Extending the Devices Profile for Web Services for secure mobile device communication

Sebastian Unger, Elmar Zeeb,
Frank Golatowski and Dirk Timmermann
University of Rostock
Faculty for CS and EE
Institute of Applied Microelectronics and Computer Engineering
Rostock, Germany
(sebastian.unger/elmar.zeeb/frank.golatowski/dirk.timmermann)@uni-rostock.de

Holger Grandy
BMW CarIT GmbH
Munich, Germany
holger.grandy@bmw-carit.de

***Abstract*—This paper describes a new security profile and propose a security architecture for the Devices Profile for Web Services (DPWS). DPWS defines a Web Service based architecture for networked embedded devices. A requirements analysis on given use cases in the field of in-vehicle-infotainment (IVI) systems showed that the original security profile of DPWS lacks of certain security issues. Thus, our proposed security profile addresses three common security problems of DPWS. The original security profile of DPWS does not support fine-grained security requirements, direct authentication between devices without a third party and it does not propose a comprehensive authorization concept. In contrast to existing research work our solution solves these problems and complies with the original specifications. As a case study two applications in the IVI domain were implemented to show that our proposed security profile meets the missing requirements.**

I. Introduction

The *Devices Profile for Web Services (DPWS)* [1] defines a subset of the WS-* specifications suitable for small-scale, embedded devices. Thus, DPWS fosters interoperability between small embedded devices and fully Web-Service-capable computers such as web servers. It also brings the advantage of using the same technology for communication among devices in an LAN and through a WAN such as the internet.

Since communication security plays an important role in real-world scenarios, DPWS defines a flexible security model by means of interchangeable security profiles. Furthermore, the specification recommends such a security profile. This paper analyzes the security mechanisms of DPWS regarding a use case in the field of in-vehicle-infotainment (IVI), identifies weaknesses and proposes a solution. The paper is structured as follows: As the DPWS specification defines an optional security profile, it is explained and discussed in section II. Section III gives an overview over complementary specifications and related work regarding security in DPWS. Since the DPWS security profile is not suitable under certain – yet not uncommon – conditions this paper proposes a new security profile and a way how to implement it in section IV. After a short glance at a prototype implementation and an IVI use case in section V an outlook and a conclusion will be given in sections VI and VII.

II. Basics Concepts

DPWS defines a subset of existing and specially created WS-* specifications which enables small-scale embedded devices to communicate over Web Services. A DPWS-*Device* consists of exactly one *Hosting Service* and an arbitrary number of *Hosted Services*. The Hosting Service represents the Device itself and is responsible for *discovery* and *description*. Discovery is the process of several DPWS-Devices and Clients finding each other in a local network. The messages and communication flow are defined by the *WS Dynamic Discovery (WSDD)* specification ([2]). After discovering each other, description takes place. This means that Clients and Devices exchange metadata information about the capabilities of the Devices and their Hosted Services. The Hosted Services implement the actual functionality of a Device. DPWS-*Clients* can not only access a Device's Hosted Services by traditional request-response-communication. Instead, the specification *WS-Eventing* ([3]) provides a publish-subscribe-mechanism. Since in real world scenarios, communication security plays a fundamental role, DPWS provides a flexible security mechanism by means of *security profiles*. A profile can be considered as a set of rules devices agree on for securing their communication.

A. The DPWS Security Profile

The DPWS standard defines a generic optional security profile. It basically relies on X.509.v3 certificates and TLS/SSL-secured transport channels. To optionally ensure message integrity and authenticity during the discovery-process, messages can be signed using the *WSDD Compact Signature* format, which is a lightweight derivate of XML Signature ([4]). The keys for creating and verifying signatures are provided by the certificates' public key infrastructure. The certificates are also used to establish TLS/SSL connections to provide secure channels which ensure message integrity and confidentiality. The use of secure channels is mandatory for the exchange of metadata (description phase) and optional for service invocations.

B. Requirements beyond the standard profile

The security profile recommended by the DPWS standard is supposed to define a “baseline for interoperable security” which means that it is intentionally kept simple. As a result, there are no recommendations on how to

1) exchange certificates
2) authenticate certificates without a third party
3) apply different mechanisms for providing integrity or confidentiality or
4) apply different levels of security for a service’s methods and event sources and
5) cope with authorisation

which will be discussed in further detail.

1) How to exchange certificates: Certificates are usually signed by a Certificate Authority (CA) whose certificate is also signed by another CA. This way, certificate hierarchies are set up. When a device receives an unknown certificate, it can traverse through this hierarchy until it finds a trusted CA, which means that the device has a local copy of the CA’s certificate. It can then verify that the received certificate is trustworthy and that its content has never been modified. This mechanism often demands access to the internet to be able to download the certificates of the various CAs. Besides, this can cause high traffic. An average base 64-coded X.509-certificate is of around 1 kByte of size, so each CA’s certificate needed to resolve the chain is another kByte. This causes an amount of traffic and necessary memory that can become an issue on small-scale embedded devices.

2) How to authenticate certificates without a third party: A certificate is associated with a device by containing the device’s UUID. However, there remains a problem for the user to associate a device to an identity (e.g. is it the user’s cell phone requesting some access or someone else’s phone?). Comparing the UUIDs manually is not user-friendly since they consist of sixteen bytes represented by 32 hexadecimal digits and because the device’s UUID might even be only accessible by navigating through a menu-structure. To avoid both lengthy and therefore costly certificate chain resolutions and making the user compare arbitrary looking characters (such as UUIDs), it would be desirable to establish trust between devices directly without requiring a third party.

3) How to apply different mechanisms for providing integrity or confidentiality: and

4) apply different levels of security for a service’s methods and event sources: Another issue not covered by the standard security profile is the ability to offer or require alternative security mechanisms for providing message integrity or confidentiality. An example for TLS not being a sufficient way for encryption would be multi-hop communication with end-to-end encryption. Consider a device D communication with a web server W using a smartphone S as internet gateway (fig. 1. If addressing happens on SOAP-level but TLS provides encryption on transport level, the gateway S needs to decrypt all data to access addressing information. However, encryption on message level would allow to only encrypt the SOAP-payload instead of the whole transport layer and therefore enables access to the SOAP-addressing-information without decrypting the message. Besides privacy-related issues, this also prevents intermediaries from unnecessary expensive en- and decryptions. To enable the use of alternative security mechanisms, a procedure is needed to indicate acceptable ways to provide integrity and confidentiality. Furthermore, the DPWS security profile provides no opportunity for a fine-grained security configuration. According to the specification, a device offers security by means of a secure channel and a service can decide whether to make use of it or not. It would be desirable to be able to configure for each service’s method and event source if they require any kind of security and what security mechanisms they require or offer.

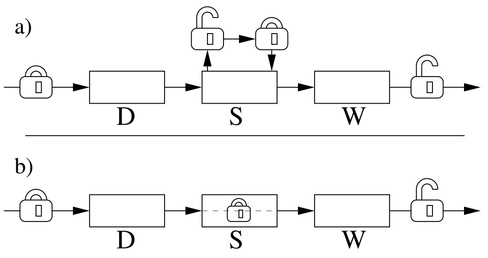

Fig. 1. Point-to-Point- (a) and End-to-End-Encryption (b)

5) How to cope with authorization: Finally, an important issue is authorization of requests. Devices’ users shall be able to give or deny permissions for different clients to access a service, its methods and event sources. An even finer-grained authorization model comes in handy if access to arbitrary resources such as data sources or physical signals shall be controlled, even without knowledge of the user.

III. Complementary Specifications and Related work

There exist a few security-related WS-* specifications as well as proposals for alternative security profiles or architectures for DPWS that will be discussed in this section. The specification WS-Security ([5]) basically defines how to embed security-related tokens into a SOAP message. Tokens can for example be cryptographic signatures, encrypted message parts or certificates. This specification is indirectly referenced by DPWS since the WS-DD Compact Signature format is derived from the signature format defined in WS-Security and XML-Signature. Besides, it describes a way to embed encrypted message parts into SOAP messages with XML-Encryption ([6]) and therefore provides encryption on message level.

To combine the advantages of symmetric and asymmetric encryption methods, the specification WS-SecureConversation ([7]) provides the message frameworks to establish secure connections. After establishing a connection encrypted with an asymmetric algorithm, a shared secret is negotiated which is used to encrypt future messages with a symmetric algorithm. Symmetric algorithms increase performance of en- and decryption by orders of magnitude.

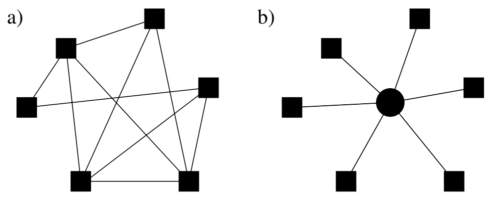

Fig. 2. Trust relations in a network without (a) and with (b) a central instance

The specification WS-Trust ([8]) defines a special Web Service within a network called the Security Token Service (STS). The STS is capable of issuing security tokens and brokers trust. That is – assumed every participant trusts the STS – that if the STS trusts a certain device, all other participants may trust this device as well. This way, trust relations may be established between a device and the STS, not between every device participating in the network which simplifies the process of establishing trust in a network with many participants (see fig. 2). The drawback is, that trust depends on a central instance which produces a single point of failure on the one hand and requires high maintenance effort on the other hand.

Since an STS is able to issue security tokens, it can also be used as a central authorization instance. Again accepting high maintenance effort, an STS could be aware of which client is allowed to access which service, method or event source, it can issue authorization tokens. If the requested service trusts the STS, it will allow or deny access based on the token's content.

Most of these described concepts have inspired some security architectural proposals. Martínez et al. proposed an architectural approach in [9] that suits well the needs of e.g. large office networks with several hundreds or even thousands of DPWS-compliant devices. The authors introduce an authorization server and an authentication server as central instances, similar to an STS described above. Besides, the use of XML-Encryption is encouraged to provide multi-hop communication with end-to-end security and independence from a particular transport protocol (TLS/SSL requires TCP). To increase efficiency, the authors recommend the use of WS-SecureConversation to establish a secure context with asymmetric algorithms and use a shared secret and symmetric algorithms within the context to exchange SOAP messages.

Hernández et al. also encourage the use of XML-Encryption instead of TLS/SSL in [10]. The authors provide a proposal for a security framework for highly resource-constrained DPWS-compliant devices (e.g. with app. 10 kByte of runtime memory). The framework essentially increases efficiency by using symmetric encryption, centralized deployment of network-wide keys and slightly altering the WS-Security specification. The drawbacks are, that in case of the network-wide keys being compromised, high maintenance effort arises to restore security and that the framework is no longer compliant with the WS-Security specification.

IV. DPWS Security for Mobile Devices

Summing up, the aspects to cover were authentication without a third party and associating a device's identity with a certificate, providing the capability to define fine-grained security configurations including alternative security mechanisms and to provide an authorization concept.

To address these problems, we developed an extended security profile for DPWS, the *WS4D Security Profile*. The extended profile was primarily developed for devices at least as powerful as cell phones. Parts of the described solution – especially regarding the developed architecture and the prototype implementation – are specialized on in-vehicle-infotainment (IVI) systems. That is, we assume that the vehicle (the DPWS-Device) offers at least a display and the mobile devices (the DPWS-Clients) offer at least a keypad. However, we will discuss opportunities to make the extended profile suitable for other use cases at the end of the section as well as to point out under which circumstances the extended profile is compliant with the original security profile.

Since the aspect of authorization does not play a role on Web Service level, it is not part of the security profile. It is discussed in the context of the developed architecture as a recommendation on how to restrict access.

A. Introducing the WS4D Security Profile

To indicate that a DPWS-Device supports the WS4D Security Profile, it uses types within the discovery and escription messages. In addition to device types (e.g. printer) and service types (e.g. print-service) the messages contain the newly defined types ws4d-secure-device and ws4d-secure-service, respectively.

The process of authentication – and therefore setting up trust – is composed by the procedures of exchanging and validating certificates as well as associating a certificate with a device identity. Considering these two parts as a single problem allows a simple approach: make two devices exchange their certificates and verify that the received certificate belongs to the device with which a trust relationship is supposed to be established. To provide this functionality, the WS4D Security Profile defines an *Authentication Service*, which is a special Hosted Service of a DPWS-Device. Its only purpose is to exchange certificates as well as a certain proof of the communication partners' identity. The term 'proof' can be substituted by any kind of knowledge – e.g. a short PIN that has been exchanged out-of-band (OOB) – or a given fact such as a dedicated connection over e.g. USB, Ethernet or RFID. The Authentication Service is also responsible for providing information about what authentication methods a device is capable of. Thus, the Authentication Service's description contains all suitable methods in its Policies. A client can chose a suitable method for authentication based on the Authentication Service's description. Section V describes a way for authentication based on exchanging a short PIN OOB.

Authentication can be considered as an additional communication phase in DPWS. Usually, there are the three phases discovery, description and invocation of services (see fig. 3).

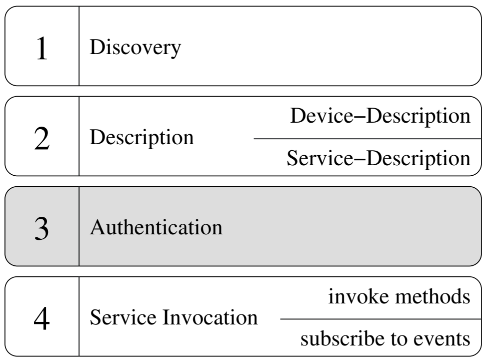

Fig. 3. Extended communication phases of DPWS

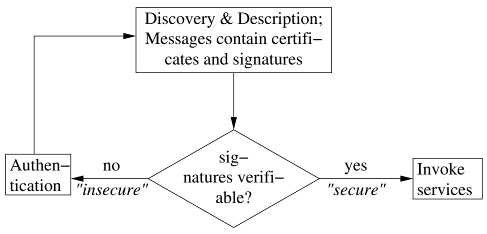
Discovery & Description; Messages contain certifi– cates and signatures
sig– natures verifi– able?
no
"insecure"
Authen– tication
yes
"secure"
Invoke services

Fig. 4. Secure and insecure discovery and description

Regarding the use of the WS4D Security Profile, discovery and description are necessary for a client to discover the device's support of the profile. At the same time, signing these messages assures authenticity and message integrity. However, received discovery and description messages initially cannot be authenticated as they are exchanged prior to the authentication phase. Thus, a client needs to distinguish between secure and insecure discovery and description processes. After authentication has taken place, buffered messages can be verified or discovery and description can be repeated. Messages' authenticity and integrity can then be validated because the parties' certificates have been exchanged and verified before (see fig. 4).

To enable different services' methods and event sources to require or support different security mechanisms, the Hosted Services' descriptions have to contain Policy sections that describe the configuration. The Policy section consists of the following parts:

1) WS4D Security Profile assertion
2) granularity: "Service" or "Method"
3) supported mechanisms for each entity

The *granularity* specifies whether the same security mechanisms are required by every method and event source of a service ("Service") or whether every method and event source has different requirements ("Method"). The *supported mechanisms* for each entity (the service or each method and event source) are expressed as groups of *tuples*. A tuple consists of a mechanism for providing message integrity and a mechanism for providing confidentiality. If no Policies are specified, it implies that all methods and event sources support or require TLS/SSL to provide message integrity and

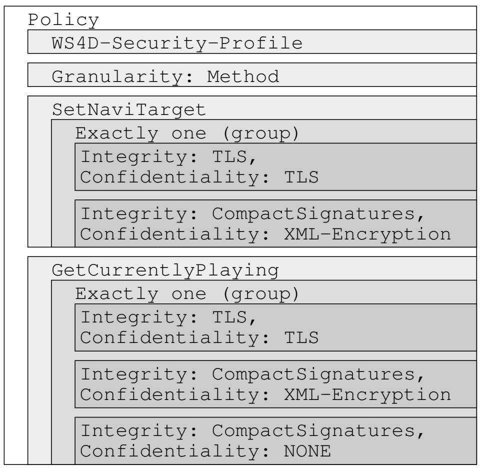

Fig. 5. Schematic example for security configuration policies

confidentiality for the sake of compliance (see end of this section). Fig. 5 shows an example. The described service contains two methods. *SetNaviTarget* supports TLS to provide integrity and confidentiality as well as a combination from WS-DD Compact Signatures and XML Encryption. The same security mechanisms are supported by *GetCurrentlyPlaying*. However, the latter one also accepts connections that don't provide confidentiality at all.

The WS4D Security Profile requires that a DPWS-Device answers with an error message in case that a request comes from a client that has not been authenticated or if the request does not match the required security mechanisms. Regarding a DPWS-Client, the WS4D Security Profile requires to distinguish between secure and insecure discovery and description phases.

B. Architecture

To realize the WS4D Security Profile as well as an effective authorization model, which is described below, the architecture shown in fig. 6 has been developed. Its mode of operation is now explained regarding a Device's initialization phase followed by the communication phases of DPWS.

When a Device is initiated (e.g. the software is started or the device is turned on) every Hosted Service registers its security requirements inside the *Security Configuration Database*. This way, the according components can check whether an incoming request fulfills the requirements.

For a DPWS-Device, there is no difference between secure and insecure discovery / description. However, for some applications it can make sense to ignore unsigned messages or to answer with an error message during these phases.

After discovery and description have been finished, a client sends a request to the Authentication Service to initiate exchange and verification of certificates. While the Authentication Service merely provides the Web Service interface, the *Authentication Engine* implements the logic to do the actual authentication (e.g. generation and validation of proofs) and to store authenticated certificates inside the *Certificate Database*.

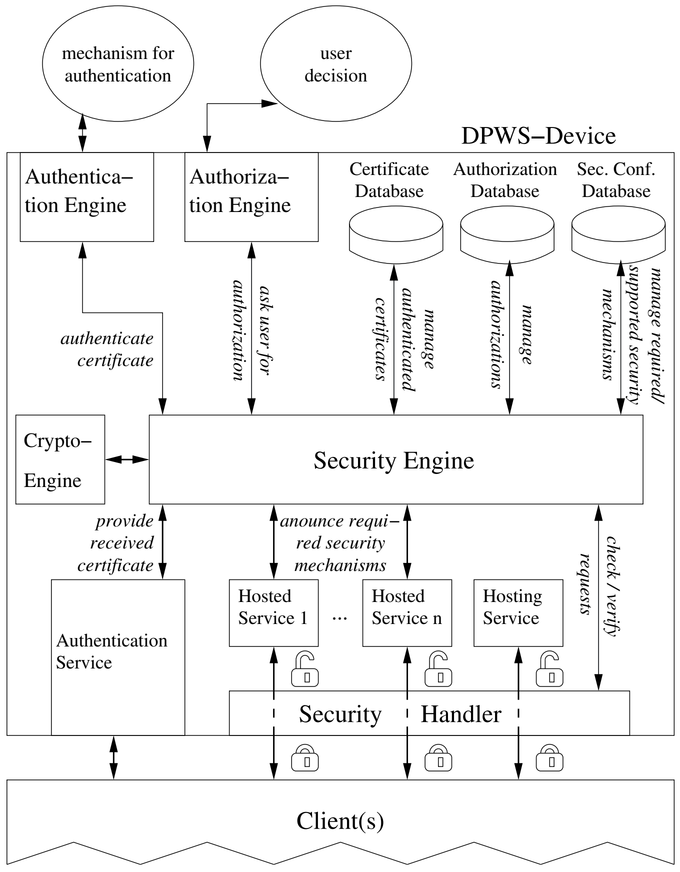

Fig. 6. Security Architecture

The components never access each other directly but always indirectly through the *Security Engine* as an intermediary. This ensures easy maintenance and interchangeability of the components (e.g. for realizing different authentication methods).

When invoking a service's method or registering to an event source, the *Security Handler* is responsible for converting secure contexts into insecure contexts and vice versa. For incoming messages (requests, subscribe messages) this means, that the Security Handler checks whether the embedded or referenced certificate has been authenticated earlier. If so, the Security Handler can make use of the *Crypto Engine* to verify the messages signature (if any) using the public key from the certificate. Eventually, it checks whether the required security mechanisms are used by accessing the Security Configuration Database. If, for example, XML-Encryption is used for providing confidentiality, the Security Handler is also responsible for decrypting the encrypted message parts and replace the ciphers with the plain text. If one of the checks fails, the Security Handler immediately replies with an according error message and discards the request. Otherwise, it will hand out the checked and if necessary decrypted message to the Hosted Service.

For outgoing messages (responses, events), the Security Handler is responsible for signing and encrypting the message where necessary.

As mentioned before, the architecture also implements an effective authorization concept. When initializing a device, authorization datasets can be saved in the *Authorization Database*. An authorization dataset consists of

- a resource (e.g. a service, method, data field or physical signal)
- a proof (e.g. a certain certificate, a certificate signed by a certain CA or a username-password-combination)
- the authorization itself (granted / denied)
- a validity timestamp to enable temporarily limited decisions (e.g. allow for today only or deny for the next 60 minutes)

Using the proposed authorization model allows the following scenario: a certain Client A is allowed to access all methods and event sources of a Hosted Service S, while Client B is only allowed to subscribe to a certain event source. Also, every client in possession of a certificate that has been signed by a certain CA or can provide a certain username-password-combination is allowed to access services for reconfiguring the security settings.

Thus, the Security Handler has another task for incoming messages: take care of authorization. This means, for an incoming request it has to check, whether there exists a suitable authorization dataset. If not, it invokes the *Authorization Engine* to ask for a decision of the user. Depending on the decision, the Authorization Engine stores a new authorization dataset and the Security Handler permits or denies access.

By using the proposed security architecture, the Hosted Services' implementations do not need to take care of any security related tasks. That is with one exception: context-based authorization. For example, with the proposed authorization concept, it is possible to generate different responses for different clients (proofs). You could consider an application where a signal (resource) is available in two qualities of accuracy. While most clients receive a response containing low accuracy, few clients can provide an additional proof allowing them to receive a result with high accuracy.

C. Possible Applications & Compliance

To be able to implement the WS4D Security Profile, a device needs to meet some requirements. First of all, it must provide the capability for some kind of OOB communication. As mentioned before this can be as simple as being able to display and/or enter a PIN code. This however implies that the DPWS-Device needs a display and the Client needs a keypad. Alternative approaches can be wired (USB, Ethernet, . . .) or short-range wireless (Bluetooth, ZigBee, RFID) communication and a push button to trigger exchange and verification of certificates. Besides these peripherals, the devices should be at least as powerful as cell phones. The CPU must be powerful enough to cope with asymmetric encryption and a device needs enough memory to store its own X.509 certificate as well as a couple of certificates that have been received by other devices. As mentioned in section III, Hernández et al. proposed an approach suitable for less powerful devices in [10] that bases on symmetric algorithms. Also, van Engelen and Zhang showed in [11] that round-trip-times decrease by nearly an order of magnitude when signing messages using symmetric HMAC algorithms compared to using asymmetric algorithms such as DSA or RSA. Nevertheless, we decided to use the public key infrastructure (PKI) deployed by X.509 certificates and the according asymmetric algorithms for the sake of compliance. This means, that devices implementing the WS4D Security Profile can interoperate with devices that support the DPWS standard security profile under certain circumstances that are now discussed.

If certificates are exchanged and verified before communication takes place (e.g. by copying them into the devices' file systems), no authentication is necessary. Thus, there is no need to distinguish between secure and insecure discovery and description messages. Furthermore, if every service and each of its methods and event sources support TLS/SSL connections (or no security at all), and if description can be processed using TLS/SSL, interoperability is given. The latter aspect is why the absence of security configurations by means of Policy sections within the description of a service is implicitly interpreted as TLS/SSL being supported, as described earlier in this section.

V. Prototype & Use Case

The described architecture was implemented to realize the WS4D Security Profile. This section briefly describes a security plug-in for the DPWS stack WS4D-gSOAP ([12]) as well as the use case applications. The former subsection also describes a way for authentication based on the OOB exchange of a one time PIN.

A. Security plug-in for WS4D-gSOAP

The DPWS stack WS4D-gSOAP is based on the well-known Web Service framework gSOAP ([13]). It is written in C and provides a flexible plug-in mechanism which makes it very well extendable. Thus, the WS4D Security Profile was developed as a security plug-in. A convenient API enables a developer to easily set up the Authentication Service in a device and initiate authentication from within a client. The API also lets developers rapidly define authorization datasets and de/activate user decisions on authorization for services or single methods and event sources.

The Security Engine has been implemented as a plug-in socket that the other components register at. This enables developers to easily interchange single components. This way, alternative Authentication or Authorization Engines can be integrated or the databases for certificates or authorization datasets may be real relational databases instead of plain text files.

The Crypto Engine is responsible for providing cryptographic functionality such as building message digests and en- and decryption. In our implementation the Crypto Engine is an interface to the popular open source library OpenSSL ([14]). Besides the cryptographic methods it also provides a convenient API to deal with X.509 certificates.

As mentioned earlier, the use cases described in the next section are located in the area of in-vehicle-infotainment.

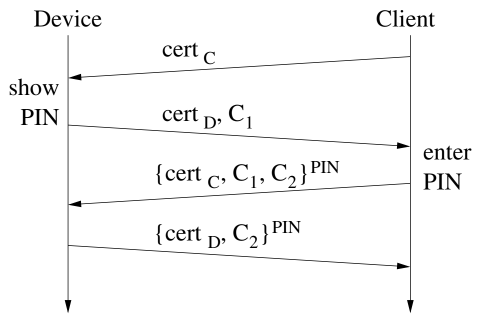

Fig. 7. Authentication Handshake based on PIN exchanged OOB

Thus, we assumed that every DPWS-device has at least a display and every DPWS-Client has some sort of keypad. The prototype implementation of the Authentication Engine therefore provides the logic for authentication based on the OOB exchange of a one time PIN. Fig. 7 shows the basic communication pattern of the developed authentication handshake. The handshake consists of two requests and two responses. The client initiates authentication by sending the first request that merely contains the client's certificate $cert_C$.

The Device generates a random one time pin (e.g. four alphanumeric characters) and a random sequence of bytes (e.g. 128), the Challenge C_1. Before responding, the Device saves the Client's certificate in the Certificate Database together with the generated PIN, and the Challenge C_1. After displaying the PIN, the Device responds with its certificate $cert_D$ and the Challenge C_1.

The Client's user can now enter the PIN. To testify the knowledge of the PIN and to prevent man-in-the-middle attacks at the same time, the second request is encrypted with a key derived from the PIN. The request contains the received Challenge C_1, a Challenge C_2 generated by the Client and the Client's certificate. Because the number of possible keys is limited due to potentially short PINS, the Challenges' purpose is to increase entropy of the cypher texts.

The Device eventually decrypts the request with the key derived from the PIN and compares $cert_C$ and C_1 with the according entries in the Certificate Database. If they match, the certificate $cert_C$ is authenticated.

Authentication of the Device happens in the same manner. The Device responds with its certificate $cert_D$ and the Challenge C_2, both encrypted with the key derived from the PIN. The Client decrypts the certificate and the challenge und compares them with the values saved earlier.

B. Use Cases

The use cases are located in the area of in-vehicle-infotainment (IVI). The board computer – the so called head unit – implements a DPWS-Device and offers two Hosted Services, the *Remote Keyboard Service* and the *Car Data Service*.

The Remote Keyboard Service can receive keystrokes of mobile devices such as smartphones and netbooks to make text input more convenient. This functionality especially addresses fellow passengers for applications such as searching the media library or browsing the world wide web. The operator of the car must have control over which device is allowed to produce text input. Eventually, it is important that the communication can not be eavesdropped. The WS4D Security Profile allows authentication of devices without a third party. Furthermore, the context-based authorization enables another feature. It's not too unlikely, that a fellow passenger may type in one application (e.g. media library) but should not be able to type in another application (e.g. entering navigation target). An authorization model on service level is not able to make this restriction. However, context-based authorization is. Consider the fellow passenger's identity as proof and the active application as resource. The proposed authorization model is then capable of distinguishing authorization to type in different applications.

The Car Data Service is supposed to provide access to all kind of information a car can offer. This includes maintenance information, current GPS position or title and artist of the currently played music. It also should be able to fire alarms e.g. in case of running low on fuel. In some cases, the Car Data Service should also enable the possibility to change some information such as the current navigation target. Some of this information – such as the currently played music – might be accessible to everyone including passengers of other cars. Thus, there is no need to provide message integrity or confidentiality. On the other hand, every passenger inside the car might be allowed to view the current GPS position and navigation target while the car's owner is the only one allowed to change the latter one. When requesting maintenance information, the owner should receive e.g. tire pressure or a date for the next inspection while only certified car repair shops should be able to access the engine's calibration interface. The WS4D Security Profile provides the ability to define different security requirements for each method and event source of a Hosted Service. The comprehensive authorization concept can limit access based on previously authenticated certificates.

The prototypes of the use cases have been implemented as simulations of real automotive scenarios. Specifically, the result are three GUI applications based on the open source graphics framework Qt4. One application simulates the head unit which offers the services, the remaining two simulate the clients (e.g. smartphones) consuming the services. DPWS functionality is provided by WS4D-gSOAP and the security requirements are provided by the implemented security plug-in.

VI. Outlook

A first version of the security plug-in for WS4D-gSOAP is available at the project web site ([12]). While the current state of the plug-in can be considered as pre-alpha, it is under constant development. To ease the process of evaluation of the security plug-in, the prototype applications have been developed as GUI applications running on a PC. However, once the security plug-in is in a stable state, the simulation applications can easily be ported to real automotive hardware.

In the current state, the security plug-in allows security configuration only at design time. To enable configuration changes at run time, we are currently developing a Security Management Service (SMS) as an interface through another dedicated Hosted Service. The SMS has intentionally not been integrated into the WS4D Security Profile as its necessity as well as its interface might be very application-specific. Another aspect to increase versatility of the WS4D Security Profile is the consideration to provide the Authentication Service with the ability to issue new certificates for clients not having own certificates.

As mentioned earlier, the WS4D Security Profile is applicable for devices at least as powerful as cell phones, since dealing with X.509 certificates and asymmetric encryption algorithms is expensive regarding memory consumption and CPU time. We are now focusing on providing DPWS security in smart environments with potentially hundreds of highly resource-constrained devices. In this context research is necessary to find solutions how to provide authenticity, message integrity, encryption and authorization in a network with many participants most likely not being able to deal with large certificates or asymmetric cryptography. To this end, the next steps will be timing analysis regarding round-trip times depending on processing power and used security mechanisms. In addition, focus will move to symmetric encryption algorithms and suitable ways for deploying shared secrets.

VII. Conclusion

This paper identifies shortcomings in the security profile proposed in the DPWS specification regarding its applicability to scenarios in which mobile devices such as cell phones are supposed to communicate securely over Web Services. After analyzing the existing security-related WS-* specifications and related work, an approach for a new DPWS security profile, the WS4D Security Profile, is presented. It provides DPWS-Devices with the facility to establish trust between devices directly and without third parties and allows fine-grained security configurations for a Device's services and event sources. At the same time, it preserves compliance with the original specifications under certain conditions. To restrict access to different services, methods and event sources, the developed architecture also proposes an authorization concept. The security-related analysis and the implementation of the IVI use cases show the effectiveness of the WS4D Security Profile and propose a way for establishing trust by exchanging a short one-time PIN out of band.

Acknowledgment

The authors would like to thank Prof. Dr. Harald Heinecke for supporting this work by fostering cooperation between BMW Car IT GmbH and the University of Rostock.

References

[1] OASIS, "Devices Profile for Web Services Version 1.1," http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.html, July 2009.

[2] OASIS, "Web services dynamic discovery (ws-discovery) version 1.1," http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html, July 2009.

[3] W3C: World Wide Web Consortium, "Web services eventing (ws-eventing)," http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/, March 2006.

[4] W3C: World Wide Web Consortium, "XML-Signature Syntax and Processing," http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, February 2002.

[5] OASIS, "Web Services Security: SOAP Message Security 1.1," http://docs.oasis-open.org/wss/v1.1/, February 2006.

[6] W3C: World Wide Web Consortium, "XML Encryption Syntax and Processing - W3C Candidate Recommendation 04 March 2002," http://www.w3.org/TR/2002/CR-xmlenc-core-20020304/, March 2002.

[7] OASIS, "WS-SecureConversation 1.3," http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html, March 2007.

[8] OASIS, "WS-Trust 1.3," http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-spec-cs-01.html, November 2006.

[9] J.-F. Martínez, M. López, V. Hernández, K. Jean-Marie, A.-B. García, L. López, C. Herrera, and C.-J. Sánchez-Alarcos, "A security architectural approach for DPWS-based devices," *CollECTeR Iberoamérica*, 2008.

[10] V. Hernández, L. López, O. Prieto, J.-F. Martínez, A.-B. García, and A. Da-Silva, "Security Framework for DPWS Compliant Devices," *Third International Conference on Emerging Security Information, Systems and Technologies*, 2009.

[11] R. A. van Engelen and W. Zhang, "Identifying Opportunities for Web Services Security Performance Optimizations," *Services, IEEE Congress on*, vol. 0, pp. 209–210, 2008.

[12] WS4D.org, "WS4D - Web Services for Devices $>>$ Stack: C (gSOAP)," http://ws4d.e-technik.uni-rostock.de/?page_id=13.

[13] R. Engelen, "gSOAP: SOAP C++ Web Services," http://www.cs.fsu.edu/~engelen/soap.html.

[14] The OpenSSL Project, "OpenSSL: The Open Source toolkit for SSL/TLS," http://www.openssl.org/.