate interactions, either raw keys or certificates are used. Finally, authorization is provided via either identity or role-based policies that can be enforced by either the device or a proxy. One difference from our work to OCF is that their use of UUIDs does not imply on an implicit link to public keys and endpoints (as is the case with DIDs). Moreover, our proposal uses attribute-based policies and credentials, which enhances flexibility for spontaneous environments. Finally, although OCF considers RESTful communications, it does not consider transport-independent security mechanisms.

Some works focused on identity management, for example, by proposing a smart city system that leverages two blockchains to manage identities and access policies, respectively (ASAMOAH et al., 2020)*. The identity blockchain is public and stores hashes of device identifiers, while the authorization blockchain is private and stores and evaluates access policies. The public one resembles our proposal in which we use a blockchain to store identifiers and related metadata, with the difference that in our work the blockchain use is optional. The approach to access control differs from our work since we perform the access decisions locally. This work also does not mention the end-to-end protection of IoT interactions.

Others also used a secure hardware module to generate an identifier locally on the IoT device (SU et al., 2020). Associated metadata and credentials are then recorded, respectively, on a blockchain and on IPFS, a distributed file system. The system is evaluated through an electric charging station use case, in which a vehicle presents a signed credential to the station to authenticate the charging transaction. This proposal is not ideal for protecting Swarms since it does not provide fine-grained authorization nor message security. Moreover, having all device credentials always public may lead to

Table 1: Related work.

Work / Features	Identification	Credentials	Authorization	Confidentiality	Integrity	Authentication	Non-Repudiation
This work	DID	VC	ABAC + CapBAC	DIoTComm	DIoTComm	DIoTComm	DIoTComm
Kovatsch et al. (2022)	URN	RPK, Certificate	RBAC, CapBAC	DTLS	DTLS	DTLS	
oneM2M... (2022)	AE-ID	RPK, Certificate	IBAC, RBAC, CapBAC	DTLS	DTLS	DTLS	
OCF... (2022)		RPK, Certificate		DTLS	DTLS	DTLS	
Seitz et al. (2021)	Flexible	PoP Token	CapBAC				
Fotiou et al. (2019)	DID	DID	IBAC				
Lagutin et al. (2019)	DID	VC	*	TLS	TLS	TLS	
Antonino et al. (2019)	DID	VC	IBAC				
Siris et al. (2020)	DID	VC	CapBAC	TLS	TLS	TLS	
Su et al. (2020)	DID	VC					
Asamoah et al. (2020)	Derived	Vector	IBAC				
Novak, Tang & Li (2018)	Audio F.	*	IBAC		Audio F.	Audio F.	
Ghosh et al. (2019)	*	Trust Level	IBAC, CapBAC				
Sharma et al. (2021)	*	RPK, Trust Level	IBAC, Trust-based	Trust-based	Trust-based	Trust-based	
Din et al. (2021)		Certificates, Trust Level				Trust-based	
Pérez et al. (2019)	*	RPK	IBAC	EDHOC + OSCORE	EDHOC + OSCORE	EDHOC + OSCORE	
Grande & Beltrán (2020)	Soft Fingerprint	Token	CapBAC	DTLS	DTLS	DTLS	

Legend: RPK - Raw Public Key; * Unclear.

privacy issues.

Some works also proposed the use of self-sovereign identity concepts, such such as using ephemeral DIDs for electric vehicle charging (ANTONINO et al., 2019). This approach creates a new DID for every new interaction, therefore the vehicle mobility cannot be tracked by the charging stations. Although this approach improves privacy, it did not address the costs associated with issuing and managing ephemeral DIDs, as well as how to link these DIDs to credentials. This proposal also does not satisfy the requirements of secure communications and flexible authorization.

Fotiou et al. (2019) proposed using decentralized identifiers along with permissioned blockchains to enable opportunistic user access in IoT systems. In their work, guests entering a network are assigned a DID which is then also inserted in an access control list on an IoT gateway. They also consider the case where multiple decision points are used to compute an authorization decision which is then used as input to a pre-defined consensus algorithm. This work is similar to ours in that it uses DIDs to identify entities in IoT systems. It differs from ours since DIDs may be generated by third-parties and authorization is primarily based on unique identifiers (not attributes). In addition, this work do not provide a layer for secure communications.

Lagutin et al. (2019) consider the case of constrained IoT devices that cannot handle DIDs or VCs. The authors propose using a DID and VC -aware OAuth server to generate authorization decisions to highly constrained IoT devices. They illustrate the solution

through a printer use case in a university setting, where a guest user gets a token from the authorization server and uses it to print a document. An implementation based on Hyperledger Indy is also provided, however no resource usage evaluation is performed. This work differs from ours in its fundamental assumption, since we seek to develop IoT agents that can fully handle their own security.

Other works have focused on providing opportunistic, or spontaneous, interactions between IoT agents. Fernando et al. (2019) explored the benefits and challenges of opportunistic selection and use of fog services in IoT contexts. Among the benefits of this approach, are the support for scalable real-time interactions as well as optimized geographical resource distribution. On the other hand, some challenges hinder the deployment of such flexible fogs, including the lack of more dynamic and scalable simulators, the sustainability of business models, and the security and privacy of these systems. Indeed, establishing trust over spontaneous links consists of a significant challenge that spans many areas.

Others yet leverage ultrasound to create spontaneous interactions between physically close devices, such as laptops and displays in the same room (NOVAK; TANG; LI, 2018). To authenticate devices, it relies on audio fingerprinting, which yielded good results in the operational phase, albeit it requires preparing a significant training set beforehand. This approach enables indoor positioning and transmission of very small packets, however it suffers from scalability and throughput limitations.

Yet another approach for obtaining flexible interactions is to rely on trust scores based on the history of past interactions. Ghosh et al. (2019) proposed a context-aware and behavior-based authorization system that takes into account the history of past interactions to compute an access decision for smart homes. While this approach is interesting to enable flexible authentication and authorization, its proposed mechanisms for behavior analysis and authorization are performed on a centralized gateway, which is incompatible with the requirements of IoT Swarms.

Sharma et al. (2021) modeled and simulated Sec-IoT, a privacy-preserving and trust-aware framework for cloud-assisted data routing between IoT nodes. Privacy is achieved by encrypting public keys with the aid of a central cloud server, while trust is derived from multiple direct and indirect sources, such as delivery capability and neighbor trust. This work differs from ours, firstly, because it focuses on network routing, while we focus on secure interactions. Moreover, it relies on a central server and does not consider the issues of decentralized identification and flexible authorization.

Din et al. (2021) uses a centralized trust agent to emit trust certificates to IoT nodes. It considers both direct trust from observed parameters and indirect trust inferred from other nodes as well as historical data. This work differs from ours in that it relies on a centralized party and does not comprise identification, secure messaging, and authorization.

Grande & Beltrán (2020) proposes an edge-centric authorization mechanism to support constrained devices that lack cryptographic capabilities. In this system, IoT devices ask an edge device for an access token, which in turn forwards the requests to a cloud server. If the request is valid, a token is generated by the server and returned to the device to be used in future interactions. It differs from our work since it relies on a centralized server to compute authorization decisions and issue an access token. Also, this work does not consider the issues of decentralized identity and secure communications.

Another work (SIRIS et al., 2020) presents different models for decentralized authorization based on smart contracts and multiple blockchains. To enhance performance, it proposes a public blockchain to store payment information and a private blockchain, to store and execute access control policies. This approach is not suitable for Swarms, which should enable computation of access decisions by the IoT devices themselves.

Pérez et al. (2019) proposed an end-to-end application-level secure key derivation scheme based on the EDHOC standard and applied it to OSCORE, a method for protection of RESTful messages in constrained environments. More specifically, the authors implemented EDHOC in a constrained device, and included an optimization to simplify the negotiation of security parameters. This approach is promising to reduce the overhead of end-to-end secure channels in IoT. Nevertheless, this work does not consider the topics topics of identity, authorization, and autonomous IoT devices.

Another effort specified Authentication and Authorization for Constrained Environments using the OAuth 2.0 Framework (ACE-OAuth) (SEITZ et al., 2021). The proposal consists in leveraging the existing OAuth token-based approach, with Proof-of-Possession (PoP) tokens, and combining it with more lightweight building blocks, such as CBOR and CoAP. Similarly to our proposal, in ACE-OAuth a binary token is obtained for use during resource requests. Still, these approaches differ in three main aspects. First, our system supports application-level authentication via Verifiable Credentials, which enables flexible and decentralized attribute-based authentication. Second, we include evaluation of ABAC policies in the scope of our solution, making it complete with respect to authorization. Third, we propose to execute all operations on devices themselves, instead of

relying on an authorization server.

Mahalle, Shinde & Shafi (2020) discusses the use of DIDs and VCs for identity management in IoT environments. The authors highlight the expected benefits of DIDs, such as being cryptographically verifiable, and raise questions regarding potential privacy issues due to reuse of identifiers. This study argues that DIDs and private keys should be controlled by third parties, such as device owners. While we agree that mechanisms for ownership and guardianship of IoT devices are necessary, our position is that it can be done via VCs, thus without exposing sensitive information such as private keys.

In a previous work (FEDRECHESKI et al., 2020) we discussed the benefits and challenges of self-sovereign identity for IoT environments. We have identified privacy, ownership, and decentralization as the most attractive features of this approach. We also provided a detailed comparison with existing models for identity, such as X.509 certificates and PGP Keys. Still, we identified many challenges for adoption of SSI in IoT environments, including the heterogeneity of devices and networks and the lack of open source tools to support development of SSI solutions in the IoT realm.

As highlighted in the previous paragraphs, there are many gaps to the development of self-sovereign agents capable of secure and spontaneous interactions in IoT Swarms. In the following sections, we present basic definitions and specific challenges to solve these gaps, and then proceed to present our system design.

2.2 Data models for digital identity

This section provides analysis and comparisons of existing data models for digital identity. We start by discussing the limitations of password-based accounts, and then proceed to compare data models based on public key cryptography. The main conclusion is that recent standards for SSI improve privacy and decentralize identification of users and IoT devices.

2.2.1 Accounts

The most basic way to identify subjects in computer systems is through *accounts*. Accounts are digital records that include a user name and password, and they identify a user. Accounts are typically stored on a server that is controlled by the service provider. Many IoT vendors require that device owners have an account with the cloud in order to configure their devices.

Account-based authentication is one of the most primitive solutions for digital identities, and although it has been used for decades in the computing industry, it has a number of drawbacks. Among these are the need for manual configuration, the lack of privacy, and the risks associated with using passwords. Since the IoT is expected to reach thousands of devices per person, it is not feasible to manually configure accounts for each one. Additionally, storing plaintext PII in a third-party system raises privacy concerns. With respect to the use of passwords, there are two main problems: password reuse and difficulty enforcing strong passwords. The most common solution is to provide recommendations on how to choose strong passwords, but this relies on people following the recommendations, which can be difficult to enforce. (TANESKI; HERIČKO; BRUMEN, 2019).

2.2.2 Models based on public key cryptography

Pretty Good Privacy (PGP) (CALLAS et al., 2007) is a system that allows individuals to prove that a public key is associated with a particular identifier, which typically includes a real name and email address. This binding, as well as any additional attributes or signatures, is stored in a document called a PGP key. PGP keys can be signed by other individuals in the PGP system in order to endorse that the key holder is who they claim to be, providing a level of trust between peers.

The X.509 certificate standard defines a format for public key certificates (PKC) that binds public keys to qualified names (COOPER et al., 2008). Although technically nothing prevents peer-to-peer signature of X.509 certificates, the vast majority of its usage is under centralized architectures, in which a trusted authority signs the certificate to make it trustworthy. In certain cases, it is useful to have a separate document that, instead of having a public key, contains only a name associated with signed attributes. To meet this demand, X.509 proposed a new standard called Attribute Certificate (AC), which contains no public key, but links to a PKC through its *subject* field (FARRELL; HOUSLEY; TURNER, 2010).

Finally, Self-Sovereign Identity is a novel approach that uses Decentralized Identifiers (SPORNY et al., 2019a) and Verifiable Credentials (SPORNY et al., 2019b) to prove possession of identifiers and attributes.

2.2.2.1 High-level comparison

The following paragraphs compares models used by the PGP, X.509, and SSI standards, according to Table 2.

Table 2: Comparison of standardized data models for digital identity.

Item	PGP	X.509		Self-Sovereign Identity	
	PGP Key	Public Key Certificate (PKC)	Attribute Certificate (AC)	DID Document (DDo)	Verifiable Credential (VC)
Goal	Prove control of public keys and identifier (plus optional attributes) Publish public keys	Prove control of public keys and identifier (plus optional attributes) Publish public keys	Prove possession of attributes	Prove control of identifier Publish public keys and service endpoints	Prove possession of attributes
Identifier	Name and Email	Qualified Name	Same as PKC	Method-specific DID	Same as DDo
Uniqueness of identifier	Global authority (DNS)	Global authority (CA)	Same as PKC	Ledger consensus / Random number gen.	Same as DDo
Public Key(s)	1 primary, N subkeys	1	n/a (points to PKC)	N	n/a (points to DDo)
Attribute(s)	Attributes field	Extensions field	Attributes field	-	subjectCredential field
Endorsement	Signature of many peers	Signature of a CA	Signature of a CA	Self-signed (optional) Indirect through VC	Signature of an Issuer
Service endpoints	-	-	n/a	Yes	n/a
Semantic schemas	-	-	-	Yes	Yes

Source: Fedrecheski et al. (2020).

Goal The main goal of PGP Keys and PKCs is to publish and prove control of public keys that are tied to identifiers. Also, in these approaches, attributes can be provided either in the same document as the public keys (PGP Key and PKC), or, in the case of X.509, in a separate document (AC). On the other hand, documents in the SSI paradigm have decoupled goals: DDos are be used to prove control of an identifier and to provide a means for establishing a secure communication; and VCs are used to prove possession of attributes.

Identifier SSI uses decentralized identifiers that can be generated automatically, for example by using strong random number generators. This enables global uniqueness and enhances privacy, when compared to PGP and X.509, which use real names and email addresses, respectively. Pseudonymous identifiers are also more suited for IoT, since devices do not have names or email addresses.

Public Key(s) PKC keys can only be used for one public key, while PGP and DDos keys can be used for multiple keys. PGP uses a primary key that is linked to an identifier, while DDos allows different types of public keys to be used.

Attribute(s) In self-sovereign identity, a DDo does not support attributes to preserve pseudonymity. Instead, all PII is handled only by VCs, which are private by default.

PGP Keys and X.509 certificates support arbitrary attributes, either via PKC extensions or dedicated ACs.

Endorsement(s) PGP Keys can be signed by one or more peers, but X.509 certificates and VCs can only be signed by a single issuer. DDos are not signed by external entities, and may be self-signed. When a DDo is written to a ledger, however, the transaction will be signed, which can be used to attest the validity of the DDo. Another way of proving endorsement over a DID is to check the signature of a VC associated with that DID. If the VC is signed by a trusted issuer, the DID can be trusted. Furthermore, with respect to who can sign the endorsements, technically it can be anyone, but there are philosophical differences. X.509, for example, was devised to work within a centralized architecture, where only trusted authorities can sign certificates. On the other end of the spectrum, PGP expects peer-to-peer signatures, which ultimately creates a Web of Trust. Finally, VCs does not make strong assumptions on the network structure, although decentralized approaches may be favorable.

Service endpoints A key feature of DDos is that it allows for secure interactions between different devices and systems, by providing a way to easily reach the owner of DID through web-based endpoints. This makes it ideal for use in web and IoT applications.

Semantic schemas only SSI-based data models allow extensibility through semantic annotations over JSON documents. The main reason for this is that these technologies only became popular after X.509 and PGP were developed.

2.2.2.2 Public key distribution

In the following, we compare three approaches for key distribution, as shown in Table 3: Raw Public Key (RPK), Public Key Certificates, and DID Document.

Raw public key In this approach, an entity would share their public key as a raw array of bytes. This is decentralized and does not disclose any personal information, however it does not allow associated metadata.

Public Key Certificate X.509 PKCs are not ideal for privacy purposes because they typically include PII in the main identifier and other attributes. Other drawbacks include

Table 3: Comparison of data models for key distribution.

Feature	RPK	PKC	DDo
Associates key material to metadata		X	X
Privacy: no PII disclosed	X		X
Key rotation does not requires re-signing	n/a		X
Serialization formats	Binary Base64	DER PEM	JSON-LD JWT
Semantic schemas			X
Decentralized: user generates the artifact	X		X
Decentralized: user carries the artifact	X	X	X
Service endpoint			X

Source: Fedrecheski et al. (2020).

the imposition of specialized serialization formats, tight coupling of keys and data, and a centralized architecture.

DID Document The main advantage of DDos is that they allow public keys to be associated with pseudonymous metadata, while also allowing key rotation without having to re-sign any associated metadata. This is possible because all signed metadata actually only exists in associated VCs. DDos also support JSON-based serialization formats, which are available in most programming languages and platforms, and can benefit from publicly available semantic schemas. As each user auto-generates their own DIDs and DDos, the management of the identifier is decentralized.

2.2.2.3 Attribute distribution

Four out of the five previously described formats can be used to prove control over attributes: PGP Keys, Public Key Certificates, Attribute Certificates, and Verifiable Credentials. Since PGP Keys are less widely used, we only compare the latter, as shown in Table 4.

Public Key Certificates The X.509 certificate standard defines a format for public key certificates (PKC) that can be used to bind public keys to qualified names (COOPER et al., 2008). PKCs are commonly used in the Internet to authenticate domain names and protect communications. Attributes can be embedded in PKCs through the use of an *extension field*. However, this approach has the drawback of requiring the whole certificate to be re-signed whenever a key is rotated or selective attribute disclosure is necessary.

Table 4: Comparison of data models for attribute distribution.

Feature	**PKC**	**AC**	**VC**
Signed attributes about a subject	X	X	X
Key rotation does not requires re-signing		X	X
Identifier differs from key material	X	X	X
Attributes decoupled from key material		X	X
Selective disclosure			X
Zero-knowledge proofs			X
Delegation		X	
Revocation	X	X	X
Serialization formats	DER PEM	DER PEM	JSON-LD JWT
Semantic schemas			X
Decentralized: user carries the artifact	X	X	X
Decentralized: Verifier decoupled from Issuer			X

Source: Fedrecheski et al. (2020).

Attribute Certificates The X.500 working group has proposed a new standard called Attribute Certificate (AC) to meet the need for signing attributes instead of public keys. ACs differ from PKCs in that they contain a name (that references a PKC) and a list of attributes, but no public key, which simplifies key rotation (FARRELL; HOUSLEY; TURNER, 2010). However, ACs are not compatible with decentralized verification, semantic attributes, and selective attribute disclosure.

Verifiable Credentials VCs are a type of digital credential that focus on binding identifiers to attributes, which enables simpler key rotation when compared to ACs. VCs are serialized in JSON, which is both human readable and lightweight to parse. VCs can be further specialized into two formats: JSON Linked Data (JSON-LD)[1], a format to serialize linked data; and JSON Web Token (JWT)[2], a widely used format to express security claims. Finally, as opposed to previous approaches, VCs are specified to be used in decentralized environments, and considers decoupled verification and compatibility with Decentralized Identifiers (DIDs).

[1]https://json-ld.org/
[2]https://jwt.io/

2.3 Models for attribute-based access control

Attribute-Based Access Control (ABAC) [3] is a system in which attributes from users, objects, and their context, are used to create access policies, thus enhancing flexibility and scalability, at the cost of increased complexity (HU et al., 2013). It emerged in the last two decades to solve scalability issues faced by previous systems, such as Access Control List (ACL) and Role-Based Access Control (RBAC) (HU et al., 2013). ABAC is capable not only of representing many traditional access control systems but also of offering new dimensions for policy creation, such as the use of context attributes in the access rules (HU et al., 2013).

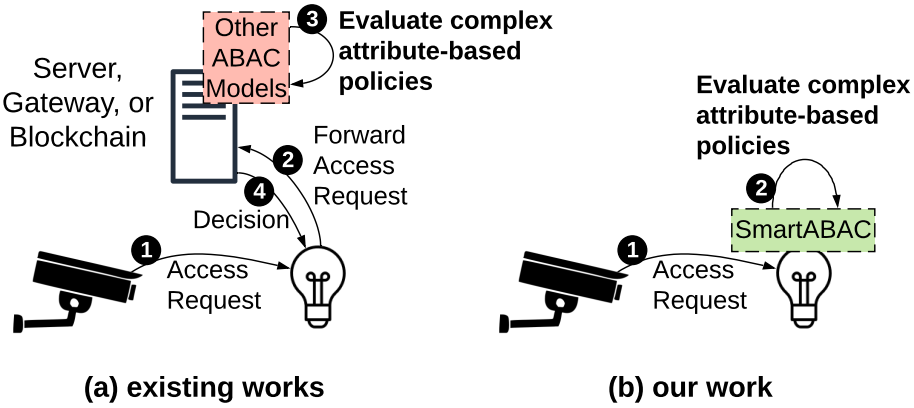
Unlike previous access control systems (SANDHU et al., 1996), however, there is still no consensus about the best way to represent attributes and policies in ABAC. For example, while some models construct policies using logic statements, others use an attribute-enumeration approach. Also, while some works emerged from the industry (PARDUCCI; LOCKHART; RISSANEN, 2013), recent academic works are shifted towards formally specified models (FERRAIOLO; ATLURI; GAVRILA, 2011; SERVOS; OSBORN, 2014; BISWAS, 2017).

Moreover, current ABAC models are primarily focused on cloud scenarios, leaving a gap for an ABAC model optimized for constrained IoT devices. Specifically, considering that device A wants to communicate with device B, most works use an external actor C to compute access decisions. For example, the actor C could be a server, a gateway, or a blockchain, as shown in Figure 1. In the following subsections, we discuss existing ABAC models and related work using ABAC in IoT.

2.3.1 ABAC Models

A widely used ABAC model in the industry is the eXtensible Access Control Markup Language (XACML) (PARDUCCI; LOCKHART; RISSANEN, 2013), which enables expressive logic-based policies through a rich data model. XACML has been explored on different scenarios, such as protecting applications based on the Fiware IoT framework (ESPOSITO; FICCO; GUPTA, 2021) and those based on publish-subscribe architectures (KIM; KIM; ALAERJAN, 2021). One drawback of XACML is that it is based on XML, a format known to be highly verbose, making it inappropriate to use in constrained IoT

[3] Not to be confused with Attribute-Based Encryption (ABE). While ABAC is used to control access to resources by intercepting access requests, ABE uses attributes to encrypt plain text data (SAHAI; WATERS, 2005).



Source: Fedrecheski et al. (2021).

Figure 1: The ABAC approach in this work vs. existing works.

devices and networks. Moreover, its use of logical language to define policies leads to monolithic policies that are complex to process and NP-complete to audit (BISWAS; SANDHU; KRISHNAN, 2016b; BISWAS, 2017).

Hierarchical Group and Attribute-Based Access Control (HGABAC) proposes a formal model for implementing Hierarchical ABAC along with a logic-based acccess policy (LAP) language (SERVOS; OSBORN, 2014). For attribute inheritance, HGABAC uses Group Hierarchies, in which users assigned to a child group can inherit the attributes of their parents. Another work (BHATT; PATWA; SANDHU, 2017) showed that it is possible to represent Group Hierarchies through attribute containment, a more straightforward approach. HGABAC uses typed attributes, which give useful information for policy building and evaluation. Finally, as it uses the LAP aproach, it faces similar complexity challenges as XACML to evaluate and audit polices.

In contrast with LAP models, other works have explored enumerated access policies (EAP). Enumerated models tend to create micro-policies, and are bound to polynomial-time algorithms (BISWAS; SANDHU; KRISHNAN, 2016a), which are simpler to process. The main drawback is that current EAP models have limited expressiveness, e.g., due to lack of context attributes and data types.

Policy Machine (PM) (FERRAIOLO; ATLURI; GAVRILA, 2011) was the first formal EAP model for access control and it is capable of representing different access control systems, such as ACL and ABAC. PM introduced the concept of attribute hierarchies via attribute containment, which is equivalent to partial ordering and allows attributes "lower" in the hierarchy to be considered *contained* within the "greater" attributes. On the other hand, the PM is a single-attribute EAP, and thus it does not allow creating conjunctive policies without adding more complexity, such as policy classes. The PM also

lacks support for numerical and contextual attributes. Even though more recent proposals used the Policy Machine to develop new models (BISWAS; SANDHU; KRISHNAN, 2016b; BHATT; PATWA; SANDHU, 2017), the mentioned limitations were not addressed.

A more recent model is $EAP\text{-}ABAC_{m,n}$, the first EAP model to use multi-attribute policies (BISWAS, 2017). While it provides a simple way to represent conjunctive policies, this model lacks support for hierarchies and data types, which decreases its scalability and expressiveness.

In the Cyberspace-Oriented Access Control (CoAC) model (LI et al., 2018), enumerated policies are modeled using general subjects, access scenes, and general objects. In particular, access scenes enable a comprehensive description of the context around interacting agents, including location, time ranges, and network paths. Still, this work does not consider attribute types and does not evaluate the model in real IoT environments.

Finally, others also defined new ABAC models that were based on EAP, and applied them to the use case of automatic policy generation (JABAL et al., 2020; KARIMI et al., 2021). These works, however, did not investigate the expressiveness of the models, nor considered execution on constrained IoT devices.

2.3.2 ABAC in IoT

Although the flexibility of ABAC makes it a great candidate for the IoT, the application of the latest advances in ABAC models to IoT systems, with few exceptions (BEZAWADA; HAEFNER; RAY, 2018; GABILLON; GALLIER; BRUNO, 2020), has been largely unexplored. In general, related works either adapt existing solutions initially designed for different use cases, such as XACML (PARDUCCI; LOCKHART; RISSANEN, 2013), or implement ad-hoc solutions (ZHANG et al., 2021), which hinders interoperability. We believe that this gap occurs mainly because a standard for ABAC has not been achieved yet (SERVOS; OSBORN, 2017); most solutions are ad-hoc, too specific, or adaptations of existing systems (PARDUCCI; LOCKHART; RISSANEN, 2013; CHEHAB; MOURAD, 2020; ALKHRESHEH; ELGAZZAR; HASSANEIN, 2020). In these conditions, we could identify ABAC being used in different domains, such as healthcare (ALNEFAIE; CHERIF; ALSHEHRI, 2019) and smart home (BEZAWADA; HAEFNER; RAY, 2018).

Recent works have also explored ABAC with blockchain for IoT, an interesting approach that enables decentralized trust. In some systems, blockchains are used for identity

management (TAN et al., 2021) and decentralized verification of attribute authenticity (GILANI et al., 2020). This approach is scalable since the ledgers act only as distributed trust anchors for verification keys (KUPERBERG, 2019; SPORNY et al., 2019b), thus requiring low storage and computational costs. It does not provide, however, a solution for local access control decisions, which is the main goal of this paper.

Others have used blockchains to store access control policies and compute access decisions (DRAMÉ-MAIGNÉ; LAURENT; CASTILLO, 2019; PAL et al., 2019), thus guaranteeing that policies and decisions are trusted. This approach, however, requires that devices query the blockchain to ask for an authorization decision. Furthermore, it may expose private attributes since the blockchains are public. Even though recent works (ZHANG et al., 2021) tackled the problem of protecting attributes by using a permissioned blockchain, it still imposes transaction costs, latency, and overhead for access decision computation that is often measured in seconds (PAL et al., 2019).

Several works have used ABAC to protect applications using the Message Queuing Telemetry Protocol (MQTT). Among them is MQTT ABAC (GABILLON; GALLIER; BRUNO, 2020), one of the first models in the literature to propose an ABAC model specifically tailored for IoT. It is optimized towards expressiveness and can express various kinds of contextual rules, such as frequency of messages. One limitation of this work is that its logical language can only protect entities that use the MQTT protocol. A different approach proposed a protocol-agnostic ABAC model focused on intelligent transportation system (ITS), and applied it to a scenario that uses MQTT (GUPTA et al., 2021). Named ITS-ABAC$_{\mathrm{G}}$, this model uses group hierarchies to assign attributes to entities, and relies on a logic language to specify access policies. Others have also used existing ABAC models for MQTT environments, and focused on enabling decentralized authorization by using an MQTT broker proxy (COLOMBO; FERRARI; TÜMER, 2021). An important difference from these approaches to ours is that, due to the broker-oriented architecture of MQTT, the authorization is computed by an intermediary device.

Another work proposed to execute access control decisions within constrained devices and combining it with AI-assisted context inference (CHEHAB; MOURAD, 2020). Acknowledging the overhead of XACML, they adopt a more lightweight approach based on set-based algebra. However, this solution does not work on highly constrained devices, since it was only demonstrated on a smartphone with a 1.4 GHz CPU and 1 GB of RAM, which acts as a gateway for more limited IoT devices.

3 A DECENTRALIZED FRAMEWORK TO PROTECT SWARM AGENTS

Resource sharing in open environments is a key component of the Swarm Computing approach, since it enables cooperation among IoT agents (COSTA et al., 2015; BIASE et al., 2018b). A pressing challenge in this scenario consists in creating a flexible security layer, capable of protecting resource sharing in the Swarm, while at the same time keeping it open for cooperative interactions. In other words, building a secure Swarm implies in building a system that concurrently supports the properties of openness, cooperation, decentralization, and security. This chapter presents our efforts towards the development of such a system.

Note: A significant portion of the solutions presented in this section has been previously published in the following works:

- Fedrecheski et al. (2019): Attribute-Based Access Control for the Swarm With Distributed Policy Management.
- Fedrecheski et al. (2020): Self-sovereign identity for IoT environments: a perspective.
- Fedrecheski et al. (2021): SmartABAC: enabling constrained IoT devices to make complex policy-based access control decisions.
- Fedrecheski et al. (2022): A low-overhead approach for self-sovereign identity in IoT.

3.1 Framework Blocks

Security is primarily concerned with mitigating threats towards computer systems (SHIREY, 2007). Threats are often modeled considering a threat model, such as the STRIDE threat model (HERNAN et al., 2019), which stands as an acronym to the

threats of **s**poofing, **t**ampering, **r**epudiation, **i**nformation disclosure, **d**enial of service, and **e**levation of privilege. Different techniques can be used to mitigate each of these threats, ranging from packet filtering, to access control enforcement and cryptography-based mechanisms. In this work, we investigate and propose methods related to the latter techniques.

Specifically, we propose a layered framework to protect the Swarm, encompassing authorization, authentication, confidentiality, and identification. On each layer, we propose secure mechanisms that satisfy requirements of the Swarm approach, such as openness, decentralization, and support for heterogeneous devices and networks.

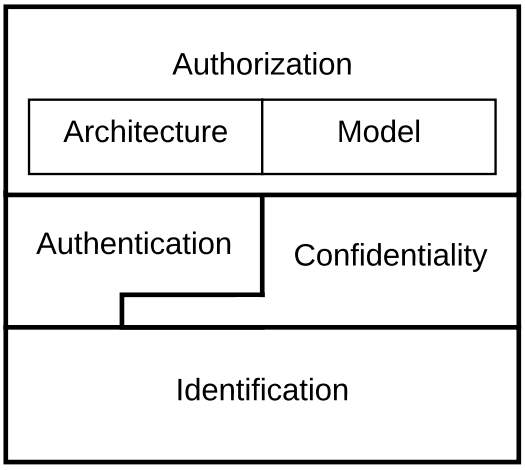


Figure 2: The proposed framework is divided in four layers.

At the top, lies the Authorization layer, which deals with high-level permissions for resource sharing. This layer has two main components. The first one specifies the architectural elements, and the second defines the model, i.e., the relations (i.e. policies and requests) and algorithms used to compute access control decisions. We propose an attribute-based access control (ABAC) approach, since it enable creation of expressive and scalable authorization policies. This layer aims to mitigate the threat of elevation of privilege, in which an agent performs an action it is not supposed to perform.

The next layer comprises authentication and confidentiality. The Authentication layer consists of a model for credentials as well as a set of cryptographic algorithms to create and validate them. This is required to ensure that access requests do not use spoofed credentials, which in turn could be used to bypass authorization policies. Another way to break authorization would be through an eavesdropping attack, in which an unauthorized party gains access to sensitive data. To mitigate this, we employ cryptographic

algorithms to encrypt data, which happens at the Confidentiality layer. As shown in Figure 2, authentication does not necessarily depends on confidentiality, however, encrypting credentials is useful since it improves privacy.

Finally, there is the Identification layer, which defines how to manage identities in the context of the Swarm. Specifically, we propose a self-sovereign identification mechanism that enables agents to automatically generate and manage their own identifiers. This is a fundamental layer, on top of which all other entities in our framework are built upon, therefore it is placed at the bottom of the stack.

The next sections describe each layer in detail, and the latest section in this chapter presents how these layers work together as a whole to secure Swarm systems.

3.1.1 Authorization

Authorization consists in "the problem of determining the operations (e.g., read and write) that subjects (e.g., users and services) can perform on objects (e.g., files and network connections)" (JAEGER; ZHANG; EDWARDS, 2003). In the context of the Swarm, this means that only authorized entities shall have permission to interact with a given agent. The authorization layer for the Swarm must satisfy the Swarm requirements, such as device heterogeneity and openness for resource sharing.

In this work we provide authorization by combining Attribute-Based Access Control (ABAC) and Capability-based Access Control (CapBAC). The main characteristic of ABAC is the use of attributes assigned to subjects and objects, as well as attributes that can be extracted from the environment. The structure that defines the access control rules in ABAC systems is the *policy*, which consists of a collection of permissions built in terms of attributes. Since ABAC uses attributes rather than identifiers, it scales and enables authorization flexibility, at the cost of increased complexity. To authorize interactions between Swarm agents using ABAC, an architecture is needed to specify the building blocks and the authorization flow (HU et al., 2013).

In the Capability-Based Access Control approach, computing entities carry a token which encode the permissions possessed by the entity itself (DENNIS; HORN, 1966). For example, a camera might carry a token that specifies its right to toggle an IoT lamp. This token is then presented to the target of the interaction, which verifies the validity of the token before accepting it.

The CapBAC approach enables decentralized authorization (HERNÁNDEZ-RAMOS

et al., 2016), however it depends on a way to decide how to handle to authorization tokens. Since the latter can be provided by ABAC, in this work we combine the ABAC and CapBAC approaches, thus providing a decentralized and expressive authorization mechanism for IoT devices. Our contributions to achieve decentralized ABAC are presented in the next two subsections, and the integration with CapBAC tokens is described along with the operation flow (Section 3.2).

3.1.1.1 ABAC Architecture

In this sub-section we present the ABAC architecture. First, we show the architecture as defined by the National Institute of Standards and Technology (NIST) (HU et al., 2013). Next, we present how we adapted it to the Swarm context.

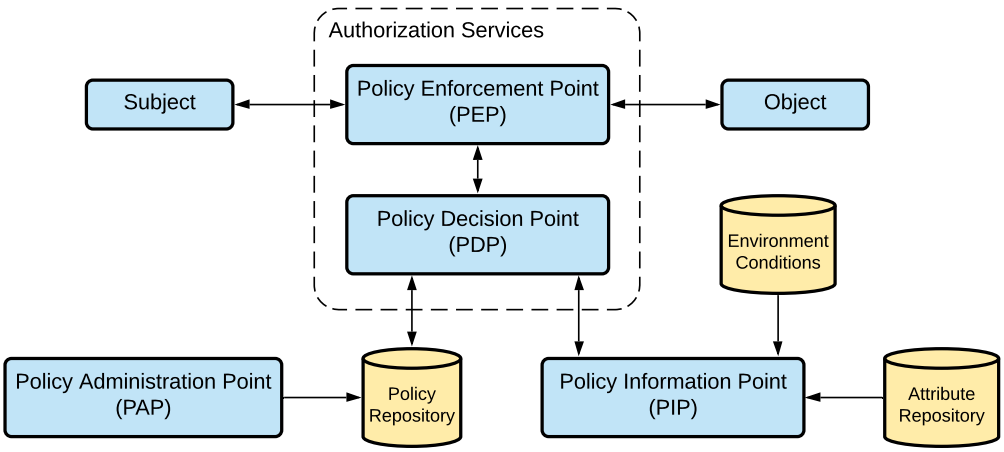


Figure 3: ABAC architecture proposed by NIST.

The ABAC architecture, shown in Figure 3, involves a subject, an object, data sources and separate functional blocks referred to as *points* (HU et al., 2013). The subject is the entity trying to perform a given operation, and the object is either a resource to be accessed or an entity that will handle the operation. The functional points determine whether the operation execution should be authorized. The Policy Enforcement Point (PEP) is responsible for intercepting a message from a subject to an object, and for forwarding an access request to the Policy Decision Point (PDP). The PDP, then, reads the policies from the repository, fetches attributes and environmental conditions from the Policy Information Point (PIP), and uses that information to make a decision, i.e., compute whether the access request shall be *allowed* or *denied*. This decision will be returned to the PEP, which will forward the message to the object, in case the request is "allowed", or otherwise return an "unauthorized" message to the subject. Finally,

the Policy Administration Point (PAP) is responsible for presenting a user interface that allows an administrator to create, view, and edit policies.

The incorporation of the ABAC architecture in the Swarm must consider the characteristics of the latter, particularly decentralization and scalability. To achieve that, we propose a re-organization of the elements from the original ABAC architecture in order to fit it within the Swarm architecture, as shown in Figure 4.

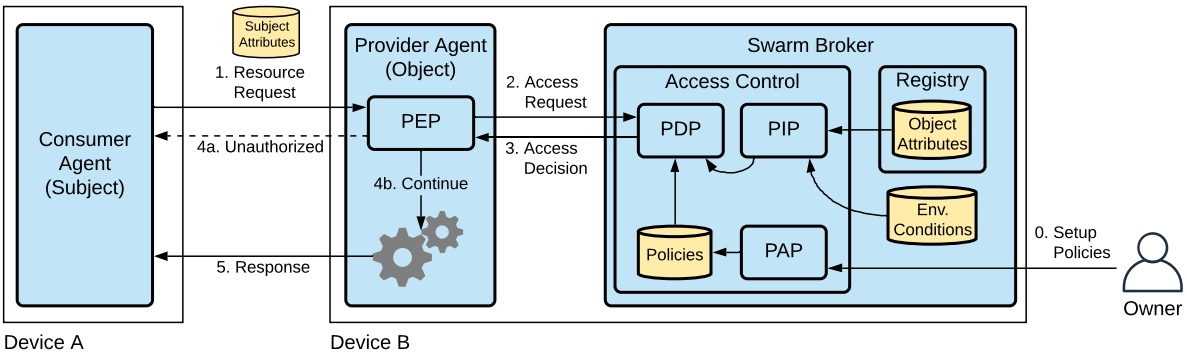


Figure 4: ABAC Architecture adapted and incorporated into the Swarm.

The subject and object entities are represented by two Swarm Agents, the initiator and the responder. Thus, the responder agent implements the PEP to intercept requests and forward them to the PDP. The PDP runs in the broker, and it is responsible for evaluating the access policies and returning an access response to the PEP. If the access is denied, the responder immediately returns an unauthorized status to the initiator; otherwise, it allows the resource access to continue. Another modification to the original ABAC architecture is the location of the attributes. Since the responder always registers itself in its associated broker, the broker has all of the object attributes, therefore they can be loaded normally by the PIP. The subject attributes, on the other hand, cannot be loaded locally since they come from an external agent, therefore they must be presented by the responder as part of the resource request message. Finally, an administrator could update access policies for Swarm devices by accessing a user interface made available by the PAP.

The proposed architecture offloads much of the authorization complexity to the Swarm Broker. This is aligned with the Swarm proposal, in which the Broker is expected to provide platform services that can be reused by other Swarm Agents (COSTA et al., 2015), e.g., Device B could have more agents, all of which would be able to use the same PDP. Consequently, this simplifies the receiver code, which only needs to implement a PEP in order to leverage the Swarm authorization layer. The PEP is simpler than the

PDP since the former just intercepts requests, whereas the latter must implement a policy evaluation engine.

3.1.1.2 ABAC Model

An access control model describes the entities, relations, and algorithms needed to express access requests and policies, and compute access decisions (HU et al., 2013). In this sub-section we propose SmartABAC: a fast, concise, and expressive ABAC model that can be executed in constrained IoT devices, and can therefore be used to protect resource sharing in the Swarm. The design of the SmartABAC model was subject to a set of requirements, summarized below:

- Expressiveness: allows creating policies that can represent a wide variety of scenarios. Since Swarm systems include Smart Homes, Smart Cities, Industrial IoT, and many more, the underlying structure of the access policies must be flexible and allow the expression of reasonably complex rules. Furthermore, it should comprise contextual attributes, as it is crucial for cyber-physical systems to use environment characteristics, such as geolocation and timestamp, to make an access decision.

- Performance: runs inside constrained devices. The ideal ABAC system for the IoT must use efficient algorithms to evaluate and manage policies to be embedded into devices with limited processing and power resources.

- Conciseness: low policy footprint. While policies are required to be expressive, the number of bytes necessary for representing policies must be as low as possible due to low storage and bandwidth resources in IoT devices.

- Scalability: supports large numbers of devices. A model for access control in IoT systems must pose a minimal overhead as the network grows, potentially, to a global scale of trillions of devices. A generic approach that uses attributes as parameters for expressing access rules is necessary so that a growing number of devices does not map directly to a similar growth in the number of policies.

In the following paragraphs we present the constructs and trade-offs considered in our model proposal.

ABAC policies can be represented in two distinct ways, Logic-based Access Policies (LAP) or Enumeration-based Access Policies (EAP) (BISWAS; SANDHU; KRISHNAN, 2016a). In LAP, the format most commonly adopted, policies are expressed as

logic statements that evaluate to true or false. In EAP, policies are defined by enumerating the accepted values. For example, a policy that "allows family members to access security appliances" would be represented as `subject.role == FamilyMember and object.type == SecurityAppliance` in LAP, and as {`(role, FamilyMember), (type, SecurityAppliance)`} in EAP. When compared to LAP, EAP presents the following benefits: forms micro-policies that are easy to share and to update, has a canonical policy structure that facilitates automated manipulation, and can be parsed in polynomial time (as opposed to LAP, in which some operations are NP-complete) (BISWAS; SANDHU; KRISHNAN, 2016a; BISWAS; SANDHU; KRISHNAN, 2016b). The drawbacks of EAP are well documented and include: policy creation may be more laborious, policies can grow larger in size, and it may become complex to implement conjunctive policies, i.e., those having the effect of a logical `and` operator (BISWAS; SANDHU; KRISHNAN, 2016a).

A useful concept to model attributes is data types, which prevents ambiguity and enhances policy expressiveness. Although some LAP models use data types, no EAP model has considered the concept so far. Among the benefits of a typed EAP model is the expression of numerical ranges, which mitigate the problem of huge enumerated policies. For example, to express the rule "age must be greater than 18", instead of writing {`(age, 18), (age, 19), ..., (age, 100)`}, one could write {`(range, age, (min, 18))`}. Note that the use of the attribute type *range* allows for an expressiveness gain in the policy, avoiding the drawback of EAP, which is the enumeration of all possible values. Another benefit of typed attributes is that attributes can be nested by using an *associative array* type, which allows compatibility with widely used standards, such as JavaScript Object Notation (JSON)[1] and Concise Binary Object Representation (CBOR)[2]. For example, a policy can use the following set of nested attributes to specify a device manufacturer: `(manufacturer: (id: m1, name: pear))`.

Thus, we introduce this new concept that can be viewed as a hybrid of logic and enumerated policies, whereby the benefits of EAPs, such as canonical form and micro-policy, are maintained, while the flexibility of type-based conditionals increases policy expressiveness, improves readability, and reduces policy size, contributing to the system scalability. Moreover, another contribution is the introduction of nested attributes in an ABAC model, which makes it directly compatible with existing standards for signed attributes, such as the Verifiable Credentials data model (SPORNY et al., 2019b).

A common drawback of adopting enumerated policies is that they require complex

[1] https://tools.ietf.org/html/rfc7159
[2] https://tools.ietf.org/html/rfc7049

mechanisms to represent conjunctive policies. Although previous works showed that single and multiple-attribute EAPs are equally expressive in their theoretical power (BISWAS; SANDHU; KRISHNAN, 2016a), the expression of conjunctive policies in a single attribute model leads to an attribute explosion, making it less useful in practice. We identified that a possible solution to alleviate this issue is to use multiple attributes in the policies. In a single-attribute model, a single attribute is used in each field of a policy rule, e.g., in `(hvac, read, temperature)`, each attribute is atomic. In a multiple-attribute model, many attributes can be listed in each field, e.g., in `({hvac, kitchen}, {read}, {temperature})`, each attribute is a set. In other words, EAP models require multiple attributes to be able to efficiently express conjunctions. For example, the single-attribute policy above cannot express the rule "devices *of type hvac that are in the kitchen* can read the temperature", while the multi-attribute policy can. Since our model uses EAP, we adopt multiple attributes to support conjunctions.

As the number of attributes grows in large systems, attribute hierarchies can be used to simplify policy management. For example, instead of specifying duplicate policies for `father`, `mother`, and `uncle`, hierarchical relationships among the attributes can be created to enable the creation of a single policy that addresses `adulFamilyMembers`. Different ways to express hierarchies have been proposed, from simple solutions, such as attribute containment (FERRAIOLO; ATLURI; GAVRILA, 2011), to more complex constructs that use hierarchical groups to assign attributes to users (SERVOS; OSBORN, 2014). Combining these techniques has also been explored, showing that attribute containment can be used to represent hierarchical groups (BHATT; PATWA; SANDHU, 2017). Thus, policy scalability can be enhanced by delegating part of the complexity of the domain to attribute hierarchies, which reduces the number of policies to be managed in each IoT device.

This way, we propose SmartABAC, an Attribute-Based Access Control model with the following features: typed, nested, and hierarchical attributes, as well as enumerated and multi-attribute policies.

Our model is specified in first-order logic as shown in Table 5. It is divided in five segments with definitions for attributes, policies, requests, hierarchical relationships, and an authorization function.

Segment I presents general definitions of SmartABAC. An access control model dictates how the rules for a subject to access a given object are defined. In SmartABAC, access is defined by the operations (Op) that can be executed by a subject on an object.

Table 5: The SmartABAC model expressed in first-order logic.

I. General definitions

- SA, OA, and CA – set of subject, object, and context attributes, respectively.
- $A = SA \cup OA \cup CA$ – the set of all possible attributes.
- Op – the set of all operations.
- $type : A \rightarrow \{string, number, range, assoc\_array\}$ – denotes the type of an attribute.
- $name : A \rightarrow N$ – denotes the name of an attribute, where N is a non-bounded set of names.
- $value : A \rightarrow V$ – denotes the value of an attribute, where V is a non-bounded set of values.

II. Policy definition

- $P = 2^{Op} \times 2^{SA} \times 2^{OA} \times 2^{CA}$ – the set of all possible policies.

III. Request definition

- $A_r = \{a_{req} \in A \mid type(a_{req}) \in \{string, number, assoc\_array\}\}$ – set of attributes of requests.
- $SA_r = SA \cap A_r$ – the set of subject attributes allowed in a request.
- $OA_r = OA \cap A_r$ – *idem*, for object attributes.
- $CA_r = CA \cap A_r$ – *idem*, for context attributes.
- $R = 2^{Op} \times 2^{SA_r} \times 2^{OA_r} \times 2^{CA_r}$ – the set of all possible requests.

IV. Hierarchy definition

- $A_{str} = \{a \in A \mid type(a) = string\}$ – set of attributes with type string, that have hierarchical relationships.
- $expand : 2^{A_{str}} \rightarrow 2^{A_{str}}$, defined as

$expand(A) : \{a\} \cup \{a_{succ} \in A_{str} \mid a \prec a_{succ}\} \forall a \in A_{str}$ – where $\prec$ denotes a partial order.

V. Authorization function

- $matchAttrs : 2^{A_r} \times 2^{A} \rightarrow \{true, false\}$ – checks whether all request attributes match the policy attributes, according to their types.

$$matchAttrs(A_r, A) : \begin{cases} ar = ap, & \text{for } type(ar) = type(ap) \wedge type(ap) \in \{number, string\}, \forall ar \in A_r, ap \in A \\ ar \in ap, & \text{for } type(ar) = number \wedge type(ap) = range, \forall ar \in A_r, ap \in A \\ name(ar) = name(ap) \wedge & \\ matchAttrs(value(ar), value(ap)), & \text{for } type(ar) = object \wedge type(ap) = object, \forall ar \in A_r, ap \in A \end{cases}$$

- $isAuthorized : R \times 2^{P} \rightarrow \{true, false\}$, defined as

$isAuthorized(r, P) : (p \in P)[r_1 \subseteq p_1 \wedge matchAttrs(expand(r_2), p_2) \wedge matchAttrs(expand(r_3), p_3) \wedge matchAttrs(expand(r_4), p_4)]$,
where p_i and r_i correspond to the i-th element of the respective tuple in P and R,
i.e., (operations, subject, object, and context attributes)

Source: Fedrecheski et al. (2021).

Subjects and objects are defined by their attributes (respectively, SA and OA). Since in the IoT the surrounding context may be relevant during the access decision, SmartABAC also considers context attributes (CA).

Each attribute has an associated name, value, and data type. The valid types for attributes are string, number, range, and assoc_array. The use of data types is a novel element in SmartABAC, as it enhances expressiveness and reduces policy size. Strings are interpreted as character lists, and numbers can be either integers or real. Ranges are applied over numbers and represent an inclusive interval between a minimum and a maximum value. Finally, the assoc_array (associative array) type enables the creation of key-value pairs, effectively allowing attribute nesting. The use of this data type in ABAC models is a novel contribution of our work, and it is useful for compatibility with popular serialization formats and standards for signed attributes (SPORNY et al., 2019b).

Segment II, Policy Definition, specifies the domain of access policies in SmartABAC.

The policies determine which subjects (SA) can perform which operations (Op) over which objects (OA) under which context (CA). Thus, a policy is a set comprising the enumeration of operations and attributes.

Segment III, Request Definition, defines the domain of possible requests. In SmartABAC, an access request is evaluated against a set of policies. The structure of a request is similar to that of the policy, i.e., an enumeration of attributes and operations. The main difference from the policy definition is an attribute type restriction, as requests in SmartABAC only support the following types: `number`, `string`, and `assoc_array`, i.e. they do not support the `range` type. The reason is that, while it is common for policies to use ranges, e.g., "age between 18 and 60", requests tend to use concrete values, such as "age equal to 25".

Segment IV, Hierarchy Definition, presents the method to expand the attributes of a given request, mapping associated attributes from an attribute hierarchy defined as a partial order relation. In other words, a more generalized attribute may be mapped to a set of more specific attributes. The partial order is defined over attributes of type `string` in the `expand` function, which allows the retrieval of all attributes that succeed a given attribute, i.e., $expand(a_{prev})$ will return a set containing all the attributes higher in the hierarchy, along with a_{prev} itself. For example, a partial order can be defined between attributes `camera` and `appliance`. When applying the function `expand` over the attribute `camera`, the result will be `(camera, appliance)`. This way, policies can be specified using attributes higher in the hierarchy, e.g., `appliance`, which reduces the number of policies and simplifies policy management.

Finally, Segment V, Authorization Function, describes functions for evaluating a request and deciding whether or not it should be allowed. The decision is computed by checking whether all the request operations and attributes satisfy at least one of the available access policies. Regarding operations, a request satisfies a policy if its operations are a subset or equal to those of the policies. With respect to attributes, a request satisfies a policy if the result of calling the function `matchAttrs` is true for all attributes in the policy. The `matchAttrs` function works by applying the correct operator based on attribute name and data type. Specifically, number and string attributes are compared using an equality operator, while a membership operator is used to verify ranges. When an attribute is of type associative array, the `matchAttrs` function is called recursively[3]. Thus, the `isAuthorized` function takes a request and a list of policies and uses `matchAttrs` to

[3]Attributes in real-world applications tend to have only a few levels of nesting. That said, if necessary, a threshold value can be set in the implementation to limit the depth of recursive calls.

check whether at least one policy satisfies that request. Note that `isAuthorized` also uses `expand` to obtain attributes considered "higher" in a hierarchy of attributes. The result of `isAuthorized` is true if at least one of the policies satisfies the request or false otherwise.

By combining elements of enumerated access policies, data types, and hierarchies, SmartABAC enables constrained IoT devices to run complex access policies.

3.1.2 Authentication

The next layer in our proposed framework, following from Authorization, is Authentication. According to the Internet Security Glossary, authentication consists in having an entity proving the validity of certain attributes about itself (SHIREY, 2007). In the Swarm, these entities are Swarm agents, and their attributes can range from specific identity-based claims, to generalized fields linked to an agent. For example, an agent that implements the functionality of a security camera might have an unique identifier, along with a few more generic attributes, e.g., `type: securityCamera, location: outdoor`. In this section, we focus on authentication of attributes, leaving unique identity authentication to the other layers.

As shown in Figure 4, an initiator agent presents its attributes to the responder during an interaction. The challenge that authentication must solve, then, consists in enabling the responder to trust these attributes. For example, an attacker could create a Swarm agent that presents spoofed attributes that would in turn make it pass through the authorization mechanism.

A widely used mechanism to enable attribute authentication is the digital signature, which enables a recipient to validate data origin and integrity (SHIREY, 2007). Among the different digital signing algorithms available, in this work we selected the Edwards-curve Digital Signature Algorithm (EdDSA), along with the Ed25519 curve (BERNSTEIN et al., 2011; JOSEFSSON; LIUSVAARA, 2017). This selection is justified (BERNSTEIN et al., 2011) due to the following properties:

- It is considered infeasible to break, since it requires around 10^{128} operations to forge a signature.
- It requires small public and private keys, i.e. 32 bytes for each key part, as opposed to larger keys from different curves or those that use different cryptographic schemes, such as the RSA (RIVEST; SHAMIR; ADLEMAN, 1978).

- The generated signature is small, requiring only 64 bytes. Signature and key sizes are important since bandwidth may be scarce in IoT networks.
- It generates deterministic signatures, which are not subject to repeated nonce attacks (JOSEFSSON; LIUSVAARA, 2017).

In addition to a signature, authentication of attributes generally require a data structure to hold a few elements that enable a verifier to check authenticity. The elements that need to be present in this structure include:

- Subject identifier: a unique identifier of the entity described by the signed attributes.
- Issuer identifier: a unique identifier of the entity that signed the attributes.
- Expiration: a point in time after which the signature ceases to be valid, effectively revocating the attributes.
- Issuance date: a point in time when the attributes were created.
- Attributes: the actual set of application-specific attributes, usually represented by a key-value mapping.

These elements are present in standardized approaches for secure serialization and presentation of attributes, as discussed in Section 2.2.2.3. Thus, we adopted the Verifiable Credentials data model for attribute representation in the Swarm, with two modifications. First, we serialize the credentials using Concise Binary Object Representation (CBOR) (BORMANN; HOFFMAN, 2020) instead of using JSON-LD, in order to optimize credential size. CBOR was developed to be a JSON alternative for IoT applications, since it provides similar semantics with smaller serialization footprint. Second, we only adopt signing algorithms based on elliptic-curves and do not adopt algorithms based on zero-knowledge proofs, since they are too expensive to run on IoT devices. This approach enables the use of authenticated attributes while having a smaller footprint and leveraging the advantages of the VC data model, such as flexible key rotation and decentralized verification.

Our VC-based credential is wrapped as a CBOR Object Signing and Encryption (COSE) envelope (SCHAAD, 2017). The COSE specification defines how to create signed and encrypted objects that can be serialized in CBOR. Specifically, a signed COSE object is defined as an array containing (1) an integrity-protected header, (2) an unprotected

header, (3) the signed content (payload), and (4) the signature. The goal of the COSE headers is to provide information regarding the type of algorithm used to perform the signature.

An example VC that adopts the COSE envelope format is shown in Listing 3.1. Since the actual COSE object is a binary blob, we use the CBOR diagnostic notation (BORMANN; HOFFMAN, 2020), which is loosely based on JSON and enable blobs to be displayed in a human readable way. Thus, Listing 3.1 must be interpreted as a regular JSON document, with the exception of the grey items between slashes, which are comments about the meaning of each key or value. For example, line 3 could be read as "the signature algorithm, specified by code `1`, is EdDSA, specified by code `-8`".

```
1  [
2      / protected header / {
3          / algorithm / 1: -8 / EdDSA /
4      },
5      / unprotected header / {},
6      / payload / {
7          / issuer / 1: "did:sw:ShbRLB6VsoNAizbwYw9Qz5",
8          / subject / 2: "did:sw:PwnqEmgnJ4V8qfVEnd7w16",
9          / issued at / 6: 1640362938,
10         / expiration / 4: 1671898938,
11         / credential_subject / 21: {
12             "owner": "did:sw:PwnqEmgnJ4V8qfVEnd7w16",
13             "type": "camera",
14             "household": {"id": "home-1"}
15         }
16     },
17     / signature / "bc12491f ... truncated ... a684ad03"
18 ]
```

Listing 3.1: Example of Verifiable Credential enveloped using CBOR and COSE.

The protected header of the VC specifies the signing algorithm, while the unprotected header is empty. The signed payload include the issuer, subject, relevant timestamps, and a set of attributes. The issuer and subject of a VC are represented by their respective DIDs, and the timestamps follow the Unix format. Finally, the attributes, presented in the `credential_subject` field inside the payload, represent application-specific information

about the subject. In the example of Listing 3.1, these include the type of a Swarm agent, its owner, and an identification of the household it is installed in. A Swarm agent verifies a credential by checking if the signature in the VC matches the public key of the issuer.

The lifecycle of a credential begins when an issuer signs the VC and delivers it to the holder agent, which can then use it to authenticate in Swarm environments. The end of life of a credential occurs when its expiration time is passed. It is expected that additional mechanisms for credential revocations may be implemented, such as revocation lists, however they are out of scope of this document.

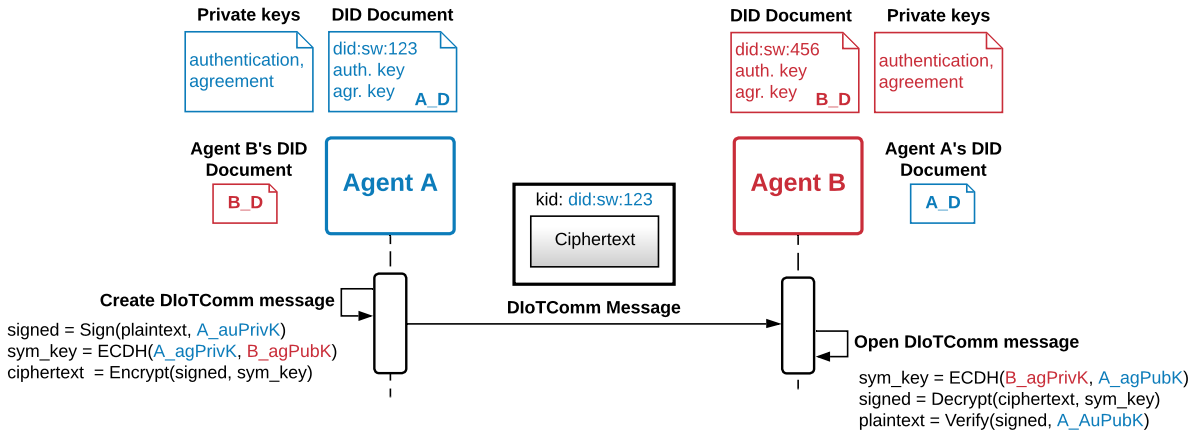
3.1.3 Confidentiality

The next layer for securing the Swarm is confidentiality, and it serves to counter the threat of information disclosure, in which an attacker gets access to data exchanged between two entities.

Confidentiality is most commonly achieved by encrypting data using an established symmetric encryption algorithm, such as the Advanced Encryption Standard (AES) (DAEMEN; RIJMEN, 2001). It is also common to use asymmetric cryptography to facilitate management of cryptographic keys, such as the Diffie-Hellman (DH) key exchange and its elliptic-curve version, ECDH. These algorithms, in turn, are wrapped in a protocol which usually provides mechanisms for key distribution, binding to network protocols, and protection against common attacks, such as message replay.

Transport Layer Security (TLS) is a widely used protocol in internet applications that provides both encryption and authentication (RESCORLA, 2018). The encryption part is carried over the payload of TCP segments, thus guaranteeing confidentiality between two TCP/IP network nodes. An alternative to TLS is also defined for UDP communications, named Datagram TLS (DTLS). Although TLS and DTLS are mature and widely used, one drawback of both is the lack of support for end-to-end security over multiple network hops. Additionally, both protocols were designed to authenticate entities using public key certificates. As we discuss in more detail in the next section, this works adopts DIDs for agent identification, and therefore is not directly compatible with TLS nor DTLS.

To enable confidential and authenticated communications based on DIDs, the DIDComm protocol has been proposed (HARDMAN, 2020). It allows agents to use the keys referenced in their DID Documents to encrypt and sign messages, so that they can be securely transmitted and verified by the responder. To do so, DIDComm adopts the



Source: Adapted from Fedrecheski et al. (2022).

Figure 5: An overview of the DIoTComm protocol.

structure and algorithms defined in the JSON Object Signing and Encryption (JOSE) standard (JONES, 2015), which allows use of existing schemes for message encryption and authentication. Furthermore, it is agnostic of both DID method and transport protocol. DIDComm also defines a set of message headers to identify a message sender and responder, as well as a message type, a unique message identifier, and other optional metadata. A significant downside of DIDComm, however, is the use of JSON which causes unnecessary overhead in constrained networks, common in IoT scenarios.

Thus we propose **DID-based IoT Communication (DIoTComm)**, an alternative to DIDComm that is tailored for IoT networks, i.e., it uses a more concise serialization method and simpler message headers. While in DIDComm a JSON-based format is used for message protection, DIoTComm uses COSE, which defines both a message format and a set of lightweight security algorithms, leading to small-footprint protected messages. Another difference is that, while DIDComm supports a few types of message headers, DIoTComm has only a single header, the *key id*, which contains the DID of the initiator, as shown in Figure 5. We are able to omit the responder id since the decryption by a responder other than the intended one will fail. We also consider that other fields such as message type and message id, if needed, can be handled at the payload layer, e.g., if the payload is a CoAP message, its header would include a method, a path, and a unique message identifier.

DIoTComm leverages the structure defined by COSE messages which consists of: an integrity-protected header, an unprotected header, a payload, and optional extra fields. We define that the sender id in DIoTComm must be the binary-encoded DID of the message sender. To save space, the sender may choose to omit the fixed part of the DID

prefix, i.e. `did:`, and encode only the method and the random part of the DID. This id must be carried within the "key id" field of the COSE protected header. If the message must be encrypted, the payload will contain the cipher-text. If it must be signed, the payload will contain the plain-text, and the message will have the signature as a fourth parameter. In cases where a message must be protected both for non-repudiability and confidentiality, the plain-text is first signed then encrypted. Creation of such protected envelopes is described in the following paragraphs.

For message signature, the sending agent signs the message with its private authentication key. The receiving agent can verify the signature using the public key available in the sender's DDo. The COSE algorithm used is EDDSA, i.e., the Edwards Curve Digital Signature Algorithm applied over curve Ed25519 keys (SCHAAD, 2017). This enables short and fast signatures that do not depend on strong random number generators.

Regarding encryption, the sending agent will derive an encryption key using its private agreement key, along with the public agreement key available in the receiving agent's DDo. A similar process is then executed by the receiving agent to decrypt the message, wherein the receiving agent uses its private agreement key and the sending agent's public agreement key to obtain the decryption key. The COSE algorithm used for key derivation is the ECDH-SS-HKDF-256, which uses an elliptic curve Diffie-Hellman with two static keys (the private key of one party and the public key of another), along with a key derivation function based on SHA-256 (SCHAAD, 2017). The COSE algorithm used for content encryption is AES-CCM-16-64-128, that is the Advanced Encryption Standard in CCM mode with a 64-bit tag and a 13-bytes nonce.

3.1.4 Identification

The final layer for security in the Swarm consists in how to uniquely identify agents in a decentralized and scalable way. In this work we use the Self-Sovereign Identity approach to identify Swarm agents. In SSI, each entity has full control of its own identity. Informally, this approach can be compared with a physical wallet, in which a subject carries all of his documents with himself. Formally, the self-sovereign identity of an agent is the union of all of its identifiers and attributes across different domains, which are managed in a decentralized way (FERDOUS; CHOWDHURY; ALASSAFI, 2019).

The Decentralized Identifiers (DID) specification (SPORNY et al., 2019a) defines a standardized format for self-sovereign identifiers and associated metadata. A DID is

composed of a DID method[4] prefix and a namespace-specific identifier (NSI) (SPORNY et al., 2019a). The prefix always start with the string `did:` and is followed by a method name and a colon, e.g., the prefix for the Tangle DID method is `did:tangle:` (TANGLEID, 2019). The NSI is a globally unique identifier, usually randomly generated, whose size and other parameters are specified by the DID method. A truncated example of a DID is: `did:tangle:WILTZRG...Q99NA9999`. The primary use for DIDs is to uniquely identify an entity in a decentralized way.

Each DID is usually associated to a DID Document (DDo), which contains related metadata such as public keys and service endpoints (SPORNY et al., 2019a). DDos are usually serialized in JSON, and although a CBOR serialization is specified (SPORNY et al., 2019a), none of the currently registered DID methods uses it. A typical JSON-based DDo occupies 500 bytes or more.

We propose agent identification as a fully self-sovereign procedure. Each agent generates its own identifier, in the format of a DID, as well as its own identity metadata, in the format of a DDo, which contains service endpoints and public keys. This approach allows devices to fully own and control their identity without depending on third parties (FEDRECHESKI et al., 2020). To enable discoverability, though, agents may choose to anchor their DDos on an Identity Blockchain, which acts as a decentralized source of truth for identity metadata. This allows agents to dynamically resolve the DDo associated with a specific agent, given that its DID is known. For example, if Agent 1 knows the DID of Agent 2, the DDo of Agent 2 can be obtained by querying the blockchain.

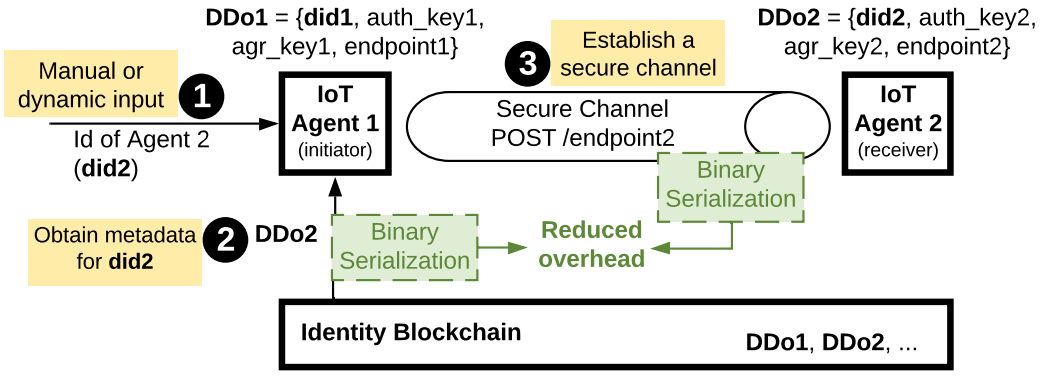
Once agents are identified, i.e. with locally generated DIDs and a way to share DDos, they are ready to securely communicate, as described in Section 3.1.3. While that section focused on the encryption and signatures, this section presents how agents can obtain each other's DID Document.

We consider interactions to involve an initiator agent and a responder agent, and optionally the Identity Blockchain. If initiator and responder are pre-provisioned with each other's DDo, communication can begin immediately, without the need to contact a third party, as shown Figure 5. If, on the other hand, an agent is only given the DID of another agent (for example, when verifying the signature in a Verifiable Credential), it needs to contact the blockchain to retrieve its respective DDo[5], as shown in Figure 6. Then, once the initiator has the DDo of the responder, it can extract the endpoint to find out where to send the messages and use the public key to protect the messages, i.e.,

[4] A "DID method" is an extension of the DID specification;

[5] Note that the DDo can be cached after the first use.

derive a session key for encryption.



Source: Adapted from Fedrecheski et al. (2022).

Figure 6: DID resolution and usage along with DIoTComm.

These procedures, however, may be of limited use in constrained IoT networks due to the overhead of DID resolution. The identified related work adopting Self-Sovereign Identity in IoT either do not consider the limitations of constrained networks, or address it by creating a centralized adaptation layer.

In this section we propose a set of extensions and optimizations to reduce the overhead of transmitting DDos and protecting interactions between self-sovereign IoT agents. First, we propose a lean DID method that specifies the minimum needed metadata for DDos in the IoT. Then, we propose an alternative serialization mechanism for DDos, named CBOR-based DID Documents for IoT (CBOR-DI), that can reduce DDo size up to four times.

3.1.4.1 The Swarm DID Method

A *DID method* consists of a set of definitions about the format of DIDs and DDos, as well as on how to perform management operations (SPORNY et al., 2019a). In this section, we propose the Swarm DID Method (`did:sw:`), which will enable self-sovereign identification of IoT agents. Although it was motivated by the Swarm architecture, it is sufficiently generic to be used in general IoT architectures.

Requirements Previously, we established that the self-sovereign approach can satisfy the requirements of privacy and decentralization for IoT devices (FEDRECHESKI et al., 2020). We now specify the remaining requirements that need to be tackled in order to enable self-sovereign identification of devices in heterogeneous networks. First, both the DID and the DDo must be short since they may be carried over constrained networks.

Second, the DID should carry enough randomness to be able to identify a very large number of devices, e.g., if considering one thousand devices per person, the target address space would be in the order of trillions. And third, the DDo should support at least one service URL, needed to allow remote service invocations. The next sections specify the DID and DDo according to these requirements.

Decentralized Identifier (DID) In the Swarm DID method, each device is responsible for autonomously generating its own DID. A DID is composed of a prefix and a namespace-specific identifier (NSI). For the part of the prefix that identifies the DID method in use, we chose the two-letter string `sw`. Thus the full prefix is `did:sw:`. Next, we define the NSI as a short byte array of size 16, that must be generated using a strong random number generator. The address space is 2^{128}, what leaves more than 2^{88} unique identifiers for each device, considering 2^{40} devices (around 1 trillion) and a uniform distribution. A full example of a Swarm DID with a Base58-encoded NSI is `did:sw:TTbs19FJKYf6jXzS1dbnqe`.

DID Document (DDo) Additional metadata about a DID can be stored in a DID Document. In a generic way, the DDo usually carries the DID itself, one ore more public keys, and zero or more endpoints (SPORNY et al., 2019a). Existing DID methods define that the DDo will contain the DID itself and at least one authentication key (SOVRIN...,) (OCKAM...,). We adopt this design for the Swarm DID method, since it provides identification and authentication. Next, unlike most DID methods, we propose the use of at least one static agreement key, since it enables the creation of a secure channel without transmission of ephemeral keys, thus saving bandwidth in constrained networks.

One consideration we take in order to shorten the DDo is that both the authentication and the agreement keys must use an optimized cipher suite with respect to public key sizes. While many elliptic curves satisfy this requirement, we adopt the curves X25519 and Ed25519, which have the smallest public keys among those defined in the COSE specification (SCHAAD, 2017), i.e., 32 bytes.

Next, to allow referencing specific keys within a DDo, keys may have an arbitrary identifier that is unique in the scope of the DDo. We define the following automated way to generate a short key id: compute the hash (SHA-2) of the public key, and truncate it to the first eight bytes. The resulting entropy is enough to avoid collisions since a DDo is expected to have only a few public keys (usually only two).

Finally, a DDo in the Swarm must support at least one service endpoint to allow remote service invocations. The main parameter for each endpoint is an URL, which enables remote agents to message the owner of the DDo. Optionally, endpoints can also have a type tag and an id that shall be unique within the DDo. Both the service type and the id are application-dependent, and if used, they should be short. Figure 7 (a) shows an example DID Document according to the Swarm DID Method, serialized in JSON.

3.1.4.2 Optimized DDo Serialization with CBOR-DI

Serialization mechanisms have direct impact on the size of messages transferred across a network, and range from simple raw bytes encoding to complex structured data, such as the eXtensible Markup Language (XML)[6]. While the binary approach has the benefit of conciseness, a structured approach facilitates arbitrary manipulation. Formats such as the JavaScript Object Notation (JSON) have provided a reasonable trade-off, with the benefit of being human-readable. The general specification for DIDs (SPORNY et al., 2019a) uses JSON as its main format, and most existing DID methods rely on JSON as well.

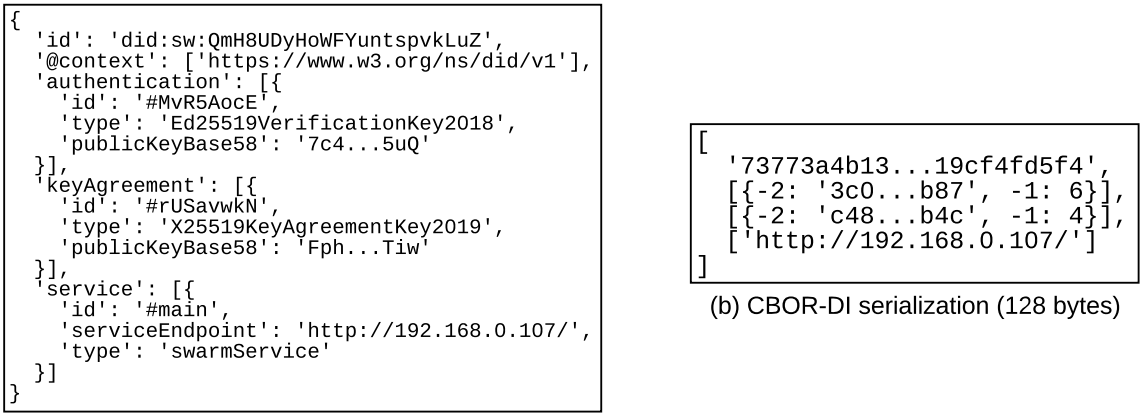
We provide a JSON-based serialization for the Swarm DID method, as shown in Figure 7 (a). It contains an identifier (DID), two public keys, and a service endpoint. The random part of the DID is serialized in Base58, since JSON does not allow encoding of raw bytes. The keys have each an id and a value, both encoded in Base58, and a type indicating its format and usage. Similarly, the service contains an id, a type, and an endpoint URL. After trimming white spaces, the JSON document occupies 497 bytes.

Although human-readable and relatively short, the JSON-based DDo still cannot be transmitted over low-power IoT networks, e.g., LoRaWAN (ALLIANCE, 2017) only supports packets of up to 242 bytes. Fragmentation could be used, at the expense of increased spectrum usage and latency. What is needed is a more concise representation for DDos that allow transmission on constrained networks.

The Concise Binary Object Representation (CBOR) is a JSON-compatible serialization mechanism that uses a binary encoding. Although the DID specification considers direct conversion from JSON to CBOR (SPORNY et al., 2019a), our tests have shown that on average this approach only reduces document size in 20%, i.e., achieving 415 bytes.

Considering this limitation, we present a novel serialization method named CBOR-

[6]https://www.w3.org/TR/2008/REC-xml-20081126/



(a) JSON serialization (497 bytes)

(b) CBOR-DI serialization (128 bytes)

Source: Adapted from Fedrecheski et al. (2022).

Figure 7: Example of Swarm DID Document serialized in JSON and CBOR-DI.

based DID Documents for IoT (CBOR-DI) that reduces size of DDos in up to 75%. We do so by adopting a binary approach and transmitting only the strictly necessary parts of a DID Document, as exemplified in Figure 7 (b). Specifically, we implement the following modifications when comparing to the JSON serialization:

- Use CBOR as serialization mechanism.
- Use an array instead of a key-value mapping so that the elements have a fixed order: DID, verification keys, agreement keys, and service endpoints.
- Remove the `did:` prefix of the DID, and only use the method designator, e.g., use `sw:` instead of `did:sw:`.
- Encode the DID and the key values as raw bytes instead of Base58[7]. Note that the average overhead of Base58 is close to 30%.
- Use the key format defined in the CBOR Object Signing and Encryption (COSE) specification (SCHAAD, 2017), which uses integers instead of strings to reduce size. For example, the mapping "type: Ed25519VerificationKey2019" is translated into "-1: 6". Similarly, note that in Figure 7 (b), the integer -2 points to the public part of the key. The full table with rules for key representation is available in the Key Objects section of COSE (SCHAAD, 2017).
- Finally, ignore optional fields in service endpoints and only include the URL.

[7]https://tools.ietf.org/id/draft-msporny-base58-01.html

As shown in Figure 7 (b), CBOR-DI achieves a DDo size of only 128 bytes, enabling DDo transmission in constrained networks, while losing no essential information. Also, by leveraging existing standards, such as CBOR and COSE, it fosters interoperability. Furthermore, the conversion process between JSON and CBOR-DI can be automated by applying a small set of mapping rules, e.g., convert between JSON and CBOR, Base58 and binary, and JSON keys and COSE keys. Finally, although we proposed CBOR-DI in the context of the Swarm DID method, the technique is generic and could be easily extended to reduce DDo size in other methods as well.

3.2 Operation Flow

This section assembles together the blocks presented in the previous parts of this chapter, and show how they work together to provide secure and decentralized communications in the Swarm. We start by presenting a data model that enables fully self-sovereign IoT agents, as well as a mechanism for establishing and executing a secure interaction using that data model. Figure 9 presents the interaction setup along with an instance of the agent configuration, while Figure 10 presents the interaction execution. More details are provided in the next subsections.

3.2.1 Self-sovereign IoT agents

In the Swarm approach, IoT agents are expected to be independent and to interact autonomously. In this section we present our approach to enable such behavior. It consists in embedding an Autonomous Agent Description (AAD) document that an agent can use to interact, authenticate, and make access decisions. We refer to this configuration as the *agent data model*. This data model is designed to enable IoT agents to be completely self-sovereign with respect to identity, cryptographic keys, attributes, and access policies.

The elements that constitute the Agent Data Model and their relationship are presented in Figure 8. We explain each element in detail as follows:

- Agent: an autonomous Swarm entity capable of providing or consuming resources. The agent stores the full data model locally and may publish parts of it to the appropriate registries. For example, the DID Document may be published to an Identity Blockchain, and public credentials may be sent to Swarm registries to facilitate device discovery.

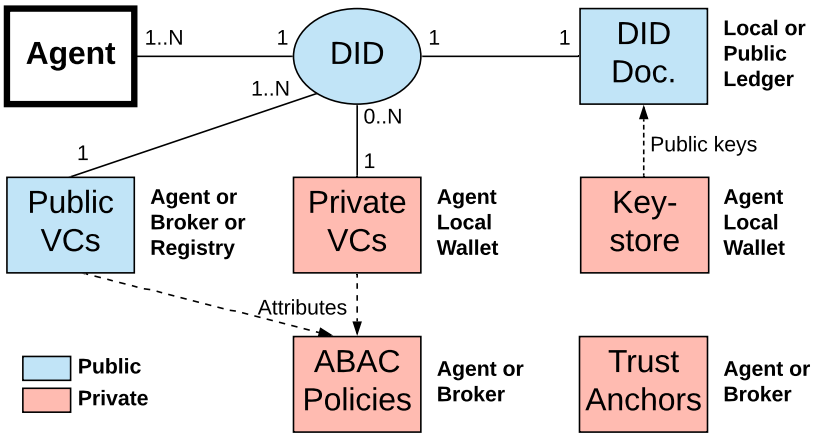


Figure 8: Agent description data model.

- DID: a public and unique Decentralized Identifier that is generated by the agent itself. The agent uses the DID to distinguish itself from other agents in the Swarm. DIDs may also be used to authenticate and authorize interactions, and can serve as addresses in payment transactions during resource sharing (BIASE et al., 2018b). The format of the DID follows the rules defined in the Swarm DID Method, i.e., a random 16-byte sequence preceded by the *did:sw:* prefix. An example DID in the Swarm is *did:sw:Jp1tXfv3XfQE8DRHJeXuDL*.

- DID Document (DDo): A structured document that contains the DID that identifies an agent, public keys, and service endpoints. The endpoints in a DDo are Uniform Resource Locators (URLs) used to interact with an agent. The public keys can be used for establishment of a secure channel via a key agreement algorithm, and also for verification of digital signatures. As defined in the Swarm DID method, the keys are based on Edwards Curves which have short public keys (32 bytes) and whose security does not depend on secure random number generators. In this work we extend the Swarm DID method, and add a broker reference, which is a DID that points to the broker to which its agent is associated. This reference is useful to perform management operations such as the establishment of service level agreements between agents, which is mediated by their respective brokers (BIASE et al., 2018b). DDos can be serialized in a number of formats, including JSON, CBOR, and (as proposed in Section 3.1.4.2) CBOR-based DID Documents for IoT (CBOR-DI), the latter being particularly useful in constrained networks due to its reduced size.

- Cryptographic Keys: a set of asymmetric keys used by the agent, including both public and private parts. These keys are used to digitally sign messages and to

derive symmetric session keys used for encryption. The private part of the keys must never leave the agent storage and should be stored in a secure storage medium, e.g. a trusted platform module. The public parts of these keys can be openly shared, and thus they are present in the DID Document.

- Verifiable Credentials: signed attributes associated to a DID. Its format is based on the Verifiable Credentials (VCs) specification (SPORNY et al., 2019b), which presents a data model for signed attributes. A credential states that a specific set of attributes about a subject has been signed by an issuer at a specific point in time, and it is valid until a certain expiration time. Credentials can be used to establish ownership and authenticate agents during interactions. For example, ownership can be established by having the device owner issue a credential stating that it is owner of the device, referencing the device's DID in the credential's *subject* field, and storing the credential in the device. Authentication is performed by verifying the credential signature, as described in Section 3.1.2. Although originally VCs use a JSON-based serialization, we use a binary serialization based on COSE (SCHAAD, 2017) since the latter is more compact and thus more appropriate for constrained IoT networks. A credential is composed of an issuer, a subject, a set of attributes, a timestamp, an expiration date, and a signature. Keys from an agent's keystore are used to sign new credentials, and the respective public keys from a DDo are used to verify the validity of a credential. We assume the existence of private and public credentials, where the former are stored only locally on the device, while the latter may be stored on brokers and aid in the Swarm discovery mechanism.

- Trust Anchors: a private list of agents that are considered valid credential issuers. That is, when evaluating a credential, an agent should first verify if the credential issuer is legitimate, and it does so by checking if it is in the list of trust anchors. For example, there could be two valid credentials stating the type of an agent, one issued by the agent's owner and another by an unknown agent. Only the credential signed by the owner would be trusted. As a special case, the trust anchor could correspond to the DID of a Reputation Blockchain, as described in a previous work (BIASE et al., 2018b). In this case, the trust anchor would serve as an indirect trust layer to enable a more flexible authentication scheme. More specifically, a reputation threshold can be defined, where only credentials from agents whose reputation satisfy the threshold would be considered trusted.

- ABAC Policies: a private list of attribute-based policies used to authorize agent interactions, specified according to the SmartABAC model (FEDRECHESKI et al.,

2021). These policies are locally enforced by agents that share resources, and dictate which subjects are allowed to execute which operations on which objects, under specific context restrictions. Subjects, operations, objects, and contexts are defined in terms of attributes, which can be extracted from agent credentials or from the environment. For example, a policy can state that an agent of type persona can access devices of type lamp during daytime. As shown in previous works (FEDRECHESKI et al., 2021), evaluation of access policies using the SmartABAC model has negligible execution time and can be embedded in IoT devices, which improves privacy, latency, and reliability.

Figure 9 exemplifies what this configuration look like for two agents: a persona, meaning the virtual representation of a user; and a lamp. Each agent has a keystore, a DID Document, and a set of Verifiable Credentials. Thus, the persona agent in Figure 9 has private keys, a DID Document, and one credential that contains the attribute "friendOf: did:alice". If an agent will act as a responder, as is the case of the lamp agent, it must also have a list of trust anchors and a list of attribute-based policies. The trust anchors enable the lamp to know which credentials to trust, and the policies specify what interactions are allowed.

The Swarm agent data model enables full autonomy for Swarm agents to interact securely. It allows identification using DIDs, and agent description using public credentials. It enables confidentiality, integrity, and non-repudiation via the keys available in the Keystore and in the DID Document. Credentials can be used to authenticate an agent using attributes whose validity is ensured by the trust anchors. Finally, ABAC policies regulate agent interactions in a flexible way. We will now detail the mechanisms that leverage the data model to protect interactions.

3.2.2 Controlling interactions using attributes

An interaction is composed of an initiator agent, or subject, a responder agent, or object, and an arbitrary communication channel. This includes how to to get the agents to know about each other in the first place, what has already been solved by the Swarm Broker (COSTA et al., 2015), which provides a Discovery service, enabling agents to dynamically find each other based on semantic queries (CALCINA-CCORI et al., 2018). Thus, when an initiator finds a potential responder in the Swarm, a spontaneous interaction can take place.

In this dynamic scenario, in which an initiator contacts a responder, possibly for the

first time, **how can the responder decide whether a given interaction should be allowed (or denied)?** In other words, there must be a way for the responder to characterize the initiator either as an authorized or unauthorized party.

At a high level, our proposed solution relies on attribute-based authentication and authorization, as described in Sections 3.1.2 and 3.1.1. It works by authenticating the initiator with its attributes, and by having the responder evaluating access policies to check whether they match the attributes of the involved parties. The attributes of the initiator are stored and presented in the form of Verifiable Credentials, an open format for signed attributes in decentralized environments (SPORNY et al., 2019b). The access policies are defined using the SmartABAC model (FEDRECHESKI et al., 2021), which enables expressive and lightweight ABAC policies, suitable for being embedded in IoT devices.

Even though we have developed an efficient ABAC model (SmartABAC), computing an attribute-based authorization involves other costs, such as the transmission of the attributes of the initiator. To address this, we combine ABAC with the CapBAC approach, which uses tokens to encode authorization decisions. We do so by caching the attribute-based authorization response into a signed capability token, which can be used by the initiator multiple times. Such token encodes the initiator and responder identities, what kind of interaction is allowed, and for how long it is valid. The benefits of the token-based approach include smaller size when compared to credentials, faster verification when comparing to access policies, and improved privacy since it only discloses pseudonymous identifiers (DIDs).

Thus, in summary, an initiator proves its right to interact with a responder by using a capability token, which was programmatically generated by the responder after locally evaluating ABAC policies. This way, the system retains the flexibility of the decentralized attribute-based approach, while keeping the reduced overhead of token-based authorization. That is, since the token can be re-used, computation and network resources are saved, as the token is smaller than Verifiable Credentials and simpler to parse than SmartABAC policies. Our approach also enables decentralized processing by using a lightweight ABAC model that can be embedded in IoT devices, and by employing self-sovereign identity. The latter includes use of DIDs and VCs to identify and authenticate devices, and will be discussed further in the next subsection.

The process of obtaining an access token is presented in Figure 9. The first step consists in the establishment of an end-to-end secure channel by using DIoTComm (FE-

DRECHESKI et al., 2022), a DID-aware application-layer security protocol, proposed in Section 3.1.3. After creating the secure channel, the initiator requests an authorization token by presenting its credentials, i.e., signed attributes. The responder's Broker verify the credentials signatures considering the list of trust anchors, and compares the credential's attributes with the ones defined in the ABAC policies. If the policies are satisfied, a token is generated, and then returned to the initiator (not shown in the diagram). Finally, note that as per the Swarm architecture (COSTA et al., 2015), the Broker usually runs in the same device as the responder agent, therefore the access decision is computed locally. The whole process will be explained with more details in the next subsections.

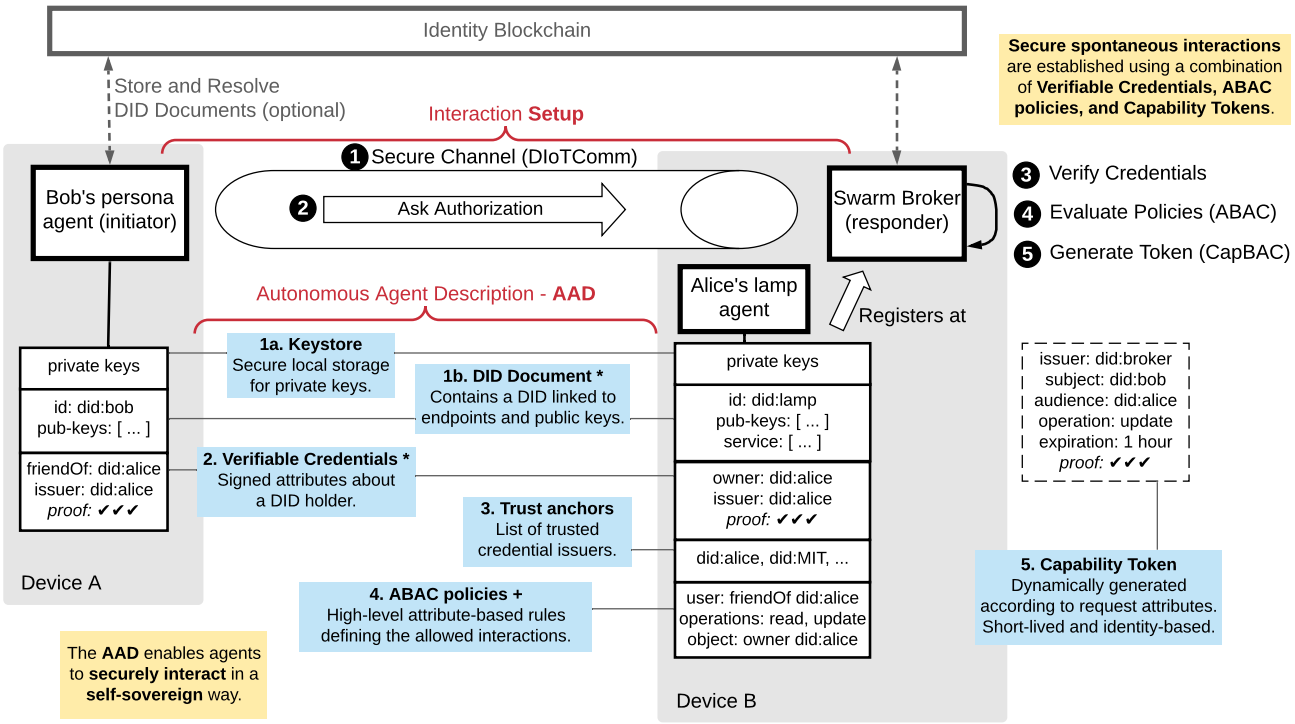


Figure 9: Setup of a secure spontaneous interaction between self-sovereign Swarm agents.

Next, as shown in Figure 10, the initiator can use the token to prove its right to interact with the responder. It does so by establishing a secure channel using DIoTComm, and sending a message along with the capability token. The responder agent, then, verifies whether the token satisfies the attempted interaction, and if the token is valid, the interaction is allowed to proceed.

Finally, note that in both the setup and execution phases, an identity blockchain may be involved. It serves as a decentralized source of truth for DID Documents, which primarily map DIDs to public keys and endpoints. As explained in section 3.1.4.1, this feature enables spontaneous secure communications between peers.

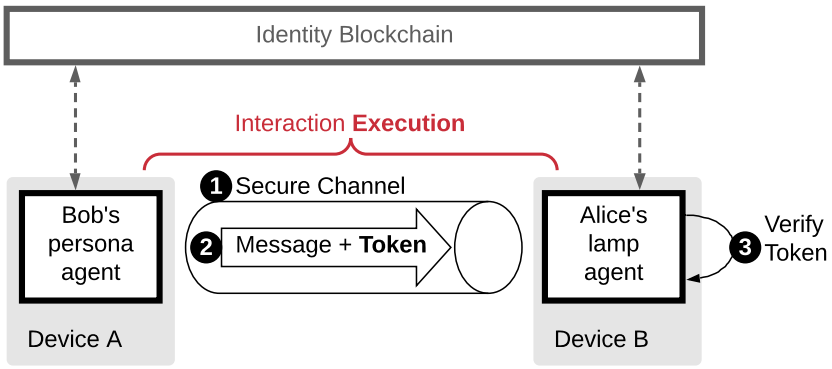


Figure 10: Secure interaction execution.

3.2.3 Confidentiality and pseudonymous Authentication

When two agents interact, the first level of security consists in providing confidentiality and pseudonymous authentication. The proposed framework does so by establishing a secure channel for communication (Step 1 of the interaction setup, shown in Figure 9). Confidentiality is implemented through message encryption using strong algorithms. Pseudonymous authentication relies on DIDs, enabling agents to verify their respective identifiers. Since DIDs are linked to DID Documents, and therefore to their respective keys, DID-based authentication enable agents to verify the origin of a confidential message.

The specific workings of this layer are provided by the DIoTComm protocol, described in Section 3.1.3. In DIoTComm, the initiator creates a confidential and authenticated binary message based on the data available in the Agent Description. That is, the message is encrypted with private part of the keys stated in the DID document, and it is authenticated based on the DID of the sender. Since DIoTComm messages are built at the application layer, they enable establishment of end-to-end security over heterogeneous protocols. Moreover, its use of binary encoding reduces overhead when comparing to existing alternatives for DID-based secure messaging (FEDRECHESKI et al., 2022).

3.2.4 Attribute-based authorization

After a secure channel is established, application-specific attributes and policies are used to verify whether an initiator agent can perform a specific operation on a responder agent – steps 2, 3, and 4 of Figure 9. Figure 11 details the steps involved in the authorization decision.

Upon receiving an authorization request that contains the attributes of the initiator,

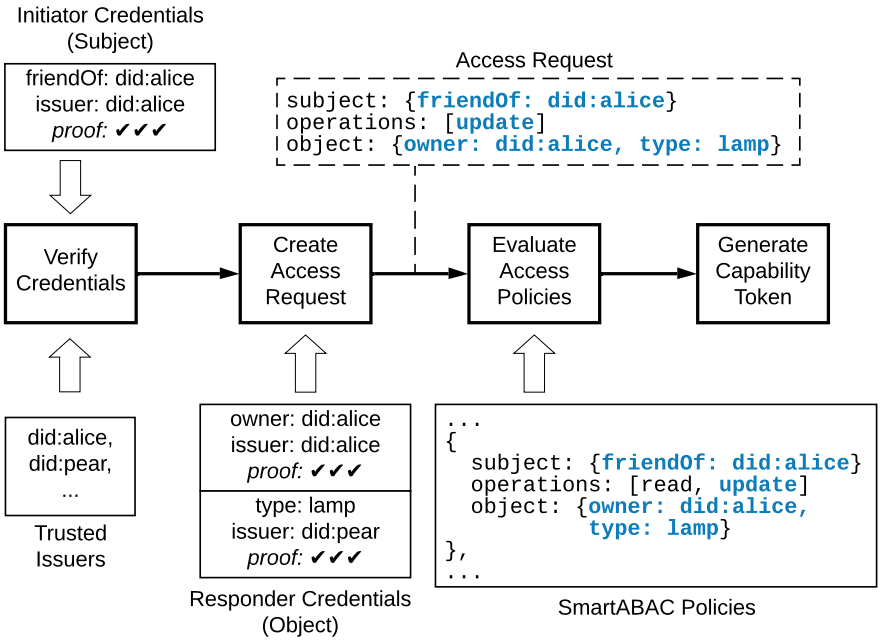


Figure 11: Steps of attribute-based authorization flow.

a responder agent will verify whether these attributes were signed by a trusted issuer. If that is the case, the next step is to create an access request to serve as input for the authorization function. The access request contains all the attributes from the initiator (subject), the intended operation, and all attributes of the responder (object). It might also include context attributes, such as current time and location. The access request is then compared to a set of attribute-based policies.

In this work we adopt the SmartABAC model for the authorization policies, since it allows creation of complex access policies that are fast to evaluate. If the access request matches at least one of the policies, the request is authorized. In SmartABAC, the matching rules consider the data type of each attribute to achieve flexibility in the access policies while maintaining a fast execution time. The matching of attribute-based credentials and high-level access policies is an important feature to enable secure spontaneous interactions in the Swarm.

3.2.5 Capability token generation

After a request is authorized, a short-lived capability token is generated, as per step 5 of Figure 9. It binds the identities (DIDs) of the initiator, responder, and token issuer (usually the responder's broker), as well as the authorized operation and an expiration time. The latter can be extracted from the policy that authorized the request, in case it contains a time restriction. For example, if the policy states that requests are allowed

between 1pm and 6pm, and the access request happens at 4pm, the token will be valid for two hours. If the access policy does not specify a time restriction, a sensible default should be used, ideally less than 24 hours.

Figure 12 shows an example token. It contains the DIDs of the initiator (subject), the responder (audience), and the broker (issuer). It also contains the operation, encoded in terms of Representational State Transfer (REST) methods, therefore they can be easily mapped to protocols such as HTTP or CoAP. The token body also holds an expiration time and a context restriction. Finally, the token has a digital signature computed by the issuer, which can be later verified by the responder, i.e. the token's audience.

```
{
  "subject": "did:sw:BaPShMy4Uh1fDgM61gPDMD",
  "audience": "did:sw:2WV5BUwcwQbw5TWQby835T",
  "issuer": "did:sw:9gc8hYeRD1D3vqBCZZ1CWc",
  "operation": {
    "method": "PUT",
    "entry": "/brightness"
  },
  "expiration": 1629330172,
  "context": {},
}
```

```
<digital signature by "issuer">
```

Figure 12: Example capability token.

The token in Figure 12 is shown in JSON for display purposes only, as in practice the token is serialized using COSE and relies on abbreviations to further reduce token size. We reuse the abbreviations defined in the CBOR Web Token (CWT) specification (JONES et al., 2018), and add new Swarm-specific abbreviations for the newly added `operation` and `context` keys.

3.2.6 Interaction Execution

The execution of an interaction between two Swarm agents is the final step and ultimate goal of the interaction setup process that was described in the previous section. In this step, shown in Figure 10, two agents interact with reasonable guarantees that their exchanged messages are confidential, integrity-protected, authenticated, and authorized.

The execution starts with the initiator agent, which assembles a DIoTComm message whose payload contains a capability token and an application-specific message. Upon reception and decryption of the message, the responder verifies the token by checking its fields as follows. The subject, audience, and issuer fields of the token must be equal, respectively, to the initiator, the receiver, and the Broker where the receiver is registered

at. Next, the token operation must match both the request method and path, and the token expiration date must not be in the past. If and only if all of the previous conditions holds true, the signature is verified, and if it is valid, the interaction is accepted by the responder. After it is finished, a response, protected by DIoTComm, is sent back to the initiator. This procedure can be run many times with the same token, as long as the token is not expired. To renew a token, an initiator must re-run the previously defined interaction setup procedure.

3.3 Concluding remarks

This chapter presented a framework for protection of interactions between Swarm agents. It presented solutions using a layered approach, from attribute-based authorization down to self-sovereign agent identification. Then, it described the operational flow for the establishment of a secure interaction, as well as the data model required for configuration of each agent. Next, we will present how this architecture was actually implemented to demonstrate that the solution is viable.

4 FRAMEWORK IMPLEMENTATION

This chapter describes the implementation of the proposed framework and its integration with pre-existing Swarm infrastructure, such as the Swarm Broker and the SwarmLib. Table 6 presents a summary of the software projects created or significantly modified to support the implementation of the proposed framework. These include five libraries and three Swarm Agents that are part of the Swarm infrastructure, which have some level of interdependency. Most of the libraries were created from scratch, and all the agents but the did-chain already existed and were updated to support the new security layer. A few example agents were also created, which are not directly shown in the table, but are discussed later in the text. Documentation on how to use each software described in Table 6 can be found in its respective repository link. The remainder of this chapter describes the implementations for each layer in detail.

Table 6: Software repositories created or modified as a result of this work.

Name	**Layer**	**Type**	**Description**	**Contribution**	**Depends on**
Swarm Broker[1]	All	Specialized Agent	Control plane services for IoT Swarms	Heavily modified	SmartABAC, cose-elixir
did-chain[2]	Identification	Specialized Agent	Storage of DID Documents	Created from scratch	SwarmLib, swarm-sec
economy-miner-python[3]	N/A	Specialized Agent	Economy and Reputation blockchains	Slightly modified	SwarmLib, swarm-sec
SwarmLib[4]	All	Library	Enables the creation of Swarm Agents	Heavily modified	swarm-sec
swarm-sec[5]	Authentication, Confidentiality	Library	Abstracts the handling of COSE messages	Created from scratch	-
smart-abac-elixir[6]	Authorization	Library	SmartABAC authorization and storage functions	Created from scratch	-
smart-abac-c[7]	Authorization	Library	SmartABAC authorization functions	Created from scratch	-
cose-elixir[8]	Authentication, Confidentiality	Library	Partial implementation of the COSE standard	Created from scratch	-

[1] ⟨https://github.com/swarm-citi-usp/iot-broker-elixir⟩ [2] ⟨https://github.com/swarm-citi-usp/did-chain⟩
[3] ⟨https://github.com/swarm-citi-usp/economy-miner-python⟩
[4] ⟨https://github.com/swarm-citi-usp/swarm-lib-python⟩ [5] ⟨https://github.com/swarm-citi-usp/swarm-sec⟩
[6] ⟨https://github.com/swarm-citi-usp/smart-abac-elixir⟩ [7] ⟨https://github.com/swarm-citi-usp/smart-abac-c⟩
[8] ⟨https://github.com/geonnave/cose-elixir⟩

4.1 The SwarmOS Framework

The Swarm (also referred to as SwarmOS) is a bio-inspired framework constituted by autonomous smart objects (RABAEY, 2011). In a parallel with swarms of bees, with specialized bees contributing to a common goal; the Swarm is composed of specialized devices whose interaction solves a common problem. The Swarm network behaves like an organism and shows an organized behavior that results in an emergent collective intelligence (COSTA et al., 2015). Analogously to specialized honeybees, heterogeneous devices in the Swarm might range from high-power processing servers to low-power wearable devices. Human participation in the Swarm network is also of crucial importance.

The SwarmOS framework is composed of networked IoT devices, with no dependencies on the network topology. As shown in Figure 13, IoT device runs one or more Swarm agents. A Swarm agent is a uniquely identifiable logical entity that is able to interact with the Swarm, and there are two main types of agents. One type is the Broker agent, which implements services such as registry, discovery, and reputation, which enable other agents to find, establish cooperating relations, and trust each other. The second type of agent is the application agent, which implement specific functionalities that vary according to each application. Such agents register themselves on Brokers, which in turn enables other agents to perform discovery operations through a network of connected Brokers.

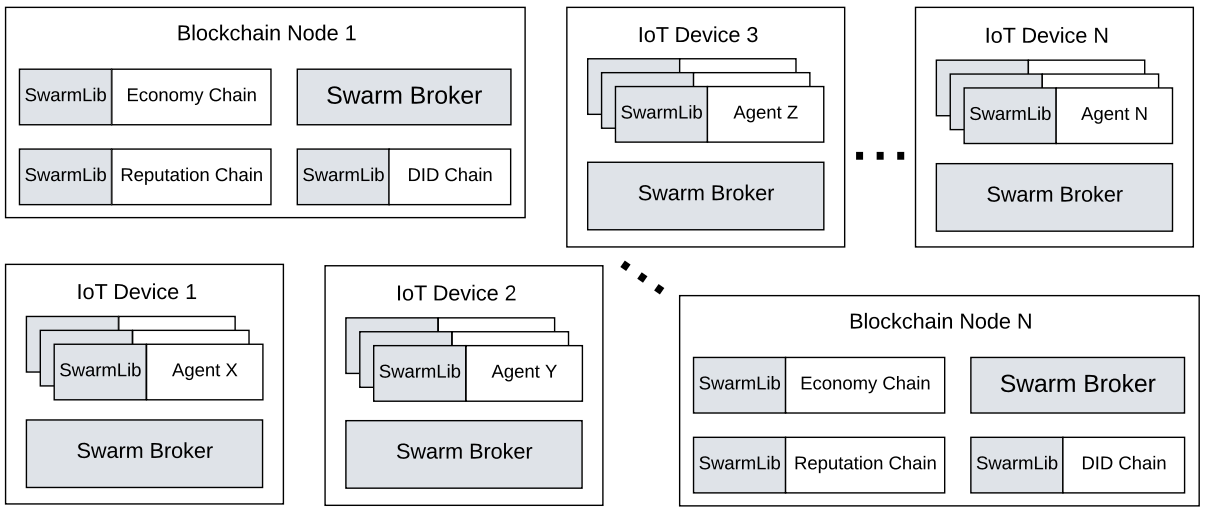


Figure 13: The SwarmOS framework organization.

Since much of the functionality implemented by application agents is shared, it was extracted to a library called SwarmLib. Figure 13 shows an example with many IoT devices, each of which runs one Broker, a few application agents (or simply, agents). As described in previous works, the SwarmOS framework has economic mechanisms for

incentivising resource sharing, as well as a reputation scheme to improve trust in the network (BIASE et al., 2018b). These mechanisms are implemented through blockchains, which in turn are modeled as Swarm agents as well. Finally, as proposed in the current work, a third blockchain was added, the DID Chain, whose objective is to serve as a decentralized and secure layer for DID Document storage.

In this thesis, significant contributions were made to both the Broker and the SwarmLib. Figure 14 illustrate the added contributions contributions highlighted in blue; on the bottom-right side of the blue boxes, libraries developed in this thesis (shown in Table 6) are also shown. For example, implementing the Authorization module in the Broker required the development and use of the cose-elixir and smart-abac-elixir libraries. Regarding the Broker, three main modules have been added. The first implements secure communication channels using DIoTComm, the probocol described in Section 3.1.3. Another module implements authentication of attributes, as described in Section 3.1.2 and step 3 of Figure 9. Finally, the authorization module evaluates attribute-based policies, based on the SmartABAC model (Section 3.1.1.2), and generates capability tokens according to Section 3.2.5.

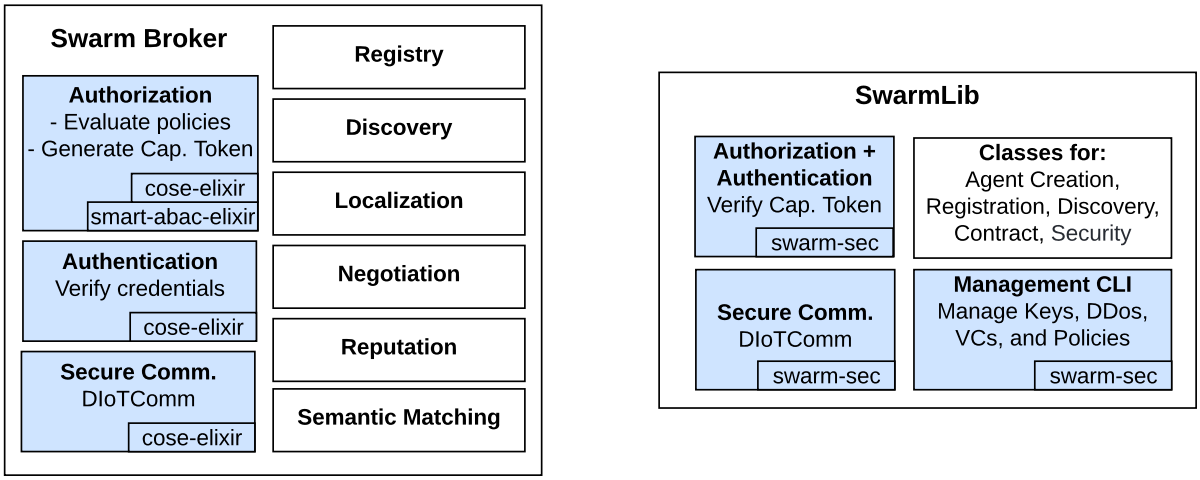


Figure 14: Contributions to the SwarmOS framework, highlighted in blue.

In the SwarmLib, analogous security modules have been added as well. Confidentiality was achieved by adding a module that implements DIoTComm; and Authentication and Authorization were achieved by implementing a module that verifies capability tokens, as described in Section 3.2.6. In addition, the SwarmLib was also integrated with a command-line interface (CLI) tool that enable several management operations, including: creation of a full agent data model instance, issuance and inclusion of verifiable credentials, registration of DID Documents in the DID Chain, and the definition of SmartABAC policies.

4.2 Authorization

The authorization layer was implemented partly in separated libraries, and partly as additions to the Swarm Broker and the SwarmLib. The SmartABAC model, described in Section 3.1.1.2, was implemented as the smart-abac-elixir and smart-abac-c standalone libraries. The Elixir version was created to facilitate integration with the Swarm Broker, which is written in Elixir, and the C version is meant to allow execution on highly constrained devices. In both implementations, all functions of the SmartABAC model were implemented, while the Elixir implementation featured additional utilities, such as storage and JSON and CBOR serialization functions. In order to realize the *expand* function described in Table 5, both versions feature a breadth-first search algorithm which can traverse the graph of hierarchical attributes. This approach works since the hierarchy is a directed acyclic graph.

The Broker also received modifications to handle authorization. It used smart-abac-elixir as a library in order to store and evaluate SmartABAC policies, and was updated to handle the remaining of the attribute-based authorization flow (Figure 11). These include attribute extraction from credentials, credential verification, construction of access requests, calling of the policy evaluation functions, and generation of the capability token. To generate the token, it relied in the provided cose-elixir library, which implements the CBOR Web Token standard (JONES et al., 2018).

The main changes to the SwarmLib regarding authorization include the ability to intercept requests and to verify authorization tokens. The purpose of request interception is to verify whether whether an initiator is authorized to perform a given request. This was achieved by adding a Python decorator on each endpoint that should be authorization-protected, a flexible design that makes it simple for implementers to choose which parts of the agent should be public, and which should be protected. For such protected endpoints, the presence of a valid capability token would be checked via the provided library swarmsec, which implemented verification of CBOR Web Tokens; swarmsec, in turn, delegated the handling of COSE objects to the external library pycose[1].

4.3 Authentication

In the context of our framework, both messages and agents can be authenticated. Message authentication is specified in DIoTComm, whose implementation builds upon

[1] https://github.com/TimothyClaeys/pycose

cose-elixir and swarm-sec in the Broker and in the SwarmLib, respectively. It works based on the fact that AES-CCM provides authenticated encryption, therefore all messages that are successfully decrypted, are also authentic.

Agent authentication refer to the presentation of signed attributes, in the form of verifiable credentials, during an interaction setup. The implementation in the SwarmLib leverages swarmsec and pycose to generate such credentials, in the format of CBOR Web Tokens. The verification of VCs by the Broker is implemented using the cose-elixir library, during the authorization flow.

4.4 Confidentiality

The confidentiality of messages is provided by DIoTComm, which was implemented in both the Broker and the SwarmLib. In the SwarmLib, the construction of COSE (DIoTComm) messages is delegated to swarm-sec, which in turn uses the pycose library. Specifically, the function enc_encrypt_ecdh_ss in swarm-sec takes a payload, a sender (private) key, and a receiver (public) key, and returns an encrypted COSE object. In doing so, it specifies the key derivation function (ECDH with HKDF of 256 bits), the AES-CCM algorithm, and generates a nonce that combines timestamp and a random part. Both the Broker and the SwarmLib leverage DIoTComm to ensure that their incoming and outgoing messages are confidential. For incoming traffic, messages are intercepted at early stages (usually an extension of the server library) to enable decryption. For outgoing messages, functions implemented in the SwarmLib and in the Broker construct secure messages before they are sent through the network.

4.5 Identification

Agent identification was implemented by providing capabilities to handle Decentralized Identifiers in both the SwarmLib and the Broker. Specifically, the SwarmLib now provides functions for creation, storage, and serialization of DIDs and DID Documents, as well as registration and updates of DID Documents at the Identity Blockchain. To facilitate creation of new identities, a command line program named swarm_manager was created as part of the SwarmLib. The Broker was modified to read and handle DIDs and DID Documents, and to be able to resolve DDos from the Identity Blockchain.

The Identity Blockchain is an important part of the identification layer. Due to the

complexity of implementing a full blockchain, we considered it out of scope of this work, and provided instead a mock implementation, which we refer to as did-chain. The mock provides an API for creation and updates of DID Documents, and uses a nosql database (MongoDB) to store the DDos. Nevertheless, the mocked did-chain is secure in that it only accepts creation and updates of DDos that have been signed by the owner of the DID contained in that DDo. Read operations, on the other hand, can be performed by anyone in the network, since DIDs are public and pseudonymous by design. In addition, updates can only update the endpoints, which usually change due to the agent acquiring new IP addresses as it roams through different networks.

4.6 Example: creating a camera agent using the SwarmLib

The framework described so far deals with a broad set of security principles, and it would be unwise to let their implementation to the discretion of the user. For this reason, the functionality that is common for all Swarm agents is abstracted by the SwarmLib. In order to illustrate how the SwarmLib can be actually used, this section presents the full implementation of an example Swarm agent. It consists in less than 30 lines of Python code, while providing all of the security mechanisms presented in Chapter 3. The chosen agent is a camera agent, which has one authorization-protected endpoint for returning frames, as shown in Listing 1.

First, the necessary libraries are imported, including the Agent and Wallet classes of the SwarmLib. Next, the Wallet is initialized with a storage location, and the camera_object is loaded with the information of the agent data model, which contains the agent's Keys, DID Document, Credentials, and Access Policies (see Section 3.2.1). Since the camera will act as a provider, it enables the use of a server, which uses the flask library[2]. It also initializes the actual camera device.

Next, from lines 11 to 20, an endpoint that accepts the GET method on the /frame method is defined and implemented. Since line 11 calls flask through the camera_agent object, only requests that have valid DIoTComm messages will be accepted, thus providing identification, confidentiality, and authentication. Next, line 12 specified that authorization tokens must be enforce for this route, thus asserting that only authorized initiators will have access to the frame. Lines 13 to 19 implement aquisition of the camera image, and line 20 creates and send a DIoTComm response to the initiator.

[2]https://flask.palletsprojects.com/en/2.1.x/

```python
from flask import request
from swarm_lib import Agent, Wallet
from labrador_camera import LabradorWebcam
import cbor2, logging, os, cv2

Wallet.init(os.environ.get("SWARM_BASE_DIR"))
camera_agent = Agent.from_config(os.environ.get("SWARM_AGENT") or "camera")
camera_agent.enable_flask_app(__name__)
webcam_device = LabradorWebcam(device=os.environ.get("WEBCAM_DEVICE") or 0)

@camera_agent.flask_app.route('/frame', methods=["GET"])
@camera_agent.enforce_authorization_token
def frame():
    ok, frame = webcam_device.read()
    if ok:
        frame_jpg = cv2.imencode('.jpg', frame)[1].tobytes()
        resp_data = cbor2.dumps({"result": "ok", "value": frame_jpg})
    else:
        resp_data = cbor2.dumps({"result": "error", "value": ok})
    return camera_agent.flask_app.secure_response(request.remote, resp_data, 200)

if __name__ == "__main__":
    camera_agent.register_identity()
    camera_agent.register_at_broker()
    camera_agent.serve_with_flask(debug=False)
```

Listing 1: Example of a Swarm camera agent implemented using the SwarmLib.

Finally, the last three lines are executed once whenever the script is executed. First, the agent registers (or updates) its DID Document a the Identity Blockchain; next, it registers its full agent data model (except private keys) at its broker of trust; and at last it enables the server to make the endpoint available to the Swarm network.

4.7 Concluding remarks

This chapter presented how the proposed framework was implemented and integrated within the Swarm. Relevant details of the implementation of each layer were discussed, and a real example of a Swarm agent was presented. The next chapter will provide an evaluation of each layer, as well as the full integrated work, to demonstrate its feasibility to run in real IoT hardware and actually protect Swarm agents.

5 EVALUATION AND RESULTS

So far we have proposed and implemented a solution for protecting interactions between IoT devices in Swarm environments. This chapter presents a comprehensive evaluation of such solution. First, we evaluate the mechanisms for authorization, then we measure the overhead of the identification and protection mechanisms. Finally, we present an evaluation of the complete solution.

Note: A significant portion of the evaluation and results presented in this section has been previously published in the following works:

- Fedrecheski et al. (2021): A low-overhead approach for self-sovereign identity in IoT.
- Fedrecheski et al. (2022): SmartABAC: enabling constrained IoT devices to make complex policy-based access control decisions.

5.1 Evaluation of the SmartABAC model

This section presents an evaluation of the SmartABAC model, including a use case, a comparison with existing ABAC models, and a performance analysis on constrained and non-constrained platforms.

5.1.1 Smart Home Use Case

We provide a Smart Home use case to demonstrate the application of our model in a realistic situation. It is worth mentioning that SmartABAC improves availability and privacy, specially relevant in this scenario. First, it involves personal devices whose usage patterns represent highly sensitive information, therefore it is essential, from a user perspective, to keep access control decisions private. Second, having home devices

depending on external authorization services exposes owners to availability issues if these services fail temporarily or permanently. Table 7 shows the use case participants and their respective attributes, which are discussed below:

1. Alice, Bob, and Carl: users that live in the household "home-1". Carl is Alice and Bob's son. Alice is the owner of most devices in this household, except for the other two user devices (Bob and Carl). Their respective attributes will determine their access to different devices in the house.

2. Lamp1 and Lamp2: smart lamps whose status can be queried and updated. Note that, while their "type" and "manufacturer" attributes have distinct values, both lamps have similar functionality.

3. Camera1: a security camera located outside the house from which frames can be read.

Table 7: Use case actors and associated attributes.

Alice	**Bob**	**Carl**	**Lamp1**	**Lamp2**	**Camera1**
id: alice owner: alice type: user manufacturer: pear household: (id: home-1, role: mother) lastName: smith age: 34	id: bob owner: bob type: user manufacturer: pear household: (id: home-1, role: father) lastName: smith age: 36	id: carl owner: carl type: user manufacturer: sberry household: (id: home-1, role: child) lastName: smith age: 10	id: lamp1 owner: alice type: lamp manufacturer: pear household: (id: home-1)	id: lamp2 owner: alice type: lightBulb manufacturer: sberry household: (id: home-1)	id: camera1 owner: alice type: securityCamera manufacturer: pear household: (id: home-1) location: outdoor

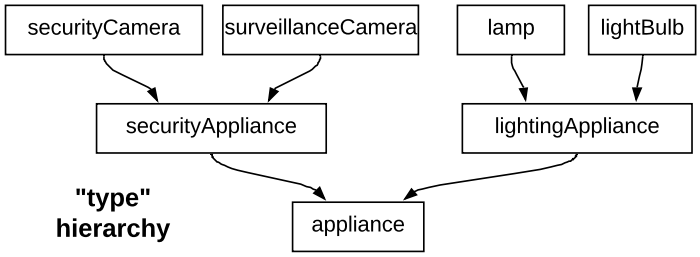
Source: Fedrecheski et al. (2021).

Table 8: SmartABAC policies that satisfy the given use case.

Id	**Description**	**Subject Attributes**	**Operations**	**Object Attributes**	**Context Attributes**
p1	Alice can do anything with her own devices	id: alice	create, read, update, delete	owner: alice	
p2	Adults can use security appliances	age: (min: 18) household: (id: home-1)	read, update	type: securityAppliance household: (id: home-1)	
p3	Children can check and update any lamp when it is dark outside	household: (id: home-1, role: child)	read, update	type: lightingAppliance household: (id: home-1)	outdoorLuminosity: (max: 33)
p4	Camera1 can read and toggle Lamp1	id: camera1	read, update	id: lamp1	
p5	Users with reputation of at least 4 can contract outdoor cameras during the day	reputation: (min: 4)	contract	type: securityCamera household: (id: home-1) location: outdoor	hour: (min: 8, max: 18)
p6	A specific device can read frames from Camera1 during a 5-minute time window	id: some-device-x	read	id: camera1	year: 2020, month: 6 day: 30, hour: 17 minute: (min: 20, max: 25)

Source: Fedrecheski et al. (2021).

Given the described participants and their attributes, Table 8 presents six policies, identified by column "Id" to facilitate future reference, and whose purpose is written in plain English in column "Description". For example, policy p2 regulates the interaction



Source: Fedrecheski et al. (2021).

Figure 15: Example hierarchy for the "type" attribute.

between adult users and security appliances. The remaining columns contain the policy itself, expressed in terms of the allowed operations and the subject, object, and context attributes. Returning to the policy p2 example, the subject attributes contain a range-typed attribute for age (that must not be less than 18) and a nested attribute that identifies a specific household. It further specifies that read and update access will be given only to security appliances in the same household. Note also that some object attributes in the policies refer to "type" attributes that are not present in Table 7. The link between these attributes is expressed by the hierarchy in Figure 15. For example, both lamp and lightBulb have lightingAppliance as a successor in the hierarchy. This allows creating higher-level rules, reducing the number of required policies, and simplifying policy administration.

5.1.2 Comparison with existing models

Table 9 presents a comparison of SmartABAC and related work. The criteria used in the comparison are discussed below.

5.1.2.1 Model Features

In Section 3.1.1.2, we highlighted the advantages of using a particular set of features (nested, typed, and hierarchical attributes, and enumerated and multi-attribute policies) to design an ABAC model that is both lightweight and expressive. When comparing to the works in Table 9, it first comes to attention that none of the enumeration-based models supports typed attributes, limiting their expressiveness. Moreover, PM and CoAC fully support hierarchies, while other models do not support. Also, note that hierarchy support for MQTT ABAC is limited, since it is fixed to MQTT protocol entities. Finally, all the models but PM support multi-attribute policies, a feature that facilitates the expression of logical conjunctions. None of the existing models concurrently supported the features

Table 9: Comparison of ABAC models for IoT environments.

	Criteria	Smart-ABAC	PM[1]	EAP-ABAC$_{m,n}$[2]	HG-ABAC[3]	XACML[4]	MQTT ABAC[5]	CoAC[6]	Block. ABAC[7]	ITS-ABAC$_G$[8]	Auto. ABAC[9]	Pol. ABAC[10]
Model features	Typed	✓			✓	✓				✓*		
	Hierarchical	✓	✓		✓		✓*	✓		✓		
	Enumerated	✓	✓	✓				✓			✓	✓
	Multi-attribute	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
Expressiveness	Context-aware	✓			✓	✓	✓	✓	✓*	✓*	✓*	
	Numeric ranges	✓			✓	✓	✓	✓	✓*			
	Nested attributes	✓										
	Protocol-agnostic	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Supports policies from use case	p1	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
	p2	✓										
	p3	✓										
	p4	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
	p5	✓										
	p6	✓			✓	✓		✓	✓			
Implementation	Evaluation	✓	✓		✓	✓			✓	✓	✓	✓
	Open source	✓	✓		✓	✓						
Architecture	Designed for constrained devices	✓										
	Standalone access decision	✓										

[1] Ferraiolo, Atluri & Gavrila (2011). [2] Biswas (2017). [3] Servos & Osborn (2014).
[4] Chehab & Mourad (2020), Parducci, Lockhart & Rissanen (2013), Esposito, Ficco & Gupta (2021), Kim, Kim & Alaerjan (2021).
[5] Gabillon, Gallier & Bruno (2020). [6] Li et al. (2018). [7] Zhang et al. (2021).
[8] Gupta et al. (2021). [9] Karimi et al. (2021). [10] Jabal et al. (2020).
[*] Limited support.

Source: Adapted from Fedrecheski et al. (2021).

proposed in the SmartABAC model.

5.1.2.2 Expressiveness

This criterion refers to a qualitative evaluation regarding the capabilities of ABAC models.

- Context-awareness: in the Internet of Things, a common requirement is to have authorization decisions linked to contextual factors, such as time of day or geolocation. Among the compared works, the PM and EAP-ABAC$_{m,n}$ do not support this feature. Blockchain ABAC and ITS-ABAC$_G$ support it but have limitations, since the former only supports time and the latter only considers location.

- Numeric ranges: comparison among numbers is essential in a rule-based system to avoid extensive enumeration of all the possible values. While most existing models support ranges, the PM and EAP-ABAC$_{m,n}$ models do not, and the ABAC model by (ZHANG et al., 2021) has limited support since it only works with time ranges.

- Nested attributes: attributes either have atomic values, such as `role=adult`, or have nested values, such as `degree=(year=2020, level=phd)`. This is common, for example, in the Verifiable Credentials specification for portable identity (SPORNY et al., 2019b), which can be applied to IoT environments to encode attributes (FE-

DRECHESKI et al., 2020). *None of the previous ABAC models had support for nested attributes.*

- Protocol-agnostic: since IoT applications tend to span over heterogeneous networks and protocols, we consider an essential feature for an ABAC model not to be tied to a specific protocol. The MQTT ABAC model, as the name implies, does not support this capability.

5.1.2.3 Policies From Use Case

As mentioned, all of the example policies in Table 8 are expressed using SmartABAC. This section evaluates whether other models could be used to create the same policies.

- PM: the only policies fully supported by the PM are p1 and p4. Other policies are not supported because PM supports neither typed attributes nor context attributes.
- EAP-ABAC$_{m,n}$: similarly to the PM, this model only supports p1 and p4.
- HGABAC: since this model uses a rich policy language, it fully supports policies p1, p4, and p6. However, it does not support nested attributes, rendering it unable to encode policies p2, p3, and p6 directly.
- XACML: this model supports p1, p4, and p6, but cannot express other policies due to the lack of nested attributes.
- MQTT ABAC: since it only supports operations from the MQTT protocol (publish, subscribe, and deliver), it cannot encode any of the policies from Table 8.
- CoAC: this is an expressive model, however, it also does not support nested attributes. Therefore it is unable to directly represent policies p2, p3, and p5.
- Blockchain ABAC: this model has support for basic attributes, including support for time-based context. Still, it is not able to express policies p2, p3, and p5.
- ITS-ABAC$_G$: this model supports policies p1 and p4, however it cannot express policies p2, p3, p5, and p6 dues to lack of the range type.
- Automatic ABAC: although this model supports policies p1 and p4, it cannot express other policies due to lack of the range type.
- Polisma ABAC: similarly, this model can only express policies p1 and p4 due to lack of the range type.

5.1.2.4 Implementation

Implementing an ABAC model helps to validate the feasibility of the system. Most works evaluated their implementation and considered factors such as policy evaluation time, policy size, and others. Unfortunately, although many did implement their models, only a subset of them feature open-source implementations, i.e., PM, HGABAC, and XACML. This hinders the possibilities of comparison, since in the absence of the source code, it is not possible to craft similar policies and execute tests under an homogeneous environment. Regarding our model, we implemented SmartABAC in two programming languages, Elixir for faster prototyping and C for integration in embedded systems. The source code is available and linked on Table 10.

5.1.2.5 Architecture

Among the compared works, all but one (CHEHAB; MOURAD, 2020) aimed to run within constrained devices. Their solution, however, was only tested in a smartphone with a 1.4 GHz CPU and 1 GB RAM, which cannot be considered a constrained device (BORMANN; ERSUE; KERANEN, 2014). On the other hand, SmartABAC was designed to run even on microcontrollers with just a few MHz of clock and less than 512 KB of RAM. Henceforth, SmartABAC allows standalone access decision since no remote server is needed to evaluate the policies.

5.1.3 Performance comparison with open-source models

To compare the execution time among different models, we first selected the models that had available open-source implementations, i.e., PM, HGABAC, and XACML. Then, we selected the policies from our use case that are supported by these models, i.e., p1, p4, and p6. Even though the PM does not support context attributes, as required by p6, we built a partial p6 policy (without context information) and included PM in the experiment. Results are available in Table 10.

Time to evaluate policies We measured the time, in milliseconds, to evaluate 1 request against policies p1, p4, and p6, repeatedly for 3000 times. These tests were run on a laptop with a 1.8 GHz quad-core CPU and 8 GB of RAM, since it was a platform capable of running all the available implementations. Even though our Elixir implementation of SmartABAC was almost twice as slow as that of XACML, it is still more than 10 times

Table 10: Comparison of open-source ABAC implementations.

Criteria	**Smart-ABAC**		**PM**	**HGABAC**	**XACML**
Time (ms) to run 1 request against policies p1, p4, and p6, 3000 times	0,18 (C[a])	70 (Elixir[b])	1112 (Ruby[c])	840 (Python[d])	46 (Java[e])
Average policy size (bytes)	85		52	111	3433
Lines of code	449	616	1277	1854	19269

[a] ⟨https://github.com/swarm-citi-usp/smart-abac-c⟩
[b] ⟨https://github.com/swarm-citi-usp/smart-abac-elixir⟩
[c] ⟨https://github.com/mdsol/the_policy_machine⟩
[d] ⟨https://github.com/dservos/HGABAC⟩
[e] ⟨https://github.com/authzforce/core⟩

Source: Adapted from Fedrecheski et al. (2021).

faster than PM and HGABAC. On the other hand, our C implementation of SmartABAC was 255 times faster than its closest competitor, a Java implementation of XACML.

It is worth mentioning that we were unable to directly compare the performance of SmartABAC to works that did not share their software as open source. Still, we can provide a discussion on apparent performance considering their respective results. For instance, the ITS-ABAC$_G$ work took 0.08 milliseconds to evaluate 10 requests against a single policy in a cloud server; and Blockchain ABAC mentions that their authorization flow is limited to the throughput of the underlying blockchain, which in the case of Ethereum 1.0 supports only 15 transactions per second. Therefore, although more work is needed to provide direct comparisons, our proposal seem to be more efficient than existing works.

Policy Size Since IoT devices may be constrained in storage and bandwidth, embedded access policies must have a compact size. To compare the policy size among models, we measured the size of policies p1, p4, and p6 for each model and computed the average. Some models already have a serialization format for policies. For example, XACML uses the eXtensible Markup Language (XML) and HGABAC uses the HGABAC Policy Language (HGPL), so we expressed and measured the policies in these formats. For SmartABAC, we used CBOR, a compact and extensible data format commonly used in IoT applications. We also used CBOR to represent the Policy Machine graph as an adjacency list since it originally did not provide a serialization mechanism.

The line "Policy size" in Table 10 contains the results. The larger policy is that from XACML, which is about 40 times larger than SmartABAC policies. This is mainly due to the verbosity of XML encoding. SmartABAC is also more compact than HGABAC, while it uses more bytes than PM. Still, the small footprint of SmartABAC policies allows it to store up to 3000 policies in a device with 255 kB of memory.

Lines of Code Besides performance, the size of a codebase in storage-constrained platforms should be considered. Even though we acknowledge that such a comparison depends on the programming language and features implemented, we include it for reference. We used the Linux utility `wc`[1] to count lines, which ignores comments and blank lines. Moreover, observing that the PM repository provided different storage adapters, we only included the in-memory adapter in the count. Table 10 shows the results, wherein source lines of code sizes vary from almost twenty thousand for XACML, to well under one thousand for SmartABAC.

5.1.4 Performance of SmartABAC on different platforms

In this section, we measure the performance of two SmartABAC implementations on different platforms, as shown in Table 11. Laptop and Labrador are non-constrained devices running Debian 10, while ESP32 and Pulga consist of MCUs that run bare-metal software. A C-based implementation was tested in all platforms, while an Elixir-based implementation was tested only on the Laptop and Labrador platforms.

Recall that, as shown in Table 7, the same attribute may be shared by many entities. For example, many thousands of cameras could have the `type: camera` attribute. Moreover, this effect is increased when considering hierarchies, as shown in Figure 15. Thus, a single SmartABAC policy could regulate access between many IoT devices. Aiming to emulate large scale scenarios, we measured policy evaluation time in two configurations: thousands of authorization requests against a few policies; and one request against thousands of policies.

Table 12 shows the test descriptions and results. In the first test, we evaluated one request against the six policies defined in Table 8, repeated it 3000 times, and measured the total time. This scenario mimics an IoT device receiving thousands of authorization requests in a short period. The C implementation showed a negligible delay on Laptop and Labrador platforms and an acceptable delay on the two resource-constrained platforms.

[1] https://linux.die.net/man/1/wc

Table 11: Description of the platforms used during evaluation of SmartABAC.

Platform	**Description**
Laptop	Intel i7 quad-core 1.8GHz, 8 GB RAM
Labrador	ARM Cortex A9 quad-core 1.3GHz, 2 GB RAM
ESP32	Xtensa LX6 dual-core 240MHz, 520 KB RAM
Pulga	ARM Cortex-M4F single-core 32MHz, 256 KB RAM

Source: Fedrecheski et al. (2021).

Table 12: Performance of SmartABAC on different platforms.

Test Description	**Platform**	**SmartABAC (C)**	**SmartABAC (Elixir)**
Time (ms) to run 1 request against 6 policies, 3000 times	Laptop	0,5	76
	Labrador	5,5	830
	ESP32	88	n/a
	Pulga	176	n/a
Time (ms) to run 1 request against 3000 policies	Laptop	0,04	1
	Labrador	0,12	6
	ESP32	1	n/a
	Pulga	4,8	n/a

Source: Fedrecheski et al. (2021).

The delay of using Elixir in Laptop was also acceptable, while its evaluation delay in Labrador could imply a noticeable delay in response time.

In the second test, we considered a device in a highly dense network, containing thousands of policies. Hence, we measured the time needed to evaluate one request against 3000 policies. The evaluation time in this scenario was negligible for all the platforms and implementations tested. Even for the most resource-constrained device, a 32 MHz MCU, less than five milliseconds were necessary to process 3000 policies.

5.1.5 Discussion

In this section, we compared SmartABAC with existing works to demonstrate its expressiveness and ran tests on different platforms to show its ability to run even on constrained devices. Next, we discuss some of the implications and limitations of our work.

Contributions to ABAC in IoT In previous work, the use of ABAC in IoT has been considered useful for its flexibility and granularity, however, it has been pointed as lacking features such as heterogeneity and offline mode (OUADDAH et al., 2017). With SmartABAC, we hope to have broadened the applicability of ABAC to IoT systems in

general. By testing it on embedded devices, we have shown that our model is lightweight, has potential for offline mode, is distributed across devices, and can be executed in heterogeneous platforms.

Enabling embedded policy evaluation As shown, most existing works rely on a third party to evaluate policies, as it has been considered impractical for devices to compute their own decisions (RAVIDAS et al., 2019). With our work, we hope to have proposed a counterexample in which policy evaluation algorithms run within constrained devices with no significant overhead.

Benefits of device autonomy SmartABAC allows the creation of IoT access control systems that have enhanced privacy and reliability, and reduced latency. Regarding privacy, by keeping access control decisions within IoT devices, external servers are prevented from logging or monitoring the interactions of other IoT agents. Reliability is also improved since reducing dependency on third parties implies that outages in remote servers will not affect policy decisions, and network issues are less likely to interfere. Moreover, latency is reduced. As shown, evaluating a single request against thousands of policies can take as little as 5 milliseconds on a constrained device. Even for optimized edge applications, a round-trip to an IoT gateway rarely takes less than 10 ms (NIKAEIN; VASILAKOS; HUANG, 2018).

Limitations In this work we focused on providing an authorization layer and policy model for constrained devices in the Internet of Things. We intentionally do not provide mechanisms to authenticate attributes or to protect messages against eavesdropping and other attacks, as we consider those out of the scope of our work. Similarly, we also do not include a formal security analysis, as we consider that it would more appropriate over the authentication and encryption layers. Thus, we leave topics such as attribute authentication, message protection, and security analysis to future work.

5.2 Evaluation of the Swarm DID Method and DIoT-Comm

We evaluate our system in three steps. First, we use a widely known threat model to check the security of our system. Then, we compare the size of DIDs, DDos, and

Table 13: Mitigations of DIoTComm against the STRIDE threat model.

Threat	Element under threat	Attack mitigation	Technique
Spoofing of identity	Blockchain identity, IoT agent identity	Authentication via Authenticated Encryption and/or Digital Signature	AES-CCM with tag, EDDSA
Tampering of data	Message to register or query DDo, Message between IoT agents	Data integrity via Authenticated Encryption	AES-CCM with tag
Repudiation of messages	Message to register or query DDo, Message between IoT agents	Non-repudiation via Digital Signature	EDDSA
Information disclosure	Message between IoT agents	Confidentiality via Authenticated Encryption	AES-CCM with tag
Denial of service	Blockchain node IoT agent	Network analysis and other techniques (out of scope)	(out of scope)
Elevation of Privilege	Update blockchain operation IoT agent	Only a DID owner can update a DDo's keys Policy list (upper layer, out of scope)	(out of scope)

DIoTComm messages with existing mechanisms. And finally, we measure the performance of our proposal.

5.2.1 Security Analysis

In this section, we analyse possible threats to our system and explain how they can be mitigated. To do so, we use the STRIDE threat model, an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege (HERNAN et al., 2019). Table 13 summarizes how these threats might affect different elements of our system, along with our approach to mitigate them. The term *message* in the table should be interpreted as a DIoTComm message. Similarly, *identity* refers to the DID, DID Document, and associated private keys of a given agent.

The first threat we consider is identity spoofing, which happens when a malicious entity pretends to be someone else. In our system, the worst case would happen if the Identity Blockchain is targeted by such an entity, which would cause the identity of all IoT agents to be fundamentally insecure. Another way this threat could be explored is if an attacker pretended to be an existing IoT agent, for example, by forging its sender id. DIoTComm mitigates these threats at two levels. In the first layer, it uses an authenticated encryption algorithm (AES-CCM), which not only encrypts data, but also appends a one-time message authentication code to the ciphertext. This enable the data origin to be authenticated. The second layer of mitigation uses digital signatures to further ensure authentication of IoT agents. This is provided by the EDDSA algorithm, which uses agents' private and public key pairs to, respectively, sign and verify signatures.

When receiving a message, an agent must ensure it has not been tampered along the way. This should be checked not only by IoT agents, but also by the Identity Blockchain, which must check that new DID Document registrations arrived without modifications. This threat can be mitigated by having messages protected with either authenticated

encryption, digital signatures, or both. As mentioned, DIoTComm already provides such protection.

Some IoT applications may impose legal penalties to agents that cause harm to its surrounding environment. Such agents may try to masquerade their malicious activities by simply alleging that they did not execute any malicious command, what is known as message repudiation. Agents using DIoTComm mitigate this threat by digitally signing exchanged messages. Since only the private key holder is able to produce a valid signature, agents can be made accountable against malicious behavior.

A basic and important requirement of secure systems consists in preventing information disclosure threats, wherein unauthorized parties might get access to confidential data. Thus, messages exchanged between IoT agents, or between agents and the Identity Blockchain, should be encrypted using state-of-the art algorithms. Our system provides such protection by employing the AES-CCM symmetric encryption algorithm with keys of 128 bits.

While some threats can be mitigated by employing cryptographic algorithms, others require techniques of broader scope. That is the case with the denial of service threat, which may vary in its sophistication and can cause availability issues. At the protocol level, DIoTComm mitigates this threat by using nonces, which enable agents to efficiently discard repeated messages. Additional techniques to counter denial of service threats can be employed, however they are out of the scope of our proposal.

Another threat that does not depend exclusively on the secure communications layer is elevation of privilege, in which a party gains access to unauthorized operations. For example, an attacker could try to update a DID Document of another agent in the blockchain. An identity-based access control scheme enables the Identity Blockchain to mitigate this threat: every time a DID Document is registered or updated, it must match the identity of the request initiator. On the other hand, elevation of privilege between IoT agents may follow different rules depending on each use case, and thus are considered out of scope. Nevertheless, it is recommended that appropriate access control policies are configured and enforced during agent interactions.

In addition to the mentioned threats, DIoTComm also prevents replay attacks by using a sliding window with previously received nonces. Thus, we conclude that our system is prepared to resist against common threats in IoT environments.

Table 14: Comparison of DID Swarm with existing DID Methods.

Prefix	**Focus on IoT?**	**DID serialization**	**DID size**	**DDo ser.**	**DDo size**
did:sw:	Yes	binary	**19**	CBOR-DI	**128**
did:sov:	No	text	30	JSON	499
did:ockam:	Yes	text	39	JSON	779
did:io:	Yes	text	49	JSON	1112
did:v1:	No	text	54	JSON	1182
did:tangle:	Yes	text	92	JSON	853

Source: Fedrecheski et al. (2021).

5.2.2 DID and DDo sizes

In this section we measure the size of our proposed DID and DID Document, and compare it to five existing DID methods, as shown in Table 14. The methods `did:ockam` (OCKAM...,), `did:io` (IOTEX...,), and `did:tangle` (TANGLEID, 2019) were selected since they are specifically tailored for IoT applications. The other two, `did:sov` (SOVRIN...,) and `did:v1` (VERES...,), were selected as references since they provide both a complete specification and a mature open source implementation.

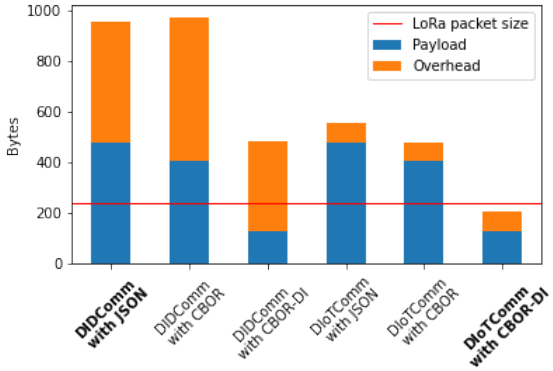
As shown in Table 14, our proposed method, `did:sw`, has the smaller DID size, occupying only 19 bytes when using the binary serialization described in Section 3.1.4.2. Among the methods tailored for IoT, `did:ockam` has the second smaller DID, requiring 39 bytes. In order to compare DID Documents, we built documents with equivalent configurations, i.e., having two public keys and, when applicable, one service endpoint[2]. The measured DDos were extracted from the specification of each DID method, and a second public key was added when the example contained only one. As shown in Table 14, the `did:sw` method has the smallest DDo size, which represents a reduction of almost 75% when comparing to the second smallest DDo. These results confirm that the methods proposed in sections 3.1.4.1 and 3.1.4.2 indeed reduced DID and DDo sizes when comparing to previous works.

5.2.3 Secure Envelope Overhead

In this section, we compare the overhead of using DIDComm and DIoTComm to protect DID Documents and application messages for transmission in constrained networks.

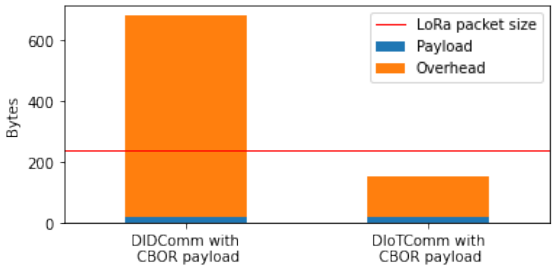
We start by measuring the size of a signed message containing a DID Document, using both DIDComm and DIoTComm. In doing this, we use different DDo serializations:

[2]Some DID methods do not use endpoints.



Source: Fedrecheski et al. (2022).

Figure 16: Size and overhead for DID Documents sent within a signed message (step 2 of Figure 6).



Source: Fedrecheski et al. (2022).

Figure 17: Size and overhead for regular messages sent within a signed and encrypted message (step 3 of Figure 6).

JSON, CBOR, and CBOR-DI. Figure 16 shows the results. The three leftmost bars use the DIDComm message envelope[3], while the three rightmost bars use DIoTComm. We also highlight the threshold for transmission of LoRa packets when considering Data Rate 6, which allows packets of up to 242 bytes. Note that, although the overhead is significantly reduced when adopting DIoTComm, the only scenario in which a DDo can be transmitted over a LoRa network is when DIoTComm and CBOR-DI are combined.

In the next chart, shown in Figure 17, we consider a 21-bytes application message serialized in CBOR and sent over DIDComm and then over DIoTComm. Differently from the previous chart, this message is not only signed, but also encrypted, i.e. the messages are nested with two layers of headers. This confers the DIoTComm version an even higher compression rate, with an overhead at least five times smaller.

5.2.4 Runtime

In this section we measure the runtime of our proposed operations to enable self-sovereign identification of IoT agents. We start by measuring the time needed to encode a Swarm DID Document in three different formats: JSON, CBOR, and CBOR-DI. Next, we measure the time to generate non-repudiable DIoTComm messages, and compare to the time needed for DIDComm messages. The tests were conducted on two platforms. The first, nicknamed "Laptop", is a Quad-core Intel i7 1.8 GHz laptop with 20 GB of RAM. The second, nicknamed "SBC", is a Quad-core ARM 1.3 GHz Single Board Computer with 2 GB RAM. The software used was the SwarmLib, which uses the Python programming

[3]The size of DIDComm with CBOR is larger than DIDComm with JSON due to overhead of Base64 encoding.

Table 15: DID Document (de)serialization mean time (ms).

Operation	**Device**	**JSON**	**CBOR**	**CBOR-DI**
Serialization	Laptop	0.3 (±0.0)	0.3 (±0.0)	0.2 (±0.0)
	SBC	4.9 (±0.1)	4.9 (±0.1)	2.3 (±0.1)
Deserialization	Laptop	0.4 (±0.0)	0.4 (±0.0)	0.1 (±0.0)
	SBC	6.1 (±0.5)	6.1 (±0.1)	1.5 (±0.1)

Source: Fedrecheski et al. (2022).

language.

Runtime to (de)serialize DID Document In this test we seek to evaluate whether there are performance differences among three distinct methods to serialize and deserialize a DID Document: the existing JSON and CBOR approaches, and our proposed CBOR-DI. We used our Python implementation of the SwarmLib to generate an in-memory DID Document containing two public keys and one service endpoint, similar to the one shown in Figure 7(a). We then measured the time to (de)serialize it using the three methods mentioned.

As shown in Table 15, CBOR-DI takes slightly less time than JSON and CBOR to serialize on the Laptop, and approximately half of the time on the SBC. Regarding deserialization, CBOR-DI uses one third of the time when compared to JSON and CBOR on both platforms. We believe that the speedup in processing is mostly due to the smaller amount of data that goes into CBOR-DI.

Runtime to (un)protect messages In this test we seek to understand what are the performance impacts of our proposed DIoTComm versus the existing DIDComm protocol. To do so, we created an 100-byte message and then measured the time necessary to protected it, i.e., sign, encrypt, and serialize to the appropriate format. We also did the inverse operation, that is to deserialize and decrypt the message, and verify its signature. Each operation was performed 100 times, from which we extracted the mean and standard deviation. The encryption, decryption, serialization, and deserialization functions were implemented as part as the SwarmLib.

Results, shown in Table 16, show that DIoTComm is slightly faster than DIDComm in both platforms tested, i.e., a laptop and a single-board computer. Since the cryptographic algorithms are very similar, the differences mainly amounts to serialization, which requires slightly less data processing in the case of DIoTComm.

Table 16: Runtime to (un)protect 100-byte payload (ms).

Operation	**Device**	**DIDComm**	**DIoTComm**
Encryption and	Laptop	0.7 (±0.1)	0.6 (±0.0)
Serialization	SBC	9.3 (±0.1)	7.0 (±0.1)
Decryption and	Laptop	0.6 (±0.0)	0.5 (±0.0)
Deserialization	SBC	7.9 (±0.1)	6.5 (±0.1)

Source: Fedrecheski et al. (2022).

5.2.5 Discussion

This subsection present topics that capture the main impacts and limitations of our work.

Security An important question to address is whether our proposal could affect security or introduce new vulnerabilities. The operations applied to reduce DDo size, i.e., the Swarm DID method and CBOR-DI, have no impact on security, since only the data serialization is changed. In other words, the data model is different, but the cryptographic keys and algorithms used are standardized (SCHAAD, 2017). Moreover, DDos are always protected when transmitted using the DIoTComm protocol. We have also shown that that DIoTComm prevent the threats of spoofing, tampering, repudiation, and information disclosure.

Differences to DIDComm Although DIoTComm can be seen as a binary version of DIDComm, some simplifications have been made to our protocol. First, it does not incorporate routing, since we assume that it could be implemented on top of it, for example by using CoAP messages with proxy options as the DIoTComm payload. The second main difference is that, since DIoTComm uses static keys to derive a symmetric key, it does not provide forward secrecy. This approach is required to avoid the transmission of an ephemeral public key, which would increase payload size by more than 32 bytes. Future work should address a solution that considers forward secrecy into DIoTComm.

Asynchronous messages It is worth to highlight that, just as DIDComm, DIoTComm works asynchronously, e.g. similar to email messages. This approach enables communication even when one of the end parties is temporarily offline (provided it can be buffered in-transit), and it also does not require a session establishment phase. On the other hand, this approach requires key agreement to be performed before every message, so it may be more processing intensive than session-based protocols. A possible solu-

tion, which we leave for future works, would be to support the option to choose among synchronous and asynchronous messaging modes.

Size of identity metadata We achieved significant reduction of DIDs and DDo sizes when comparing to existing solutions. Note that this reduction did not require information loss nor the use of pre-configured static parameters. Moreover, it relied on standardized serialization guidelines provided by the COSE specification, which facilitates interoperability (SCHAAD, 2017). Overall, it represents a significant improvement to allow use of self-sovereign identification in constrained IoT networks.

Size of protected messages By using a binary serialization based on the COSE standard (SCHAAD, 2017), we were able to greatly reduce the size of security overhead for signed and encrypted messages, when compared to DIDComm, the current solution for message protection in SSI communication.

Performance impacts A valid question regarding the improvements on message sizes concerns the impacts on performance. In our evaluation, we demonstrated that our message size optimizations did not cause performance penalties. On the contrary, since the messages demand less bytes to be processed, serialization and deserialization becomes faster than the original alternatives.

Self-sovereign identity in constrained networks A fundamental question with respect to adoption of SSI in IoT environments consists in how to support it in constrained environments. By reducing the overhead involved in transmission of identity metadata and message protection, we have shown that IoT devices could use self-sovereign identity even when their communication link is constrained to only a few hundred bytes per message.

5.3 Evaluation of a complete interaction

Whereas previous sections evaluated specific parts of the proposed system, such as SmartABAC and DID Swarm, this section evaluates a complete interaction between two Swarm agents running in real IoT hardware.

5.3.1 Defining and Preparing a Use Case

The evaluation is focused on a specific interaction: Bob tries to toggle a Lamp (Figure 18). Each agent embeds all of its identity information, as described in the Agent Data Model (Section 3.2.1). This includes their DIDs, credentials, and ABAC policies. In the case of the Lamp, one of its policies states that "friends of the Lamp's owner are allowed to toggle the Lamp".

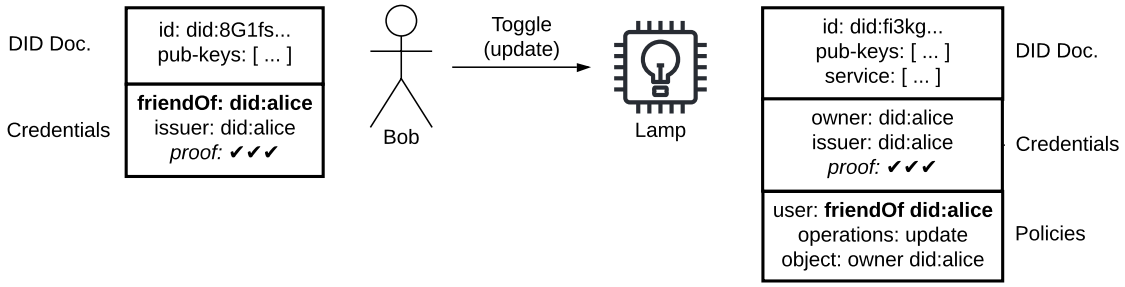


Figure 18: The agent "Bob" interacts with a "Lamp" agent.

The agents were setup in a laboratory environment, using the Labrador Single Board Computer (SBC), as it has both hardware and software support for running the Swarm Broker and the SwarmLib. The Labrador has the following CPU and memory: ARM Cortex A9 quad-core 1.3GHz; 2 GB of RAM. An LED connected to the Labrador GPIO was used to simulate the lamp.

The implementation of the use case involves the creation of five agents across three devices, being two single board computers (Labrador) and one Laptop. The first Labrador runs one broker and the bob agent; the second one runs another broker and the lamp agent; finally, the laptop runs the alice agent. The setup is shown in Figure 19.

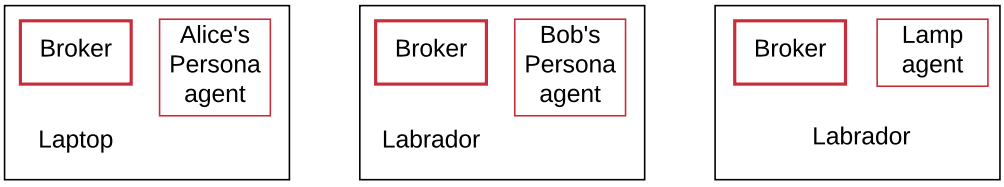


Figure 19: The use case devices and agents.

To prepare the agents for the use case, we use the swarm_manager tool to execute the following steps:

- Create five agents: broker_lab1, bob, broker_lab2, lamp, broker_laptop and alice.

- Store the DID Document of the DID Chain.
- Register the DDos of the new agents at the DID Chain.
- Since the lamp is a provider, we configure it with its supported operations: update (on/off) and read current status.
- Emit credentials to the participants, following the attributes assignment defined in Table 7. For example, Bob has attributes such as "friendOf: alice", and Lamp has attributes "purpose: lighting" and "owner: alice".
- Create access control policies, according to the definitions in Table 8. Thus, the "Lamp" got one policy stating that it could be used by friends of Alice.

5.3.2 Methods

The evaluation is focused on packet sizes and latency over the setup and use steps. To briefly recall, in the setup step, the initiator asks the responder for an authorization token, which is generated in case the attributes of the initiator match the policies of the responder; next, the initiator uses the token to interact directly with the responder (see Chapter 3). We executed the interaction using Bob and Lamp as initiator and responder, respectively, and collected metrics regarding packet sizes and count for each request/response. Next, we repeated the interaction 50 times to measure the time spent at each phase of the interaction.

The packet size and count were obtained using Wireshark, a tool for network analysis. All communications taken during the setup and use steps were measured, which implies recording the packets sent not only, e.g., between Bob and Broker, but also the packets sent between the agents and the Identity Blockchain.

To measure latency, the swarm_lib and the Broker were modified with the addition of a flag-enabled metrics module that records how much time each function took to execute. The functions selected for measurement include the time to close and open a message (using DIoTComm), resolve a DID Document, verify credentials, evaluate SmartABAC policies, as well as build and verify the access token. After the data is collected, the participants would send their collected logs to a server which would aggregate them into a single file. The results were then analysed using the Jupyter Notebook data analysis tool, and graphics were generated using the matplotlib Python library.

During preliminary tests we identified latency bottlenecks that could be fixed with the use of a cache, so we added a bounded cache and repeated the latency measurements.

5.3.3 Results

Results of measuring packet sizes are shown in Table 17, and latency is shown in Figure 20.

Table 17: Packet sizes in the "Bob toggles Lamp" use case.

Step	**Setup**						**Use**			
Network call	**Bob to Chain**		**Bob to Broker**		**Broker to Chain (3x, avg)**		**Bob to Lamp**		**Lamp to Chain**	
	Req	**Res**	**Req**	**Res**	**Req**	**Res**	**Req**	**Res**	**Req**	**Res**
Packet sizes (bytes)	269	380	928	735	202	725	454	228	269	300
Total bytes	5093						1251			
Packet Count	10						4			
Average pkt size	509,3						312,75			

Packet sizes are divided by interaction step and sub-divided into specific network calls, the latter being needed since interactions usually comprise networks calls between two agents plus the identity blockchain. Note that some measurements happen more than once, e.g., the Broker resolves three DIDs at the Blockchain, for which the average size is shown. Looking at individual packet sizes, it is clear that the most expensive network call is between Bob and the Lamp's Broker, the reason being that Bob must present all of its credentials so that the Broker can use them to evaluate the ABAC policies. Nevertheless, it is also worth noting that the Broker has to consult the blockchain three times to resolve the DIDs of the initiator and of the credential issuers. The latter may be alleviated, however, by the use of caching mechanisms.

In general, the use step requires less network resources when compared to the interaction setup. The total amount of bytes sent in the use step was about 25% of the amount sent in the setup step; the former also required less than half the number of packets. When aggregating these numbers, we find that, on average, the use step required 38.7% smaller packets when compared to the setup step, which is expected since the setup steps involve the transference of all the public attributes of the initiator over the network.

Latency results are summarized in Figure 20, where items a, b, c, and d show the average time taken at each step, with a per-function-per-request granularity. For example, the leftmost bar in item 20a shows that the first request from Bob to Broker took almost 500 milliseconds, of which around 200 ms account for Network and Broker processing combined; similarly, item 20b shows that most requests were served in less than 70 ms, with the costliest operations being to DIoTComm encryption and decryption (open and

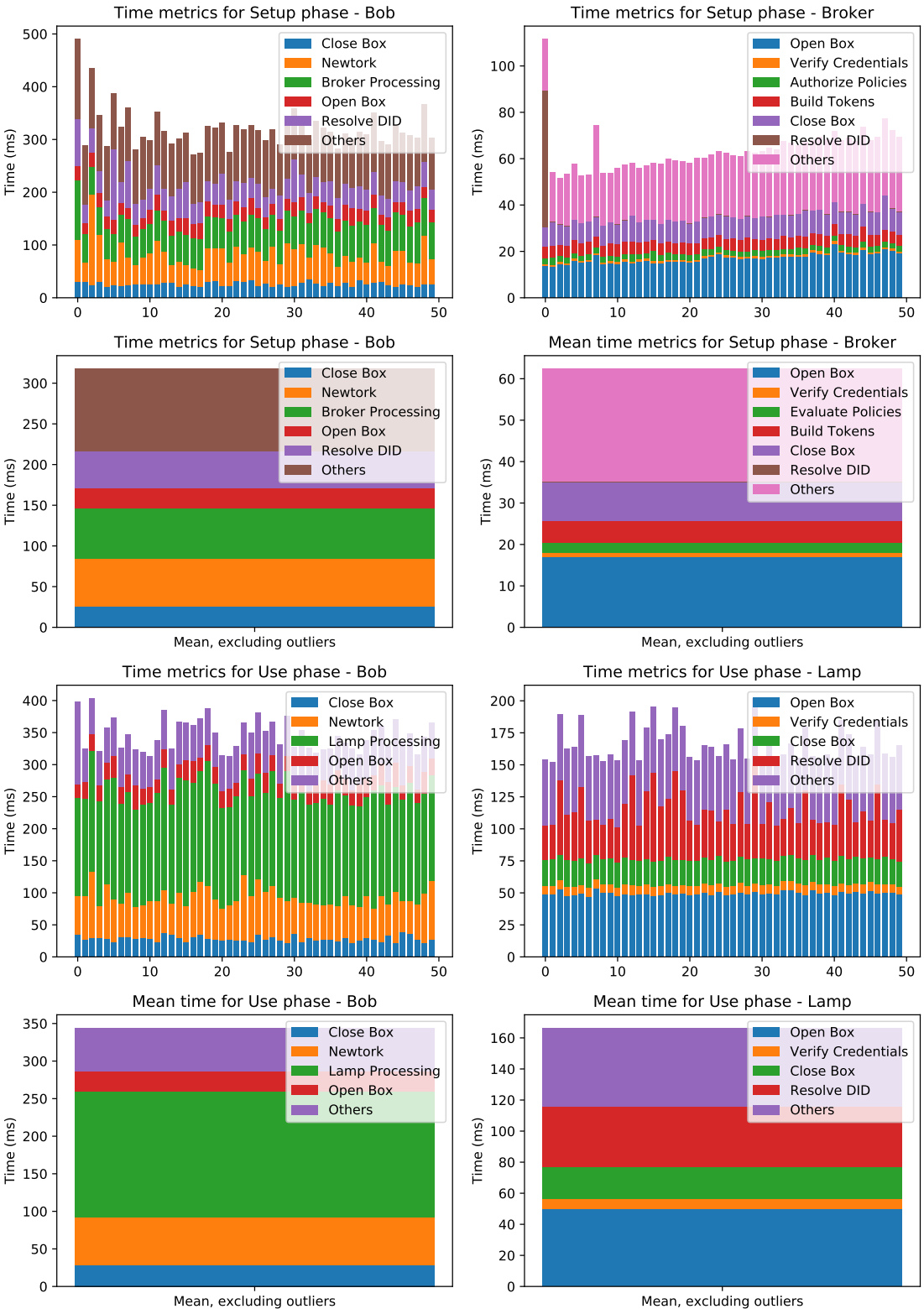


Figure 20: Results.

close, respectively).

Overall, interactions take around 300 milliseconds on average, with the most expensive operations depending on the specific step and actor involved.

5.4 Considerations about constrained devices

The implementation of the proposed framework, with the exception of the SmartABAC model, used high-level and garbage-collected languages, i.e. Python and Elixir. On one hand, this approach enabled faster prototyping of the framework and full integration with the existing SwarmOS infrastructure, i.e. the Broker and the SwarmLib. On the other hand, it prevents tests on many smaller and more constrained IoT devices.

The latest version of the terminology for constrained node-networks (BORMANN et al., 2022) defines a set of classes for IoT devices, ranging from Class 0 (the most constrained, with 10 KiB of RAM and 100 KiB of flash memory) to Class 19 (the least constrained, such as a server). The full version of the SwarmOS was tested on a Single Board Computer, which corresponds to Class 15. The SmartABAC model was tested on a microcontroller that corresponds to Class 3, i.e, about 100 KiB of RAM and up to 1000 KiB of ROM.

Nevertheless, it is possible to analyse the suitability of the framework to devices of Class 3 or less, by obtaining estimates about the execution of certain operations. The most CPU-intensive operations in the framework are those that involve performing cryptographic algorithms, such as signing, deriving keys, and encrypting data. According to Lachner & Dustdar (2019), who evaluated several cryptographic algorithms on typical off-the-shell IoT hardware, the most expensive algorithms involve signature verification and generation (ECDSA), and key derivation (ECDH). The authors provide measurements on an 240 MHz Xtensa microprocessor, showing that the execution times for these operations are, respectively, 78, 50, and 26 milliseconds. The signatures were performed on a payload of 700 bytes (similar to the size of an interaction setup payload, as seen in Table 17). The authors also show that operations involving data encryption (AES) and hashing (SHA256) are much faster, in the order of 1 MB/s. Detailed execution times are shown in Table 18, which also correlates the algorithms to specific parts of our proposed framework.

It is then possible to estimate the minimum amount of time needed to execute the steps of the framework, considering its cryptographic operations, as shown in Table 19. Note that the execution times are adjusted to how many times each operation is executed. For example in the setup step, we assume that the responder has to verify three credentials. Since this means resolving the DID Documents of three issuers, it implies in performing DIoTComm three times, which is then added to the DIoTComm interaction with the initiator, completing four DIoTComm interactions. Thus, the total time spent deriving

Table 18: Execution time for main cryptographic operations used by the proposed framework, considering 240 MHz IoT CPU. All values based on Lachner & Dustdar (2019), except the last line which was measured in this work.

Technology	**Operation**	**Algorithm**	**Time (ms)**
CapBAC DIoTComm Verifiable Credentials	Verify token Verify message Verify credential	EDDSA verification	78
Verifiable Credentials DIoTComm CapBAC	Sign credential Sign message Sign token	EDDSA signing	50
DIoTComm	Derive keys	EDDH	26
DIoTComm	Derive keys	SHA256 (HKDF)	0,001
Decentralized Identifiers	Generate DDo	Ed25519 key generation	49
DIoTComm	Encrypt / decrypt	AES 128 CTR	0,0003
SmartABAC	Evaluate policies	SmartABAC authorize	1

keys is $26 * 4 = 104$ milliseconds. Similar reasoning applies to all computed times in Table 19.

Table 19: Estimated time (ms) to execute each step on a constrained IoT node.

Setup				**Execution**			
Initiator	**Time**	**Responder**	**Time**	**Initiator**	**Time**	**Responder**	**Time**
Verify message	78,0	Verify message	312,0	Verify message	78,0	Verify token	78,0
Sign message	50,0	Verify credential	234,0	Sign message	50,0	Verify message	78,0
Derive keys	26,0	Sign message	200,0	Derive keys	26,0	Sign message	50,0
Encrypt / decrypt msg	0,0	Derive keys	104,0	Encrypt / decrypt msg	0,0	Derive keys	26,0
		Generate token	50,0			Encrypt / decrypt msg	0,0
		Encrypt / decrypt msg	0,0				
		Evaluate policies	1,0				
	154,0		901,0		154,0		232,0

It is clear, then, that the setup step takes at least 1 second, and the execute step takes close to 400 milliseconds. Taken alone these times may be reasonable for certain applications, however they could be improved, especially when considering energy consumption and availability issues. The easiest part to improve is to simply remove message signature from DIoTComm, since AES-CCM already provides authentication (signatures could be used only in applications that need non-repudiability), which would save $78 + 50 = 128$ milliseconds on all interactions. Another possible improvement is to use a MAC-based token, instead of one based on digital signatures. Since MACs use hashing and symmetric cryptography, token creation and verification time would decrease to just a few microseconds. This would require an architectural change, however, since the responder of the execution step (a regular agent) would have to somehow obtain the MAC secret key from

the responder of the setup step (the Broker).

Yet another possible improvement would be to adapt DIoTComm to use a three-way handshake, which would reduce key derivation step to only once per agent. One alternative would be to integrate it with Ephemeral Diffie-Hellman Over COSE (EDHOC) (SELANDER; MATTSSON; PALOMBINI, 2022), which provides a secure application-layer key establishment mechanism. This approach would also bring the benefit of providing forward-secrecy to all interactions. On the other hand, it would spend some time during the key establishment phase, which in turn might be problematic in highly constrained networks, such as LoRaWAN, which has a severely restricted downlink channel.

5.5 Concluding remarks

When developing the framework, two key questions were whether it could actually protect Swarm agents, and whether it could run in constrained devices. To answer these questions, this chapter presented evaluations of each of the framework layers.

The authorization layer was first evaluated to demonstrate its capability to protect Swarm agents in a smart home use case. Next, its capabilities were compared to the existing models in the literature, and its performance was assessed in heterogeneous computing platforms. This enabled us to confirm that the hybrid approach used in SmartABAC could indeed provide expressive access policies while offering a negligible performance overhead, even when run on highly constrained devices.

For the authentication and confidentiality layers, we first analysed DIoTComm under STRIDE threat model, showing that they employ the needed primitives to protect agent interactions. For instance, we pointed how identity spoofing is prevented by using authenticated encryption at the protocol level, and how confidentiality is achieved by encrypting messages between IoT agents using AES-CCM. We further analysed DIoTComm in terms of its overhead, and showed that it can work even on bandwidth-constrained networks. Nevertheless, there is still room for improvement on the messaging protocol overhead, which we leave for future works.

Finally, the identity layer was evaluated with respect to raw size and serialization time. The size evaluation demonstrated that the proposed DID Swarm method is leaner than pre-existing DID methods, which justifies its proposal as a new method. In addition, the newly proposed serialization mechanism, CBOR-DI, was shown to be faster than regular serialization methods (JSON and CBOR), taking at most 2.3 milliseconds on a

single-board computer.

In conclusion, the evaluation of the authorization, authentication, confidentiality, and identity layers demonstrated that the proposed framework can (1) protect Swarm agents using various techniques applicable at each layer, and (2) is sufficiently lightweight to run at least on single-board computers.

6 CONCLUSION

This dissertation investigated and developed a mechanism to enable secure interactions in IoT Swarms. One key challenge was that many existing security approaches for the Internet and the IoT rely on centralized architectures, such as authorization and naming servers. Thus, we investigated whether we could provide alternatives that were both secure and could minimize dependency on third parties, i.e., make devices autonomous with respect to their own security.

The chosen approach was to specify a full-stack framework for security in IoT Swarms, following two main principles. First, to ensure autonomy from centralized parties, Swarm agents should be self-sovereign, which means that they should handle as much as possible of the security mechanisms by themselves, going from identification up to the authorization layer. The second principle was to use attribute-based access control, which enable creation of flexible and high-level policies necessary to specify authorization rules considering the dynamic and cyber-physical nature of Swarm systems.

The framework was presented in layers, following a top-down approach. First, the authorization layer was described, adapting best-practices from ABAC guidelines to create a decentralized ABAC architecture. The layer also included a hybrid approach for building access policies, by means of the SmartABAC model, which enables embedded evaluation of policies even on highly constrained devices.

Next, the authentication layer was presented, which included both message authentication via the DIoTComm protocol, and agent authentication by sharing of Verifiable Credentials. The DIoTComm protocol was also instrumental for the confidentiality layer, responsible for protecting interactions from eavesdroppers. Since it works at the application level, DIoTComm also work across heterogeneous networks, a common scenario in both traditional IoT and Swarms.

Finally, a layer for lean and self-sovereign identification of IoT agents was proposed. It consists of the Swarm DID Method, a concrete instantiation of the Decentralized Iden-

tifiers specification, and a new serialization mechanism that enables use of DIDs on constrained networks. A mock Identity Blockchain was created to enable implementation and tests of real Swarm agents.

The implementation consisted in major updates to the Swarm Broker and the SwarmLib, as well as the development of new libraries and updates to existing Swarm Agents. Among the libraries created from scratch to support this work, we highlight an Elixir and a C implementation of the SmartABAC model. A new agent was also created to provide the Identity Blockchain functionality.

The evaluation first measured different aspects of SmartABAC, DIoTComm, and the Swarm DID Method, then assessed the fully integrated system on a use case on real hardware. One key implication of the SmartABAC evaluation was that it refuted the previously held notion of ABAC being inherently unsuitable for constrained devices (RAVIDAS et al., 2019), as we were able to run it on a 35MHz ARM microcontroller with negligible latency.

The evaluation of DIoTComm has shown that it addresses the STRIDE threat model by re-using well-knonwn algorihtms for ciphering, authenticating, and signing messages. It also demonstrated that the effort to reduce its size enabled packets to fit in constrained networks. The size reduction is, in fact, best achieved when coupled with another proposal of this work, the Swarm DID method, which provides lean DID Documents with a new approach for serialization, named CBOR DID Documents for IoT (CBOR-DI). Compared to prior methods, it achieves documents more than four times smaller.

Finally, the integrated system was evaluated on a use case with real IoT hardware consisting of two single-board computers. Metrics show that the average packet size is less than 500 bytes per request, with latency on the range of 300 to 350 milliseconds.

In conclusion: This dissertation achieved the goal of researching a framework for protecting interactions between agents in IoT Swarms. It demonstrated that (1) the self-sovereign approach is possible, not only to identify agents, but to fully describe their operation, up to the authorization layer; and that (2) attribute-based access control can be leveraged to offer expressive policies for dynamic IoT environments while imposing negligible performance and storage overhead.

Regarding future work, a number of paths can be explored. The Swarm DID Method can be ported to constrained IoT hardware, and tested on real networks. The DIoTComm protocol can be improved considering emerging standards that enable smaller security

overhead. And finally, the integrated framework can be ported to and tested on more constrained devices, as well as deployed on larger networks to prove its scalability.

REFERENCES

AFZAL, S. et al. Analysis of web-based iot through heterogeneous networks: Swarm computing over lorawan. *Sensors*, MDPI, v. 22, n. 2, p. 664, 2022.

ALKHRESHEH, A.; ELGAZZAR, K.; HASSANEIN, H. S. Daciot: Dynamic access control framework for iot deployments. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 12, p. 11401–11419, 2020.

ALLEN, C. *The path for self-sovereign identity*. 2016. ⟨http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html⟩. Accessed: 2020-02-13.

ALLIANCE, L. *LoRaWAN®Specification v1.1*. 2017. Disponível em: ⟨https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/⟩.

ALNEFAIE, S.; CHERIF, A.; ALSHEHRI, S. Towards a distributed access control model for iot in healthcare. In: IEEE. *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*. [S.l.], 2019. p. 1–6.

ANTONINO, A. et al. *A privacy-preserving approach to grid balancing using scheduled electric vehicle charging*. Tese (Doutorado) — Aalto University, 2019.

ASAMOAH, K. O. et al. Zero-chain: A blockchain-based identity for digital city operating system. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 10, p. 10336–10346, 2020.

BARABAS, C.; NARULA, N.; ZUCKERMAN, E. Defending internet freedom through decentralization: back to the future. *The Center for Civic Media & The Digital Currency Initiative MIT Media Lab*, 2017.

BERNSTEIN, D. J. et al. High-speed high-security signatures. In: SPRINGER. *International Workshop on Cryptographic Hardware and Embedded Systems*. [S.l.], 2011. p. 124–142.

BEZAWADA, B.; HAEFNER, K.; RAY, I. Securing home iot environments with attribute-based access control. In: ACM. *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*. [S.l.], 2018. p. 43–53.

BHATT, S.; PATWA, F.; SANDHU, R. Abac with group attributes and attribute hierarchies utilizing the policy machine. In: ACM. *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*. [S.l.], 2017. p. 17–28.

BIASE, C. L. D. et al. Swarm minimum broker: an approach to deal with the internet of things heterogeneity. In: IEEE. *2018 Global Internet of Things Summit (GIoTS)*. [S.l.], 2018. p. 1–6.

BIASE, L. C. D. et al. Swarm economy: a model for transactions in a distributed and organic iot platform. *IEEE Internet of Things Journal*, IEEE, 2018.

BIASE, L. C. D. et al. Swarm assistant: an intelligent personal assistant for the swarm. In: IEEE. *2019 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2019. p. 1–2.

BIASE, L. C. D. et al. Surveillance swarm: Gathering resources for finding targets in a distributed global network of smart devices. In: IEEE. *2020 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2020. p. 1–2.

BISWAS, P. *Enumerated Authorization Policy ABAC Models: Expressive Power and Enforcement*. Tese (Doutorado) — The University of Texas at San Antonio, 2017.

BISWAS, P.; SANDHU, R.; KRISHNAN, R. A comparison of logical-formula and enumerated authorization policy abac models. In: SPRINGER. *IFIP Annual Conference on Data and Applications Security and Privacy*. [S.l.], 2016. p. 122–129.

BISWAS, P.; SANDHU, R.; KRISHNAN, R. Label-based access control: An abac model with enumerated authorization policy. In: ACM. *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*. [S.l.], 2016. p. 1–12.

BORMANN, C.; ERSUE, M.; KERANEN, A. *Terminology for Constrained-Node Networks*. [S.l.], 2014. Disponível em: ⟨https://tools.ietf.org/html/rfc7228⟩.

BORMANN, C. et al. *Terminology for Constrained-Node Networks*. [S.l.], 2022. Disponível em: ⟨https://www.ietf.org/id/draft-ietf-lwig-7228bis-00.html⟩.

BORMANN, C.; HOFFMAN, P. E. *Concise Binary Object Representation (CBOR)*. RFC Editor, 2020. RFC 8949. (Request for Comments, 8949). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8949.txt⟩.

CALCINA-CCORI, P. et al. Swarmgen: a framework for automatic generation of semantic services in an iot network. In: IEEE. *2020 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2020. p. 1–5.

CALCINA-CCORI, P. et al. Agile servient integration with the swarm: Automatic code generation for nodes in the internet of things. In: *Proceedings of the International Conference on Future Networks and Distributed Systems*. [S.l.: s.n.], 2017. p. 1–6.

CALCINA-CCORI, P. C. et al. Enabling semantic discovery in the swarm. *IEEE Transactions on Consumer Electronics*, IEEE, v. 65, n. 1, p. 57–63, 2018.

CALCINA-CCORI, P. C. et al. Describing services geolocation in iot context. In: IEEE. *2019 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2019. p. 1–2.

CALCINA-CCORI, P. C. et al. Location-aware discovery of services in the iot: a swarm approach. In: IEEE. *2019 Global IoT Summit (GIoTS)*. [S.l.], 2019. p. 1–6.

CALLAS, J. et al. *OpenPGP Message Format*. [S.l.], 2007. ⟨http://www.rfc-editor.org/rfc/rfc4880.txt⟩. Disponível em: ⟨http://www.rfc-editor.org/rfc/rfc4880.txt⟩.

CHEHAB, M.; MOURAD, A. Lp-sba-xacml: Lightweight semantics based scheme enabling intelligent behavior-aware privacy for iot. *IEEE Transactions on Dependable and Secure Computing*, IEEE, 2020.

COLOMBO, P.; FERRARI, E.; TÜMER, E. D. Regulating data sharing across mqtt environments. *Journal of Network and Computer Applications*, Elsevier, v. 174, p. 102907, 2021.

COOPER, D. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. [S.l.], 2008. ⟨http://www.rfc-editor.org/rfc/rfc5280.txt⟩. Disponível em: ⟨http://www.rfc-editor.org/rfc/rfc5280.txt⟩.

COSTA, L. C. et al. Swarm os control plane: an architecture proposal for heterogeneous and organic networks. *IEEE Transactions on Consumer Electronics*, IEEE, v. 61, n. 4, p. 454–462, 2015.

DAEMEN, J.; RIJMEN, V. Reijndael: The advanced encryption standard. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, Miller Freeman Inc., v. 26, n. 3, p. 137–139, 2001.

DENNIS, J. B.; HORN, E. C. V. Programming semantics for multiprogrammed computations. *Communications of the ACM*, ACM New York, NY, USA, v. 9, n. 3, p. 143–155, 1966.

DIN, I. U. et al. Lighttrust: lightweight trust management for edge devices in industrial internet of things. *IEEE Internet of Things Journal*, IEEE, 2021.

DRAMÉ-MAIGNÉ, S.; LAURENT, M.; CASTILLO, L. Distributed access control solution for the iot based on multi-endorsed attributes and smart contracts. In: IEEE. *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. [S.l.], 2019. p. 1582–1587.

ESPOSITO, C.; FICCO, M.; GUPTA, B. B. Blockchain-based authentication and authorization for smart city applications. *Information Processing & Management*, Elsevier, v. 58, n. 2, p. 102468, 2021.

ESQUIAGOLA, J. et al. Performance testing of an internet of things platform. In: *IoTBDS*. [S.l.: s.n.], 2017. p. 309–314.

FARRELL, S.; HOUSLEY, R.; TURNER, S. *An Internet Attribute Certificate Profile for Authorization*. [S.l.], 2010.

FEDRECHESKI, G. et al. Attribute-based access control for the swarm with distributed policy management. *IEEE Transactions on Consumer Electronics*, IEEE, v. 65, n. 1, p. 90–98, 2019.

FEDRECHESKI, G. et al. SmartABAC: enabling constrained iot devices to make complex policy-based access control decisions. *IEEE Internet of Things Journal*, IEEE, 2021.

FEDRECHESKI, G. et al. A low-overhead approach for self-sovereign identity in iot. In: IEEE. *2022 Global Internet of Things Summit (GIoTS)*. [S.l.], 2022. p. (in press).

FEDRECHESKI, G.; COSTA, L. C.; ZUFFO, M. K. Elixir programming language evaluation for iot. In: IEEE. *Consumer Electronics (ISCE), 2016 IEEE International Symposium on*. [S.l.], 2016. p. 105–106.

FEDRECHESKI, G. et al. Self-sovereign identity for iot environments: a perspective. In: IEEE. *2020 Global Internet of Things Summit (GIoTS)*. [S.l.], 2020. p. 1–6.

FERDOUS, M. S.; CHOWDHURY, F.; ALASSAFI, M. O. In search of self-sovereign identity leveraging blockchain technology. *IEEE Access*, IEEE, v. 7, p. 103059–103079, 2019.

FERNANDO, N. et al. Opportunistic fog for iot: challenges and opportunities. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 5, p. 8897–8910, 2019.

FERRAIOLO, D.; ATLURI, V.; GAVRILA, S. The policy machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, Elsevier, v. 57, n. 4, p. 412–424, 2011.

FOTIOU, N. et al. Enabling opportunistic users in multi-tenant iot systems using decentralized identifiers and permissioned blockchains. In: *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. [S.l.: s.n.], 2019. p. 22–23.

GABILLON, A.; GALLIER, R.; BRUNO, E. Access controls for iot networks. *SN Computer Science*, Springer, v. 1, n. 1, p. 24, 2020.

GHOSH, N. et al. Softauthz: A context-aware, behavior-based authorization framework for home iot. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 6, p. 10773–10785, 2019.

GILANI, K. et al. A survey on blockchain-based identity management and decentralized privacy for personal data. In: IEEE. *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. [S.l.], 2020. p. 97–101.

GRANDE, E.; BELTRÁN, M. Edge-centric delegation of authorization for constrained devices in the internet of things. *Computer Communications*, Elsevier, v. 160, p. 464–474, 2020.

GUPTA, M. et al. An attribute-based access control for cloud enabled industrial smart vehicles. *IEEE Transactions on Industrial Informatics*, IEEE, v. 17, n. 6, p. 4288–4297, 2021.

HARDMAN, D. *DIDComm Messaging*. [S.l.], 2020. Https://identity.foundation/didcomm-messaging/spec/.

HERNAN, S. et al. *Uncover Security Design Flaws Using The STRIDE Approach*. 2019. Disponível em: ⟨https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach⟩.

HERNÁNDEZ-RAMOS, J. L. et al. Dcapbac: embedding authorization logic into smart things through ecc optimizations. *International Journal of Computer Mathematics*, Taylor & Francis, v. 93, n. 2, p. 345–366, 2016.

HU, V. C. et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, Citeseer, v. 800, n. 162, 2013.

IOTEX DID Method Specification. Disponível em: ⟨https://github.com/iotexproject/iotex-did/blob/master/README.md⟩.

JABAL, A. A. et al. Polisma-a framework for learning attribute-based access control policies. In: SPRINGER. *European Symposium on Research in Computer Security*. [S.l.], 2020. p. 523–544.

JAEGER, T.; ZHANG, X.; EDWARDS, A. Policy management using access control spaces. *ACM Transactions on Information and System Security (TISSEC)*, ACM, v. 6, n. 3, p. 327–364, 2003.

JONES, M. *JSON Web Algorithms (JWA)*. RFC Editor, 2015. RFC 7518. (Request for Comments, 7518). Disponível em: ⟨https://www.rfc-editor.org/info/rfc7518⟩.

JONES, M. et al. *CBOR Web Token (CWT)*. RFC Editor, 2018. RFC 8392. (Request for Comments, 8392). Disponível em: ⟨https://www.rfc-editor.org/info/rfc8392⟩.

JOSEFSSON, S.; LIUSVAARA, I. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC Editor, 2017. RFC 8032. (Request for Comments, 8032). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8032.txt⟩.

KARIMI, L. et al. An automatic attribute based access control policy extraction from access logs. *IEEE Transactions on Dependable and Secure Computing*, IEEE, 2021.

KIM, H.; KIM, D.-K.; ALAERJAN, A. Abac-based security model for dds. *IEEE Transactions on Dependable and Secure Computing*, IEEE, 2021.

KOVATSCH, M. et al. *Web of Things (WoT) Architecture*. 2022. Disponível em: ⟨https://www.w3.org/TR/wot-architecture/⟩.

KUPERBERG, M. Blockchain-based identity management: A survey from the enterprise and ecosystem perspective. *IEEE Transactions on Engineering Management*, IEEE, v. 67, n. 4, p. 1008–1027, 2019.

LACHNER, C.; DUSTDAR, S. A performance evaluation of data protection mechanisms for resource constrained iot devices. In: IEEE. *2019 IEEE International Conference on Fog Computing (ICFC)*. [S.l.], 2019. p. 47–52.

LAGUTIN, D. et al. Enabling decentralised identifiers and verifiable credentials for constrained internet-of-things devices using oauth-based delegation. In: *Workshop on Decentralized IoT Systems and Security (DISS)*. [S.l.: s.n.], 2019.

LEE, E. A. et al. The swarm at the edge of the cloud. *IEEE Design & Test of Computers*, IEEE, n. 3, p. 8–20, 2014.

LI, F. et al. Cyberspace-oriented access control: A cyberspace characteristics-based model and its policies. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 2, p. 1471–1483, 2018.

MAHALLE, P. N.; SHINDE, G.; SHAFI, P. M. Rethinking decentralised identifiers and verifiable credentials for the internet of things. In: *Internet of Things, Smart Computing and Technology: A Roadmap Ahead*. [S.l.]: Springer, 2020. p. 361–374.

NIKAEIN, N.; VASILAKOS, X.; HUANG, A. Ll-mec: Enabling low latency edge applications. In: IEEE. *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. [S.l.], 2018. p. 1–7.

NOVAK, E.; TANG, Z.; LI, Q. Ultrasound proximity networking on smart mobile devices for iot applications. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 1, p. 399–409, 2018.

OCF - Specifications. 2022. Disponível em: ⟨https://openconnectivity.org/developer/specifications/⟩.

OCKAM DID Method Specification. Disponível em: ⟨https://github.com/ockam-network/did-method-spec/blob/master/README.md⟩.

oneM2M - Release 3. 2022. Disponível em: ⟨https://www.onem2m.org/technical/published-drafts/release-3⟩.

OUADDAH, A. et al. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, Elsevier, v. 112, p. 237–262, 2017.

PAL, S. et al. On the integration of blockchain to the internet of things for enabling access right delegation. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 4, p. 2630–2639, 2019.

PARDUCCI, B.; LOCKHART, H.; RISSANEN, E. Extensible access control markup language (xacml) version 3.0. *OASIS Standard*, p. 1–154, 2013.

PÉREZ, S. et al. Application layer key establishment for end-to-end security in iot. *IEEE Internet of Things Journal*, IEEE, v. 7, n. 3, p. 2117–2128, 2019.

RABAEY, J. M. The swarm at the edge of the cloud-a new perspective on wireless. In: IEEE. *VLSI Circuits (VLSIC), 2011 Symposium on*. [S.l.], 2011. p. 6–8.

RAVIDAS, S. et al. Access control in internet-of-things: A survey. *Journal of Network and Computer Applications*, Elsevier, 2019.

RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC Editor, 2018. RFC 8446. (Request for Comments, 8446). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8446.txt⟩.

RESCORLA, E.; MODADUGU, N. *Datagram Transport Layer Security Version 1.2*. RFC Editor, 2012. RFC 6347. (Request for Comments, 6347). Disponível em: ⟨https://rfc-editor.org/rfc/rfc6347.txt⟩.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, ACM New York, NY, USA, v. 21, n. 2, p. 120–126, 1978.

SAHAI, A.; WATERS, B. Fuzzy identity-based encryption. In: SPRINGER. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. [S.l.], 2005. p. 457–473.

SANDHU, R. S. et al. Role-based access control models. *Computer*, IEEE, v. 29, n. 2, p. 38–47, 1996.

SCHAAD, J. *CBOR Object Signing and Encryption (COSE)*. RFC Editor, 2017. RFC 8152. (Request for Comments, 8152). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8152.txt⟩.

SEITZ, L. et al. *Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)*. [S.l.], 2021. Work in Progress. Disponível em: ⟨https://datatracker.ietf.org/doc/draft-ietf-ace-oauth-authz/46/⟩.

SELANDER, G.; MATTSSON, J. P.; PALOMBINI, F. *Ephemeral Diffie-Hellman Over COSE (EDHOC)*. [S.l.], 2022. Work in Progress. Disponível em: ⟨https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-17.html⟩.

SERVOS, D.; OSBORN, S. L. Hgabac: Towards a formal model of hierarchical attribute-based access control. In: SPRINGER. *International Symposium on Foundations and Practice of Security*. [S.l.], 2014. p. 187–204.

SERVOS, D.; OSBORN, S. L. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, ACM, v. 49, n. 4, p. 65, 2017.

SHARMA, D. K. et al. An opportunistic approach for cloud service based iot routing framework administering data, transaction, and identity security. *IEEE Internet of Things Journal*, IEEE, 2021.

SHIREY, R. W. *Internet Security Glossary, Version 2*. RFC Editor, 2007. RFC 4949. (Request for Comments, 4949). Accessed: 2021-10-13. Disponível em: ⟨https://rfc-editor.org/rfc/rfc4949.txt⟩.

SIRIS, V. A. et al. Decentralized authorization in constrained iot environments exploiting interledger mechanisms. *Computer Communications*, Elsevier, v. 152, p. 243–251, 2020.

SOVRIN DID Method Specification. Disponível em: ⟨https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html⟩.

SPORNY, M. et al. *Decentralized Identifiers (DIDs) v1.0*. [S.l.], 2019. Https://www.w3.org/TR/did-core//.

SPORNY, M. et al. *Verifiable Credentials Data Model 1.0*. [S.l.], 2019. Https://www.w3.org/TR/2019/REC-vc-data-model-20191119/.

SU, Y. et al. Secure decentralized machine identifiers for internet of things. In: *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*. [S.l.: s.n.], 2020. p. 57–62.

TAN, L. et al. A blockchain-empowered access control framework for smart devices in green internet of things. *ACM Transactions on Internet Technology (TOIT)*, ACM New York, NY, v. 21, n. 3, p. 1–20, 2021.

TANESKI, V.; HERIČKO, M.; BRUMEN, B. Systematic overview of password security problems. *Acta Polytechnica Hungarica*, v. 16, n. 3, 2019.

TANGLEID. *TangleID DID Method Specification*. 2019. Disponível em: ⟨https://github.com/TangleID/TangleID/blob/develop/did-method-spec.md⟩.

VERES One DID Method Specification. Disponível em: ⟨https://w3c-ccg.github.io/did-method-v1/⟩.

ZHANG, Y. et al. Attribute-based access control for smart cities: A smart contract-driven framework. *IEEE Internet of Things Journal*, IEEE, 2021.

ZUFFO, M. et al. *IoT in Brazil: An Overview From the Edge Computing Perspective*. [S.l.]: IERC Cluster Book, 2020.

APPENDIX A – ADDITIONAL PUBLICATIONS

This chapter lists the research articles co-authored by the author during the development of this dissertation, and represent indirect results of this research.

BIASE, L. C. D. et al. Swarm economy: a model for transactions in a distributed and organic iot platform. *IEEE Internet of Things Journal*, IEEE, 2018.

AFZAL, S. et al. Analysis of web-based iot through heterogeneous networks: Swarm computing over lorawan. *Sensors*, MDPI, v. 22, n. 2, p. 664, 2022.

BIASE, C. L. D. et al. Swarm minimum broker: an approach to deal with the internet of things heterogeneity. In: IEEE. *2018 Global Internet of Things Summit (GIoTS)*. [S.l.], 2018. p. 1–6.

ZUFFO, M. et al. *IoT in Brazil: An Overview From the Edge Computing Perspective*. [S.l.]: IERC Cluster Book, 2020.

CALCINA-CCORI, P. C. et al. Enabling semantic discovery in the swarm. *IEEE Transactions on Consumer Electronics*, IEEE, v. 65, n. 1, p. 57–63, 2018.

BIASE, L. C. D. et al. Swarm assistant: an intelligent personal assistant for the swarm. In: IEEE. *2019 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2019. p. 1–2.

CALCINA-CCORI, P. C. et al. Location-aware discovery of services in the iot: a swarm approach. In: IEEE. *2019 Global IoT Summit (GIoTS)*. [S.l.], 2019. p. 1–6.

BIASE, L. C. D. et al. Surveillance swarm: Gathering resources for finding targets in a distributed global network of smart devices. In: IEEE. *2020 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2020. p. 1–2.

CALCINA-CCORI, P. et al. Swarmgen: a framework for automatic generation of semantic services in an iot network. In: IEEE. *2020 IEEE International Conference on*

Consumer Electronics (ICCE). [S.l.], 2020. p. 1–5.

CALCINA-CCORI, P. C. et al. Describing services geolocation in iot context. In: IEEE. *2019 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2019. p. 1–2.

CALCINA-CCORI, P. et al. Agile servient integration with the swarm: Automatic code generation for nodes in the internet of things. In: *Proceedings of the International Conference on Future Networks and Distributed Systems*. [S.l.: s.n.], 2017. p. 1–6.

FEDRECHESKI, G.; COSTA, L. C.; ZUFFO, M. K. Elixir programming language evaluation for iot. In: IEEE. *Consumer Electronics (ISCE), 2016 IEEE International Symposium on*. [S.l.], 2016. p. 105–106.

ESQUIAGOLA, J. et al. Performance testing of an internet of things platform. In: *IoTBDS*. [S.l.: s.n.], 2017. p. 309–314.