DPWSec: Devices Profile for Web Services Security

Sebastian Unger and Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering
University of Rostock, Germany
Email: sebastian.unger@uni-rostock.de, dirk.timmermann@uni-rostock.de

***Abstract*—As cyber-physical systems (CPS) build a foundation for visions such as the Internet of Things (IoT) or Ambient Assisted Living (AAL), their communication security is crucial so they cannot be abused for invading our privacy and endangering our safety. In the past years many communication technologies have been introduced for critically resource-constrained devices such as simple sensors and actuators as found in CPS. However, many do not consider security at all or in a way that is not suitable for CPS. Also, the proposed solutions are not interoperable although this is considered a key factor for market acceptance.**

Instead of proposing yet another security scheme, we looked for an existing, time-proven solution that is widely accepted in a closely related domain as an interoperable security framework for resource-constrained devices. The candidate of our choice is the Web Services Security specification suite. We analysed its core concepts and isolated the parts suitable and necessary for embedded systems. In this paper we describe the methodology we developed and applied to derive the Devices Profile for Web Services Security (DPWSec). We discuss our findings by presenting the resulting architecture for message level security, authentication and authorization and the profile we developed as a subset of the original specifications. We demonstrate the feasibility of our results by discussing the proof-of-concept implementation of the developed profile and the security architecture.

***Keywords*—Applied Cryptography; Authentication; Cyber-Physical Systems (CPS); DPWS; Intelligent Environments; Internet of Things (IoT); Usability**

I. Introduction

Security is a crucial design goal for visions such as the Internet of Things, Smart Homes or Ambient Assisted Living that all heavily utilize cyber-physical systems. It is recognized, that these technologies will not reach the consumer market unless security issues are solved in a user-friendly way. Nonetheless, when new technologies are developed, security is often considered as making things expensive and complicated.

Another enabling key factor is interoperability. Consumers hesitate to adapt new technologies as they fear the so-called vendor lock-in. That is, components bought once are incompatible with components from different manufacturers. Considering that interoperability needs to be provided on functional and security level does not render the challenge easier.

It would thus be desirable to provide a security framework that can be adapted and employed by different communication technologies to foster secure interoperability between them. It would be rather undesirable though to completely reinvent a whole security framework from scratch as this is cumbersome and error-prone. A sound approach should reuse an existing time-tested solution that is already widely deployed and accepted in a domain as closely related as possible.

The candidate of our choice is the Web Service Security specification suite, because it provides a comprehensive security framework for loosely-coupled distributed systems. While the underlying communication technology (Web Services) is already adapted to the realm of embedded devices by the means of the Devices Profile for Web Services (DPWS), the security specifications are designed to suit desktop PCs and web servers, not critically resource-constrained devices as found in CPS. In this paper we describe how we isolated the core concepts of the WS Security specifications and readjusted them for use in IoT- or AAL-scenarios and how we identified and eliminated specification parts that are not suitable or not necessary for CPS-incorporating scenarios.

In the remainder of this paper we describe the state of the art as well as the basics of DPWS and the WS Security suite in sct. II. We then describe which aspects were essential for deciding which core concepts to extract from the specifications and adapt to the Devices Profile for WS Security in sct. III. DPWSec and its core concepts are discussed in IV. We implemented a proof-of-concept and describe it in sct. V before concluding our paper in sct. VI.

II. State of the Art & Basic Principles

We give a very brief survey over existing communication technologies we considered and show their security features. After that, we give a short introduction to Web Services, the Web Service security specification suite and DPWS.

A. Existing middleware technologies

In the past, numerous communication middlewares for smart environments have been introduced. Universal Plug and Play (UPnP) is a technology for spontaneous autonomous device connection and together with its A/V profile DLNA it is widely deployed in the consumer area. Selén states in [1] that due to its optional nature, the security profiles in UPnP are rarely implemented. The once planned-to-be successor of UPnP named DPWS is covered in more detail below.

Besides these specifications, there is a plethora of middlewares that were proposed by academic research within the past decade. Only few of them exhaust security (e.g. [2]), most of them cover only selected aspects (e.g. [3], [4]), employ existing techniques that do not suit the demand of embedded devices (e.g. [5], [6]) or completely ignore security (e.g. [7], [8]). If security is covered, interoperability to related technologies is never a concern.

B. Web Services

Web Services are the most-used technology to implement service-oriented architectures (SOA) in enterprise environments, as they are well-suited to model business processes. Their design goal is to couple functional units as loosely as possible to provide high degrees of flexibility, reusability and exchangeability. They rely on the SOAP protocol which is based on XML. Besides the base core specifications there exists a plethora of different complementing documents that define e.g. dynamic discovery, management, streaming functionality, asynchronous messaging patterns, description or metadata exchange functionality. To provide a certain degree of interoperability, so-called profiles exist that narrow down subsets of specifications.

C. Web Services security specification suite

In enterprise environments communication security plays a vital role. The following security-related Web Service specifications exist to guarantee the four basic security objectives confidentiality, integrity, authenticity and authorization. The base specification *WS-Security* ([9]) provides message level security by defining how to embed cryptographic signatures and cipher texts into SOAP messages and how to provide message freshness through timestamps. Token profiles complement this specification by defining how to embed e.g. SAML-tokens into SOAP messages. A message can only be secured within a certain security context, that is within a trust relationship with established credentials such as a session key. *WS-SecureConversation* ([10]) defines the format for Security Context Tokens (SCT) to be exchanged to establish such a context. *WS-Trust* ([11]) defines the Security Token Service (STS) as a special entity in a network. It is a trusted party in a network that can issue, renew, cancel and validate security tokens such as an SCT or SAML tokens for authorization. When federating security related information among different administrative domains to e.g. provide single-sign-on solutions for separated enterprise networks, *WS-Federation* ([12]) provides a comprehensive framework. Among its features are definitions for different trust models, an STS enhancement to request authorization, federated sign-in and sign-out, pseudonyms and enhanced metadata definitions. *WS-Policy* ([13]) defines the designated way to express requirements and assertions for a service. Policies can indicate which cryptographic algorithms are supported or which authentication methods are acceptable.

D. The Devices Profile for Web Services

What was planned to be the successor of UPnP became the independent technology DPWS ([14]) and brings Web Services to resource-constrained devices. While it was planned for printers or scanners in enterprise networks, DPWS increasingly gains momentum in medical appliances ([15]), Wireless Sensor Networks ([16]) and the automation industry ([17]).

DPWS *Devices* host *Services* that offer *Operations* to be invoked by *Clients*. Due to the underlying SOAP Web Services, DPWS provides a classic request response messaging pattern, as well as asynchronous communication by employing WS-Eventing. A central registry is not necessary as DPWS supports dynamic discovery. Also, it allows Clients to retrieve metadata about the Device itself as well as its hosted Services.

The security concept of DPWS relies on profiles that can be understood as a set of rules two parties agree on prior to communicating. One optional profile is part of the DPWS specification. It employs Transport Level Security (TLS) to optionally secure communication. As this is not applicable for discovery messages that are sent via UDP multicast, these messages can be signed by a compact signature mechanism described in WS Dynamic Discovery. The necessary credentials are provided by X.509 certificates and their matching private keys. The process of exchanging certificates is not specified.

In addition, the literature proposes several profiles for certain use cases. The profile introduced in [18] is vulnerable against MITM-attacks, and the solution from [19] imposes vulnerability for replay-attacks. There is one approach for DPWS-security in office spaces ([20]) and another profile proposes a profile for automotive infotainment scenarios ([21]). They both rely on technologies such as X.509-Certificates and public-key-infrastructures that do not take into account the presence of critically resource-constrained devices in a network.

III. Methodology to develop DPWSec

In this section we describe the methodology we developed to isolate DPWSec from the WS Security specifications suite, so we explain the basis for our decision on how to alter the original specifications.

A. Requirements analysis

The first step on our way to develop DPWSec was a thorough requirements analysis. We analyzed suitable attacker models and collected functional requirements from the literature and existing scenarios. For the sake of brevity, only the most relevant final conclusions can be laid out here. First, we realized that all scenarios incorporate devices that can be grouped in either of the following categories: Tiny devices, giant devices and multimedia devices. Tiny (e.g. sensors, simple actuators) means devices with very limited resources, maybe communicating wirelessly and maybe battery-driven and usually have very limited user interface (UI) capabilities. Giant devices (e.g. multimedia servers or settop boxes) are on the other end of the scale. Their resources are virtually unlimited, they usually have a constant power supply and may use wired communication. Their UI capabilities are often limited as well but are likely to be extensible. The third class, multimedia devices (e.g. TV sets, smartphones, digital picture frames, ..), are special in that they are very visible to their users. Their resources usually are limited but not critically. They incorporate and offer a large number of UI capabilities.

Given these three categories, a security framework needs to allow for a very heterogeneous variety of devices in terms of computing power, communication capabilities, communication protocols, energy supply and UI capabilities.

Secure communication with the outside world is a crucial feature, mostly in elderly-care scenarios, where medical data

needs to be exchanged with physicians or in case of an emergency. Also, special care must be taken to make encrypted persistence of sensitive data as easy as possible.

While not as obvious in small apartments, larger office spaces may contain dozens of rooms with dozens to hundreds of smart devices each. It must be possible to organize this plethora of devices in logical administrative groups that can be securely connected. This scalability requirement becomes even more of an issue when it comes to interconnecting different smart homes or different enterprise networks with each other. Finally, when devices get sold or stolen they must reveal as little information as possible and the remaining devices must be informed about the withdrawal.

B. Offloading efforts

The most outstanding property the requirements analysis revealed is the heterogeneity of the devices regarding available resources. The Web Service Security specifications suite however assumes virtually unlimited resources across all participants. Thus, DPWSec should exhaust the original specifications' possibilities to offload efforts from the 'weak' to the 'strong' participants in a network. However, the less powerful devices should not completely rely on the existence of more powerful ones to not introduce single points of failure.

Efforts that can be offloaded very well especially concern trust establishment. Here, a trustworthy strong intermediary may take over fetching, parsing and matching policies to negotiate parameters and ease trust establishment by dynamically bridging existing gaps in trust chains and later shortcut them. Also, authorization decision and propagation can be handled very well by strong participants in a network.

C. Eliminating Specification Parts

Besides finding strategies for offloading efforts, the second important method to carve out DPWSec is simplifying the specifications which in turn simplifies implementation. That is, identifying and eliminating specification parts that are not necessary, not useful or not suitable for our scenarios.

We identified certain patterns in the differences between regular Web Services and WS for embedded devices that led to a good guideline for which specification parts to eliminate.

1) The DPWS Communication Model: The DPWS communication model as depicted in [14] strictly distinguishes a Device hosting services and Clients that invoke the services' operations. This implies that a logical device should not necessarily have client functionality and vice versa. It was a design goal for DPWSec to maintain this separation.

2) Hosting and hosted Services: In the Web Service world, a service is an isolated autonomous entity and represents the top level of the organizational hierarchy to enable the envisioned loose coupling. However, there is a difference for DPWS where a logical Device is represented by its *Hosting Service* that offers several strongly related *Hosted Services*. The hierarchical top level thus is the Device that groups several Hosted Services. Therefore it might be beneficial to consider the services on one Device related and not isolated.

3) Statelessnes: An important design goal of Web Services is loose coupling. A key factor to achieve this goal is statelessness. When a service is perfectly stateless, it can fulfill its task completely with the given parameters and without the need of prior knowledge and its results may not depend on prior requests. Although it is possible to design Web Services to remain completely stateless even for authenticated requests, it becomes rather complex, interferes with the mentioned DPWS Communication model and does not consider the relatedness between hosting and hosted services.

4) No Multihop Security: When it comes to message level security, that is signing and encrypting a message, the according specifications respect the possibility that parts of a message are encrypted or signed for entities different to the immediate receiver so it can simply bypass these parts. On rare occasions, use cases are constructed where this feature could be useful. However, we could not identify any necessity for multihop security in the requirements. As this feature introduces a vast amount of overhead to the specifications as well as to single messages, we decided to drop the according specification parts.

IV. The Devices Profile for WS Security

By applying the developed methodology to the original specifications we could develop both an architecture for a security framework and the profile DPWSec in parallel. We are discussing both in this section, however, for the sake of brevity and comprehensibility we will focus on the security framework's concepts. The complete profile is published online alongside with the open source prototype implementation.

A. General aspects

Before diving into the security-related parts, we propose some general enhancements.

1) Client IDs: In the world of DPWS, only Devices have IDs that they can be addressed with. Clients do not because they never need to be addressed. However, to be identified we specify that clients need unique IDs as well that they provide during authentication and when invoking a service's operation.

2) Resource-saving discovery: When discovering devices in a network, often only trusted devices are of interest. To indicate this, a client sets a header flag and devices only answer if they trust the requesting client. This behaviour is compatible to devices not supporting the profile.

3) Two-stage discovery and description: Discovering devices and fetching their description must happen before authentication and must be marked untrusted. They must be repeated in a trusted way before relying on them.

4) Device classes and types: We distinguish three device types: tiny devices, giant devices and multimedia (or UI) devices. They are identified in the network by their functionality. Tiny devices can be of type "AuthenticationEndpoint" as they need to offer a service for direct authentication. The same holds true for giant devices. In addition they can be of type "Authenticator" and/or "SynchronousAuthorizer". A UI device can be identified by its types "UI-Authenticator" and/or "AsynchronousUIAuthorizer". The related concepts will be discussed in the remainder of this section.

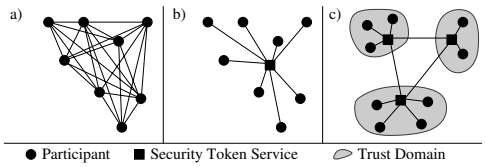

Fig. 1. Trust relationships a)w/o STS b)w/ STS c)w/ STS in trust domains

B. Message Level Security

Most existing communication technologies employ Transport Layer Security (TLS) to establish secure communication channels, including DPWS itself. It is discussed in [22] why TLS is not the optimal choice for embedded device communication. The authors propose the WS Compact Security scheme to embed digital signatures as well as encrypted payloads into SOAP messages and show that this introduces no significant performance drawbacks over TLS. We employ this partial solution as it relies on existing specifications and fits well into our methodology especially regarding eliminating specification parts that are not ideal for embedded devices.

From the original WS Security specification ([9]) we employ the chapters regarding ID References (4), the Security Header (5) and the Security Timestamps (10) to ensure message freshness. The chapters 6 – 9 are not applicable to embedded devices or are superseded by the WS Compact Security scheme.

C. Connection Level Security

WS-SecureConversation ([10]) specifies the Security Context Token (SCT). It represents a trust relationship with related key material that is exchanged during authentication. The specification also defines how to request and issue these tokens and how to renew and cancel them. The chapters 4, 7 and 8 are not applicable or implicitly covered at different places.

D. Authentication for Smart Environments

The specification WS-Trust ([11]) defines an entity in a network that is responsible for issuing security tokens such as an SCT to represent a trust relationship. These entities are called Security Token Services (STS) and may also broker trust between trusted third entities, so not all participants need to directly authenticate each other (fig. 1a)) but only authenticate with an STS (fig. 1b)). It is also allowed for multiple STSes to exist to increase robustness or to segment a network in different trust domains (fig. 1c)). The devices hosting an STS for brokered authentication have the type "Authenticator" and are typically giant devices. Tiny devices can denote one or more Authenticators as their brokers which makes them responsible for brokering their trust relationships.

Brokering trust is only one half of a working authentication framework. The missing half is direct trust establishment between two entities. In existing solutions for embedded systems this relies on X.509 certificates that are installed prior to deployment. This appears feasible for devices from one manufacturer but it is unlikely to work as a manufacturer-independent solution. Instead we decided to base direct authentication on the out-of-band (OOB) exchange of one-time pins. This provides an easy way to authenticate two devices without relying on existing infrastructures and after devices have been deployed. Specifically, we employ a protocol presented in [23] based on an elliptic curve Diffie Hellman (ECDH) key establishment and became part of the IEEE 802.15.6 specification ([24]).

The definitions of WS-Trust are used to issue tokens from direct and from brokered authentication and this way combines these methods. Even tiny devices host a simple STS which only offers tokens for direct authentication and provides the interface to accept established SCTs resulting from brokered authentication. This way all devices offer a unified interface.

There remains an issue with direct authentication: devices need to be able to establish an OOB channel. Not every device can display numerical PINs or provides a way to enter them. Thus we are using a concept described in [25] which uses multimedia devices to translate OOB channels when necessary.

To keep the necessary efforts for tiny devices as little as possible they follow this scheme:

1) If it has its own broker already, request authentication with target from broker. Authentication done. End here.
2) If target has a broker, (in)directly authenticate with it and declare it own broker. Go to step 1.
3) If there is any Authenticator device, (in)directly authenticate with it and declare it own broker. Go to step 1.
4) (In)directly authenticate with target.

(In)direct authentication is depicted by the following scheme:

1) Discover UI-Authenticators in a network that support common OOB channels. Request indirect authentication from UI-Authenticator with target. Authentication done.
2) If no matching UI-Authenticators are available, fetch and parse metadata from target and direct an authentication request to target if possible.

It should be noted, that incorporated Authenticators or UI-Authenticators take care of fetching the target's policies to select the correct authentication mechanisms and cryptographic algorithms to be used in the trust context to be established. This is completely transparent for the targets and heavily reduces efforts for requestors. Also, every (in)direct authentication involves the user which prevents the wrong (potentially malicious) devices to be authenticated. While UI-Authenticators simply bypass direct authentication messages, Authenticators work according to the following scheme when they receive a request to authenticate with a target:

1) If requestor is not trusted, then fail. End here.
2) If it is target's broker, generate SCT and deliver it to target and requestor. Authentication done. End here.
3) If target has no brokers, (in)directly authenticate with it and become it's broker. Go to step 2.
4) If target has a broker that it trusts, request brokered authentication from target's broker on behalf of requestor. Deliver SCT to requestor. Target's broker delivers SCT

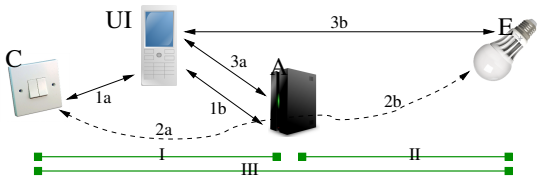

Fig. 2. Authentication Flow in a completely untrusted environment

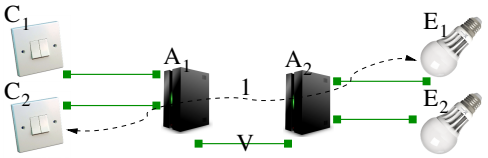

Fig. 3. Authentication Flow in an already established environment

to target. Authentication done. End here.

5) If target has a broker it doesn't trust, (in)directly authenticate with target's broker. Go to step 4.

Combining these concepts leads to the communication flow depicted in fig. 2. A Client C wants to authenticate with an Endpoint E. Since neither C nor E have any brokers at this point, C discovers an arbitrary Authenticator (A). Because there is a suitable UI-Authenticator in the network (UI) C requests indirect authentication with A in 1a and 1b which results in the trust relationship I between C and A. At the same time A becomes C's broker, so C now requests brokered authentication with E from A in 2a. Because E has no brokers yet, A requests indirect authentication with it in 3a and 3b and becomes its broker (II). Now that A is E's broker, it can finally respond to C's request for brokered authentication (2a) and supplies an SCT to both C and E (in 2b), resulting in an immediate trust relationship III between C and E.

Although this introduces some communication overhead, it is completely allotted to the strong participants. The whole set of advantages can be seen in larger scenarios as depicted in figure 3. In this scenario, all clients C_i trust their broker A_1 and all endpoints E_i have the broker A_2. If C_2 requests authentication with E_1 only A_1 and A_2 need to authenticate each other directly. After this, brokered authentication can be established between every other participant of the network. No further user interaction is necessary and the tiny devices do not have to fulfil expensive cryptographic handshakes.

E. Authorization

Regarding the four basic security objectives, authorization is special in that there is no cryptography involved. Instead, it can be viewed as an application which delivers an authorization decision based on information it has. This can be as simple as a static table containing access rights e.g. which user may read a resource or write on it. This information could be more dynamic e.g. by asynchronously asking a privileged user for a decision. Or it could be collected by a complex system that incorporates a host of context information and makes its decisions autonomously without user interaction as it is envisioned in several use cases (e.g. [26]).

Considering the diversity of possible authorization concepts it is of high importance to provide an infrastructure that is flexible enough to not be affected by changes in the application that derives decisions from known information.

To meet this requirement, we completely employ the specification parts of the Web Services world that cover authorization. Because deciding on authorization is not a communication problem, this only incorporates [12, cpt.9] to request authorization and the SAML tokens specified in [27], restricted to *AuthzDecisionStatements* to populate authorizations.

Based on this versatile message framework, we developed an authorization concept that again incorporates the three device classes to offload as much effort as possible from the tiny to the giant and UI devices and to be easy to use and requires as little user interaction as possible.

We introduce the type *SynchronousAuthorizer* that maintains authorization rules such as which client may access which resource under which circumstances. These rules are supplied synchronously e.g. by a user accessing a web interface. A *SynchronousAuthorizer* is usually hosted on a giant device. Complementary, UI devices can host *AsynchronousUIAuthorizers*. They offer a service that is accessed by *SynchronousAuthorizers* to asynchronously ask a user for a decision.

A tiny device has a *Primary Authorizer* which is a *SynchronousAuthorizer* that it populates in its policies. A client needs to request authorization to access a resource from the *Primary Authorizer* of the device that hosts this resource. To give a device the possibility to maintain its authorization on its own, it may host its own *Primary Authorizer*. To not introduce single points of failure and to distribute load, *Primary Authorizers* may nominate further *SynchronousAuthorizers* to make authorization decisions.

V. Prototype Implementation

We implemented a proof of concept that consists of two tiny devices (switch, light bulb), two UI devices hosting a UI-Authenticator and an AsynchronousUIAuthorizer each and two giant devices hosting an Authenticator and a SynchronousAuthorizer each. Documentation as well as all source code and schematics are freely available as open source at [28].

A. Materials

The following material, tools and hardware were used.

1) JMEDS: The Java Multi Edition DPWS Stack (JMEDS, [29]) is an open source Java DPWS implementation ideal for rapid prototyping. We used the latest development snapshot.

2) Bouncy Castle: The *Legion of the Bouncy Castle* provides ([30]) a crypto library for Java we used in version 1.49. For the UI authenticator (see below) the Android derivative spongycastle ([31]) was used in version 1.47.

3) Raspberry Pi: The tiny devices were implemented on Raspberry Pi Model B computers. These are embedded Linux platforms with a 700MHz ARM11 CPU and 512MB RAM. Such a platform is not exactly resource constrained, but extremely inexpensive runs Java Virtual Machines. Thus, it allows rapid prototyping and implementing proof-of-concepts.

B. Devices: From tiny through UI to giant

The light bulb and the switch are both implemented in Java with JMEDS and are executed on a Raspberry Pi Model B. While the switch can be attached to a Pi's GPIO, the bulb needs to be interfaced using a relay and a driver circuit. The bulb hosts an STS supporting direct authentication and accepts SCTs from its brokers. It is also capable of flickering a PIN in its binary form to be captured by a UI device's camera. The switch forms the DPWS Client and provides the capability that a PIN can be 'tapped in' in its binary form for authentication.

As multimedia devices we use two LG Nexus 4 Android smartphones. We developed an Android app that hosts a UI-Authenticator as well as an AsynchronousUIAuhenticator and incorporates JMEDS. The app's core is a collection of plugins providing support for different authentication mechanisms. Besides, the app offers DPWS Device capabilities to expose an STS. The STS bypasses direct authentication messages between requestor and authentication target and translates the OOB channels. Besides, the STS can be accessed by SynchronousAuthorizers to request a user's authorization decision.

The two identical giant devices host an "Authenticator" and therefore their STS is capable of brokering trust contexts. Besides, they support direct authentication through the authenticated ECDH protocol and incorporate a comprehensive authentication client part. They completely support the scenario depicted in fig 3. They also host a SynchronousAuthorizer that maintains authorizations that have been provided statically or that have been asynchronously requested from a user through an AsynchronousUIAuthorizer. The applications can run as a daemon on dedicated hardware and are controlled by a simple command line interface that can be executed remotely on a different host.

VI. Conclusion and Outlook

In this paper we introduce the Devices Profile for Web Services Security (DPWSec). It relies on the time-tested, widely-deployed and well-accepted Web Services Security specification suite. We present how we derived DPWSec suitable for the needs of cyber-physical systems from the original specifications and present our findings by discussing the profile and how it respects the four basic security objectives confidentiality, integrity, authenticity and authorization. We give insight into the developed security framework and the proof-of-concept implementation.

The first future step to follow is to finalize the profile documents. After this we will exchange the underlying communication technology. Since we want to prove that our solution is not only applicable for DPWS but for every service-oriented communication technology that provides certain features it is our goal to identify these features and adapt our solution to at least one different communication technology.

References

[1] K. Selén, "Upnp security in internet gateway devices," in *TKK T-110.5190 Seminar on Internetworking*, 2006.

[2] M. Eisenhauer *et al.*, "A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *SECON Workshops '09. 6th Annual IEEE Communications Society Conference on*, June 2009.

[3] "A b3g service platform: The ist plastic project," PLASTIC Consortium, Tech. Rep.

[4] M. Román *et al.*, "Gaia: a middleware platform for active spaces," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 4, 2002.

[5] IST Amigo Project, "Ambient intelligence for the networked home environment (projektbeschreibung)," September 2004.

[6] M. Handte *et al.*, "D4.1 secure middleware specification - version 1.4," Peces - Pervasive computing in embedded systems, Tech. Rep., feb 2010.

[7] J. Barton and T. Kindberg, "The cooltown user experience," Hewlett Packard Laboratories Palo Alto, Tech. Rep., 2001.

[8] A. Saffiotti and M. Broxval, "Peis ecologies: Ambient intelligence meets autonomous robotics," in *sOc-EUSAI conference on Smart Objects and Ambient Intelligence*, Grenoble, France, Oktober 2005.

[9] OASIS, "Web services security: Soap message security 1.1," Feb 2006.

[10] ——, "Ws-secureconversation 1.3," March 2007.

[11] ——, "Ws-trust 1.3," November 2006.

[12] ——, "Web services federation language (ws-federation) version 1.2," May 2009.

[13] W3C: World Wide Web Consortium, "Web services policy 1.5 - framework," September 2007.

[14] OASIS, "Devices profile for web services version 1.1," Juli 2009.

[15] S. Pöhlsen *et al.*, "A dpws-based architecture for medical device interoperability," in *World Congress on Medical Physics and Biomedical Engineering*, 2009.

[16] G. Moritz *et al.*, "Devices profile for web services in wireless sensor networks: Adaptations and enhancements," in *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 2009.

[17] ——, "Web services on deeply embedded devices with real-time processing," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 2008.

[18] A. Muller *et al.*, "An assisted device registration and service access system for future home networks," in *Wireless Days (WD), 2009 2nd IFIP*, Dezember 2009, p. 5.

[19] V. Hernández *et al.*, "Security Framework for DPWS Compliant Devices," *Third International Conference on Emerging Security Information, Systems and Technologies*, 2009.

[20] J.-F. Martínez *et al.*, "A security architectural approach for DPWS-based devices," *CollECTeR Iberoamérica*, 2008.

[21] S. Unger *et al.*, "Extending the devices profile for web services for secure mobile device communication," in *Internet of Things Conference - TIoPTS Workshop*, 2010.

[22] S. Unger, S. Pfeiffer, and D. Timmermann, "Dethroning transport layer security in the embedded world," in *5th International Conference on New Technologies, Mobility and Security (NTMS)*, 2012.

[23] J.-M. HO, "A versatile suite of strong authenticated key agreement protocols for body area networks," in *8th International Conference on Wireless Communication and Mobile Computing*. IEEE, 2012.

[24] I. C. Society, "Ieee standard for local and metropolitan are networks – part 15.6: Wireless body area networks," 2012.

[25] S. Unger and D. Timmermann, "Bridging the gap for authentication in smart environments," in *Computers and Communications (IEEE ISCC 2014), 19th IEEE Symposium on*, Funchal, Portugal, June 2014.

[26] C. Dixon *et al.*, "An operating system for the home," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, April 2012.

[27] OASIS, "Assertions and protocols for the oasis security assertion markup language (saml) v2.0," Mrz 2005.

[28] S. Unger. (2014) Open source prototype implementation. [Online]. Available: http://bit.ly/1uZNT3R

[29] Materna GmbH. (2013) Jmeds (java multi edition dpws stack) — free development software downloads at sourceforge.net. [Online]. Available: http://sourceforge.net/projects/ws4d-javame/

[30] The Legion of the Bouncy Castle. (2013) bouncycastle.org. [Online]. Available: http://www.bouncycastle.org/

[31] GitHub, Inc. (2013) rtyley/spongycastle. [Online]. Available: https://github.com/rtyley/spongycastle