Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2013

H. Tschofenig
Nokia Siemens Networks
J. Gilger
RWTH Aachen University
M. Kovatsch
ETH Zurich

February 25, 2013

A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks

draft-tschofenig-lwig-tls-minimal-02.txt

Abstract

Transport Layer Security (TLS) is a widely used security protocol that offers communication security services at the transport layer. The initial design of TLS was focused on the protection of applications running on top of the Transmission Control Protocol (TCP), and was a good match for securing the Hypertext Transfer Protocol (HTTP). The Stream Control Transmission Protocol (SCTP), as a more recent connection-oriented transport protocol, also benefits from TLS support. Subsequent standardization efforts lead to the publication of the Datagram Transport Layer Security (DTLS) protocol, which allows TLS payloads to be exchanged on top of the User Datagram Protocol (UDP), and the Datagram Congestion Control Protocol (DCCP).

TLS can be customized in a variety of ways and every feature has a certain cost. To offer input for implementers and system architects this document provides guidance for the usage of TLS and DTLS features for smart objects and constraint node networks.

Status of this Memo

Table of Contents

1. Introduction

The IETF published three versions of Transport Layer Security: TLS Version 1.0 [RFC2246], TLS Version 1.1 [RFC4346], and TLS Version 1.2 [RFC5246]. Section 1.1 of [RFC4346] explains the differences between Version 1.0 and Version 1.1; those are small security improvements, including the usage of an explicit initialization vector to protect against cipher-block-chaining attacks, which all have little impact for the size of the code. Section 1.2 of [RFC5246] describes the differences between Version 1.1 and Version 1.2. TLS 1.2 introduces a couple of major changes with impact to size of an implementation. In particular, prior TLS versions hardcoded the MD5/SHA-1 combination in the pseudorandom function (PRF). As a consequence, any TLS Version 1.0 and Version 1.1 implementation had to have MD5 and SHA-1 code even if the remaining cryptographic primitives used other algorithms. With TLS Version 1.2 the two had been replaced with cipher-suite-specified PRFs. In addition, the TLS extensions definition [RFC6066] and various AES ciphersuites [RFC3268] got merged into the TLS Version 1.2 specification.

All three TLS specifications list a mandatory-to-implement ciphersuite: for TLS Version 1.0 this was TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA, for TLS Version 1.1 it was TLS_RSA_WITH_3DES_EDE_CBC_SHA, and for TLS Version 1.2 it is TLS_RSA_WITH_AES_128_CBC_SHA. There is, however, an important qualification to these compliance statements, namely that they are only valid in the absence of an application profile standard specifying otherwise.

All TLS versions offer a separation between authentication and key exchange, and bulk data protection. The former is more costly performance- and message-wise. The details of the authentication and key exchange, using the TLS Handshake, vary with the chosen ciphersuite. With new ciphersuites the TLS feature-set can easily be enhanced, in case the already large collection of ciphersuites, see [TLS-IANA], does not match the requirements.

Once the TLS Handshake has been successfully completed the necessary keying material and parameters are setup for usage with the TLS Record Layer, which is responsible for bulk data protection. The TLS Record Layer could be compared with the IPsec AH and IPsec ESP while the Handshake protocol can be compared with the Internet Key Exchange Version 2 (IKEv2). The provided security of the TLS Record Layer depends also, but not only, on the chosen ciphersuite algorithms; NULL encryption ciphersuites, like those specified in RFC 4785 [RFC4785], offer only integrity- without confidentiality-protection. Example ciphersuites for the TLS Record Layer are RC4 with SHA-1, AES-128 with SHA-1, AES-256 with SHA-1, RC4 with SHA-1, RC4 with MD5

It is worth mentioning that TLS may also be used without the TLS Record Layer. This has, for example, been exercised with the work on the framework for establishing a Secure Real-time Transport Protocol (SRTP) security context using the Datagram Transport Layer Security (DTLS [RFC4347]) protocol (DTLS-SRTP [RFC5763]).

2. Communication Relationships

A security solution is strongly influenced by the communication relationships [RFC4101], which will have an impact on the code size. Having a good understanding of these relationships will be essential.

Consider the following scenario where a smart meter transmits its energy readings to other parties. The public utility has to ensure that the meter readings it obtained can be attributed to a specific meter in a household. It is simply not acceptable for public utility to have any meter readings in transit or by a rogue endpoint (particularly if the attack leads to a disadvantage, for example financial loss, for the utility). Users in a household may want to ensure that only certain parties are able to read their meter; privacy concerns come to mind.

In this example, a sensor may only ever need to talk to servers of a specific utility or even only to a single pre-configured server. Clearly, some information has to be pre-provisioned into the device to ensure the desired behavior to talk only to selected servers. The meter may come pre-configured with the domain name and certificate belonging to the utility. The device may, however, also be

configured to accept one or multiple server certificates. It may even be pre-provisioned with the server's raw public key, or a shared secret instead of relying on certificates.

Lowering the flexibility decreases the implementation overhead. TLS-PSK [RFC4279], if shared secrets are used, or raw public keys used with TLS [I-D.ietf-tls-oob-pubkey] require fewer lines of code than employing X.509 certificate, as it will be explained later in this document. A decision for constraining the client-side TLS implementation, for example by offering only a single ciphersuite, has to be made in awareness of what functionality will be available on the TLS server-side. In certain communication environments it may be easy to influence both communication partners while in other cases the existing deployment needs to be taken into consideration.

To illustrate another example, consider an Internet radio, which allows a user to connect to available radio stations. A device like this will be more demanding than an IP-enabled scale that only connects to the single Web server offered by the device manufacturer. A threat assessment may even lead to the conclusion that TLS support is not necessary at all.

Consider the following extension to our earlier scenario where the meter is attached to a home WLAN network and the interworking with WLAN security mechanisms need to be taken care of. On top of the link layer authentication, a transport layer or application layer

security mechanism needs to be implemented. Quite likely the security mechanisms will be different due to the different credential requirements. While there is a possibility for re-use of cryptographic libraries (such as the SHA-1, MD5, or HMAC) the overall code footprint will very likely be larger.

3. Design Decisions

To evaluate the required TLS functionality a couple of high level design decisions have to be made:

- What type of protection for the data traffic is required? Is confidentiality protection in addition to integrity protection required? Many TLS ciphersuites also provide a variant for NULL

encryption. If confidentiality protection is required, a carefully chosen set of algorithms may have a positive impact on the code size. For example, the RC4 stream cipher code size is 1,496 bytes compared to 7,096 bytes for AES usage. Re-use of crypto-libraries (within TLS but also among the entire protocol stack) will also help to reduce the overall code size.

- What functionality is available in hardware? For example, certain hardware platforms offer support for a random number generator as well as cryptographic algorithms (e.g., AES). These functions can be re-used and allow to reduce the amount of required code.

- What credentials for client and server authentication are required: passwords, pre-shared secrets, certificates, raw public keys (or a mixture of them)? Is certificate handling necessary? If not, then the ASN.1 library as well as the certificate parsing and processing can be omitted. If pre-shared secrets are used then the big integer implementation can be omitted.

- What TLS version and what TLS features, such as session resumption, can or have to be used? In the case of DTLS, generic fragmentation and reordering requires large buffers to reassembly the messages, which might be to heavy for some devices. Possible optimizations of DTLS for constrained environments are currently under investigation, e.g., [I-D.hartke-core-codtls].

- Is it possible to design only the client-side TLS stack, or necessary to provide the server-side implementation as well?

- Which side will be more powerful? Resource-constrained sensor nodes running CoAPS might be server only, while nodes running HTTPS are most like clients only that post their information to a normal Web server. The constrained side will most likely only implement a single ciphersuite. Flexibility is given to a more powerful counterpart that supports many different ciphersuite for various connected devices.

- Is it possible to hardwire credentials into the code rather than loading them from storage? If so, then no file handling or parsing of the credentials is needed and the credentials are

already available in form that they can be used within the TLS

implementation.

4. Conclusion

The IAB published a document, RFC 5218 [RFC5218], on the success criteria for protocols. A "wildly successful" protocol far exceeds its original goals, in terms of purpose (being used in scenarios far beyond the initial design), in terms of scale (being deployed on a scale much greater than originally envisaged), or both. TLS is an example of such a wildly successful protocol. It can be tailored to fit the needs of a specific deployment environment. This customization property offers the basis for a relatively small code footprint. The communication model and the security goals will, however, ultimately decide about the resulting code size; this is not only true for TLS but for every security solution.

More flexibility and more features will translate to a bigger footprint. Generic complaints about the large size of TLS stacks are not useful and should be accompanied by a description of the assumed functionality. To support the author's opinion this position paper provides information about the amount of required code for various functions and considers most recent work from the IETF TLS working group for the support of raw public keys.

The author is convinced that TLS is a suitable security protocol (with the standardized extensions) for usage in many smart object deployments. Only minor extensions, as currently being developed in the IETF TLS working group, are needed to support an even larger set of use cases. There are, however, cases where the security goals ask for a security solution other than TLS. With the wide range of embedded applications it is impractical to design for a single security architecture or even a single communication architecture.

5. Security Considerations

This document discusses various design aspects for reducing the footprint of TLS implementations. As such, it is entirely about security.

6. IANA Considerations

This document does not contain actions for IANA.

7. Acknowledgements

The author would like to thank the participants of the Smart Object Security workshop, March 2012.

At the time of writing the following DTLS implementations were available:

Contiki DTLS: http://code.google.com/p/contiki-dtls/

Californium: https://github.com/mkovatsc/Californium/tree/security

TinyDTLS: http://tinydtls.sourceforge.net/

8. References

8.1. Normative References

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

8.2. Informative References

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC3268] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC4785] Blumenthal, U. and P. Goel, "Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS)", RFC 4785, January 2007.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.

[I-D.hartke-core-codtls]
Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments",

draft-hartke-core-codtls-02 (work in progress), July 2012.

[RFC5218] Thaler, D. and B. Aboba, "What Makes For a Successful Protocol?", RFC 5218, July 2008.

[RFC4101] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005.

[TLS-IANA]
IANA, "Transport Layer Security (TLS) Parameters: http://www.iana.org/assignments/tls-parameters/

tls-parameters.xml", Oct 2012.

Authors' Addresses

Hannes Tschofenig
Nokia Siemens Networks

Linnoitustie 6
Espoo, 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Johannes Gilger
RWTH Aachen University
Mies-van-der-Rohe-Str. 15
Aachen, 52074
Germany

Phone: +49 (0)241 80 20 781
Email: Gilger@ITSec.RWTH-Aachen.de

Matthias Kovatsch
ETH Zurich
Universitaetstrasse 6
Zurich, CH-8092
Switzerland

Phone: +41 44 632 06 87
Email: kovatsch@inf.ethz.ch