A Middleware for Secure Integration of Heterogeneous Edge Devices

Arthur Desuert, Stéphanie Chollet, Laurent Pion and David Hély
Univ. Grenoble Alpes, Grenoble INP, LCIS
F-26000 Valence, France
{firstname.lastname}@lcis.grenoble-inp.fr

***Abstract*—Connected devices are being deployed at a steady rate, providing services like data collection. Pervasive applications rely on those edge devices to seamlessly provide services to users. To connect applications and edge devices, using a middleware has been a popular approach. The research is active on the subject as there are many open challenges. The secure management of the edge devices and the security of the middleware are two of them. As security is a crucial requirement for pervasive environment, we propose a middleware architecture easing the secure use of edge devices for pervasive applications, while supporting the heterogeneity of communication protocols and the dynamism of devices. Because of the heterogeneity in protocols and security features, not all edge devices are equally secure. To allow the pervasive applications to gain control over this heterogeneous security, we propose a model to describe edge devices security. This model is accessible by the applications through our middleware. To validate our work, we developed a demonstrator of our middleware and we tested it in a concrete scenario.**

***Index Terms*—IoT, Security, Middleware, Pervasive Applications, Modeling**

I. Introduction

Connected devices are linked together using cloud infrastructures as well as fog and edge networks [1]. The resulting data collection and remote control capacities can help the development of pervasive applications fulfilling Mark Weiser's vision of the *Invisible Computing* [2], by providing useful services embedded in the physical world.

To guarantee the successful adoption of pervasive applications, security is a key factor and several challenges remains in this area [3]. For instance, the secure integration and management of connected devices in pervasive applications is challenging because of the heterogeneity and the dynamism of those devices. The classical security requirements for a device interacting with an application are:

- **authentication**, which allows the device to ensure it communicates with the expected application. Several levels of authentication exist:
  - **none**: authentication is not used. In that case, there is no guarantee that either side of the communication is the expected participant.
  - **one-way**: the device authenticates the application as genuine. The device trusts the application, however the application has no guarantees about the device.
  - **mutual**: both sides of the communication authenticate the other side as genuine.

  Authentication, one-way or mutual, often relies on:
  - asymmetrical cryptography and a key-pair, or
  - symmetrical cryptography and a shared secret.
- **confidentiality**, which guarantees that data sent by the device to the application can be read only by the application and not by unknown participants. Confidentiality is mostly achieved using cryptography algorithms and secrets.
- **integrity**, which guarantees that the application is able to check if the data it receives is identical to data which was emitted by the device. Integrity can be achieved using hash functions, error correction codes, etc..

Security requirements are described from the device perspective, but the needs are symmetrical for the application. Implementing security features to meet those requirements in the application makes the development more complex and error-prone. One popular approach to reduce application complexity is the use of a middleware.

We propose in this article a secure fog middleware to ease the secure integration of edge devices into cloud-based pervasive applications. The middleware deals with the security, the heterogeneity and the dynamism of edge devices. It hides this complexity from the pervasive applications and it proposes instead a consistent interface to easily use the devices capabilities. Moreover, the middleware informs the applications of the security requirements fulfilled or not by the devices. This security transparency allows the applications to integrate security criteria in their selection process.

The main contributions of this article are:

- the **design of a secure middleware architecture** easing the development of secure pervasive applications,
- the **design of a data model** to describe the protocol and hardware security features of connected devices,
- the **development of a demonstrator** and its validation in a concrete scenario.

This article is organized as it follows: Section II examines recent middleware solutions and their approach of security. Section III gives an overview of the security heterogeneity of protocols and hardware in pervasive environments. In Section IV, we present our approach to build a secure middleware, while Section V details the middleware architecture. Before the conclusion, an implementation of our work and a validation scenario are presented in Section VI.

II. Related Works

Research is active on the topic of IoT middleware as underlined by Razzaque *et al.* in a 2016 survey which draws up a list of sixty one IoT middleware [4]. Razzaque *et al.* classify the middleware according to their design approach like service-oriented middleware or agent-based middleware, while mentioning that some middleware solutions combine several designs. In the rest of the article, we will focus on service-oriented middleware as it is the design we use in our global approach presented in Section IV. We present a set of recent middleware solutions for IoT with their main features and their approach on security. In particular, we focused on the management of the devices security and the security of the middleware solution itself. We also examined if the applications are informed of the deployed security features.

The *Hydra* middleware [5], currently known as *LinkSmart*[1], is designed to ease the development of Ambient Intelligence applications. The middleware uses a service-oriented approach to provide interoperability between connected devices and applications. It supports several protocols to communicate with the devices, like ZigBee[2] or Bluetooth[3]. The description of the devices is based on the OWL[4], OWL-S[5] and SAWSDL[6] ontologies. The middleware interfaces for applications use Web Services technologies such as the SOAP protocol [6]. Concerning security, the middleware features a security layer which takes into account security goals such as confidentiality and authenticity. The security relies on Web Services mechanisms enhanced by ontologies. It is not indicated if those mechanisms are compatible with the WS-Security specifications [7], the security standard for Web Services.

KASOM [8], for Knowledge-Aware and Service-Oriented Middleware, is designed to manage Wireless Sensors and Actuators Networks (WSAN) and to enable Service-Oriented Computing in those environments. Wireless communication protocols like ZigBee are supported to communicate with the edge devices. The middleware relies on two key components: Knowledge Management services and the Contextual Model. Knowledge Management services manage the data generated by heterogeneous device networks, while the Contextual Model semantically describes the network resources. The available network services are offered by the middleware using a REST approach to ease the communication between conventional networks and WSAN. The middleware architecture features a security module dedicated to the internal security management of the middleware, by enforcing access control to the services for instance. The security features of the communication with devices, like confidentiality and integrity, are not detailed and it is not clear wether the applications have access to security information about the services.

[1]https://linksmart.eu/
[2]https://csa-iot.org/all-solutions/zigbee/
[3]https://www.bluetooth.com/
[4]https://www.w3.org/TR/owl-overview/
[5]https://www.w3.org/Submission/OWL-S/
[6]https://www.w3.org/TR/sawsdl/

ubiREST [9] is an improved version of the *ubiSOAP* middleware [10]. The goal of *ubiSOAP* is to manage several network links between service producers, which are usually connected objects, and service consumers, which are usually pervasive applications. The middleware supports several network protocols, wired and wireless, such as Bluetooth and UMTS[7]. The goal of the middleware is to select the best available network link to exchange messages between producers and consumers respecting the Quality of Service threshold defined by the consumer. The *ubiSOAP* middleware used the SOAP protocol to offer its services, without mentioning the use of WS-Security. The *ubiREST* middleware moves to a P-REST architecture, defined as a refinement of the REST style. Device security is briefly mentioned and appears to be part of the Quality of Service metric. The selection of the security features and their impact on the metric remain unexplained. The internal security of the middleware is not mentioned.

In.IoT [11] is a middleware solution focused on easing the registration of connected devices and on ensuring their performance. *In.IoT* uses a modular architecture and a service-oriented approach to abstract the connected devices and to provide interoperability. The middleware supports the HTTP [12], MQTT [13] and CoAP [14] protocols to communicate with the devices. Security is also a key focus of the solution with the implementation of security restrictions for the MQTT protocol and the authentication of applications using the services. For connected devices, the middleware mainly relies on credentials authentication. They are generated during the device registration and then transmitted to the device through a network link. However, the secure handling of those credentials by the device is not detailed. Moreover, it is not precised wether the pervasive applications are aware of this security when using a service.

Table I summarizes for each middleware the features related to the security of connected devices and wether the pervasive applications using the middleware services are informed of the security measures related to the services they use.

TABLE I
SECURITY IN MIDDLEWARE SOLUTIONS

Middleware	Device Security	Middleware Security
Hydra [5]	Not indicated	Application authentication
KASOM [8]	Not indicated	Not indicated
ubiREST [9]	Not indicated	Not indicated
In.IoT [11]	Device authentication	Application authentication

Middleware	Security-Awareness of Applications
Hydra [5]	Not indicated
KASOM [8]	Not indicated
ubiREST [9]	Low (Quality of Service)
In.IoT [11]	Not indicated

From this review on IoT middleware solutions, we notice that device and middleware security are not systematically addressed and thus they remain open challenges in the domain. Moreover the middleware solutions which take devices

[7]https://www.3gpp.org/technologies/keywords-acronyms/103-umts

security into account often consider the security of the communication link, but they rarely take into account the device hardware security. Hardware security plays an important role in secure communications and thus should be considered as well when evaluating a connected device security. Lastly, the diversity of communication protocols and hardware involved in the IoT combined with the diversity of concrete security mechanism creates heterogeneous security levels. Yet, we notice the pervasive applications are rarely informed of this underlying heterogeneity. This can lead to applications trusting services with a weak security because the available security indicators are not accurate enough. In the next section, we show the diversity of protocols and hardware security to support the need for more accurate security descriptions.

III. Edge Security

In this section, we first present a set of protocols used for edge communication with their security features, and second a set of hardware security mechanisms which can be deployed on edge devices.

A. Protocol Security

We present current protocols used for communicating with IoT devices, with a focus on their security features.

The **Hypertext Transfert Protocol** (HTTP) is a protocol standardized by the Internet Engineering Task Force (IETF) [12]. It allows the exchange of data between two equipment following a client/server paradigm. HTTP is a high level protocol and relies on lower network protocols, such as TCP and Wi-Fi, to establish a connection with an edge device. The protocol in itself does not offer security features such as data confidentiality or host authentication. However, it is often used with the Transport Layer Security (TLS) [15] protocol which provides those security features. The association of HTTP with the TLS protocol is commonly called HTTPS. The TLS protocol supports the encryption of exchanged data using several ciphers such as AES or Chacha20, to ensure data confidentiality. It also supports simple and mutual authentication mechanisms based on a shared secret or X.509 certificates. TLS is also used by other high level protocols such as MQTT [13] and CoAP [14] to cover their security needs. The SOAP protocol [6], used for instance by Web Services, can also benefit from TLS security as it is often used over HTTPS in addition to the WS-Security specification [7], its own security layer.

ZigBee is a wireless communication protocol defined by the Connectivity Standards Alliance[8]. Devices communicating with ZigBee are organized in networks. To join an existing network, a device needs to recover a key named the Network Key. Each ZigBee network has a dedicated Network Key. To retrieve this key, several procedures are available [16]:

- the device can contact the equipment managing the network, called the ZigBee Coordinator in ZigBee terminology, to retrieve the Network Key. A predefined key is often used to ensure the confidentiality of the exchange; however this predefined key is publicly available in the ZigBee specification which creates an attack vector [17].
- In ZigBee 3.0, the device and ZigBee Coordinator can mutually authenticate using an install code. An install code is a random secret assigned to the device during its manufacturing. This code has to be shared with the ZigBee Coordinator, which is done by a user or an installer for instance. This secret is then used to derive a shared key enabling authentication and confidentiality between the new device and the ZigBee Coordinator.
- finally, the Network Key can be manually loaded into the device by a user or an installer for instance.

ZigBee ensures the confidentiality and authentication of data in transit with the use of the Advanced Encryption Standard (AES) symmetric cipher in CCM* mode with keys of 128 bits [17]. ZigBee also ensures data integrity using a Message Integrity Code (MIC) generated by the AES-CCM*.

Bluetooth Low Energy (BLE) is a wireless communication protocol defined by the Bluetooth specification[9]. It is primarily designed for IoT applications constrained in energy and which only need to transmit small amounts of data [18]. BLE supports several authentication levels. The level selected for a given connection depends on the following parameters:

- the selection (or not) of the authentication option during the negotiation of connection options.
- the input and output capabilities of each device. For instance, the presence of a keyboard or a screen on the devices.

Depending on the connection parameters, the exchanged data can be ciphered to ensure confidentiality. To do so, BLE uses the AES symmetric cipher in CTR mode with a shared key of 128 bits. The key is generated during the connection. Data integrity and authentication are ensured by Message Authentication Codes (MAC) which are computed using the AES cipher in CCM mode.

To summarize this overview of IoT communication protocols, Table II presents the security features supported by those protocols. All protocols support mutual authentication, data confidentiality and integrity.

TABLE II
Security in IoT protocols.

Protocols	Authentication levels			Authentication means
	None	One-way	Mutual	
HTTP(S)	✓	✓	✓	Shared secret, certificate
Zigbee	✓	✓	✓	Shared secret
BLE	✓	✓	✓	User validation

Protocols	Confidentiality	Integrity
HTTP(S)	Configuration dependent	Configuration dependent
Zigbee	Yes	Yes
BLE	Configuration dependent	Configuration dependent

[8]https://csa-iot.org/

[9]https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

This overview highlights the heterogeneous implementation of common security requirements in several communication protocols. Another key point is the recurrent presence of a shared secret for authentication. This implies that this secret needs to be stored by devices.

B. Edge Device Security

Using a secure protocol is not enough to guarantee the security of a whole IoT solution. Section III-A highlights the recurrent use of shared secrets in IoT protocols to authenticate devices and/or ensure the confidentiality of data. Security offered by those protocols mainly relies on the use of standardized algorithms and the trust in a given secret. If a secret is compromised, the security properties inherited from this secret are compromised as well. Expected attributes of a storage solution for secrets are the following:

- **secure**: the solution should protect the confidentiality of stored secrets and prevent unwanted extraction attempts.
- **resource-efficient**: the solution should mobilize the most limited hardware ressources to perform its task.
- **low additional cost**: the solution should be affordable and accessible to even low-cost devices.

We reviewed current solutions which are used to store secrets.

1) Non-volatile memory: It is possible to store secrets directly in non-volatile memory. As such memory is widely available in embedded systems, it does not imply additional cost. However, the security of this solution is often low. This type of memory can be easily scanned by an attacker who can extract secrets if they are stored in plaintext. One countermeasure to such attack is to cipher the secret before storing it, but it also requires to store the key used which creates a circular problem.

2) Secure Element: It is a dedicated microprocessor chip which is usually installed aside the main microprocessor of a device, with communication ensured by a secure hardware bus. A Secure Element (SE) is specifically designed to store confidential data and to process cryptographic operations. It includes sensors to detect intrusion attempts and to react accordingly, by erasing the internal storage for instance. It is also designed to be tamper proof against hardware attacks such as power side channel or fault injection attacks. Those security properties are often qualified by a certification authority and must be compliant with security standards, such as the Common Criteria (CC) for instance. While with SE, one can reach a very high security level, it induces an extra cost to the device in terms of components and engineering [19].

3) Trusted Execution Environment: Another solution for secure storage is to use a microprocessor chip with embedded security features like a secure enclave, which splits the microprocessor in two distinct parts:

- a *normal context*, also named Rich Execution Environment (REE), where the main applications can run and interact with the outside world, and
- a *secure context*, also named Trusted Execution Environment (TEE), where sensitive operations can be executed and secrets securely stored, with very limited interactions with the outside world.

Context switches can only happen in well-defined entry/exit points to prevent illegal accesses. This solution can be easier to implement but it is less secure than a SE because it does not include antitampering protections [19].

4) Physical Unclonable Functions technology: It is a hardware block which reacts to an input stimulus named the challenge by emitting an output stimulus named the response. Physical Unclonable Functions (PUF) [20] originality is that its output not only depends on the challenge but also on some intrinsic properties of the hardware components, such as the geometry of its transistors. Because those properties mainly depend on manufacturing process variations, it is statistically highly unlikely or almost impossible to produce two identical PUF, hence its unclonable property.

PUF circuits have some intrinsic security properties. First, their unclonability which guarantees an almost-nil risk of duplication. Second, PUF are active circuits, thus no information are available when the device or the PUF are powered down, as opposed to non-volatile memories storing sensitive data. Those properties make PUF circuits good candidates for a secure storage solution with a reasonable integration cost.

To conclude, Table III highlights the diversity of storage solutions and the heterogeneity of the security involved. This allows connected device designers to pick the solution which fits best the constraints and security needs of each project. However, this heterogeneity must be taken into account and properly managed by the IoT middleware designers to successfully integrate various references of connected devices. Pervasive applications should also be informed in an adapted way of this heterogeneity, as it can be a meaningful criterion when selecting a service.

TABLE III
COMPARISON OF SOLUTIONS FOR SECRET STORAGE.

Storage Solution	Security	Resource efficiency	Additional Cost
Non-volatile Memory	None	High	None
Secure Element	High	Medium	Medium to High
TEE	Medium	Medium	Medium
PUF	Medium	High	Low

To achieve these main goals of managing the heterogeneous security of connected devices and to allow the pervasive applications to select services based on their underlying security, we present our middleware design approach in the next section.

IV. Global Approach

In this section, we present our global approach to ease the secure integration of connected devices into pervasive applications while supporting the heterogeneity and the dynamism of the devices.

We propose a new middleware solution deployed at the fog level which eases the secure connection between the pervasive applications and the connected devices. The middleware uses Service-Oriented Computing (SoC) principles [21] to abstract the connected devices heterogeneous capabilities as services accessible through a uniform interface. The list of services proposed by the middleware is updated dynamically. Moreover our solution manages the security of the connected devices, dealing with the heterogeneity of secure protocols and key management to provide trustworthy services to the pervasive applications. Together with the middleware, we propose a set of data structures to concretely describe these security features and to present them to the pervasive applications. The global architecture of our solution is presented in Fig. 1.

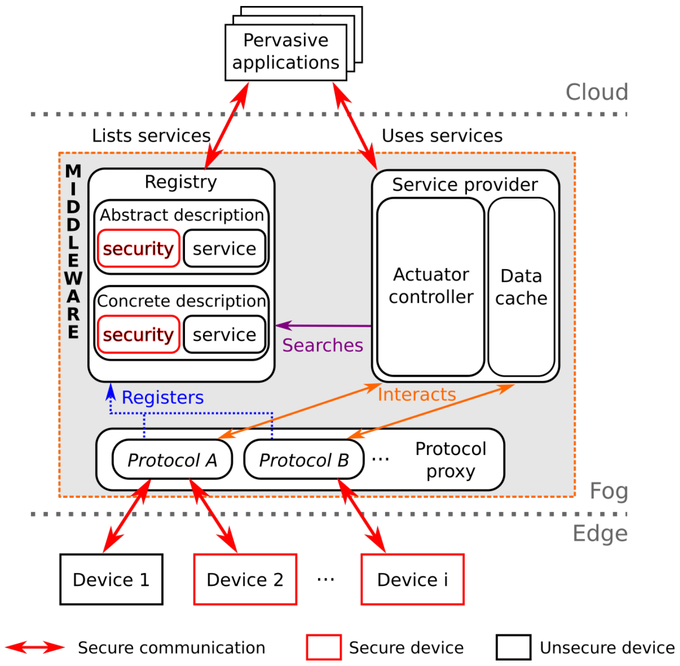

Fig. 1. Global architecture.

Our middleware is composed of three main modules, as illustrated by Fig. 1 :

- the **protocol proxy** module, which manages the communication between the middleware and the connected devices. This module can dynamically load or unload submodules which act as proxies for a specific communication protocol (*e.g*, HTTPS, ZigBee...), managing the protocol heterogeneity presented in Section III-A. Each submodule implements and manages the security features associated with its specific protocol. Those submodules ensure the communication with the devices using the appropriate security configuration depending on the pervasive application requests via the **service provider** module. They can also register new connected devices and their services, if supported.
- the **registry** module, which allows pervasive applications (*i.e.*, service consumers) to list the services available through the middleware and their description. The service description is divided in two parts: an abstract part which describes the service specifications and a concrete part which holds the necessary information to contact the service provider. The security description of the service is also stored in the registry and split in a similar manner (an abstract part and concrete one). The pervasive applications have only access to the abstract description to find the appropriate services. The concrete description is used internally by the **service provider** module.
- the **service provider** module, which manages the requests of pervasive applications to access specific services. This module is in charge of invoking the chosen service and sending back a response to the application. The **service provider** module separates the services in two types: sensors and actuators. Sensor services are associated to devices which send measures to the middleware. Those measures are temporarily stored in the *data cache* submodule and served by the corresponding services. Actuator services are associated to devices which accept control commands. Those devices are managed by the *actuator controller* submodule which handles issues such as concurrent access.

The middleware fog deployment and its service-oriented architecture offer several advantages [1]:

- it allows the **integration of a wide range of devices**, including those which can communicate only on local networks. Such devices would not be usable with cloud-based solutions which require a connection to the Internet.
- it **lowers the latency** when interacting with the devices compared to cloud-based solutions. This can be of great value when controlling actuators for instance. This can also improve the lifespan of the energy-constrained devices by reducing their communication cost.
- it **reduces the exposure of the devices to remote threats**. The devices communicate with the middleware solution through local networks which are less exposed than worldwide networks such as the Internet. The middleware is in charge of establishing the secure communication with the remote applications, taking this charge away from the devices.
- it **hides the heterogeneity** of service protocols and security mechanisms. Pervasive applications have only access to the abstract descriptions. The abstract descriptions do not contain details on the implementation. Consequently, it facilitates the device integration in pervasive applications for developers.
- it ensures a **low-coupling** between the pervasive applications and the services they use, which helps dealing with the dynamism of the services. The pervasive applications do not directly connect to the services, they only need to have access to the abstract descriptions of the services.

Having presented the global architecture of our solution to simplify the secure use of connected devices by pervasive applications, we detail its main components in the next section.

V. Middleware architecture

In the following sections, we present in details the core modules providing the basic functionalities of the middleware.

A. Protocol Proxy Module

The **protocol proxy** module provides communication with the edge devices. Its main goals are:

- to manage the heterogeneity of communication protocols, allowing the integration of a wide range of edge devices
- to manage the heterogeneity of protocols security features
- to adapt various devices data format into the middleware internal representation

To reach those goals, the module is divided into independent submodules. Each submodule manages a specific communication protocol and its security features.

We defined a generic submodule architecture with a set of interfaces and common data structures. This architecture is illustrated in Fig. 2.

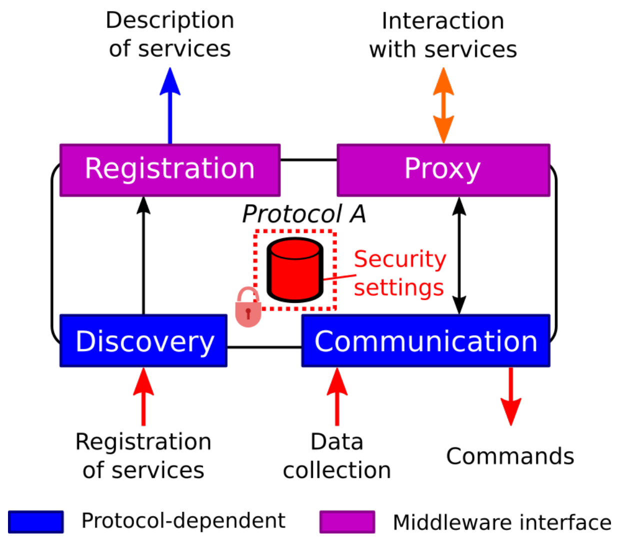

Fig. 2. Generic submodule architecture for protocol proxy.

The interfaces are divided in two types: protocol-dependent and middleware. Protocol-dependent interfaces need to be adapted to a specific protocol capabilities and may not be fully implemented. For instance, a protocol could not feature service discovery mechanisms. Those interfaces can accept data in various formats. Middleware interfaces ensure the communication with the middleware internals, the others core modules for instance. They use a precise data format to ensure the compatibility of the various middleware modules. Finally, the architecture includes a protected space for storing the security settings related to the protocol, the private key and the associated certificate of a HTTPS interface for instance.

Using a generic architecture and defined interfaces simplifies the development and the deployment of new submodules to support additional and future protocols.

B. Register Module

The **registry** module is in charge of storing the data about the services available through the middleware. It updates those data as new services are registered or as they leave. The data are stored using a data model designed to describe the services and the security linked to those services. The model is divided in two layers:

- the *abstract description layer* contains high level information about a service and its security, like the service interface or the supported security features. Such information are useful to select a service.
- the *concrete description layer* contains the technical details of a service and its security, like the address of the service provider or the supported authentication means. Such information are useful to securely use a service.

This model defines a clear separation of concerns between the service selection which is done by the pervasive applications with the abstract layer, and the secure interactions with the services which are carried on by the middleware with the concrete layer. This separation of concerns already exists in service-oriented technology [22] such as for Web Services [23] thanks to the WSDL description [24], containing an abstract part and a concrete one, and also a security description with a WS-Policy profile [25]. However, this solution has been implemented only for Web Services and not for other service technologies such as UPnP [26], DPWS [27] or iPOJO [28]. WS-Policy has been rarely used by the developers due to the complexity to create security profiles.

The service part of our model is generic and can be extended. If necessary, more specific models [28], [22] can be used under the condition that those models respect a set of basic constraints, having a distinction between actuator services and sensor services for instance.

For the security part, our goal is to propose a data model to describe the concrete security features implemented by a service provider. Fig. 3 gives an overview of this model.

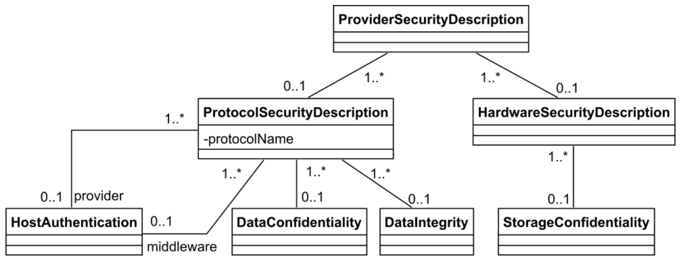

Fig. 3. Overview of the concrete security description model.

The *ProtocolSecurityDescription* class describes the security features of the communication protocol, while the *HardwareSecurityDescription* class describes the security implemented at the device level. The model is structured around the classical security requirements identified in the beginning of this article: authentication, confidentiality and integrity. Those requirements are reached using security primitives like encryption or signature, and those primitives are concretely implemented by secure algorithms (*e.g.*, SHA or ECDSA), specific hardware (*e.g.* PUF) and the manipulation of cryptographic secrets (*e.g.*, public/private keys).

For instance, Fig. 4 details the description of the authentication requirements, represented by the *HostAuthentication* class.

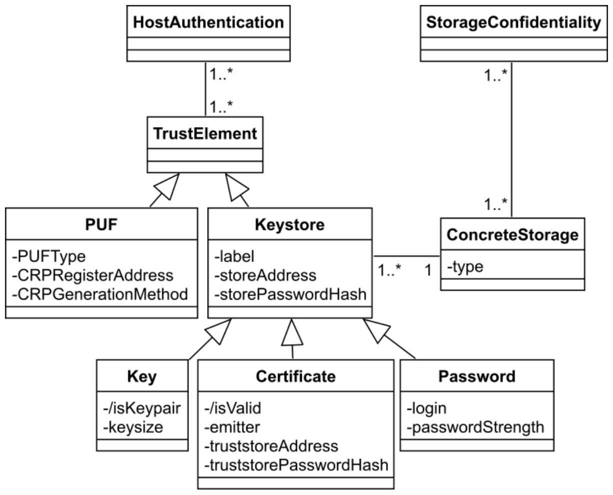

Fig. 4. Authentication part of the concrete security description model.

Authentication relies on the use of one or several trust elements, represented by the *TrustElement* class. We consider common authentication means such as password-based authentication described by the *Password* class, with the *passwordStrength* attribute evaluating the difficulty to guess the password, and certificate-based authentication described by the *Certificate* class, with the *isValid* attribute indicating if the certificate is in its validity period or not. Those authentication means are based on a secret which needs to be stored, usually on the equipment itself. We describe this need with the *Keystore* super class linked to the *ConcreteStorage* class. Thus, we take into account the security of the storage to evaluate the authentication strength. We also consider new authentication technologies like PUF-based authentication, described by the *PUF* class, which relies on the generation and exchange of Challenge-Response Pairs (CRP) [29].

This concrete security description can vastly change from one provider to another, due to the heterogeneity of protocol and hardware security highlighted in Section III. Moreover, the diversity of concrete security algorithms and hardware makes the direct comparison of concrete security description a challenging task. For those reasons, those definitions are not shared with the pervasive applications. Instead, we developed an abstract security description which indicates the status of the security requirements (either supported or not) and which features a *security level*. The model of this description is illustrated by Fig. 5.

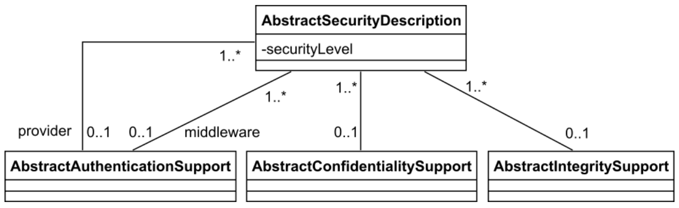

Fig. 5. Abstract security description model.

The *securityLevel* attribute of the *AbstractSecurityDescription* class is a numerical metric computed from the concrete security description to evaluate the quality of the implemented security mechanisms. The metric ranges from 0 to 100, 100 representing full confidence in the service security. For instance, if the confidentiality relies on a state-of-the-art algorithm and if the key used is adequately sized, the *security level* is likely to be high. Conversely, if the algorithms used are known to be outdated, vulnerable or if the secrets are not securely stored, the *security level* will be low. Using this metric, the pervasive applications can quickly estimate the security of the services they use without diving into the technical details.

C. Service Provider Module

The **service provider** module is used by pervasive applications to access the services listed by the middleware. The module provides a uniform interface to the applications and manages concurrent accesses to the services. It is divided in two main parts: the data cache dedicated to sensor services and the actuator controller dedicated to actuator services.

The data cache is a temporary storage for data associated with the sensor service of a connected device. When a pervasive application invokes a sensor service, the service provider module looks up in the service data cache. If a valid data is present, the data is deleted from the cache and sent back to the application. Else, the service is considered unavailable and the request fails.

The actuator controller generates and sends commands to the concrete actuator devices, using the **protocol proxy** module. It handles and regulates the concurrent accesses to an actuator service to respect the actuator limits.

VI. Middleware Implementation and Example

In this section, we introduce the demonstrator developed using the middleware architecture presented previously. We also detail a use case illustrating the validity of our solution.

A. Implementation

To implement the middleware architecture, we used the Java language and the Quarkus[10] framework which are adapted to the development of RESTful services and micro-services. This is coherent with the modular architecture of our middleware and with our goal to propose a set of uniform and secured interfaces to the pervasive applications. The modules of our architecture are implemented as micro-services and they offer

[10]https://quarkus.io/

to pervasive applications HTTP REST interfaces secured by JWT tokens[11] on HTTPS protocol. The middleware is able to authenticate the applications accessing the services and to enforce access control on sensitive services (*e.g.* the services to edit the configuration of some connected devices).

Our current demonstrator supports the HTTP and MQTT protocols to interact with connected devices. Each protocol is managed by a dedicated protocol proxy submodule with support for several security features. For instance, the HTTP proxy submodule supports simple and mutual authentication using X.509 certificates, as well as data integrity and confidentiality when the TLS protocol is used. The MQTT proxy supports password authentication.

Next, we illustrate the use of our middleware with a practical scenario.

B. Validation scenario

We consider a smart-home application which needs to detect the presence of persons. Our middleware is deployed in a house, illustrated in Fig. 6, with two connected devices which provide the detection service.

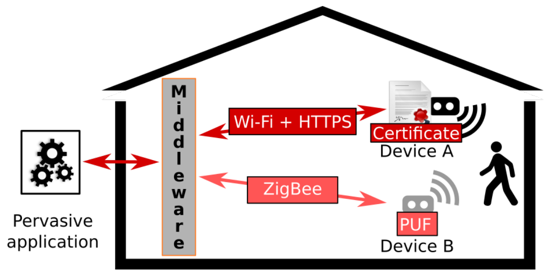

Fig. 6. Presence detection use case.

The devices use different detection technologies (*e.g.*, infrared or ultrasound), yet they both provide the same high level *detectPresence* service. Moreover the two devices have heterogeneous security features, like different secure protocols and a different authentication mean: a certificate for the *Device A* and an embedded PUF for the *Device B*. Those differences result in a unequal security level for the two devices, the same being true for the provided services. We will show how our middleware solution allows applications to be aware of this difference during the service selection.

Assuming the middleware solution is already deployed in the considered home, the first step is the registration of the devices and their services. This step is performed by a trained installer who sets up the devices, configures their security and connectivity features, then generates the appropriate service and security descriptions based on the models describes in Section V-B. At the end of this step, the middleware manages the secure connection with the devices and it offers their services to pervasive applications. In our scenario, it means the two devices are properly connected to the middleware and their *detectPresence* service is registered.

[11]https://datatracker.ietf.org/doc/html/rfc7519

The second step is the search for services. This step is performed by a pervasive application. The application uses the secure interface of the **register** module to enumerate the available services. The **register** module uses the *abstract description layer* of each service to build a list, it encodes this list using the JSON format[12] and sends it to the application. The list presents for each service its name, its input parameters and output values. Moreover, each service has one to several security profiles defined by an identifier and a set of high level security properties, like the support of data confidentiality or the device authentication. Those security properties are completed by a security level, a metric which evaluates the overall security of the service as previously detailed in Section V-B. These information can be integrated into a service selection algorithm [30] used by the application to ensure the selected service and profile respect the application security needs. In our scenario, the smart-home application querying the **register** interface for the available services gets a list including only the *detectPresence* service with two security profiles to choose from, corresponding to the two concrete devices. Fig. 7 presents a snippet of this list in JSON format, focusing on the security attributes.

```
[
  {
    "serviceName": "detectPresence",
    [...]
    "securityProfiles":
    [
      {
        "id": "a110[...]8453"’
        "securityLevel": 75,
        "providerAuthentication": true,
        "confidentiality": true,
        [...]
      },
      {
        "id": "9ea4[...]bb74",
        "securityLevel": 60,
        "providerAuthentication": true,
        "confidentiality": true,
        [...]
      }
    ]
  }
]
```

Fig. 7. Snippet of the services list.

Once a service is selected by a pervasive application, the third and final step is the service invocation. The application uses the secure interface of the **service provider** module to transmit the service name and the desired security profile identifier, and optionally the input parameters. The **service provider** module handles all the necessary steps to retrieve the service output, including the secure connection with connected devices through their heterogeneous protocols. The application gets either the service output or an error message if the service is unavailable for instance. The application implements only the communication protocol used by the middleware interfaces, HTTPS in our demonstrator, and does not manage

[12]https://datatracker.ietf.org/doc/rfc8259

additional protocols dedicated to the connected devices such as ZigBee in our scenario. This reduces the complexity of applications and it allows the developers to focus on the application added value. Moreover the handling of cryptography secrets and protocols security features is centralized by the middleware, avoiding its duplication in each pervasive application. This also reduces the security risks linked to potential implementation errors.

Through this scenario, we have validated a concrete use case of our demonstrator based on the middleware architecture proposed in this article.

VII. Conclusion and Perspectives

To summarize, after a review of recent academic middleware solutions followed by a review of IoT protocols and embedded device security, we proposed a new middleware architecture with an accent on security considerations to guarantee trustworthy services to pervasives applications. Following a secure by design approach, our middleware manages seamlessly the heterogeneity of IoT protocols and their security features to securely connect to edge devices and to offer their services to the pervasives applications. As edge device security can not be reduced to the security of the communication protocol only, we also propose a model to take into account both the communication security and the edge device embedded security. This model is fully integrated in our middleware solution and is used to label the security of the services offered by the middleware. Thus, the applications are aware of the security features supported by a given service and they can use this information as a selection criteria.

We have developed a demonstrator of our middleware using the micro-service approach to validate its benefits. We have tested our demonstrator in a concrete scenario.

In future works, we will focus on improving the security description and in particular its generation. We aim at developping a model which can involve several actors, where a connected device manufacturer will be able to propose a base security description and local actors such as an edge device installer will be able to personalize this description to fit the local context and improve its relevancy. We will also work on the design and implementation of logging functions in the middleware to support traceability and to ease security audits.

Aknowledgment

This project has received funding from the Trust Chair of the Grenoble-INP Foundation.

References

[1] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk, and S. A. Malik, "Fog/Edge Computing-Based IoT (FECIoT): Architecture, Applications, and Research Issues," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4118–4149, Jun. 2019.

[2] M. Weiser, "The Computer for the 21st Century," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3–11, Jul. 1999. [Online]. Available: https://dl.acm.org/doi/10.1145/329124.329126

[3] Y. I. Alzoubi, V. H. Osmanaj, A. Jaradat, and A. Al-Ahmad, "Fog computing security and privacy for the Internet of Thing applications: State-of-the-art," *Security and Privacy*, vol. 4, no. 2, Mar. 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/spy2.145

[4] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[5] M. Eisenhauer, P. Rosengren, and P. Antolin, "HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems," in *The Internet of Things*, D. Giusto, A. Iera, G. Morabito, and L. Atzori, Eds. New York, NY: Springer, 2010, pp. 367–373.

[6] W3C, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," Apr. 2007. [Online]. Available: https://www.w3.org/TR/soap12/

[7] OASIS, "Web Services Security: SOAP Message Security 1.1," Feb. 2004. [Online]. Available: https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[8] I. Corredor, J. F. Martínez, M. S. Familiar, and L. López, "Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 562–576, Mar. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804511001111

[9] M. Caporuscio, M. Funaro, C. Ghezzi, and V. Issarny, "ubiREST: A RESTful Service-Oriented Middleware for Ubiquitous Networking," in *Advanced Web Services*, A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. New York, NY: Springer, 2014, pp. 475–500. [Online]. Available: https://doi.org/10.1007/978-1-4614-7535-4_20

[10] M. Caporuscio, P.-G. Raverdy, and V. Issarny, "ubiSOAP: A Service-Oriented Middleware for Ubiquitous Networking," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 86–98, Jan. 2012.

[11] M. A. A. da Cruz, J. J. P. C. Rodrigues, P. Lorenz, V. V. Korotaev, and V. H. C. de Albuquerque, "In.IoT—A New Middleware for Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7902–7911, May 2021.

[12] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," Internet Engineering Task Force, Request for Comments RFC 7231, Jun. 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7231

[13] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta, "MQTT Version 5.0," Mar. 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[14] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," Internet Engineering Task Force, Request for Comments RFC 7252, Jun. 2014. [Online]. Available: https://datatracker.ietf.org/doc/rfc7252

[15] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Aug. 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446

[16] NXP, "Maximizing Security in ZigBee Networks," p. 10, Jan. 2017. [Online]. Available: https://www.nxp.com/docs/en/supporting-information/MAXSECZBNETART.pdf

[17] T. Zillner and S. Strobl, "ZigBee exploited - The good, the bad, the ugly," 2015. [Online]. Available: https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly.pdf

[18] J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu, "BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy," 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/wu

[19] Global Platform, "The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market," Jun. 2015. [Online]. Available: https://globalplatform.org/wp-content/uploads/2018/04/GlobalPlatform_TEE_Whitepaper_2015.pdf

[20] R. S. Pappu, "Physical one-way functions," Thesis, Massachusetts Institute of Technology, 2001. [Online]. Available: https://dspace.mit.edu/handle/1721.1/45499

[21] M. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, Dec. 2003, pp. 3–12.

[22] S. Chollet and P. Lalanda, "Security Specification at Process Level," in *2008 IEEE International Conference on Services Computing*, vol. 1, Jul. 2008, pp. 165–172.

[23] W3C, “Web Services Architecture,” Feb. 2004. [Online]. Available: https://www.w3.org/TR/ws-arch/

[24] ——, “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,” Jun. 2007. [Online]. Available: https://www.w3.org/TR/wsdl/

[25] ——, “Web Services Policy 1.5 - Framework,” Sep. 2007. [Online]. Available: https://www.w3.org/TR/ws-policy/

[26] UPnP Forum, “UPnP Device Architecture 2.0,” Apr. 2020. [Online]. Available: https://openconnectivity.org/upnp-specs/UPnP-arch-DeviceArchitecture-v2.0-20200417.pdf

[27] OASIS, “Devices Profile for Web Services Version 1.1,” Jul. 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html

[28] C. Escoffier, R. S. Hall, and P. Lalanda, “iPOJO: an Extensible Service-Oriented Component Framework,” in *IEEE International Conference on Services Computing (SCC 2007)*, Jul. 2007, pp. 474–481.

[29] G. E. Suh and S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation,” in *2007 44th ACM/IEEE Design Automation Conference*. San Diego, California: Association for Computing Machinery, Jun. 2007, pp. 9–14. [Online]. Available: https://doi.org/10.1145/1278480.1278484

[30] S. Chollet, V. Lestideau, P. Lalanda, Y. Maurel, P. Colomb, and O. Raynaud, “Building FCA-Based Decision Trees for the Selection of Heterogeneous Services,” in *2011 IEEE International Conference on Services Computing*, Jul. 2011, pp. 616–623.