Dethroning Transport Layer Security in the Embedded World

Sebastian Unger, Stefan Pfeiffer and Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
Germany
Email: {firstname.lastname}@uni-rostock.de

***Abstract*—Developing a security concept feasible for a large number of cooperating embedded devices is crucial for the practical relevance of visions such as the Internet of Things, Ambient Assisted Living or Pervasive Computing. However, designing these concepts in conjunction with new technologies usually is more expensive. Thus, new technologies often rely on existing security techniques on lower layers of the network stack. Especially the Transport Layer Security (TLS) protocol suite is a very popular solution because TLS is a widely spread and well accepted protocol. However, TLS is not ideal for the realm of embedded devices and it does not provide all the necessary features. We state that a comprehensive security framework for large distributed systems of embedded devices has to be implemented on application level. We investigate on the applicability and adaptability of the Web Service (WS) Security specification suite since it offers such a desired comprehensive security framework. Furthermore, its base technology – Web Services – has already been ported to the domain of embedded devices by means of the Devices Profile for Web Services (DPWS). However, WS Security imposes a dramatic overhead on message sizes and parsing efforts for providing confidentiality, integrity and authenticity. Thus, in this paper we discuss two variants of WS Security that decrease the imposed overhead but remain compatible to the classical WS Security. A performance analysis shows that our proposed solutions have no significant drawback compared to the state of the art – TLS – but even provide a richer feature set.**

I. Introduction

The visions of the Internet of Things (IoT), Ambient Assisted Living (AAL) and Pervasive Computing (PC) state that everyday objects will make our lives easier in the future by assisting us in virtually everything we do. However, these visions will never become reality without solving the very complex evolving security issues ([1]). Unfortunately, coping with these issues is rather unpopular since security increases development and also device costs and often decreases usability ([2]). For these reasons, there exists a trend to rely on security on lower layers of the network stack. In particular, the protocol Transport Layer Security (TLS) ([3]) is very popular as an implementation exists for virtually every relevant platform. This way, there is no necessity to develop a comprehensive security framework at the time a new technology is designed. Although TLS is a widely spread and well-adopted high-performance protocol it does not completely suit the demands of embedded devices to securely interconnect. To come by the shortcomings of TLS a comprehensive security framework on application layer is vital. A possible candidate is found in the Web Service Security suite (e.g. [4], [5], [6]). It defines a complete set of security mechanisms and concepts for large distributed systems. Since TLS has a couple of disadvantages over security concepts on application layer, it leads to the question of how much of an advantage TLS's performance actually is. The fact that to our best knowledge there exists no performance comparison between TLS and an implementation of WS-Security is the motivation of this paper. On the other hand, Web Service Security imposes a dramatic overhead in terms of message sizes and parsing efforts. Therefore, it is not directly applicable to the domain of embedded devices. Especially in cases where the payload is small message size overhead can easily sum up to an order of magnitude.

Thus, the main contribution of this paper is the proposal of two variants of the WS-Security message format being more suitable for resource-constrained devices. We compare the performance of our approaches by means of round trip times to TLS as the current state of the art. The remainder of this paper is structured as follows: in section II we briefly cover basic principles and motivate in more detail. Our measurement setup and methodology is described in section III. In sections IV and V we describe our proposals for compact representations of WS-Security. Finally, we sum up this paper in section VI and give a short outlook.

II. Basic Principles and Motivation

In this section we briefly cover relevant basic principles and explain our motivation to examine WS Security as the application layer security's representative for our comparisons.

TLS (aka. Secure Socket Layer (SSL)) is the de-facto standard for establishing secure channels between two machines. Simplified, TLS is a protocol suite which consists of two parts. First, there are protocols to establish a connection and to negotiate parameters. In this paper we focus on the second part, the Record Protocol which caries payload data at provides confidentiality and message integrity. To keep message sizes low and therefore save bandwidth, TLS can optionally compress application data prior to its encryption. When transmitting data over a channel secured by TLS the single, ordered steps on sending side are fragmentation of data (if necessary), compression of data (if desired), calculation of

978-1-4673-0229-6/12/$31.00 ©2012 IEEE

Message Authentication Code (MAC) over compressed data and encryption of the resulting message (payload and MAC).

Due to its transparency for applications TLS is very popular for establishing a secure channel between two machines. But when it comes to securely connecting embedded devices TLS has several drawbacks. First, if authentication is desired (which it is in most cases), the devices have to cope with X.509 certificates and asymmetric cryptography. These techniques require the devices to handle and transmit a high amount of data and to run complex algorithms. Different authentication mechanisms, that might be more feasible for a distributed system of embedded devices, are not supported. Also, if multihop scenarios are considered, there is no way to encrypt only parts of a message so that sensitive data is protected while e.g. meta information remains accessible to every intermediate. Another aspect to consider is that due to the aforementioned transparency data arrives inside an application without protection. Encryption on application level can prevent caching sensitive data in clear text and can foster secure data storage.

For these reasons, it is our future goal to develop a comprehensive security architecture for IoT, AAL and PC scenarios. For this purpose we are going to investigate on how to use an existing, well-know and well-proven security framework from a different domain and how to adapt it to the special needs of critically resource-constrained devices. In particular, we decided to adapt the Web Service Security suite because it comprises a comprehensive security framework and because its base technology – Web Services – is already adapted to the embedded world by means of the Devices Profile for Web Services (DPWS, [7], [8]). Our first step of this adaption is described in this paper by introducing a compact representation of mechanisms used in WS-Security: XML-Encryption ([9]) and XML-Signature ([10]). This is desirable to reduce message sizes which leads to lower power consumption, because – especially for wireless communication – transmission time is the highest source of power dissipation. A real-world example is our test case 1 (see sct. III): Applying a minimalistic combination of XML-Encryption and XML-Signature to a 218 byte SOAP envelope multiplies its size to 1615 byte which could already break the very minimalistic TCP/IP-implementations often found in critically resource-constrained devices where packet fragmentation is not supported.

III. Measurement Setup and Methodology

To compare our proposals' performance by means of round trip times we used the following hardware setup and measurement methodology. The client software runs on a desktop PC with Linux as operating system. The server software is executed on a Fox Board LX832 which is an embedded platform running Linux as well, coming with an 100MHz Axis RISC CPU and 32MB of RAM. Client and Server are connected via Ethernet while the client machine uses a dedicated interface which does not handle any other load.

Instead of acquiring current time before and after a large number of requests to compute the arithmetic mean we decided to log the time difference between every request and response and compute the RTTs' medians. This method is known to be more robust against outliers.

		<Body>in bytes	<Envelope>in bytes
Case 1 (1x5)	tx	89	218
	rx	107	236
Case 2 (1x50)	tx	134	263
	rx	152	281
Case 3 (10x5)	tx	308	437
	rx	326	455

TABLE I
Message sizes

We used OpenSSL [11] as implementation of TLS as well as its implementation of cryptographic algorithms and we used gSOAP 2.8.13 as Web Service messaging framework and implemented our approaches as gSOAP plugins. The implementation is planned to become part of the DPWS stack WS4D-gSOAP available at the WS4D website ([12]). The focus of this work lies solely on the performance analysis of providing message integrity and encryption, while the nonrecurring (or at least infrequent) connection and trust establishment is explicitly out of scope. Thus, we assume that some setup has been taken place earlier and keys for encryption and signatures are referenced by an ID. To be able to focus on the TLS's record protocol, the client sends a dummy request to the server for setting up a TCP connection and a TLS session. After this request the connection is kept active to reuse the TCP connection and the TLS session so connection setup does not influence the measurement of RTTs.

We used three types of SOAP messages, all with a different payload. Case 1 transported a string of five bytes, case 2 a string of 50 bytes and case 3 carried an XML structure that contained ten strings of 5 bytes each. The resulting message sizes are depicted in table I.

For all cases, we used the symmetric ciphers RC4 and AES with 128bit wide keys because the DPWS specification ([7, R4060, R4061]) requires the former and recommends the latter one. SHA-1 is used as hash algorithm for every digest.

IV. A compact Version of WS-Security for embedded Devices

In this section we present our proposal for a compact version of WS-Security: WS Compact Security (WS CSec). After comparing its round trip times to those of TLS, we discuss our approach and our results.

A. Compact Versions of XML-Signature and XML-Encryption

In our proposal for a compact version of WS-Security, we especially focus on providing message integrity and confidentiality. When DPWS and its related specification WS-Discovery ([13]) were specified the authors considered providing message integrity and authenticity for messages sent via UDP and thus specified the Compact Signature format ([13, cpt.8.2]). Simplified, an XML-Signature consists of the following parts: First, for every signed XML element within

a document there is a <SignedInfo>-Block which includes the used algorithms for canonicalization and building the signature, a reference to the current element within this document, the digest of this element and the algorithm that it was computed with. Furthermore an XML-Signature includes the signature value itself and eventually a complex three level structure containing an identifier for the used key. All this information can be compressed to a Compact Signature format in a lossless way which consists of a single, empty XML element with the following five attributes: the scheme which denotes the used algorithms, a key identifier, a comma-separated list of references of the signed elements and a list of prefixes and eventually the signature value. Summed up, this means a more compact format of the signature and the key identifier and – more importantly – implicitly the same algorithms and PrefixList for every signed element. The elements' digests are not submitted since they have to be calculated on the receiver side anyways. For samples signatures as found in [13, cpt.8.2], where only a single element of a SOAP-Envelope is signed, this results in 918 bytes and 14 XML elements with six attributes for an XML-Signature. A Compact Signature expresses the same information in 220 bytes with a single element and five attributes. However, it is likely that more elements are signed – for e.g. three elements, an XML-Signature would grow to 2010 bytes in 32 elements with 18 attributes whereas the Compact Signature stays at a nearly constant size of 228 bytes for a single element with five attributes. It should be stressed again that a bidirectional mapping between an XML-Signature and a Compact Signature is easily achievable. By providing a simple adapter, Compact Signatures are compatible to existing, classical XML-Signature implementations. The only restriction of a Compact Signature is that every element of a message must be signed with the same set of algorithms.

In addition to dramatically reducing message sizes a Compact Signature has the advantage of reducing the necessary parsing effort. Embedding encrypted message parts into a SOAP message by means of XML-Encryption does not produce such a high message overhead as XML-Signature does but still introduces several additional XML elements to be parsed. Therefore we introduced an additional element to the security header introduced by WS-Discovery containing relevant information for encrypted payloads. The overall structure of a security header as described by WS-Discovery and enhanced by our approach is depicted in figure 1. Inspired by Compact Signature, our proposal for an additional encryption header field is compatible to a classical XML-Encryption structure in that a simple adapter can easily translate between these two formats.

Besides, we introduced two conventions to decrease parsing overhead even further. The "Refs" attributes specify which elements to encrypt and which to consider when building the signature respectively. Omitting them results in an implicit default to encrypt the payload (so, the whole content of the SOAP Body is replaced by an <EncryptedData> element) and to digest the whole SOAP Envelope to build the signature.

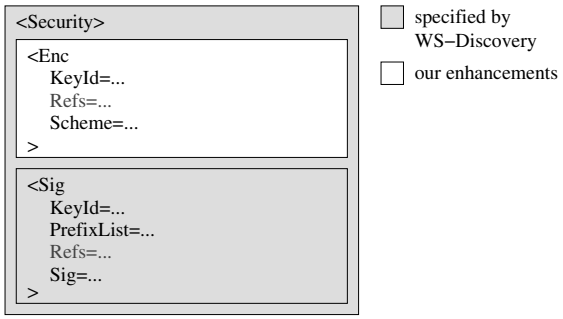


Fig. 1. WS Compact Security Header

Type		SOAP *(+HTTP)*	TLS		WS CSec	
			RC4	AES	RC4	AES
Case 1	tx	218 *(382)*	238	245	749	777
	rx	236 *(372)*	222	229	773	801
Case 2	tx	263 *(427)*	248	261	809	841
	rx	281 *(417)*	232	245	832	865
Case 3	tx	437 *(601)*	293	309	1041	1077
	rx	455 *(591)*	277	293	1065	1097

TABLE II
MESSAGE SIZES IN BYTES FOR WS COMPACT SECURITY COMPARED TO TLS

B. Performance Comparison to TLS

In this section, we will present the resulting message sizes and the measured round trip times of the previously discussed approach. Table II shows the sizes of the SOAP messages depending on the test case (compare to table I) as well as the sizes of the corresponding TCP payload (SOAP + HTTP header). Furthermore, the table shows TCP payloads for the case that the SOAP messages are sent through a TLS connection. Eventually, SOAP message sizes are shown for the case that integrity and confidentiality are provided by means of WS Compact Security. According to these values TLS' compression decreases message sizes by 40%-50% with higher reduction ratios for longer messages. The Compact Security scheme increases messages sizes by approximately factor 2.5 to 3.5 with lower effect for larger payloads which results from the nearly constant length of the security header.

The resulting round trip times (RTT) are depicted in table III. It shows that sending SOAP messages over a channel secured by TLS has no noticeable effect (factor 1.1 to 1.3)

		Round Trip Times in ms		
		Case 1	Case 2	Case 3
No Security		16.7	17.0	17.6
TLS	RC4	18.5	18.8	22.4
	AES	20.0	20.6	24.8
WS CSec	RC4	29.4	30.2	41.1
	AES	39.1	40.1	51.8

TABLE III
ROUND TRIP TIMES (MEDIANS) OF WS COMPACT SECURITY COMPARED TO TLS

on the measured round trip times compared to using no encryption at all. However using the proposed WS Compact Security scheme increases round trip times by factor two to three, depending on message size and cipher algorithm. The following discussion will show, why it is still desirable to employ WS Compact Security formats.

C. Discussion

From the results in tables II and III one can derive the following: First, providing confidentiality and integrity with TLS has no noticeable effect on round trip times. While there certainly is a computational overhead which results from de/compression, en- and decryption and building and verifying MACs, this overhead is compensated by the smaller message sizes. This leads to the conclusion, that using TLS's Record protocol has no drawbacks compared to using no security at all. It could even be advantageous if battery-driven, wireless scenarios are considered where radio time is very expensive.

Second, we find proofs for WS Compact Security having structural disadvantages over TLS that result in higher round trip times. Although WS Compact Security is compact compared to the classical formats, TLS is even compacter. This especially results from not treating the message's signature and its cipher text independently but combining them in one Record. Furthermore, this results in lower computational overhead because payload and message digest are encrypted in one run. Besides, TLS compresses payload data prior to encrypting it which results in even smaller messages.

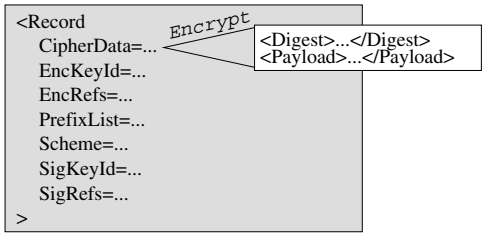
However, we see that if it is desired to apply security mechanisms on application level or if functionality is necessary TLS cannot provide (e.g. different authentication methods or multihop communication), our proposed WS Compact Security scheme can be used. By providing very simple adapters that map between this scheme and classical XML-Signature and XML-Encryption elements, compatibility to existing WS-Security solutions is preserved. Though round trip times are noticeably higher compared to TLS they can be considered as being of the same order of magnitude. Therefore, WS Compact Security does not introduce any severe disadvantage.

V. Security Records: Mapping TLS on WS-Security

Due to the aforementioned disadvantages of WS Compact Security over TLS, we introduce another approach, the WS Security Records (WS-SecRec).

A. WS Security Records

As discussed in section IV-C, the most important advantage of TLS over WS Compact Signature is its essentially compacter format, combining integrity and confidentiality in one encryption run. To leverage this advantage for security mechanisms on application layer we developed the WS Security Record based on the scheme described for WS Compact Security. The WS Security Record format is depicted in figure 2. The content of the <Body> element is substituted by a <Record> element. The original content is encrypted together with the message digest and the resulting cipher is the Record's CipherData attribute. Tables IV and V show the resulting message sizes and round trip times compared to our WS Compact Security approach and to TLS. The results show that although WS Security Record messages are still noticeably larger than TLS records (factor two to three), round trip times are only slightly higher. Especially for small payloads – which are more common in embedded scenarios – securing a message with a WS Security Record takes only 1.18 to 1.45 times as long.



Encrypt

<Digest>...</Digest>
<Payload>...</Payload>

Fig. 2. WS Security Record

Type		WS CSec		WS SecRec		TLS	
		RC4	AES	RC4	AES	RC4	AES
Case 1	tx	749	777	508	524	238	245
	rx	773	801	532	544	222	229
Case 2	tx	809	841	568	588	248	261
	rx	832	865	592	598	232	245
Case 3	tx	1041	1077	800	820	293	309
	rx	1065	1097	825	844	277	293

TABLE IV
Message sizes in bytes for WS Security Records compared to WS Compact Security and TLS

B. Pushing "smaller" to "tinier": Compression

Compression significantly increases the performance in TLS especially regarding message sizes. To benefit from compression we compressed the complete HTTP layer using the same algorithm as TLS does (Deflate, [14]). The Deflate algorithm can be tuned using two parameters, compression quality and chunk size. The former is a value between 0 (No compression, fastest) and 9 (highest compression ratio, slowest). The latter denotes the internal buffer memory available for compression – little memory results in slow compression. We tested every

		Round Trip Times in ms		
		Case 1	Case 2	Case 3
WS CSec	RC4	29.4	30.2	41.1
	AES	39.1	40.1	51.8
WS SecRec	RC4	21.6	22.4	31.8
	AES	28.5	29.9	40.5
TLS	RC4	18.5	18.8	22.4
	AES	20.0	20.6	24.8

TABLE V
Round Trip Times (medians) of WS Security Records compared to WS Compact Security and TLS

	Round Trip Times in ms			Request size	Ratio
	1024	2048	4096	(in bytes, Case 2)	
0	31.1	31.1	31.0	871	—
1	38.0	38.1	37.9	599	0.69
2	37.9	38.2	37.9	597	0.69
3	38.1	38.2	37.9	597	0.69
4	38.5	38.7	38.5	589	0.68
5	38.8	38.9	38.5	591	0.68
6	38.7	38.9	38.5	591	0.68
7	38.7	38.8	38.6	591	0.68
8	38.7	38.9	38.6	591	0.68
9	38.7	38.9	38.7	591	0.68

TABLE VI
ROUND TRIP TIMES (MEDIANS) OF WS SECURITY RECORDS COMPARED TO WS COMPACT SECURITY AND TLS

combination of compression quality set to zero to nine and chunk sizes of 1024, 2048 and 4096 bytes for both RC4 and AES. However, we focus on RC4 only and a single test case for the sake of brevity. The results in table VI show, that compressing the whole HTTP layer leads to a compression ratio of only approx. 69%. The reason is that the ciphertext's entropy is too high to be compressed. However, compressing the payload prior to encrypting it is not reasonable because this would compress only a very small part of the message (again, assuming small payloads). However, although payloads shrink round trip times significantly grow. This means that the benefit from smaller packages is smaller than the computational overhead introduced by compression and decompression.

C. Discussion

In this section, we introduced the WS Security Records to benefit from TLS's advantages in security mechanisms on application layer. We showed that especially for small payloads round trip times are nearly equal which means there is no drawback compared to TLS in leveraging our proposed solution for setups similar to the one described in section III. A security solution on application layer even provides more flexibility and thus a richer feature set compared to TLS as described in section II. Compression does not bring any benefit in our wired setup but solely increases round trip times. Although WS Security Records lack of the high degree of interoperability compared to WS Compact Security they form a promising instrument for an approach towards providing a comprehensive security framework on application level.

VI. CONCLUSION AND OUTLOOK

In this paper we describe the feature set of TLS in its function as the default way to secure connections between embedded devices. We show that there is no noticeable increase in round trip times when TLS is used compared to leveraging no security mechanisms at all. However, we also show that it is often not desirable to rely on TLS because it requires resource-constrained devices to deal with X.509 certificates and asymmetric cryptography, it does not provide a flexible authentication framework and does not support multihop communication. Furthermore, unlike encrypting the whole transport layer, encrypting data on application level can foster secure data storage of sensitive data. To give an example on approaches for security on application layer we leverage the Devices Profile for Web Service (DPWS) and the WS Security specification. Since the latter one is not directly applicable to the use on embedded devices we propose two messaging formats, WS Compact Security and WS Security Records. We compare our approaches to TLS in terms of message sizes and round trip times. We show that especially our WS Security Record format is not noticeably slower than TLS in our experimental setup. Thus, using our approach to implement security mechanisms on application layer brings no disadvantages compared to TLS but even provides a higher degree of flexibility and thus a richer feature set.

If interoperability to classical WS Security implementations is desired, we recommend the use of the proposed WS Compact Security format because it is mappable by very simple adapters. Though it is slower than WS Security Records, its round trip times are of the same order of magnitude compared to those of TLS while providing a richer feature set.

Regarding our described approaches, we plan a more detailed investigation on the impact of compression on the power dissipation of devices with low-power wireless communication. In these scenarios communication is the most expensive part and compression could be a promising approach to reduce power dissipation even if it increases round trip times.

Furthermore, we will continue developing a security framework for embedded resource-constrained devices by adapting a long term well proven solution from a different domain, the WS Security specification suite.

REFERENCES

[1] M. Mora, "Dke-symposium: Aal – ideen bewegen, visionen leben," *VDE dialog*, no. 6, p. 8, November / December 2011.
[2] D. Masak, *Digitale Ökosysteme - Serviceorientierung bei dynamisch vernetzten Unternehmen*. Springer, Berlin, 2009.
[3] T. Dierks and C. Allen, "Rfc 2246: The tls protocol version 1.0," Internet Engineering Task Force (IETF), Tech. Rep., January 1999.
[4] K. Lawrence *et al.*, "Web services security: Soap message security 1.1," OASIS, Tech. Rep., February 2006.
[5] ——, "Ws-secureconversation 1.3," OASIS, Tech. Rep., March 2007.
[6] ——, "Ws-trust 1.3," OASIS, Tech. Rep., November 2006.
[7] OASIS, "Devices profile for web services version 1.1," Juli 2009.
[8] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski, "Lessons learned from implementing the devices profile for web services," in *Proceedings of the 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007)*, February 2007, pp. 229–232.
[9] T. Imamura, B. Dillaway, and E. Simon, "Xml encryption syntax and processing – w3c recommendation," W3C: World Wide Web Consortium, Tech. Rep., December 2002.
[10] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "Xml signature syntax and processing (second edition) – w3c recommendation," W3C: World Wide Web Consortium, Tech. Rep., June 2008.
[11] The OpenSSL Project, "Openssl: The open source toolkit for ssl/tls," http://www.openssl.org, 1998-2009.
[12] WS4D.org, "Ws4d – web services for devices," http://www.ws4d.org, 2011.
[13] OASIS, "Web services dynamic discovery (ws-discovery) version 1.1," July 2009.
[14] P. Deutsch, "Rfc 1951: Deflate compressed data format specification version 1.3," 1996.