This is a pre-print. The final version will appaear in the proceedings of the 8th IFIP International Conference on New Technologies, Mobility and Security that will be held from 21 to 23 November 2016 in Larnaca - Cyprus.

An IoT middleware for enhanced security and privacy: the RERUM approach

George Moldovan* Elias Z. Tragos†, Alexandros Fragkiadakis†, Henrich C. Pöhls‡, and Daniel Calvo§

*SIEMENS SRL, Romania, †FORTH-ICS, Greece, ‡University of Passau, Germany, §Atos, Spain
Contact author: George Moldovan {george.moldovan@siemens.com}

***Abstract*—The Internet of Things (IoT) presents itself as a promising set of key technologies to provide advanced smart applications. IoT has become a major trend lately and smart solutions can be found in a large variety of products. Since it provides a flexible and easy way to gather data from huge numbers of devices and exploit them to provide new applications, it has become a central research area lately. However, due to the fact that IoT aims to interconnect millions of constrained devices that are monitoring the everyday life of people, acting upon physical objects around them, the security and privacy challenges are huge. Nevertheless, only lately the research focus has been on security and privacy solutions. Many solutions and IoT frameworks have only a minimum set of security, which is a basic access control. The EU FP7 project RERUM has a main focus on designing an IoT architecture based on the concepts of Security and Privacy by design. A central part of RERUM is the implementation of a middleware layer that provides extra functionalities for improved security and privacy. This work, presents the main elements of the RERUM middleware, which is based on the widely accepted OpenIoT middleware.**

I. Introduction

The recent advances in Information and Communication Technologies (ICTs) and especially in the domains of the Wireless Sensor Networks (WSNs) and the Internet of Things (IoT) have opened up huge opportunities for the development of "smart" products. Nowadays, a large number of companies in many domains are trying to embed digital technologies in their products, aligning themselves to the IoT trend. In this respect, there are now available smart products, such as light-bulbs, thermostats, watches, doors, fridges and more. Moreover, the latest trends in ICT are not only to embed ICT technology on the products, but also to connect them to the Internet, using technologies of the IoT. Then, products can deliver information to the companies or receive commands and updates from remote servers through IoT applications.

Despite the fact that IoT has attracted a lot of research in the last decade, the focus on enhancing the security and privacy of IoT products has been quite limited, which resulted to a new threat landscape. With hundreds or thousands of smart devices around us monitoring our everyday activities or controlling physical objects, it is reasonable to assume that if the IoT systems are not secure, then either the personal information of citizens can be easily logged and leaked to third parties or attackers could exploit vulnerable devices and systems for their benefit [1].

This paper presents the approach taken by EU project RERUM for the implementation of a middleware that aims to provide advanced security and privacy functionalities. RERUM's main target is to develop a architectural model for creating IoT systems based on the concept of "security and privacy by design" [2]. In this respect, RERUM embeds security and privacy functionalities in components throughout the IoT system: starting from the devices (which are normally the weakest links), the gateways, the middleware and the applications. For brevity we only describe the RERUM advances in the field of the middleware, presenting additional functionalities implemented by RERUM on top of the OpenIoT middleware, which was taken as a basis.

II. The RERUM middleware architecture

The RERUM project[1] has based its architecture on the broadly accepted Architectural Reference Model (ARM) of the EU lighthouse project Internet of Things Architecture (IoT-A). Thus, RERUM uses similar terminology as IoT-A and adopts the basic concepts and ideas of IoT-A for the general design of its framework. However, RERUM goes one step beyond and leverages the concepts of IoT-A tailoring them to its own objectives and system requirements. In this respect, RERUM assumes the following key building blocks in an IoT system [3]: (i) the RERUM Devices (RDs) that can be constrained or unconstrained devices, equipped with one or multiple sensors and actuators and having the RERUM embedded mechanisms, (ii) the RERUM Gateway (RGs) that play the intermediate between the RDs and the rest of the RERUM system, performing functionalities, such as network and protocol translation, to allow the RDs (that mostly use 6LoWPAN or ZigBEE) to communicate with the rest of the RERUM elements that use IPv4, (iii) the RERUM middleware (RMW) that performs all functionalities for virtualisation, data processing and service provisioning, (iv) the RERUM security server that includes all security and privacy related functionalities. The security server can be developed as a standalone component talking to the MW via pre-defined interfaces or as an integrated component of the RERUM middleware. (v) the Application Server, which can be either an internal component of the RERUM system or an external component that uses the RERUM interfaces for service provisioning.

[1]EU FP7 RERUM: Reliable, Resilient and Secure IoT for Smart City applications, www.ict-rerum.eu

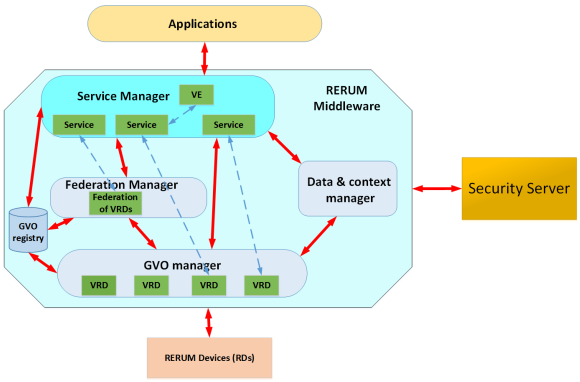

Fig. 1. RERUM middleware functional group architecture

The architecture and interconnection of these components is shown in Fig. 1. Since the focus of the paper is only on the RERUM middleware, we will not describe the rest of the components, so the readers can refer to the RERUM Deliverable D2.5 [3] for those. RERUM follows a main line of terminology for IoT, using the term Middleware (MW) to refer to the collection of functional components that are interworking for enabling the efficient provisioning of services from the devices to the users.

The MW in RERUM consists of several functional groups, as can be seen in Fig. 1. For the implementation of the RMW, the open-source middleware of the EU FP7 project OpenIoT ([4]) was selected as the most efficient solution, mainly due to its open-source nature and the fact that it was developed under the concepts of IoT-A.

The Service Manager in RERUM is responsible for handling the service requests from the applications (which maps to the Scheduler and Service Delivery components of OpenIoT), identifying which are the Virtual Entities (VEs) that are of interest for the application and tries to match the VEs to predefined templates that exist in the GVO registry (which maps to the cloud database of OpenIoT). We have to note here that following the IoT-A concepts, the VEs are the digital representations of Physical Entities, which are the objects around us and for which we want to learn some information (i.e. the chair, the room, a city square, the environment, etc.).

The Generic Virtual Object (GVO) manager (leveraging on the X-GSN component of OpenIoT) is the entity that creates and manages the digital representations of the physical objects and the RERUM Devices (that form the GVOs). The GVO manager is also a key component that handles the registration of the RERUM Devices (RDs) to the MW, by mapping the characteristics of the devices to predefined templates and storing all information to the GVO registry.

The Federation Manager is the component that creates and manages the federations of RDs, which provides advanced services to the end-users, i.e. for enabling devices to cooperate by performing service composition and orchestration. Finally, the data and context manager is the functional group that handles and processes the data gathered from the devices in order to send them to the applications according to the needs of each application (i.e. performing averages, filtering data, etc.). The data and context manager also extracts the context out of data and passes them on to the applications or to other MW functionalities (i.e. security). This component maps to OpenIoT's Service Delivery and Utility manager.

III. Security and Privacy in RERUM MW

This section provides an overview of the RERUM-developed middleware components – as extensions to the OpenIoT middleware – providing additional security, privacy and trust functionalities, as well as their interactions. In brief terms, the components presented here have four functions: (i) to offer an abstraction of heterogeneous devices by virtualising each of them as a homogeneous accessible/invocable virtualised object (the Virtual RERUM Device - VRD), (ii) to remove identifiable features (such as IP and MAC) and their requirement from any routines managing sensing or acting devices entirely, (iii) to force a Feature of Interest (FOI) based discovery process for sensors and actuators and report processed data only (e.g. averages or readings variations over a time window) only, and, finally, (iv) to provide privacy-preserving authenticity and integrity-checking mechanisms.

A. Device templates

A device template represents a semantically described abstraction of a specific RD type, containing a description of only its functional relevant features and addressable subcomponents. More specifically, the template contains information on the device model, hardware components (i.e. RAM, CPU), the default firmware, of the available resources which can be accessed on the device, their format, their type (i.e. measurement units, accuracy and sensing-interval in case of sensors, etc) and a description of how these resources are to be accessed (e.g. of what command to invoke in order to access a specific measurement or to trigger a certain command).

For each model or variation (i.e. identical hardware devices with various firmware), a single template and a relevant identification code are stored in the RMW. Initially, the templates can be either provided by a MW administrator or by the RDs themselves upon registration (only when the reported template is unknown).

B. Federation templates

A federation template represents an if-then-logic based construct, consisting of a set of required device templates, a description of the required input and output resources (of the RDs), and a set of rules which define their interaction (a basic example of a federation template would be "if too-warm then open-window" case), representing basic automation tasks supported directly by the RMW.

While the role of device templates is to abstract heterogeneous, distinctive devices as homogeneous virtualized objects and support a consistent access pattern to any of the available RDs (either sensors and actuators), the role of templates is: (i) to provide an extensible, transparent automated services to

end-users, and (ii) to provide automated services which would normally require for an end-user to have access to specific individual RDs only or which would require a granularity in the discovery of the required RDs which would rise privacy and security concerns. A federation logic meant to exclude malfunctioning temperature nodes ran by an end-user, for example, would require the user to be able to identify and link RDs providing temperature readings in a series of readings.

C. Device registration

Upon registering with the middleware, a RDs is registered in two of the middleware components: in the RG, and in the RMW. The RG, which is responsible for the creation and communication with the underlying, local sensor network, only registers the IP (v4 or v6) or MAC addresses. It then proceeds to generate a 128 bit random, unique identifier, which is mapped to the local address and then reported to the RMW alongside the template identifier reported by the newly registered RD. The RMW links and stores the reported unique identifier to the device template, and then proceeds to expose the newly added virtualised RD (we will refer in the remainder of this paper to both the virtualised and the physical RERUM devices as RD, although the RMW will only access and read to the virtualised representation) to relevant subsequent queries, based on the template's features.

Withholding the local address from the RMW has two advantages: first, it avoids address collisions in the global device registry. And second, it removes the need of storing relevant traceable identity-related RD data, such as IP or MAC.

In addition, a second optional registration message provided by the RD to the RG and forwarded to the RMW includes a list of allowed data consumers and processing goals. E.g., a sensor might disallow the use of its resources by private corporations, and only limit its usage to state-run agencies:

$$\{128\ bit\ node\ identifier,\ model\ identifier,\\ optional\ privacy\ constraints\ definition\}$$

D. Temporary per-client data buffers

Storing data over a longer period of time poses security and privacy, as well as legal challenges, such as unauthorized access due to malicious intrusions leading to data leakage, long-term data integrity, its segregation, physical disruption, unauthorized access, or complying to multiple legal requirements concerning the storage of personal and private data.

A design choice therefore was to not provide a long-term storage option for the RD readings. Instead, temporary buffers are created for only those readings which are relevant for an application or a client's request, and which are meant only to store the RD data until its consumption. More specifically, only those RDs whose data is of relevance to an end-client have their data sensing and transmission enabled, data which is then temporary stored in a non-persistent buffer only accessible by a single specific end-client. The administrator of the middleware can also adjust the time to live flag for such data buffers, e.g. for them to only contain the readings of the last five minutes, if those older are not of relevance or considered to be obsolete.

The removal of the buffers occurs automatically once the end-clients activity ends. Even in the cases in which multiple clients issue similar data requests, data is duplicated and stored in the per-client queue. The implementations based on a third-party solution, namely making use of the RabbitMQ framework [8], integrated into the RMW. The messaging queues act as buffers, and have both the time to live flags and the auto-expiration flags set: the former ensures that older sensor readings will automatically be removed after a configured period, while the latter ensures that after a certain inactivity period, the queues themselves are deleted.

In order to comply with privacy, confidentiality and legal requirements, the access to the RD data (and more directly, enabling of the sensing and transmission of data on the RDs themselves) is verified during the process of evaluating each client's discovery request through the Security Server.

E. Feature of interests

Features of interest (FOIs), are digital representations of a physical or geographical object. FoIs logically represent the entity which an RD (or a group of RDs) is able to observe or manipulate through sensing or actuating. FOIs are already part of the OpenIoT middleware. More specifically, the FOIs represent optional meta-data associated to each registered sensor, and which may provide additional human readable information for each available sensor.

In RERUM, the FOIs take a central role in the registration and the discovery of RDs. The gateways assign to each registered sensor reported to the RMW a set of one or more FOIs. This assignment by the RGs of FOIs to the devices can be either (i) automatically, case in which the FOIs in the gateway's reach are by default assigned to each registered sensor; or (ii) by using a gateway-owner defined configuration file which maps certain sensors types to only specific FOIs.

Each FOI can be linked to one or more RGs (an intersection, for example, can be observed by a set of nodes belonging to multiple gateways) - linking which, in the current version of the RERUM, is realized by configuring either the RMW by an administrator, or by a gateway owner through per-RGs configuration files.

F. The Stream Processor

The Stream Processor (SP) enables the execution of functions such as data aggregation, the computing of mean, minimum or maximum values and other window functions over data streaming from a set of RDs. Thus, it is possible to provide refined and filtered information or even to take decisions based on the particular requirements of client applications.

RERUM's SP is a specific implementation of the generic and scalable architecture for distributed Complex Event Processing presented in [15] . This architecture and therefore the SP is based on a Complex Event Processor (CEP) engine which consists of a state machine which executes the incoming events based on a set of rules defined in a declarative language (called DOLCE) and it is used to correlate the events that are received. The particular implementation that is part of

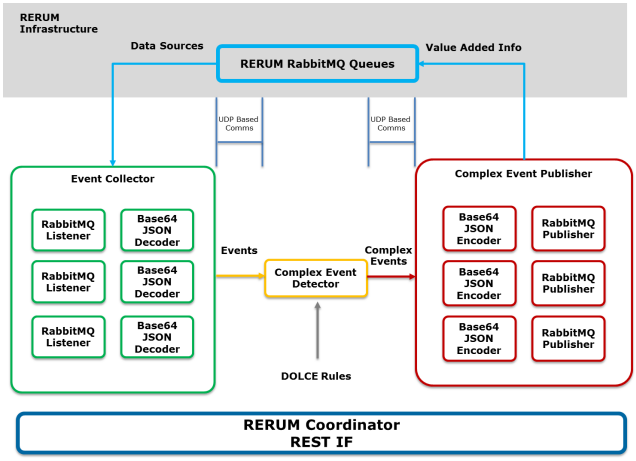

Fig. 2. Stream Processor architecture

RERUM architecture is described in Fig. 2. The Stream processor exposes a REST API that allows defining the rules (functions) that will be applied to each data stream, which is associated to a Discovery request performed by a client application regarding a FOI that groups different RDs. The Event Collector subscribes to the corresponding RMW queues in order to retrieve the data and send them to the CEP engine in the appropriate internal format. At the same time, the Complex Event Publisher listens to the output of the CEP, forwarding the added value information to the corresponding MW queues.

This flexible solution enables new functions to be seamlessly integrated as part of the RERUM environment since it only requires that the RMW accepts the new method (e.g., count data) as part of the sensors Discovery interface and passes a new DOLCE rule to the SP.

G. Data discovery

The FOIs play a determining role in the RD discovery process. More specifically, each discovery process consists of a triple which specifies (i) the FOI which the end-client intends to observe, freely available from the middleware under certain restrictions (i.e. public or private), (ii) the type or types of observations of the FOI, as units: Celsius for temperature, dB for noise etc., and the (iii) type of source processing the middleware is to apply to the raw data: average, minimum or maximum values over a time window:

$$\{FOI\ Name, unit : \{Celsius,\ dB\}, source : average\}$$

The RMW, having FOIs linked to all the registered sensors, as well as complete description of each sensors instance through its linked device template, is able to identify (and enable) relevant RDs, forwarding their date to the SP, its output then being stored in the client's data buffer. When multiple readings are requested, alongside the stream processor output values, an indicator of the type of data is also provided - determining, for example, which of the readings reflect temperature and which of them reflect the sound intensity.

Similarly, for federation, the triple replaces the source option with the federation which is to be run:

$$\{FOI\ Name, unit : \{Celsius,\ dB\}, Federation\ Name\}$$

In the case of federation, their execution involves two steps: first, of checking whether the specified FOI indeed is observed by all the required RDs and, if so, to execute the federation logic. This can be performed either through the SPIN [7] proposed standard which defines the representation of SPARQL and constraints, or through the use of more complex integrated third party framework - the JBoss SwitchYard [9].

The scope of choosing the FOIs, units and source processed as the output, respectively, is in avoiding leaking any information to the end-client regarding the actual RDs providing the readings, as well as their number and the frequency with which they perform the sensing. It therefore provides unlinkability [5] under the assumption that each type of measurements is supported by a significant large number of sensors (the l-Diversity [6] requirement).

H. Actuator discovery

The chosen approach proves challenging when discovering and controlling actuators is required. In particular, being unable to determine or address single RDs implies end-clients unable to precisely address single specific entities.

The middleware is therefore limited to two possible courses of actions: (i) for the cases in which the discovery process identifies multiple RD actuators of the same type (unit) for a specific FOI, any actuator command would be propagated to all simultaneously. (ii) To only accept discoveries and expose RD actuators when there is only a single relevant actuator unit satisfying the discovery request. Although limiting, the second option mainly contributes to the overall safety of the environments controlled by the middleware, since the invocation of a command would affect only one (expected) actuator. This representing a controlled scenario, in contrast to the first described action, in which the number of affected devices would be unknown.

RERUM has therefore decided to implement the second option, and to offer the alternative invocation of (more) complex federation templates when the controlling of the RD actuators might be performed under such automation.

I. Signatures for Integrity

Integrity is a security property of the communicated message (data or command) which is violated whenever the message becomes modified in an unauthorized way, without being detectable [11]. While unauthorized modifications that occur due to transmission errors (accidental) are usually detected by checksums [12], an active attack however requires a cryptographic level of integrity protection – digital signatures.

In order to prevent those attacks integrity protection mechanisms must be employed already on the RERUM device when it sends or receives data. For example if the device receives important commands like firmware updates via over-the-air (OTA) programming. RERUM thus empowered the devices

with the required digital signature capabilities, which proved to be possible for the RDs using elliptic curve cryptography (ECC) [13], [14]. RERUMs architecture allows end-to-end integrity protection by signing data using a JSON Sensor Signature (JSS) [14] generated on the constrained device itself (enabled by the MW by issuing a COAP request for JSS signed data). Using the initially boot strapped credentials, e.g. from a deployed flat PKI, the MW can authenticate the origin (e.g. to identify a specific RERUM device). The per device key strongly authenticates the origin, verifies that it was not maliciously tampered with and also greatly reduces the impact of key extraction attacks. Relying on cryptographically strong — ECC at least 192bit — authentication enables the MW to base its trust scoring on the identified origin and on the knowledge that data was not changed without authorization after the signature was generated on the RERUM device. RERUM's digital signature thus rules out all network and data based attacks carried out by parties with a man-in-the-middle (or privileged data flow) position. Depending on the need the MW does signed communication transparent or in an end-to-end fashion for the application. When transparent the application does not see that the actually signed values or commands. However, then the protection terminates inside the MW. Hence, RERUM strongly suggests and supports end-to-end message level integrity. The signature can then be validated by any application higher up in the IoT data-processing chain, e.g. the users smartphone or the application server. The verification key is public and can be distributed by the MW on request. Finally, since there may be selected and well-defined modifications required to packets due to privacy and confidentiality concerns, RERUM also foresees the use of redactable [17] and sanitizable signatures [18].

IV. Conclusion

This work presented the key RERUM-specific additions of the RMW implementation, based on one of the early OpenIoT MW releases, whose goal is to improve the security and privacy preserving capabilities of the MW and to add the basic functionalities upon which more advanced techniques can be build. Having the per-client data queues, for example, one can really distinguish and isolate the data that are sent to each client, and safely process it accordingly (e.g. by removing sensitive data from clients unauthorized to access certain values). Additionally, the removal of the identifiers of the sensors and their abstraction in homogeneous template-based representation significantly improves the system privacy, since the users or MW operator will not be able to identify exactly which sensor is located where and which one is sending data. At the same time though, the reliability and trustworthiness mechanisms ensure that the data are both reliable and trustworthy (leveraging on encryption and integrity protection mechanisms). For MW–to–gatway communication, RERUM specifies a virtual private network (VPN) connection, in order to ensure that only authorized gateways are sending data to the RMW in a secure way. An access control mechanism integrated with the MW is also in place.

Computationally, the observable performance-security tradeoff at the middleware layer incurred by the introduction of the security mechanisms is insignificant (less than five percent). A more significant impact, as expected, is reflected at the underlying sensor network level (RERUM devices and local gateways). Since this falls outside the scope of the paper, we refer the reader to the detailed RERUM report on the topic in [19]

Features of the RMW that have not been implemented yet are mostly related with enabling users to adjust privacy and security policies. This can be easily integrated in the RMW, which provides capabilities for activation and de-activation of data collection using a simple mechanism for starting or canceling observations via CoAP commands that originate from the MW and go through the gateway to the devices. That way, a user could be able to de-activate the gathering of data from one of his devices if he changes dynamically his privacy policies. Finally, user privacy is also ensured via the device and data anonymisation that exist on the MW through the device registration scheme and the fact that the data that arrive at a client have no identifiers for the origin of the data.

Acknowledgment

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no 609094.

References

[1] O. Vermesan, P. Friess, *Building the Hyperconnected Society: IoT Research and Innovation Value Chains, Ecosystems and Markets*, vol. 43, ed. River Publishers, 2015.
[2] E. Z. Tragos, et. al, *Enabling reliable and secure IoT-based smart city applications*, IEEE PERCOM Workshops, 2014.
[3] E. Z. Tragos, et. al., *Final System Architecture*, RERUM D2.5, 2015
[4] P. Dimitropoulos, et. al., *OpenIoT Detailed Architecture and Proof-of-Concept Specifications*, OpenIoT D2.3, 28 March 2013.
[5] A. Pfitzmann and M. Hansen, *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and identity Management* , v0.32, 2010.
[6] A. Machanavajjhala, et al., *L-diversity: Privacy beyond k-anonymity*, ACM Trans. Knowl. Discov. Data 1, 1, Article 3, 2007.
[7] H. Knublauch and J. Hendler and K. Idehen, *SPIN - Overview and Motivation*, W3C Member Submission, 2011
[8] Web Resource: rabbitmq.com. Retrieved Sept. 22, 2016.
[9] Web Resource: switchyard.jboss.org. Retrieved Sept. 20, 2016.
[10] A. Akbar, et al., *Context-aware stream processing for distributed IoT applications*, WF-IoT, IEEE, 2015.
[11] R Shirey, *RFC 4949Internet Security Glossar*y, 2007.
[12] W. Peterson and D. Brown, *Cyclic Codes for Error Detection.*, Proc. of IRE, 49(1):228235, 1961.
[13] M. Mössinger, et. al., *Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device*,Proc. of 5th workshop on IoT-SoS, IEEE, 2016.
[14] H. C. Pöhls. *JSON Sensor Signatures (JSS): End-to-End Integrity Protection from Constrained Device to IoT Application*, Proc. of esIoT workshop at the IMIS 2015, IEEE, 2015.
[15] A. Akbar, et. al., *Context-aware stream processing for distributed IoT applications*, in Proc. of WF-IoT, 2015.
[16] B. Aziz, A. Arenas, and B. Crispo, *Engineering secure Internet of Things systems*, IET book, 978-1-78561-053-0, 2016.
[17] H. C. Pöhls and K. Samelin, *Accountable Redactable Signatures*, Proc. of ARES, IEEE, 2015.
[18] H. C. Pöhls, et al., *Malleable Signatures for Resource Constrained Platforms*, Proc. of the 7th IFIP WISTP, Springer, 2013.
[19] V. Angelakis et. al., *Analysis and Evaluation of system performance and scalability*, RERUM D4.3, 2016