TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 21, 2016

P. Gutmann
University of Auckland

March 20, 2016

TLS 1.2 Long-term Support Profile
draft-gutmann-tls-lts-02

Abstract

This document specifies a profile of TLS 1.2 for long-term support, one that represents what's already deployed for TLS 1.2 but with the security holes and bugs fixed. This represents a stable, known-good profile that can be deployed now to systems that can't roll out patches every month or two when the next attack on TLS is published.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 21, 2016.

Copyright Notice

Table of Contents

1. Introduction

TLS [2] and DTLS [4], by nature of their enormous complexity and the inclusion of large amounts of legacy material, contain numerous security issues that have been known to be a problem for many years and that keep coming up again and again in attacks (there are simply too many of these to provide references for, and in any case more will have been published by the time you read this). This document presents a minimal, known-good profile of mechanisms that defend against all currently-known weaknesses in TLS, that would have defended against them ten years ago, and that have a good chance of defending against them ten years from now, providing the long-term stability that's required by many systems in the field.

In particular it takes inspiration from numerous published analyses of TLS [10] [11] [12] [13] [14] [15] [16] [17] [18] along with two decades of implementation and deployment experience to select a standard interoperable feature set that provides the best chance of long-term stability and resistance to attack. This is intended for use in systems that need to run in a fixed configuration for a long time after they're deployed, with little or no ability to roll out patches every month or two when the next attack on TLS is published.

Unlike the full TLS 1.2, TLS-LTS is not meant to be all things to all people. It represents a fixed, safe solution that's appropriate for users who require a simple, secure, and long-term stable means of

getting data from A to B. This represents the majority of the non-browser use of TLS, particularly in the embedded systems that are most in need of a long-term stable protocol profile.

[Note: Because this is a rapidly-evolving document but the posting blackout before IETF 95 makes putting new versions online in the usual location difficult, updates will temporarily be posted to http://www.cs.auckland.ac.nz/~pgut001/pubs/tls-lts.txt for comment until the draft-submission process is open again].

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].

2. TLS-LTS

The use of TLS-LTS is negotiated via TLS/DTLS extensions as defined in TLS Extensions [3]. On connecting, the client includes the tls_lts extension in its client_hello if it wishes to use the TLS-LTS profile. If the server is capable of meeting this requirement, it responds with an tls_lts in its server_hello. The "extension_type" value for this extension SHALL be TBD (0xTBD) and the "extension_data" field of this extension SHALL be empty. The client and server MUST NOT use the TLS-LTS profile unless both sides have successfully exchanged tls_lts extensions.

2.1. Rationale

The use of extensions precludes use with SSL 3.0, but then it's likely that anything still using this nearly two decades-old protocol will be vulnerable to any number of other attacks anyway, so there seems little point in bending over backwards to accomodate SSL 3.0.

3. The TLS-LTS Profile

The TLS-LTS profile specifies a few simple restrictions on the huge range of TLS suites, options and parameters to limit the protocol to a known-good subset, as well as making minor corrections to limit various attacks.

3.1. Encryption/Authentication

TLS-LTS restricts the more or less unlimited TLS 1.2 with its more than three hundred cipher suites, over forty ECC parameter sets, and zoo of supplementary algorithms, parameters, and parameter formats, to just two, one traditional one with DHE + AES-CBC + HMAC-SHA-256 +

RSA-SHA-256/PSK and one ECC one with ECDHE-P256 + AES-GCM + HMAC-SHA-256 + ECDSA-P256-SHA-256/PSK with uncompressed points:

- TLS-LTS implementations MUST support TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_PSK_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256.

  [Question: There's a gap in the suites with TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256 missing, although it's present for all manner of non-AES ciphers, should we specify TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256 or fill the current hole with TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256?].

TLS-LTS only permits encrypt-then-MAC, not MAC-then-encrypt, fixing 20 years of attacks on this mechanism:

- TLS-LTS implementations MUST implement encrypt-then-MAC [5] rather than the earlier MAC-then-encrypt.

TLS-LTS drops the IPsec cargo-cult MAC truncation, which serves no obvious purpose and leads to security concerns:

- TLS-LTS implementations MUST use full-length MAC values (for example 256 bits for SHA-256). In particular MAC values MUST NOT be truncated to 96 bits/12 bytes, removing the verify_data_length constraint in the Finished message.

TLS-LTS recommends that implementations take measures to protect against side-channel attacks:

- Implementations SHOULD take steps to protect against timing attacks, for example by using constant-time implementations of algorithms and by using blinding for non-randomised algorithms like RSA.

- Implementations SHOULD take steps to protect against fault attacks, in particular for the extremely brittle ECC algorithms whose typical failure mode if a fault occurs is to leak the private key. One simple countermeasure is to use the public key to verify any signatures generated before they are sent over the wire.

TLS-LTS signs a hash of the client and server hello messages for the ServerKeyExchange rather than signing just the client and server nonces, avoiding various attacks that built on the fact that previously-exchanged parameters weren't authenticated at that point:

- When generating the ServerKeyExchange signature, the signed_params value is updated to replace the client_random and server_random with a hash of the full ClientHello and ServerHello. In other words the value being signed becomes:

```
digitally-signed struct {
    opaque client_server_hello_hash;
    ServerDHParams params;
    } signed_params;
```

The choice of key sizes is something that will never get any consensus because there are so many completely different worldviews involved. TLS-LTS makes only general recommendations on best practices and leaves the choice of which key sizes are appropriate to implementers:

- Implementations SHOULD choose public-key algorithm key sizes that are appropriate for the situation, weighted by the value of the information being protected, the probability of attack and capabilities of the attacker(s), and the ability of the system running the TLS implementation to deal with the computational load of large keys. For example a SCADA system being used to switch a ventilator on and off doesn't require anywhere near the keysize-based security of a system used to transfer classified data.

One way to avoid having to use very large public keys is to switch keys periodically. This can be done by regenerating DH parameters in a background thread and rolling them over from time to time. If this isn't possible, an alternative is to pre-generate a selection of DH parameters and choose one set at random for each new handshake, or again rolling them over from time to time, so that an attacker has to attack n sets of parameters rather than just one.

> [Question: Should the PRF be replaced with HKDF? There's no pressing need for this, but it could be part of the general cleanup].

3.2. Message Formats

TLS-LTS sends the full set of DH parameters, X9.42/FIPS 186 style, not p and g only, PKCS #3 style. This allows verification of the DH parameters, which the current format doesn't allow:

- TLS-LTS implementations MUST send the DH domain parameters as { p, g, q } rather than { p, g }. This makes the ServerDHParams field:

```
struct {
    opaque dh_p<1..2^16-1>;
    opaque dh_g<1..2^16-1>;
    opaque dh_q<1..2^16-1>;
    opaque dh_Ys<1..2^16-1>;
    } ServerDHParams;     /* Ephemeral DH parameters */
```

The domain parameters MUST be verified as specified in FIPS 186 [8].

TLS-LTS adds a hash of all messages leading up to the calculation of the master secret into the master secret to protect against the use of manipulated handshake parameters:

- TLS-LTS implementations MUST implement extended master secret [7] to protect handshake and crypto parameters.

3.3. Miscellaneous

TLS-LTS drops the need to send the current time in the random data, which serves no obvious purpose and leaks the client/server's time to attackers:

- TLS-LTS implementations SHOULD NOT include the time in the Client/ServerHello random data. The data SHOULD consists entirely of random bytes.

TLS-LTS drops compression and rehandshake, which have led to a number of attacks:

- TLS-LTS implementations MUST NOT implement compression or rehandshake.

3.4. Implementation Issues

TLS-LTS requires that RSA signature verification be done as encode-then-compare, which fixes all known padding-manipulation issues:

- TLS-LTS implementations MUST verify RSA signatures by using encode-then-compare as described in PKCS #1 [9], meaning that they encode the expected signature result and perform a constant-time compare against the recovered signature data.

The constant-time compare isn't strictly necessary for security in this case, but it's generally good hygiene and is explicitly required when comparing secret data values:

- All operations on crypto- or security-related values SHOULD be performed in a manner that's as timing-independent as possible. For example compares of MAC values such as those used in the Finished message and data packets SHOULD be performed using a constant-time memcmp() or equivalent so as not to leak timing data to an attacker.

The TLS protocol has historically and somewhat arbitrarily been described as a state machine, which has led to a number of implementation flaws when state transitions weren't very carefully considered and enforced. A more logical means of representing the protocol is as a ladder diagram, which hardcodes the transitions into the diagram and removes the need to juggle a large amount of state:

- Implementations SHOULD consider representing/implementing the protocol as a ladder diagram rather than a state machine, since the state-diagram form has led to a number of implementation errors in the past which are avoided through the use of the ladder diagram form.

TLS-LTS protects its handshake by including cryptographic integrity checks of preceding messages in subsequent messages, defeating attacks that build on the ability to manipulate handshake messages to compromise security. What's authenticated at various stages is a log of preceding messages in the exchange. The simplest way to implement this, if the underlying API supports it, is to keep a running hash of all messages (which will be required for the final Finished computation) and peel off a copy of the current hash state to generate the hash value required at various stages during the handshake. If only the traditional { Begin, [Update, Update, ...], Final } hash API interface is available then several parallel chains of hashing will need to be run in order to terminate the hashing at different points during the handshake.

3.5. Use of TLS Extensions

TLS-LTS is inspired by Grigg's Law that "there is only one mode and that is secure". Because it mandates the use of known-good mechanisms, much of the signalling and negotiation that's required in standard TLS to reach the same state becomes redundant. In particular, TLS-LTS removes the need to use the following extensions:

- The signature_algorithms extension, since the use of SHA-256 with RSA or ECDSA is implicit in TLS-LTS.

- The elliptic_curves and ec_point_formats extensions, since the use of P256 with uncompressed points is implicit in TLS-LTS.

- The almost universally-ignored requirement that all certificates provided by the server must be signed by the algorithm(s) specified in the signature_algorithms extension is removed both implicitly by not sending the extension and explicitly by removing this requirement.

- The encrypt_then_mac extension, since the use of encrypt-then-MAC is implicit in TLS-LTS.

- The extended_master_secret extension, since the use of extended Master Secret is implicit in TLS-LTS.

TLS-LTS implementations that wish to communicate only with other TLS-LTS implementations MAY omit these extensions. Implementations that wish to communicate with legacy implementations and wish to use the capabilities described by the extensions MUST include these extensions.

3.6. Downgrade Attack Prevention

The use of the TLS-LTS improvements relies on an attacker not being able to delete the TLS-LTS extension from the handshake messages. This is achieved through the SCSV [10] signalling mechanism. [SCSV boilerplate to be filled in later, this will also require banning weak cipher suites like export ones. This is a tautology, will have to work out how to ban something that in theory has already been extinct for 15 years].

3.7. Rationale

A question that may be asked at this point is, why not use TLS 1.3 instead of creating a secure profile of TLS 1.2? The reason is that TLS 1.3 rolls back the 20 years of experience that we have with all the things that can go wrong in TLS and starts again from scratch with an almost entirely new protocol based on bleeding-edge/ experimental ideas, mechanisms, and algorithms. When SSLv3 was introduced, it used ideas that were 10-20 years old (DH, RSA, DES, and so on were all long-established algorithms, only SHA-1 was relatively new). These were mature algorithms with large amounts of of research published on them, and yet we're still fixing issues with them 20 years later (the DH algorithm was published in 1976, SSLv3 dates from 1996, and the latest DH issue, Logjam, dates from 2015.

With TLS 1.3 we currently have zero implementation and deployment experience, which means that we're likely to have another 10-20 years of patching holes and fixing protocol and implementation problems ahead of us. It's for this reason that this profile uses the decades of experience we have with SSL and TLS to simplify TLS 1.2 into a

known-good subset that leverages about 15 years of analysis and 20 years of implementation experience, rather than betting on what's almost an entirely new protocol based on bleeding-edge/experimental ideas, mechanisms, and algorithms. The intent is to create a long-term stable protocol profile that can be deployed once, not deployed and then patched, updated, and fixed constantly for the lifetime of the equipment that it's used with.

4. Security Considerations

This document defines a minimal, known-good subset of TLS 1.2 that attempts to address all known weaknesses in the protocol, mostly by simply removing known-insecure mechanisms but also by updating the ones that remain to take advantage of many years of security research and implementation experience.

5. IANA Considerations

IANA has added the extension code point TBD (0xTBD) for the tls_lts extension to the TLS ExtensionType values registry as specified in TLS [2].

6. Acknowledgements

The author would like to thank the members of the TLS mailing list for their feedback on this document.

7. References

7.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[3] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions", RFC 6066, January 2011.

[4] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[5] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, September 2014.

[6] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, April 2015.

[7] Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, September 2015.

[8] "Digital Signature Standard (DSS)", FIPS 186, July 2013.

[9] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

7.2. Informative References

[10] Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., and S. Zanella-Beguelin, "Proving the TLS handshake secure (as is)", Springer-Verlag LNCS 8617, August 2014.

[11] Brzuska, C., Fischlin, M., Smart, N., Warinschi, B., and S. Williams, "Less is more: relaxed yet compatible security notions for key exchange", IACR ePrint archive 2012/242, April 2012.

[12] Dowling, B. and D. Stebila, "Modelling ciphersuite and version negotiation in the TLS protocol", Springer-Verlag LNCS 9144, June 2015.

[13] Firing, T., "Analysis of the Transport Layer Security protocol", June 2010.

[14] Gajek, S., Manulis, M., Pereira, O., Sadeghi, A., and J. Schwenk, "Universally Composable Security Analysis of TLS", Springer-Verlag LNCS 5324, November 2008.

[15] Jager, T., Kohlar, F., Schaege, S., and J. Schwenk, "On the security of TLS-DHE in the standard model", Springer-Verlag LNCS 7417, August 2012.

[16] Giesen, F., Kohlar, F., and D. Stebila, "On the security of TLS renegotiation", ACM CCS 2013, November 2013.

[17] Meyer, C. and J. Schwenk, "Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses", Cryptology ePrint Archive 2013/049, January 2013.

[18] Krawczyk, H., Paterson, K., and H. Wee, "On the security of the TLS protocol", Springer-Verlag LNCS 8042, August 2013.

Author's Address

Peter Gutmann
University of Auckland
Department of Computer Science
University of Auckland
New Zealand

Email: pgut001@cs.auckland.ac.nz