Proposal of a Secure, Deployable and Transparent Middleware for Internet of Things

Hiro Gabriel Cerqueira Ferreira, Rafael Timóteo de Sousa Júnior, Flávio Elias Gomes de Deus, Edna Dias Canedo

Electrical Engineering Department, University of Brasília – UnB – Campus Darcy Ribeiro – Asa Norte – Brasília – DF, Brazil, 70910-900.

hiro.ferreira@gmail.com, desousa@unb.br, flavioelias@unb.br, ednacanedo@unb.br

***Abstract* — This paper proposes a security architecture for an IoT transparent middleware. Focused on bringing real life objects to the virtual realm, the proposed architecture is deployable and comprises protection measures based on existent technologies for security such as AES, TLS and oAuth. This way, privacy, authenticity, integrity and confidentiality on data exchange services are integrated to provide security for generated smart objects and for involved users and services in a reliable and deployable manner.**

Keywords — Internet of Things; Transparent Middleware; Security.

I. INTRODUCTION

Since the mainframe era, when multiple users shared one processor, computing has evolved throughout the personal computing era, when one processor was available for each user, and nowadays a new era rises proposing multiple processors for a single user [1]. Miniaturization and reduction of power consumption also has been allowing these processors to be integrated into common objects and heterogeneous devices. Wireless communications and the Internet enable these processors to exchange data, and send and receive commands to act in their environment.

These communicating things are commonly named smart objects while the expression Internet of Things (IoT) refers the computing paradigm that organizes data exchange among these things and other distributed computing entities so as to provide services to end-users [2,3]. To become fully applied, this paradigm requires a standardization effort regarding issues such as: hardware and software shells to convert plain objects into smart objects, object identification, communication channels between smart objects, and common communication language between smart objects. The expected standards must respond to requirements regarding network scalability, mobility of nodes, diversity of things and overall security. Since IoT presupposes the Internet as its main data exchange infrastructure [4], the unique identification for objects is also a problem due to ever-limited availability of IPv4 addresses.

Nevertheless, a common standard architecture is still lacking for IoT, an issue that motivates the research on architectures and middleware aimed at a better understanding of IoT requirements and the agreement on technical choices for its real enablement. Amongst these research themes a relevant one regards the issues of information security, particularly authenticity, integrity and confidentiality of communications, as well as the preservation of user privacy.

Based on a previous proposal regarding IoT middleware and usage scenarios [5,6], this paper presents and discuss a security model for IoT envisioning a scenario where devices are able to provide automatic services to users without necessarily having human intervention [1,2] and considering the scalability necessary to cope with an important usage growth rate [7], while preserving user´s privacy [8]. Besides, it presumes low energy consumption and low computational resources in smart objects, which constrains security logic and algorithms.

The remainder of this paper is organized as follows. Section II briefly discusses IoT security issues. Section III presents existing IoT middleware and associated security technologies. Section IV describes our proposed security model. This paper is concluded by presenting in Section V some open research topics related to the proposed architecture.

II. INTERNET OF THINGS AND SECURITY: RELATED WORK

This section presents security technologies that are commonly integrated into proposed IoT middleware.

A. AES-CCM

AES-CCM is the mode of the Advanced Encryption Standard when used with counter mode and cipher block chaining message authentication code (CBC-MAC) [9]. This mode is designed to providing privacy and authenticity in constrained environments and compact implementations [10]. Most wireless sensor and actuator networks (WSAN) use ZigBee [11], a technology that is commonly implemented in IEEE 802.15.4 chips which implements AES-CCM by default. Since this technology has been vastly used to secure WSANs with scalability [11], it is most suitable for IoT scenarios.

B. TLS

Transport Layer Security (TLS) is a channel-oriented protocol designed to secure data exchange between applications [12]. It uses symmetric (AES) and asymmetric (RSA) cryptography, to ensure privacy, authenticity and integrity in a communication session between client and server. HTTP over TLS, also known as HTTPS [13], is largely used over the Internet because it allows secure data

exchange between HTTP servers and clients, such as web browsers and web page servers.

Considering that the Internet is the natural infrastructure for data exchange between distributed IoT devices, TLS seems to be the choice to secure these connections among devices and applications, while allowing interoperation throughout the World Wide Web [14].

C. *oAuth*

oAuth 2.0 is a low complexity authentication protocol that allows an entity to access resources belonging to another entity within a limited scope defined by the resource owner through an issued access token [15]. This authentication suite is very scalable and is used for large enterprises such as Google, Facebook, Windows live, GitHub and others [16].

Because IoT scenarios might need to authenticate users and applications before allowing them to provide services, every entity should be identifiable. Also, good practice indicates that it should not be necessary for an entity to use other entity credentials to request or to do a service in its name. Since IoT needs simple and robust security with high scalability, oAuth is a god candidate to help with authentication and authorization.

D. *Basic middleware*

Besides the infrastructure to communicate with the external world, a IoT device needs to employ a common language, so as to understand and be understood by other entities.

The Hydra middleware, lately known as SmartLink [17], is designed for enabling any object to communicate with the world around it. This middleware provides a deployable service-oriented architecture with embedded security. Although it accomplishes important milestones, some core issues are still to be resolved in this middleware. To use Hydra in a device it is necessary to load hardcoded scripts on Hydra's main software (proxies). Also, applications, which are going to use these devices, need to talk in the proprietary Hydra's protocol.

Other publicized IoT middleware, such as IoT-A [18] and iCore [19], are yet in their requirements phase for future implementations. They promise to be scalable, allowing discovery, eventing and encompassing heterogeneity of devices, but to our best knowledge are not yet deployable.

In a previous work [5,6] we proposed a middleware aimed at providing a solution to the issues regarding the cited middlewares. Our proposed architecture is transparent to end devices and end applications. Devices and their services are added and managed within the architecture through simple web forms. Thus, no additional code is needed to be loaded. On the other side, end applications can have access to all devices functionalities through REST [20] and UPnP [21] application programming interfaces (API). A set of existing technologies, such as UPnP, REST, ZigBee and IP, are gathered so as to allow the internal and external functioning of the middleware. That makes it understandable and extensible by other researches. Also it was designed to cope with scalability, diversity, as well as devices computational and energy limitations.

This basic middleware is composed by four main entities: end application, master controller, slave controllers and end devices, as shown in figure 1.

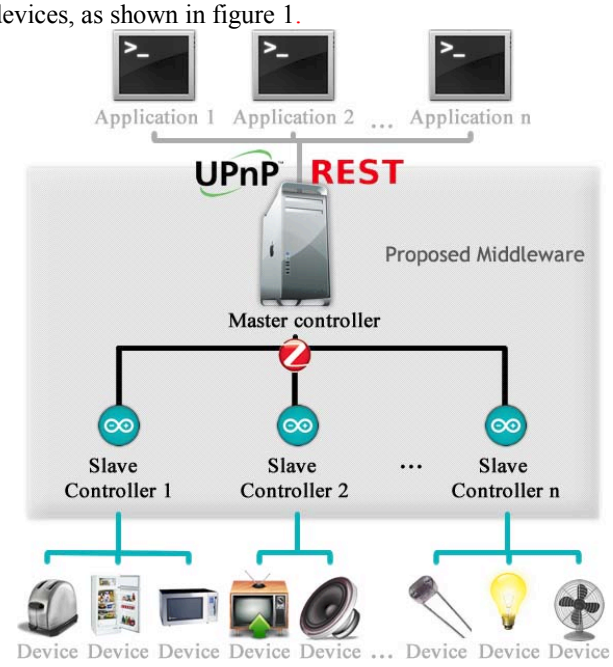

Figure 1 – Basic middleware logical architecture

The master controller is responsible for abstracting end devices, transforming them from plain objects to smart ones. This entity is also responsible for bringing slave controllers mesh networks to the regular Internet world. It holds the APIs and is responsible for the conversion of semantic requests onto a language understandable by devices. It communicates with applications through the mentioned REST and UPnP APIs. Slave controllers hold codes as simple as "write bits to a pin" and "read bits from a pin" and communicate with master controllers through ZigBee.

Indeed, the master controller can be implemented in any device, from a Raspberry Pi platform [22] to a cloud based server with cache, load balancers for applications and databases, since this controller is required just have the ability to receive UPnP and REST requests and translate/send commands to a slave controller for reading and writing values to pins through ZigBee. This makes this entity extremely scalable.

Slave controllers can be simple or complex PCBs with batteries, Arduino boards [23], Raspberry Pi platforms or Cloud based Servers. It can be any device capable of receiving serialized information and execute or receive information from objects. Also, this makes it a very scalable entity. The architecture implements all UPnP functions, such as device addressing, discovery, description, controlling, eventing and presentation, all these operations being realized in XML, SOAP, GENNA and the SSDP grammar [21].

III. PROPOSED SECURITY MODEL

As described in sections above, AES, oAuth, and TLS are very well defined and accepted security technologies.

They can be integrated to our base IoT architecture so as to fulfill this architecture security requirement.

As described, the master controller has no computational or energy constraints oppositely to end applications and slave controllers which are very heterogeneous entities and may present computational or energy limitations. Most end devices are passive plain objects controlled by slave controllers, and their security is out of our scope, as we aim to protect interactions by smart objects generated by our middleware above the plain objects.

This section covers the middleware internal and external security aspects, as well as authentication and smart objects access control list (ACL). Internal security comprehends communication between master and slave controllers, while external security encompasses communication between the master controller and applications or between the master controller and system users. Authentication and role based ACL are proposed for applications and users interactions with the master controller, which in turn generate commands that will be applied to end devices by slave controllers.

A. Internal Security

Slave controllers are supposed to be entities with few energy and computational resources so they must not handle too much policies and information processing tasks. This is the reason why this entity only deals with cryptography and integrity checks that comprise the ZigBee technology, which is already embedded on slave controllers [5].

ZigBee's AES-CCM with 128 bits key answers this requirement. It allows secure communication by providing confidentiality with block cyphering, integrity with the message integrity code (MIC) and authenticity with the message authentication code (MAC) [10].

Configuring the ZigBee modules in the master and slave controllers so as to only accept connections with encryption enabled will provide security to the connection between these devices. To make this possible, it is necessary to previously share the network key, and set ZigBee parameters "enabled encryption" (EE) to 1 and "security police handler" (EO) to 0. With this configuration, only nodes that already know the key will be able to join the network of master and slaves controllers, and consequently communicate with end applications.

Periodically, the master controller sends a new network key to its slave controllers, which are required to save this key persistently to their ZigBee modules. This operation turns the ambient even more secure against eavesdroppers.

B. External Security

Defined as the middleware most intelligent entity, the master controller centralizes most of the logical procedures and executes most of the hard processing tasks [5]. It is central middleware processor, the router between applications and slave controllers, as well as provider of abstractions of end devices.

Since it is the master controller the entity that has contact with the external world, it holds the role of securing that channel. The middleware interacts with end applications only through UPnP and REST API [5,6], both using HTTP. For that reason the master controller should perform its unicast communications over HTTPS [13] thus using TLS to enforce security, using AES and RSA to provide confidentiality, and MIC, MAC and SHA-1 provide data integrity.

C. Authentication and Access Control List

For authentication purposes, related both to using and managing the entire middleware, four system user types are proposed: *root user*, *plain object owner*, *smart object user* and *application user*.

Root users can create all the other types of users and manage all smart objects and their services. This special user is supposed to only manage the master controller's abstractions and, thus, cannot use any of the APIs services. This user can login with a browser at the administration URL *http://master_controller_ip/configure* [5] and create, view, edit and delete all system users, all system roles, all slave controllers and all end devices with its services and state variables. His credentials are hold by his session.

Plain object owners are created by *root users* and can create, update, view and remove end devices, system users and user roles. *Plain object owners* can login at the administration URL and they are supposed to physically add new plain objects at already deployed slave controllers, then add and manage these devices at the master controller. For that reason, one such user can only access entities that he has created.

Smart object users can login at the administration URL and actually use services to which they have allowance, as given by *root users* and *plain object owners*.

But, if an *Object user* wants to use other end applications, before the middleware start exchanging information with end devices, these applications are required to authenticate with the Master controller.

Applications trying to perform actions and receive updates of end devices in the name of *smart object users* should not know user credentials and must only access services to which the user allows them to. The authentication methods suite proposed by oAuth 2.0 aims to keep the resources and their owner protected. In the context of our middleware that feature represents the protection for end device's services and *smart object users*. End applications will be able to get authorization grants without using the resource owner's credentials, meaning to the *smart object user* that his username and password are safe. For that reason, the oAuth's Implicit Authorization Grant [15] method is adopted for this case.

To get the authorization grant, end applications have to first redirect the end user to the master controller URL so that a *smart object user* has to login. If he logs in with success, the master controller will ask him to allow the named application to access the named service. If allowed by the user, an access token is delivered to the application. If the user does not allow access to the service, the requester will be redirected with an empty token. With this methodology, the master controller can keep track of what the application has being doing for further audits, if necessary.

With the specified user types, and the authentication procedures (the regular login and the two oAuth token

methodologies), it is possible for the master controller to limit the access of all middleware resources to only allowed entities. The system users can be controlled separately and each one has its very specific purpose, as show in early paragraphs. The role based methodology copes with the diversity of end applications, end users and devices, because it simplifies the process to grant authorization (or not) to end users and applications on multiple devices at the same time by only adding or removing a user from a role [25].

IV. CONCLUSIONS

The themes of implementation, middleware and security for Internet of Things are currently the focus of many researchers. The effort of standardization is yet being developed.

Our previous work proposed a transparent and deployable IoT middleware, with detailed use scenarios, and this paper specifies the related security architecture, mostly based on existing and robust technologies. The resulting secure middleware allows the implementation of IoT environments, its improvement by other communities and researches, and moves the IoT paradigm closer to reality.

Regarding specific security aspects, this work details how communication and authorization should occur in order to provide privacy, authenticity, integrity and confidentiality to users, applications and devices. The proposed model considers and solves issues on scalability, diversity of devices and applications, simplicity, robustness and keeps possible the interactions with entities of low computational and energy resources.

The next steps include providing the wrap for devices that already communicate using other technologies. These devices should be put under the master controller realm to get them available under the middleware's API. Technologies such as DTMF, UPnP, Power Line and 6LoWPAN allow devices to communicate, but all in different languages. Harmonizing these devices under master controller rules would allow applications to communicate with them always using the same language. Another important issue to be solved is the mobility for end devices and, in that case, how to keep them uniquely identifiable and how to automatically transport their configurations and permissions to other master controllers.

ACKNOWLEDGMENT

The authors wish to thank the Brazilian CAPES Agency and the Brazilian Ministry of Planning, Budget and Management for their support to this work.

REFERENCES

[1] Weiser, M. (1991). The Computer for the 21st Century. Scientific American, Vol.265, no 3, pp 94-104.

[2] Giusto, D.; Iera, A.; Morabito, G.; Atzori, L. (Eds.) (2010). The Internet of Things, Springer. ISBN: 978-1-4419-1673-0.

[3] Ashton, K. (2009). That 'Internet of Things' Thing. RFID Journal, www.rfidjournal.com/article/print/4986

[4] Weber, R. H (2010). "Internet of Things – New security and privacy challenges", Computer Law & Security Report, Vol.26, Issue 1, pp 23-30, Elsevier.

[5] Ferreira, H. G. C.; Canedo, E. D.; Sousa Júnior, R. T. (2013). IoT Architecture to Enable Intercommunication Through REST API and UPnP Using IP, ZigBee and Arduino. *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp.53,60. doi: 10.1109/WiMOB.2013.6673340.

[6] Ferreira, H. G. C.; Sousa Júnior, R. T.; Canedo, Edna D. (2014). A Ubiquitous Communication Architecture Integrating Transparent UPnP and REST APIs, *International Journal of Embedded systems*, Special issue on IEEE/IFIP EUC 2013 Embedded and Ubiquitous Computing, 13-15 Nov. 2013. Inderscience publishers.

[7] Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (2013) 1645-l60, Elsevier.

[8] Atzori, L.; Iera, A.; Morabito, G. (2010). The Internet of Things: A survey, Computer Networks 54 (2010) 2787-2805, Elsevier.

[9] Whiting, D.; Housley, R.; Ferguson, N. Counter with CBC-MAC (CCM), RFC-3610. Available at: <http://tools.ietf.org/html/rfc3610> accessed on Feb 18, 2014.

[10] McGrew D.; Bailey D. AES-CCM Cipher Suites for Transport Layer Security (TLS), RFC-6655, July 2012. Available at: <https://tools.ietf.org/html/rfc6655> accessed on Feb 18, 2014.

[11] ZigBee Alliance. Available at: <http://www.zigbee.org> accessed on Feb 18, 2014.

[12] Dierks T.; Rescorlar E (2008). The Transport Layer Security (TLS) Protocol Version 1.2, RFC-5246. Available at: <http://tools.ietf.org/html/rfc5246> accessed on Feb 18, 2014.

[13] Rescolar E. (2000). HTTP Over TLS, RFC2818. Available at: <https://tools.ietf.org/html/rfc2818> accessed on Feb 18, 2014.

[14] Iwendi, C. O.; Allen, A. R. (2012). Enhanced security technique for wireless sensor network nodes. IET Conference on Wireless Sensor Systems (WSS 2012), pp.1,5, 18-19 June 2012. doi: 10.1049/cp.2012.0610

[15] Hardt D. (2012). oAuth 2.0h Authorization Framework, RFC-6749. Available at <http://tools.ietf.org/html/rfc6749> accessed on Feb 18, 2014.

[16] oAuth, open standard for authorization, 2014 Available at: < http://oauth.net/2/> accessed on Feb 18, 2014.

[17] Hydra Middleware/Link Smart Middleware, 2014. Available at: <http://www.hydramiddleware.eu> accessed on Feb 18, 2014.

[18] IoT-A, Internet of Things Architecture, 2014. Available at: <http://www.iot-a.eu> accessed on Feb 18, 2014.

[19] iCore Project, 2014. Available at: <http://www.iot-icore.eu> accessed on Feb 18, 2014.

[20] Stirbu, V., "Towards a RESTful Plug and Play Experience in the Web of Things," Semantic Computing, 2008 IEEE International Conference on, pp.512-517, 4-7. Aug. 2008. doi: 10.1109/ICSC.2008.51

[21] Universal Plug and Play Forum, Understanding Universal Plug and Play, 2000. Available at: <http://www.upnp.org/download/UPNP_understandingUPNP.doc> accessed on Feb 18, 2014.

[22] Raspberry Pi, 2014. Available at: <http://www.raspberrypi.org/> accessed on Feb 18, 2014.

[23] Arduino, 2014. Available at: <http://www.arduino.cc/> accessed on Feb 18, 2014.

[24] JSON, 2014. Available at <http://www.json.org>, accessed on Feb 18, 2014.

[25] Miorandi Daniele; Sicari, Sabrina; Pellegrini, Francesco De; Chlamtac Imrich; "Internet of things: Vision, applications and research challenges", Ad Hoc Networks 10 (2012) 1497-1516. Elsevier, April 2012.