

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 23, 2015

P. Gutmann
University of Auckland
July 22, 2014

Encrypt-then-MAC for TLS and DTLS
draft-ietf-tls-encrypt-then-mac-03.txt

Abstract

This document describes a means of negotiating the use of the encrypt-then-MAC security mechanism in place of TLS'/DTLS' existing MAC-then-encrypt one, which has been the subject of a number of security vulnerabilities over a period of many years.

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 January 23, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Conventions Used in This Document	2
2. Negotiating Encrypt-then-MAC	2
2.1. Rationale	3
3. Applying Encrypt-then-MAC	3
3.1. Rehandshake Issues	5
4. Security Considerations	6
5. IANA Considerations	6
6. Acknowledgements	7
7. References	7
7.1. Normative References	7
7.2. Informative References	7
Author's Address	7

1. Introduction

TLS [2] and DTLS [4] use a MAC-then-encrypt construction that was regarded as secure at the time the original SSL protocol was specified in the mid-1990s, but that is no longer regarded as secure [5] [6]. This construction, as used in TLS and later DTLS, has been the subject of numerous security vulnerabilities and attacks stretching over a period of many years. This document specifies a means of switching to the more secure encrypt-then-MAC construction as part of the TLS/DTLS handshake, replacing the current MAC-then-encrypt construction.

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. Negotiating Encrypt-then-MAC

The use of encrypt-then-MAC is negotiated via TLS/DTLS extensions as defined in TLS [2]. On connecting, the client includes the `encrypt_then_mac` extension in its `client_hello` if it wishes to use encrypt-then-MAC rather than the default MAC-then-encrypt. If the server is capable of meeting this requirement, it responds with an `encrypt_then_mac` in its `server_hello`. The "extension_type" value for this extension SHALL be 22 (0x16) and the "extension_data" field of this extension SHALL be empty. The client and server MUST NOT use encrypt-then-MAC unless both sides have successfully exchanged `encrypt_then_mac` extensions.

2.1. Rationale

The use of TLS/DTLS extensions to negotiate an overall switch is preferable to defining new ciphersuites because the latter would result in a Cartesian explosion of suites, potentially requiring duplicating every single existing suite with a new one that uses encrypt-then-MAC. In contrast the approach presented here requires just a single new extension type with a corresponding minimal-length extension sent by client and server.

Another possibility for introducing encrypt-then-MAC would be to make it part of TLS 1.3, however this would require the implementation and deployment of all of TLS 1.2 just to support a trivial code change in the order of encryption and MAC'ing. In contrast deploying encrypt-then-MAC via the TLS/DTLS extension mechanism required changing less than a dozen lines of code in one implementation (not including the handling for the new extension type, which was a further 50 or so lines of code).

The use of extensions precludes use with SSL3.0, 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. Applying Encrypt-then-MAC

Once the use of encrypt-then-MAC has been negotiated, processing of TLS/DTLS packets switches from the standard:

```
encrypt( data || MAC || pad )
```

to the new:

```
encrypt( data || pad ) || MAC
```

with the MAC covering the entire packet up to the start of the MAC value. In TLS [2] notation the MAC calculation for TLS 1.0 without the explicit IV is:

```
MAC(MAC_write_key, seq_num +  
    TLSCipherText.type +  
    TLSCipherText.version +  
    TLSCipherText.length +  
    ENC(content + padding + padding_length));
```

and for TLS 1.1 and greater with explicit IV is:

```
MAC(MAC_write_key, seq_num +
    TLSCipherText.type +
    TLSCipherText.version +
    TLSCipherText.length +
    IV +
    ENC(content + padding + padding_length));
```

(for DTLS the sequence number is replaced by the combined epoch and sequence number as per DTLS [4]). The final MAC value is then appended to the encrypted data and padding. This calculation is identical to the existing one with the exception that the MAC calculation is run over the payload ciphertext (the TLSCipherText PDU) rather than the plaintext (the TLSCompressed PDU).

The overall TLS packet [2] is then:

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    GenericBlockCipher fragment;
    opaque MAC;
} TLSCiphertext;
```

The equivalent DTLS packet [4] is then:

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 epoch;
    uint48 sequence_number;
    uint16 length;
    GenericBlockCipher fragment;
    opaque MAC;
} TLSCiphertext;
```

This is identical to the existing TLS/DTLS layout with the only difference being that the MAC value is moved outside the encrypted data.

Note from the GenericBlockCipher annotation that this only applies to standard block ciphers that have distinct encrypt and MAC operations. It does not apply to GenericStreamCiphers, or to GenericAEADCiphers that already include integrity protection with the cipher. If a server receives an encrypt-then-MAC request extension from a client and then selects a stream or AEAD cipher suite, it MUST NOT send an encrypt-then-MAC response extension back to the client.

Decryption reverses this processing. The MAC SHALL be evaluated before any further processing such as decryption is performed, and if the MAC verification fails then processing SHALL terminate immediately. For TLS, a fatal bad_record_mac MUST be generated [2]. For DTLS, the record MUST be discarded and a fatal bad_record_mac MAY be generated [4]. This immediate response to a bad MAC eliminates any timing channels that may be available through the use of manipulated packet data.

Some implementations may prefer to use a truncated MAC rather than a full-length one. In this case they MAY negotiate the use of a truncated MAC through the TLS truncated_hmac extension as defined in TLS-Ext [3].

3.1. Rehandshake Issues

The status of encrypt-then-MAC vs. MAC-then-encrypt can potentially change during one or more rehandshakes. Implementations SHOULD retain the current session state across all rehandshakes for that session (in other words if the mechanism for the current session is X then the renegotiated session should also use X). While implementations SHOULD NOT change the state during a rehandshake, if they wish to be more flexible then the following rules apply:

Current Session	Renegotiated Session	Action to take
MAC-then-encrypt	MAC-then-encrypt	No change
MAC-then-encrypt	Encrypt-then-MAC	Upgrade to Encrypt-then-MAC
Encrypt-then-MAC	MAC-then-encrypt	Error
Encrypt-then-MAC	Encrypt-then-MAC	No change

Table 1: Encrypt-then-MAC with Renegotiation

As the above table points out, implementations MUST NOT renegotiate a downgrade from Encrypt-then-MAC to MAC-then-Encrypt. Note that a client or server that doesn't wish to implement the mechanism-change-during-rehandshake ability can (as a client) not request a mechanism change and (as a server) deny the mechanism change.

Note that these rules apply across potentially many rehandshakes. For example if a session were in the Encrypt-then-MAC state and a

rehandshake selected a GenericAEADCiphers ciphersuite and a subsequent rehandshake then selected a MAC-then-Encrypt ciphersuite, this is an error since the renegotiation process has resulted in a downgrade from Encrypt-then-MAC to MAC-then-Encrypt (via the AEAD ciphersuite).

(As the text above has already pointed out, implementations SHOULD avoid having to deal with these cipher-suite calisthenics by retaining the initially-negotiated mechanism across all rehandshakes).

If an upgrade from MAC-then-encrypt to Encrypt-then-MAC is negotiated as per the second line in the table above then the change will take place in the first message that follows the Change Cipher Spec (CCS). In other words all messages up to and including the CCS will use MAC-then-encrypt, and then the message that follows will continue with Encrypt-then-MAC.

4. Security Considerations

This document defines an improved security mechanism encrypt-then-MAC to replace the current MAC-then-encrypt one. This is regarded as more secure than the current mechanism [5] [6], and should mitigate or eliminate a number of attacks on the current mechanism, provided that the instructions on MAC processing given in Section 3 are applied.

An active attacker who can emulate a client or server with extension intolerance may cause some implementations to fall back to older protocol versions that don't support extensions, which will in turn force a fallback to non-Encrypt-then-MAC behaviour. A straightforward solution to this problem is to avoid fallback to older, less secure protocol versions. If fallback behaviour is unavoidable then mechanisms to address this issue, which affects all capabilities that are negotiated via TLS extensions, are being developed by the TLS working group [7]. Anyone concerned about this type of attack should consult the TLS working group documents for guidance on appropriate defence mechanisms.

5. IANA Considerations

IANA has added the extension code point 22 (0x16) for the `encrypt_then_mac` extension to the TLS ExtensionType values registry as specified in TLS [2].

6. Acknowledgements

The author would like to thank Martin Rex, Dan Shumow, and 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.

7.2. Informative References

- [5] Bellare, M. and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", Springer-Verlag LNCS 1976, December 2000.
- [6] Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)", Springer-Verlag LNCS 2139, August 2001.
- [7] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC XXXX, November 2013.

Author's Address

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

Email: pgut001@cs.auckland.ac.nz

Network Working Group
Internet-Draft
Updates: 5246, 6347 (if approved)
Intended status: Standards Track
Expires: October 5, 2015

A. Langley
W. Chang
Google Inc
N. Mavrogiannopoulos
Red Hat
J. Strombergson
Secworks Sweden AB
S. Josefsson
SJD AB
April 3, 2015

The ChaCha Stream Cipher for Transport Layer Security
draft-mavrogiannopoulos-chacha-tls-05

Abstract

This document describes the use of the ChaCha stream cipher with Poly1305 in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols.

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 October 5, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. The ChaCha Cipher	3
3. The Poly1305 Authenticator	3
4. ChaCha20 Cipher Suites	3
4.1. ChaCha20 Cipher Suites with Poly1305	4
5. Acknowledgements	4
6. IANA Considerations	4
7. Security Considerations	5
8. References	5
8.1. Normative References	5
8.2. Informative References	6
Authors' Addresses	7

1. Introduction

This document describes the use of the ChaCha stream cipher in the Transport Layer Security (TLS) version 1.2 [RFC5246] protocol, as well as in the Datagram Transport Layer Security (DTLS) version 1.2 [RFC6347], or any later versions.

ChaCha [CHACHA] is a stream cipher that has been designed for high performance in software implementations. The cipher has compact implementation and uses few resources and inexpensive operations that makes it suitable for implementation on a wide range of architectures. It has been designed to prevent leakage of information through side channel analysis, has a simple and fast key setup and provides good overall performance. It is a variant of Salsa20 [SALSA20SPEC] which is one of the selected ciphers in the eSTREAM portfolio [ESTREAM].

Recent attacks [CBC-ATTACK] have indicated problems with CBC-mode cipher suites in TLS and DTLS as well as issues with the only supported stream cipher (RC4) [RC4-ATTACK]. While the existing AEAD (AES-GCM) ciphersuites address some of these issues, concerns about the performance and ease of software implementation are sometimes raised.

Therefore, a new stream cipher to replace RC4 and address all the previous issues is needed. It is the purpose of this document to describe a secure stream cipher for both TLS and DTLS that is comparable to RC4 in speed on a wide range of platforms and can be

implemented easily without being vulnerable to software side-channel attacks.

2. The ChaCha Cipher

ChaCha [CHACHA] is a stream cipher developed by D. J. Bernstein in 2008. It is a refinement of Salsa20 and was used as the core of the SHA-3 finalist, BLAKE.

The variant of ChaCha used in this document is ChaCha with 20 rounds, a 96-bit nonce and a 256 bit key, which will be referred to as ChaCha20 in the rest of this document. This is the conservative variant (with respect to security) of the ChaCha family and is described in [I-D.irtf-cfrg-chacha20-poly1305].

3. The Poly1305 Authenticator

Poly1305 [POLY1305] is a Wegman-Carter, one-time authenticator designed by D. J. Bernstein. Poly1305 takes a 32-byte, one-time key and a message and produces a 16-byte tag that authenticates the message such that an attacker has a negligible chance of producing a valid tag for an inauthentic message. It is described in [I-D.irtf-cfrg-chacha20-poly1305].

4. ChaCha20 Cipher Suites

In the next sections different ciphersuites are defined that utilize the ChaCha20 cipher combined with various message authentication methods.

In all cases, the ChaCha20 cipher, as in [I-D.irtf-cfrg-chacha20-poly1305], uses a 96-bit nonce. That nonce is updated on the encryption of every TLS record, and is formed as follows.

```
struct {  
    opaque salt[4];  
    opaque record_counter[8];  
} ChaChaNonce;
```

The salt is generated as part of the handshake process. It is either the client_write_IV (when the client is sending) or the server_write_IV (when the server is sending). The salt length (SecurityParameters.fixed_iv_length) is 4 bytes. The record_counter is the 64-bit TLS record sequence number. In case of DTLS the record_counter is formed as the concatenation of the 16-bit epoch with the 48-bit sequence number.

In both TLS and DTLS the ChaChaNonce is implicit and not sent as part of the packet.

The pseudorandom function (PRF) for TLS 1.2 is the TLS PRF with SHA-256 as the hash function.

The RSA, DHE_RSA, ECDHE_RSA, ECDHE_ECDSA, PSK, DHE_PSK, RSA_PSK, ECDHE_PSK key exchanges are performed as defined in [RFC5246], [RFC4492], and [RFC5489].

4.1. ChaCha20 Cipher Suites with Poly1305

The ChaCha20 and Poly1305 primitives are built into an AEAD algorithm [RFC5116], AEAD_CHACHA20_POLY1305, described in [I-D.irtf-cfrg-chacha20-poly1305]. It takes as input a 256-bit key and a 96-bit nonce.

When used in TLS, the "record_iv_length" is zero and the nonce is set to be the ChaChaNonce. The additional data is seq_num + TLSCompressed.type + TLSCompressed.version + TLSCompressed.length, where "+" denotes concatenation.

The following CipherSuites are defined.

TLS_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA0}
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA1}
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA2}
TLS_DHE_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA3}
TLS_DHE_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA4}
TLS_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA5}
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA6}
TLS_RSA_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA7}

5. Acknowledgements

The authors would like to thank Zooko Wilcox-OHearn and Samuel Neves.

6. IANA Considerations

IANA is requested to assign the following Cipher Suites in the TLS Cipher Suite Registry:

TLS_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA0}
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA1}
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA2}
TLS_DHE_RSA_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA3}
TLS_DHE_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA4}
TLS_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA5}
TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA6}
TLS_RSA_PSK_WITH_CHACHA20_POLY1305	=	{0xTBD, 0xTBD}	{0xCC, 0xA7}

The ciphersuite numbers listed on the last column are numbers used for ciphersuite interoperability testing, and are the suggested to IANA to assign.

7. Security Considerations

ChaCha20 follows the same basic principle as Salsa20, a cipher with significant security review [SALSA20-SECURITY][ESTREAM]. At the time of writing this document, there are no known significant security problems with either cipher, and ChaCha20 is shown to be more resistant in certain attacks than Salsa20 [SALSA20-ATTACK]. Furthermore ChaCha20 was used as the core of the BLAKE hash function, a SHA3 finalist, that had received considerable cryptanalytic attention [NIST-SHA3].

Poly1305 is designed to ensure that forged messages are rejected with a probability of $1-(n/2^{102})$ for a $16*n$ byte message, even after sending 2^{64} legitimate messages.

The cipher suites described in this document require that a nonce is never repeated under the same key. The design presented ensures that by using the TLS sequence number which is unique and does not wrap [RFC5246].

This document should not introduce any other security considerations than those that directly follow from the use of the stream cipher ChaCha20, the AEAD_CHACHA20_POLY1305 construction, (see also the Security Considerations section of [I-D.irtf-cfrg-chacha20-poly1305]).

8. References

8.1. Normative References

- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March 2009.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [I-D.irtf-cfrg-chacha20-poly1305]
Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF protocols", draft-irtf-cfrg-chacha20-poly1305-10 (work in progress), February 2015.

8.2. Informative References

- [CHACHA] Bernstein, D., "ChaCha, a variant of Salsa20", January 2008, <<http://cr.yp.to/chacha/chacha-20080128.pdf>>.
- [POLY1305]
Bernstein, D., "The Poly1305-AES message-authentication code.", March 2005, <<http://cr.yp.to/mac/poly1305-20050329.pdf>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.
- [SALSA20SPEC]
Bernstein, D., "Salsa20 specification", April 2005, <<http://cr.yp.to/snuffle/spec.pdf>>.
- [SALSA20-SECURITY]
Bernstein, D., "Salsa20 security", April 2005, <<http://cr.yp.to/snuffle/security.pdf>>.
- [ESTREAM] Babbage, S., DeCanniere, C., Cantenaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., and M. Robshaw, "The eSTREAM Portfolio (rev. 1)", September 2008, <<http://www.ecrypt.eu.org/stream/finallist.html>>.
- [CBC-ATTACK]
AlFardan, N. and K. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols", IEEE Symposium on Security and Privacy, 2013.

[RC4-ATTACK]

Isobe, T., Ohigashi, T., Watanabe, Y., and M. Morii, "Full Plaintext Recovery Attack on Broadcast RC4", International Workshop on Fast Software Encryption , 2013.

[SALSA20-ATTACK]

Aumasson, J-P., Fischer, S., Khazaei, S., Meier, W., and C. Rechberger, "New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba", 2007,
<<http://eprint.iacr.org/2007/472.pdf>>.

[NIST-SHA3]

Chang, S., Burr, W., Kelsey, J., Paul, S., and L. Bassham, "Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition", 2012,
<<http://dx.doi.org/10.6028/NIST.IR.7896>>.

Authors' Addresses

Adam Langley
Google Inc

Email: agl@google.com

Wan-Teh Chang
Google Inc

Email: wtc@google.com

Nikos Mavrogiannopoulos
Red Hat

Email: nmav@redhat.com

Joachim Strombergson
Secworks Sweden AB

Email: joachim@secworks.se
URI: <http://secworks.se/>

Simon Josefsson
SJD AB

Email: simon@josefsson.org
URI: <http://josefsson.org/>