

Network Working Group  
Internet-Draft  
Expires: May 5, 2014

A. Langley  
W. Chang  
Google Inc  
Nov 2013

ChaCha20 and Poly1305 based Cipher Suites for TLS  
draft-agl-tls-chacha20poly1305-04

Abstract

This memo describes the use of the ChaCha20 cipher with a Poly1305 authenticator in Transport Layer Security (TLS).

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 May 5, 2014.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Requirements Notation . . . . .	4
3. ChaCha20 . . . . .	5
4. Poly1305 . . . . .	7
5. AEAD construction . . . . .	8
6. Cipher suites . . . . .	10
7. Test vectors . . . . .	11
8. Security Considerations . . . . .	14
9. IANA Considerations . . . . .	15
10. References . . . . .	16
10.1. Normative References . . . . .	16
10.2. Informative References . . . . .	16
Authors' Addresses . . . . .	17

## 1. Introduction

Existing TLS [RFC5246] cipher suites either suffer from cryptographic weaknesses (RC4), major implementation pitfalls (CBC mode block ciphers) or are difficult to efficiently and securely implement in software (AES-GCM). In order to improve the state of software TLS implementations, this memo specifies cipher suites that can be fast and secure when implemented in software without sacrificing key agility.

## 2. Requirements Notation

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 RFC 2119 [RFC2119].

### 3. ChaCha20

ChaCha20 [chacha] is a stream cipher developed by D. J. Bernstein. It is a refinement of Salsa20 and was used as the core of the SHA-3 finalist, BLAKE.

ChaCha20 maps 16, 32-bit input words to 16, 32-bit output words. By convention, 8 of the input words consist of a 256-bit key, 4 are constants and the remaining four are a nonce and block counter. The output words are converted to bytes and XORed with the plaintext to produce ciphertext. In order to generate sufficient output bytes to XOR with the whole plaintext, the block counter is incremented and ChaCha20 is run again, as many times as needed, for up to  $2^{70}$  bytes of output.

ChaCha20 operates on a state of 16, 32-bit words which are initialised from the input words. The first four input words are constants: (0x61707865, 0x3320646e, 0x79622d32, 0x6b206574). Input words 4 through 11 are taken from the 256-bit key by reading the bytes in little-endian order, in 4-byte chunks. Input words 12 and 13 are a block counter, with word 12 overflowing into word 13. Lastly, words 14 and 15 are taken from an 8-byte nonce, again by reading the bytes in little-endian order, in 4-byte chunks. The block counter words are initially zero.

ChaCha20 consists of 20 rounds, alternating between "column" rounds and "diagonal" rounds. Each round applies the following "quarter-round" function four times, to a different set of words each time. The quarter-round function updates 4, 32-bit words (a, b, c, d) as follows, where  $\ll$  is a bitwise, left rotation:

```
a += b; d ^= a; d <<= 16;
c += d; b ^= c; b <<= 12;
a += b; d ^= a; d <<= 8;
c += d; b ^= c; b <<= 7;
```

The 16 words are conceptually arranged in a four by four grid with the first word in the top-left position and the fourth word in the top-right position. The "column" rounds then apply the quarter-round function to the four columns, from left to right. The "diagonal" rounds apply the quarter-round to the top-left, bottom-right diagonal, followed by the pattern shifted one place to the right, for three more quarter-rounds.

Specifically, a column round applies the quarter-round function to the following indexes: (0, 4, 8, 12), (1, 5, 9, 13), (2, 6, 10, 14), (3, 7, 11, 15). A diagonal round applies it to these indexes: (0, 5, 10, 15), (1, 6, 11, 12), (2, 7, 8, 13), (3, 4, 9, 14).

After 20 rounds of the above processing, the original 16 input words are added to the 16 words to form the 16 output words.

The 64 output bytes are generated from the 16 output words by serialising them in little-endian order and concatenating the results.

#### 4. Poly1305

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.

The first 16 bytes of the one-time key form an integer, `_r_`, as follows: the top four bits of the bytes at indexes 3, 7, 11 and 15 are cleared, the bottom 2 bits of the bytes at indexes 4, 8 and 12 are cleared and the 16 bytes are taken as a little-endian value.

An accumulator is set to zero. For each chunk of 16 bytes from the input message, a byte with value 1 is appended and the 17 bytes are treated as a little-endian number. If the last chunk has less than 16 bytes then zero bytes are appended after the 1 byte is appended until there are 17 bytes. The value is added to the accumulator and then the accumulator is multiplied by `_r_`, all mod  $2^{130} - 5$ .

Finally the last 16 bytes of the one-time key are treated as a little-endian number and added to the accumulator, mod  $2^{128}$ . The result is serialised as a little-endian number, producing the 16 byte tag. (The original specification of Poly1305 used AES to generate the constant term of the polynomial from a counter nonce. For a more recent treatment that avoids the use of a block cipher in this fashion, as is done here, see section 9 of the NaCl specification [naclcrypto].)

## 5. AEAD construction

The ChaCha20 and Poly1305 primitives are built into an AEAD algorithm [RFC5116], AEAD\_CHACHA20\_POLY1305, that takes a 32 byte key and 8 byte nonce as follows:

ChaCha20 is run with the given key and nonce and with the two counter words set to zero. The first 32 bytes of the 64 byte output are saved to become the one-time key for Poly1305. The remainder of the output is discarded. The first counter input word is set to one and the plaintext is encrypted by XORing it with the output of invocations of the ChaCha20 function as needed, incrementing the first counter word after each block and overflowing into the second. (In the case of the TLS, limits on the plaintext size mean that the first counter word will never overflow in practice.)

The reason for generating the Poly1305 key like this rather than using key material from the handshake is that handshake key material is per-session, but for a polynomial MAC, a unique, secret key is needed per-record.

The Poly1305 key is used to calculate a tag for the following input: the concatenation of the additional data, the number of bytes of additional data, the ciphertext and the number of bytes of ciphertext. Numbers are represented as 8-byte, little-endian values. The resulting tag is appended to the ciphertext, resulting in the output of the AEAD operation.

Authenticated decryption is largely the reverse of the encryption process: generate one block of ChaCha20 keystream and use the first 32 bytes as a Poly1305 key. Feed Poly1305 the additional data and ciphertext, with the length suffixing as described above. Verify, in constant time, that the calculated Poly1305 authenticator matches the final 16 bytes of the input. If not, the input can be rejected immediately. Otherwise, run ChaCha20, starting with a counter value of one, to decrypt the ciphertext.

When used in TLS, the "record\_iv\_length" is zero and the nonce is the sequence number for the record, as an 8-byte, big-endian number. The additional data is seq\_num + TLSCompressed.type + TLSCompressed.version + TLSCompressed.length, where "+" denotes concatenation.

(In DTLS, the sequence number is only 48 bits. Thus, when used in DTLS, AEAD\_CHACHA20\_POLY1305 based cipher suites use the concatenation of the 16-bit epoch with the 48-bit sequence number as a replacement for TLS's 64-bit sequence number.)



In accordance with section 4 of RFC 5116 [RFC5116], the constants for this AEAD algorithm are as follows: `K_LEN` is 32 bytes, `N_MIN` and `N_MAX` are 8 bytes, `P_MAX` and `A_MAX` are  $2^{64}$ , `C_MAX` is  $2^{64}+16$ . An `AEAD_CHACHA20_POLY1305` ciphertext is exactly 16 octets longer than its corresponding plaintext.

## 6. Cipher suites

The following cipher suites are defined which use the AEAD\_CHACHA20\_POLY1305 algorithm:

```
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256    = {0xcc, 0x13}  
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = {0xcc, 0x14}  
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256      = {0xcc, 0x15}
```

These cipher suites use the TLS PRF [RFC5246] with SHA-256 as the hash function.

## 7. Test vectors

The following blocks contain test vectors for ChaCha20. The first line contains the 256-bit key, the second the 64-bit nonce and the last line contains a prefix of the resulting ChaCha20 key-stream.

KEY: 0000000000000000000000000000000000000000000000000000000000000000  
00000000  
NONCE: 0000000000000000  
KEYSTREAM: 76b8e0ada0f13d90405d6ae55386bd28bdd219b8a08ded1aa836efcc  
8b770dc7da41597c5157488d7724e03fb8d84a376a43b8f41518a11c  
c387b669b2ee6586

KEY: 0000000000000000000000000000000000000000000000000000000000000000  
00000001  
NONCE: 0000000000000000  
KEYSTREAM: 4540f05a9f1fb296d7736e7b208e3c96eb4fe1834688d2604f450952  
ed432d41bbe2a0b6ea7566d2a5d1e7e20d42af2c53d792b1c43fea81  
7e9ad275ae546963

KEY: 0000000000000000000000000000000000000000000000000000000000000000  
00000000  
NONCE: 0000000000000001  
KEYSTREAM: de9cba7bf3d69ef5e786dc63973f653a0b49e015adbff7134fcb7df1  
37821031e85a050278a7084527214f73efc7fa5b5277062eb7a0433e  
445f41e3

KEY: 0000000000000000000000000000000000000000000000000000000000000000  
00000000  
NONCE: 0100000000000000  
KEYSTREAM: ef3fd6d6c61578fbf5cf35bd3dd33b8009631634d21e42ac33960bd1  
38e50d32111e4caf237ee53ca8ad6426194a88545ddc497a0b466e7d  
6bbdb0041b2f586b

```
KEY: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b
1c1d1e1f
NONCE: 0001020304050607
KEYSTREAM: f798a189f195e66982105fffb640bb7757f579da31602fc93ec01ac56
f85ac3c134a4547b733b46413042c9440049176905d3be59ea1c53f1
5916155c2be8241a38008b9a26bc35941e2444177c8ade6689de9526
4986d95889fb60e84629c9bd9a5acb1cc118be563eb9b3a4a472f82e
09a7e778492b562ef7130e88dfe031c79db9d4f7c7a899151b9a4750
32b63fc385245fe054e3dd5a97a5f576fe064025d3ce042c566ab2c5
07b138db853e3d6959660996546cc9c4a6eafdc777c040d70eaf46f7
6dad3979e5c5360c3317166a1c894c94a371876a94df7628fe4eaa2f
ccb27d5aaae0ad7ad0f9d4b6ad3b54098746d4524d38407a6deb3ab7
8fab78c9
```

The following blocks contain test vectors for Poly1305. The first line contains a variable length input. The second contains the 256-bit key and the last contains the resulting, 128-bit tag.

```
INPUT: 0000000000000000000000000000000000000000000000000000000000000000  
       0000  
KEY:   74686973206973203332d62797465206b657920666f7220506f6c793133  
       3035  
TAG:   49ec78090e481ec6c26b33b91ccc0307
```

```
INPUT: 48656c6c6f20776f726c6421
KEY:   746869732069732033322d62797465206b6579206666f7220506f6c793133
      3035
TAG:   a6f745008f81c916a20dcc74eef2b2f0
```

The following block contains a test vector for the AEAD\_CHACHA20\_POLY1305 algorithm. The first four lines consist of the standard inputs to an AEAD algorithm and the last line contains the encrypted and authenticated result.

```
KEY:      4290bcb154173531f314af57f3be3b5006da371ece272afa1b5dbdd110
          0a1007
INPUT:    86d09974840bde2a5ca
NONCE:    cd7cf67be39c794a
AD:       87e229d4500845a079c0
OUTPUT:   e3e446f7ede9a19b62a4677dabf4e3d24b876bb284753896e1d6
```

To aid implementations, the next block contains some intermediate values in the AEAD\_CHACHA20\_POLY1305 algorithm. The first line contains the Poly1305 key that is derived and the second contains the raw bytes that are authenticated by Poly1305.

KEY: 9052a6335505b6d507341169783dccac0e26f84ea84906b1558c05bf4815  
0fbe  
INPUT: 87e229d4500845a079c00a000000000000000e3e446f7ede9a19b62a40a00  
000000000000

## 8. Security Considerations

ChaCha20 is designed to provide a 256-bit security level. 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 AEAD\_CHACHA20\_POLY1305 algorithm is designed to meet the standard notions of privacy and authenticity. For formal definitions see Authenticated Encryption [AE].

These cipher suites require that a nonce never be repeated for the same key. This is achieved by simply using the TLS sequence number.

Only forward secure cipher suites are defined as it's incongruous to define a high-security cipher suite without forward security.

## 9. IANA Considerations

IANA is requested to assign the values for the cipher suites defined in this document from the TLS registry.

IANA is requested to assign a value for AEAD\_CHACHA20\_POLY1305 in the registry of AEAD algorithms.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.
- [chacha] Bernstein, D., "ChaCha, a variant of Salsa20.", Jan 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>>.

### 10.2. Informative References

- [AE] Bellare, M. and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", <<http://cseweb.ucsd.edu/~mihir/papers/oem.html>>.
- [naclcrypto] Bernstein, D., "http://cr.yp.to/highspeed/naclcrypto-20090310.pdf", March 2009, <<http://cr.yp.to/highspeed/naclcrypto-20090310.pdf>>.



Authors' Addresses

Adam Langley  
Google Inc

Email: [agl@google.com](mailto:agl@google.com)

Wan-Teh Chang  
Google Inc

Email: [wtc@google.com](mailto:wtc@google.com)



TLS Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: June 20, 2014

P. Gutmann  
University of Auckland  
December 17, 2013

Encrypt-then-MAC for TLS and DTLS  
draft-gutmann-tls-encrypt-then-mac-05.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 June 20, 2014.

Copyright Notice

Copyright (c) 2013 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 . . . . .	2
3. Applying Encrypt-then-MAC . . . . .	3
3.1. Rehandshake Issues . . . . .	5
4. Security Considerations . . . . .	5
5. IANA Considerations . . . . .	6
6. Acknowledgements . . . . .	6
7. References . . . . .	6
7.1. Normative References . . . . .	6
7.2. Informative References . . . . .	6
Author's Address . . . . .	6

## 1. Introduction

[2] and [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 [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 is [TBD] and the "extension\_data" field of this extension SHALL be empty.

## 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 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. 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 [2] notation the MAC calculation is:

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

for TLS 1.0 without the explicit IV and:

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

for TLS 1.1 and greater with explicit IV. 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).

In [2] notation the overall packet is then:

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

This is identical to the existing TLS layout with the only difference being that the MAC value is moved outside the encrypted data. The change for DTLS follows similarly, the only difference being that in place of the 64-bit implicit sequence number DTLS contains the two 32-bit fields 'epoch' and 'sequence\_number' between the version and length.

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. This 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 [3].

[Implementation note: There is a test server available for interop testing at <https://eid.vx4.net:443/>. This uses the "extension\_type" value 0x10, which was the first unassigned TLS extension value at the time the original specification was written, since this draft has been stalled by the TLS WG chairs for a considerable amount of time this value has now been assigned to another RFC but is currently still used for interop-testing purposes. The server has been tested successfully with several different implementations].

### 3.1. Rehandshake Issues

The status of encrypt-then-MAC vs. MAC-then-encrypt can potentially change during a rehandshake. Implementations SHOULD retain the current session state for the renegotiated session (in other words if the mechanism for the current session is X then the renegotiated session should also use X). If implementations 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

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.

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.

## 5. IANA Considerations

This document defines a new extension for TLS/DTLS.

## 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] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.
- [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.

Author's Address



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

Email: [pgut001@cs.auckland.ac.nz](mailto:pgut001@cs.auckland.ac.nz)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 4, 2014

S. Friedl  
Cisco Systems, Inc.  
A. Popov  
Microsoft Corp.  
A. Langley  
Google Inc.  
E. Stephan  
Orange  
March 3, 2014

Transport Layer Security (TLS) Application Layer Protocol Negotiation  
Extension  
draft-ietf-tls-applayerprotoneg-05

Abstract

This document describes a Transport Layer Security (TLS) extension for application layer protocol negotiation within the TLS handshake. For instances in which the TLS connection is established over a well known TCP or UDP port not associated with the desired application layer protocol, this extension allows the application layer to negotiate which protocol will be used within the TLS connection.

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 4, 2014.

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
2. Requirements Language . . . . .	3
3. Application Layer Protocol Negotiation . . . . .	3
3.1. The Application Layer Protocol Negotiation Extension . .	3
3.2. Protocol Selection . . . . .	5
4. Design Considerations . . . . .	5
5. Security Considerations . . . . .	6
6. IANA Considerations . . . . .	6
7. Acknowledgements . . . . .	7
8. References . . . . .	8
8.1. Normative References . . . . .	8
8.2. Informative References . . . . .	8
Authors' Addresses . . . . .	8

## 1. Introduction

Increasingly, application layer protocols are encapsulated in the TLS security protocol [RFC5246]. This encapsulation enables applications to use the existing, secure communications links already present on port 443 across virtually the entire global IP infrastructure.

When multiple application protocols are supported on a single server-side port number, such as port 443, the client and the server need to negotiate an application protocol for use with each connection. It is desirable to accomplish this negotiation without adding network round-trips between the client and the server, as each round-trip will degrade an end-user's experience. Further, it would be advantageous to allow certificate selection based on the negotiated application protocol.

This document specifies a TLS extension which permits the application layer to negotiate protocol selection within the TLS handshake. This work was requested by the HTTPbis WG to address the negotiation of HTTP version ([RFC2616], [I-D.ietf-httpbis-http2]) over TLS, however ALPN facilitates negotiation of arbitrary application layer protocols.

With ALPN, the client sends the list of supported application protocols as part of the TLS ClientHello message. The server chooses a protocol and sends the selected protocol as part of the TLS ServerHello message. The application protocol negotiation can thus be accomplished within the TLS handshake, without adding network round-trips, and allows the server to associate a different certificate with each application protocol, if desired.

## 2. Requirements Language

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 [RFC2119].

## 3. Application Layer Protocol Negotiation

### 3.1. The Application Layer Protocol Negotiation Extension

A new extension type ("application\_layer\_protocol\_negotiation(16)") is defined and MAY be included by the client in its "ClientHello" message.

```
enum {  
    application_layer_protocol_negotiation(16), (65535)  
} ExtensionType;
```

The "extension\_data" field of the ("application\_layer\_protocol\_negotiation(16)") extension SHALL contain a "ProtocolNameList" value.

```
opaque ProtocolName<1..2^8-1>;
```

```
struct {  
    ProtocolName protocol_name_list<2..2^16-1>  
} ProtocolNameList;
```

"ProtocolNameList" contains the list of protocols advertised by the client, in descending order of preference. Protocols are named by IANA registered, opaque, non-empty byte strings, as described further in Section 6 "IANA Considerations" of this document. Empty strings MUST NOT be included and byte strings MUST NOT be truncated .

Servers that receive a client hello containing the "application\_layer\_protocol\_negotiation" extension, MAY return a suitable protocol selection response to the client. The server will ignore any protocol name that it does not recognize. A new ServerHello extension type ("application\_layer\_protocol\_negotiation(16)") MAY be returned to the

client within the extended ServerHello message. The "extension\_data" field of the ("application\_layer\_protocol\_negotiation(16)") extension is structured the same as described above for the client "extension\_data", except that the "ProtocolNameList" MUST contain exactly one "ProtocolName".

Therefore, a full handshake with the "application\_layer\_protocol\_negotiation" extension in the ClientHello and ServerHello messages has the following flow (contrast with section 7.3 of [RFC5246]):

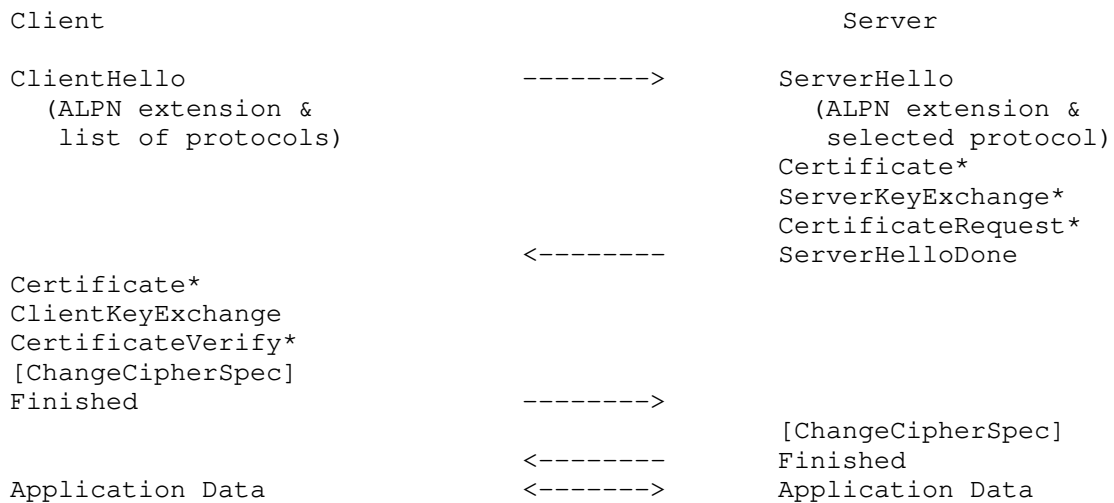


Figure 1

An abbreviated handshake with the "application\_layer\_protocol\_negotiation" extension has the following flow:

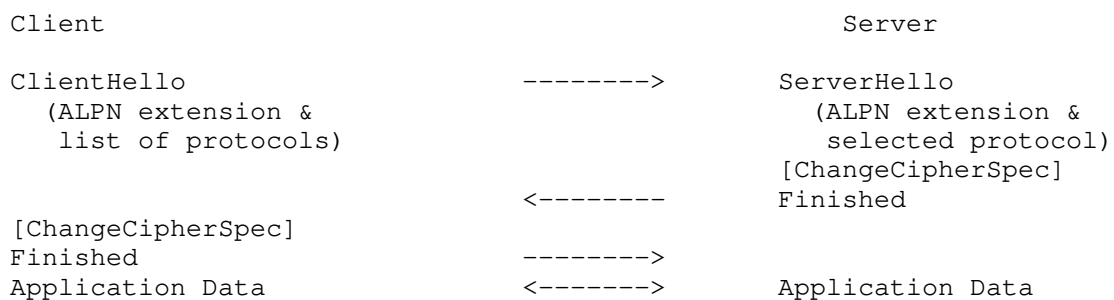


Figure 2

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

### 3.2. Protocol Selection

It is expected that a server will have a list of protocols that it supports, in preference order, and will only select a protocol if the client supports it. In that case, the server SHOULD select the most highly preferred protocol it supports which is also advertised by the client. In the event that the server supports no protocols that the client advertises, then the server SHALL respond with a fatal "no\_application\_protocol" alert.

```
enum {  
    no_application_protocol(120),  
    (255)  
} AlertDescription;
```

The protocol identified in the "application\_layer\_protocol\_negotiation" extension type in the ServerHello SHALL be definitive for the connection, until renegotiated. The server SHALL NOT respond with a selected protocol and subsequently use a different protocol for application data exchange.

### 4. Design Considerations

The ALPN extension is intended to follow the typical design of TLS protocol extensions. Specifically, the negotiation is performed entirely within the client/server hello exchange in accordance with established TLS architecture. The "application\_layer\_protocol\_negotiation" ServerHello extension is intended to be definitive for the connection (until the connection is renegotiated) and is sent in plaintext to permit network elements to provide differentiated service for the connection when the TCP or UDP port number is not definitive for the application layer protocol to be used in the connection. By placing ownership of protocol selection on the server, ALPN facilitates scenarios in which certificate selection or connection rerouting may be based on the negotiated protocol.

Finally, by managing protocol selection in the clear as part of the handshake, ALPN avoids introducing false confidence with respect to the ability to hide the negotiated protocol in advance of establishing the connection. If hiding the protocol is required,

then renegotiation after connection establishment, which would provide true TLS security guarantees, would be a preferred methodology.

## 5. Security Considerations

The ALPN extension does not impact the security of TLS session establishment or application data exchange. ALPN serves to provide an externally visible marker for the application layer protocol associated with the TLS connection. Historically, the application layer protocol associated with a connection could be ascertained from the TCP or UDP port number in use.

Implementers and document editors who intend to extend the protocol identifier registry by adding new protocol identifiers should consider that in TLS versions 1.2 and below the client sends these identifiers in the clear, and should also consider that for at least the next decade, it is expected that browsers would normally use these earlier versions of TLS in the initial ClientHello.

Care must be taken when such identifiers may leak personally identifiable information, or when such leakage may lead to profiling, or to leaking of sensitive information. If any of these apply to this new protocol identifier, the identifier SHOULD NOT be used in TLS configurations where it would be visible in the clear, and documents specifying such protocol identifiers SHOULD recommend against such unsafe use.

## 6. IANA Considerations

The IANA has updated its Registry of TLS ExtensionType Values to include the following entry:

16 application\_layer\_protocol\_negotiation

This document establishes a registry for protocol identifiers entitled "Application Layer Protocol Negotiation (ALPN) Protocol IDs" under the existing "Transport Layer Security (TLS)" heading.

Entries in this registry require the following fields:

- o Protocol: The name of the protocol.
- o Identification Sequence: The precise set of octet values that identifies the protocol. This could be the UTF-8 encoding [RFC3629] of the protocol name.

- o Specification: A reference to a specification that defines the protocol.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations of the identified protocol.

An initial set of registrations for this registry follows:

Protocol: HTTP/1.1

Identification Sequence: 0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31  
("http/1.1")

Specification: <http://tools.ietf.org/html/rfc2616>

Protocol: SPDY/1

Identification Sequence: 0x73 0x70 0x64 0x79 0x2f 0x31 ("spdy/1")

Specification: <http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1>

Protocol: SPDY/2

Identification Sequence: 0x73 0x70 0x64 0x79 0x2f 0x32 ("spdy/2")

Specification: <http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2>

Protocol: SPDY/3

Identification Sequence: 0x73 0x70 0x64 0x79 0x2f 0x33 ("spdy/3")

Specification: <http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3>

## 7. Acknowledgements

This document benefitted specifically from the NPN extension draft authored by Adam Langley and from discussions with Tom Wesselman and Cullen Jennings both of Cisco.



## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

### 8.2. Informative References

- [I-D.ietf-httpbis-http2] Belshé, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-10 (work in progress), February 2014.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

## Authors' Addresses

Stephan Friedl  
Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134  
USA  
  
Phone: (720)562-6785  
Email: sfriedl@cisco.com

Andrei Popov  
Microsoft Corp.  
One Microsoft Way  
Redmond, WA 98052  
USA

Email: andreipo@microsoft.com

Adam Langley  
Google Inc.  
USA

Email: agl@google.com

Emile Stephan  
Orange  
2 avenue Pierre Marzin  
Lannion F-22307  
France

Email: emile.stephan@orange.com

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 31, 2014

S. Josefsson  
SJD AB  
J. Strombergson  
Secworks Sweden AB  
N. Mavrogiannopoulos  
Red Hat  
November 27, 2013

The Salsa20 Stream Cipher for Transport Layer Security  
draft-josefsson-salsa20-tls-04

Abstract

This document describe how the Salsa20 stream cipher can be used in the 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 May 31, 2014.

Copyright Notice

Copyright (c) 2013 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. Salsa20 Cipher Suites . . . . .	3
2.1. Salsa20 Cipher Suites with HMAC-SHA1 . . . . .	3
3. The TLS GenericStreamCipher . . . . .	4
4. Acknowledgements . . . . .	5
5. IANA Considerations . . . . .	5
6. Security Considerations . . . . .	5
7. Algorithm Selection Background . . . . .	6
8. References . . . . .	6
8.1. Normative References . . . . .	6
8.2. Informative References . . . . .	7
Authors' Addresses . . . . .	8

## 1. Introduction

This document describe how the Salsa20 stream cipher can be used in the Transport Layer Security (TLS) version 1.0 [RFC2246], TLS version 1.1 [RFC4346], and TLS version 1.2 [RFC5246] protocols, as well as in the Datagram Transport Layer Security (DTLS) versions 1.0 [RFC4347] and 1.2 [RFC6347]. It can also be used with Secure Sockets Layer (SSL) version 3.0 [RFC6101].

Salsa20 [SALSA20SPEC] 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. Salsa20 is one of the ciphers selected as part of the eSTREAM portfolio of stream ciphers [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 ciphersuites address these issues, concerns about their performance,

on general purpose CPUs, are sometimes raised [AEAD-PERFORMANCE]. Moreover, the DTLS protocol cannot take advantage of the fast RC4 stream cipher because it does not provide random access in the key stream.

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.

## 2. Salsa20 Cipher Suites

The following variants of Salsa20 are specified. The variants provide a range of performance and security that can be selected as appropriate.

**ESTREAM\_SALSA20:** Salsa20 with 12 rounds and a 256 bit key. This cipher is the high performant eSTREAM Salsa20 with 256 bit key.

**SALSA20:** Salsa20 with 20 rounds and a 256 bit key. This is the original (conservative with respect to security) variant of Salsa20.

In the next sections different ciphersuites are defined that utilize the Salsa20 cipher combined with various MAC methods

In all cases, the pseudorandom function (PRF) for TLS 1.2 is the TLS PRF with SHA-256 as the hash function. When used with TLS versions prior to 1.2, the PRF is calculated as specified in the appropriate version of the TLS specification.

### 2.1. Salsa20 Cipher Suites with HMAC-SHA1

The following CipherSuites are defined: (note that the third column contains the suggested to IANA ciphersuite numbers)

TLS_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x10}
TLS_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x11}
TLS_ECDHE_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x12}
TLS_ECDHE_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x13}
TLS_ECDHE_ECDSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x14}
TLS_ECDHE_ECDSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x15}
TLS_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x16}
TLS_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x17}
TLS_ECDHE_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x18}
TLS_ECDHE_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x19}
TLS_RSA_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1A}
TLS_RSA_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1B}
TLS_DHE_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1C}
TLS_DHE_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1D}
TLS_DHE_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1E}
TLS_DHE_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1F}

Note that Salsa20 requires a 64-bit nonce. That nonce is updated on the encryption of every TLS record, and is set to be the 64-bit TLS record sequence number. In case of DTLS the 64-bit nonce is formed as the concatenation of the 16-bit epoch with the 48-bit sequence number.

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].

The MAC algorithm used in the ciphersuites above is HMAC-SHA1 [RFC6234].

### 3. The TLS GenericStreamCipher

The ciphersuites defined in this document differ from the TLS RC4 ciphersuites that have been the basis for the definition of GenericStreamCipher. Unlike RC4, Salsa20 requires a nonce per record. This however, does not affect the description of the GenericStreamCipher if one assumes that a nonce is optional and depends on the cipher's characteristics (in that case RC4 uses a 0 byte nonce, and Salsa20 an 8-byte nonce).

As specified in TLS [RFC5246] the MAC is computed before encryption and the stream cipher encrypts the entire block, including the MAC.

#### 4. Acknowledgements

The authors would like to thank D. J. Bernstein, David McGrew, Wan-Teh Chang, and Adam Langley for discussion and suggestions.

#### 5. IANA Considerations

IANA is requested to allocate the following numbers in the TLS Cipher Suite Registry (note that the third column contains the suggested ciphersuite numbers):

TLS_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x10}
TLS_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x11}
TLS_ECDHE_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x12}
TLS_ECDHE_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x13}
TLS_ECDHE_ECDSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x14}
TLS_ECDHE_ECDSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x15}
TLS_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x16}
TLS_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x17}
TLS_ECDHE_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x18}
TLS_ECDHE_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x19}
TLS_RSA_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1A}
TLS_RSA_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1B}
TLS_DHE_PSK_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1C}
TLS_DHE_PSK_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1D}
TLS_DHE_RSA_WITH_ESTREAM_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1E}
TLS_DHE_RSA_WITH_SALSA20_SHA1	= {0xTBD, 0xTBD}	{0xE4, 0x1F}

#### 6. Security Considerations

The security of Salsa20 is discussed in the Salsa20 security [SALSA20-SECURITY] paper. At the time of writing this document, there are no known significant security problems with the eSTREAM variant of Salsa20, nor with the original 20 round variant. As of early 2013, the best cryptanalysis breaks 8 out of 20 rounds to recover the 256-bit secret key in  $2^{251}$  operations, using  $2^{31}$  keystream pairs (see [SALSA20-ATTACK]). For more background, see the eSTREAM report [ESTREAM].

There are no ciphersuites defined in this document that utilize the variant of Salsa20 with 128-bit key material, because (due to the design of Salsa20) they provide no performance advantage over the 256-bit variant.

This document should not introduce any other security considerations than those that directly follow from any use of the stream cipher Salsa20 and those that directly follow from introducing any set of stream cipher suites into TLS and DTLS.

## 7. Algorithm Selection Background

This draft uses Salsa20, a winner of an international competition of stream ciphers (eStream), which is easily implementable without leaking information through side-channels, i.e. timing and power attacks.

Suggestions has been made to instead use Chacha [CHACHASPEC], a derivative of Salsa20 that has been shown to be 7% faster in hardware and occupy 10% less space [VLSI-IMPL]. In our opinion the performance benefits don't justify switching from a winner of an international competition to another algorithm (even if it is a derivative of it).

This draft adds a new cipher to existing TLS and DTLS implementations which is combined with the existing MAC algorithms in TLS (i.e., HMAC-SHA1). That allows the new cipher to replace the, currently known to be broken, RC4 ciphersuites, in all TLS versions.

## 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.
- [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.



- [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.
- [RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.
- [SALSA20SPEC] Bernstein, D., "Salsa20 specification", WWW <http://cr.yp.to/snuffle/spec.pdf>, April 2005.

## 8.2. Informative References

- [RFC6101] Freier, A., Karlton, P., and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, August 2011.
- [SALSA20-SECURITY] Bernstein, D., "Salsa20 security", WWW <http://cr.yp.to/snuffle/security.pdf>, April 2005.
- [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)", WWW <http://www.ecrypt.eu.org/stream/finallist.html>, September 2008.
- [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.
- [AEAD-PERFORMANCE] Krovetz, T. and P. Rogaway, "The Software Performance of Authenticated-Encryption Modes", International Workshop on Fast Software Encryption , 2011.
- [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", WWW  
<http://eprint.iacr.org/2007/472.pdf>, 2007.

[CHACHASPEC]

Bernstein, D., "ChaCha, a variant of Salsa20", WWW  
<http://cr.yp.to/chacha/chacha-20080128.pdf>, January 2008.

[VLSI-IMPL]

Henzen, L., Carbognani, F., and W. Fichtner, "VLSI  
hardware evaluation of the stream ciphers Salsa20 and  
ChaCha, and the compression function Rumba.", 2008.

Authors' Addresses

Simon Josefsson  
SJD AB

Email: [simon@josefsson.org](mailto:simon@josefsson.org)  
URI: <http://josefsson.org/>

Joachim Strombergson  
Secworks Sweden AB

Email: [joachim@secworks.se](mailto:joachim@secworks.se)  
URI: <http://secworks.se/>

Nikos Mavrogiannopoulos  
Red Hat

Email: [nmav@redhat.com](mailto:nmav@redhat.com)

UTA  
Internet-Draft  
Intended status: Best Current Practice  
Expires: August 17, 2014

Y. Sheffer  
Porticor  
R. Holz  
TUM  
P. Saint-Andre  
&yet  
February 13, 2014

Recommendations for Secure Use of TLS and DTLS  
draft-sheffer-tls-bcp-02

Abstract

Transport Layer Security (TLS) and Datagram Transport Security Layer (DTLS) are widely used to protect data exchanged over application protocols such as HTTP, SMTP, IMAP, POP, SIP, and XMPP. Over the last few years, several serious attacks on TLS have emerged, including attacks on its most commonly used cipher suites and modes of operation. This document provides recommendations for improving the security of both software implementations and deployed services that use TLS and DTLS.

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 August 17, 2014.

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
2. Conventions used in this document . . . . .	3
3. Recommendations . . . . .	3
3.1. Protocol Versions . . . . .	3
3.2. Fallback to SSL . . . . .	4
3.3. Cipher Suites . . . . .	4
3.4. Public Key Length . . . . .	6
3.5. Compression . . . . .	6
3.6. Session Resumption . . . . .	6
4. Detailed Guidelines . . . . .	6
4.1. Cipher Suite Negotiation Details . . . . .	7
4.2. Alternative Cipher Suites . . . . .	7
5. IANA Considerations . . . . .	8
6. Security Considerations . . . . .	8
6.1. AES-GCM . . . . .	8
6.2. Forward Secrecy . . . . .	8
7. Acknowledgements . . . . .	9
8. References . . . . .	9
8.1. Normative References . . . . .	9
8.2. Informative References . . . . .	10
Appendix A. Appendix: Change Log . . . . .	11
A.1. -02 . . . . .	11
A.2. -01 . . . . .	11
A.3. -00 . . . . .	12
Authors' Addresses . . . . .	12

## 1. Introduction

Transport Layer Security (TLS) and Datagram Transport Security Layer (DTLS) are widely used to protect data exchanged over application protocols such as HTTP, SMTP, IMAP, POP, SIP, and XMPP. Over the last few years, several serious attacks on TLS have emerged, including attacks on its most commonly used cipher suites and modes of operation. For instance, both AES-CBC and RC4, which together comprise most current usage, have been attacked in the context of TLS. A companion document [I-D.sheffer-uta-tls-attacks] provides detailed information about these attacks.

Because of these attacks, those who implement and deploy TLS and DTLS need updated guidance on how TLS can be used securely. Note that this document provides guidance for deployed services, as well as software implementations. In fact, this document calls for the deployment of algorithms that are widely implemented but not yet widely deployed.

The recommendations herein take into consideration the security of various mechanisms, their technical maturity and interoperability, and their prevalence in implementations at the time of writing. These recommendations apply to both TLS and DTLS. TLS 1.3, when it is standardized and deployed in the field, should resolve the current vulnerabilities while providing significantly better functionality, and will very likely obsolete the current document.

Community knowledge about the strength of various algorithms and feasible attacks can change quickly, and experience shows that a security BCP is a point-in-time statement. Readers are advised to seek out any errata or updates that apply to this document.

## 2. 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 [RFC2119].

## 3. Recommendations

### 3.1. Protocol Versions

It is important both to stop using old, less secure versions of SSL/TLS and to start using modern, more secure versions. Therefore:

- o Implementations MUST NOT negotiate SSL version 2.

Rationale: SSLv2 has serious security vulnerabilities [RFC6176].

- o Implementations SHOULD NOT negotiate SSL version 3.

Rationale: SSLv3 [RFC6101] was an improvement over SSLv2 and plugged some significant security holes, but did not support strong cipher suites.

- o Implementations MAY negotiate TLS version 1.0 [RFC2246].

Rationale: TLS 1.0 (published in 1999) includes a way to downgrade the connection to SSLv3 and does not support more modern, strong cipher suites.

- o Implementations MAY negotiate TLS version 1.1 [RFC4346].

Rationale: TLS 1.1 (published in 2006) prevents downgrade attacks to SSL, but does not support certain stronger cipher suites.

- o Implementations MUST support, and prefer to negotiate, TLS version 1.2 [RFC5246].

Rationale: Several stronger cipher suites are available only with TLS 1.2 (published in 2008).

As of the date of this writing, the latest version of TLS is 1.2. When TLS is updated to a newer version, this document will be updated to recommend support for the latest version. If this document is not updated in a timely manner, it can be assumed that support for the latest version of TLS is recommended.

### 3.2. Fallback to SSL

Some client implementations revert to SSLv3 if the server rejected higher versions of SSL/TLS. This fallback can be forced by a MITM attacker. Moreover, IP scans [[reference?]] show that SSLv3-only servers amount to only about 3% of the current web server population. Therefore, by default clients SHOULD NOT fall back from TLS to SSLv3.

### 3.3. Cipher Suites

It is important both to stop using old, insecure cipher suites and to start using modern, more secure cipher suites. Therefore:

- o Implementations MUST NOT negotiate the NULL cipher suites.

Rationale: The NULL cipher suites offer no encryption whatsoever and thus are completely insecure.

- o Implementations MUST NOT negotiate RC4 cipher suites

Rationale: The RC4 stream cipher has a variety of cryptographic weaknesses, as documented in [I-D.popov-tls-prohibiting-rc4].

- o Implementations MUST NOT negotiate cipher suites offering only so-called "export-level" encryption (including algorithms with 40 bits or 56 bits of security).

Rationale: These cipher suites are deliberately "dumbed down" and are very easy to break.

- o Implementations SHOULD NOT negotiate cipher suites that use algorithms offering less than 128 bits of security (even if they advertise more bits, such as the 168-bit 3DES cipher suites).

Rationale: Although these cipher suites are not actively subject to breakage, their useful life is short enough that stronger cipher suites are desirable.

- o Implementations SHOULD prefer cipher suites that use algorithms with at least 128 (and, if possible, 256) bits of security.

Rationale: Although the useful life of such cipher suites is unknown, it is probably at least several years for the 128-bit ciphers and "until the next fundamental technology breakthrough" for 256-bit ciphers.

- o Implementations MUST support, and SHOULD prefer to negotiate, cipher suites offering forward secrecy, such as those in the "EDH", "DHE", and "ECDHE" families.

Rationale: Forward secrecy (sometimes called "perfect forward secrecy") prevents the recovery of information that was encrypted with older session keys, thus limiting the amount of time during which attacks can be successful.

Given the foregoing considerations, implementation of the following cipher suites is RECOMMENDED (see [RFC5289] for details):

- o TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- o TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- o TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- o TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384

We suggest that TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 be preferred in general.

Unfortunately, those cipher suites are supported only in TLS 1.2 since they are authenticated encryption (AEAD) algorithms [RFC5116]. A future version of this document might recommend cipher suites for earlier versions of TLS.

[RFC4492] allows clients and servers to negotiate ECDH parameters (curves). Clients and servers SHOULD prefer verifiably random curves (specifically Brainpool P-256, brainpoolp256r1 [RFC7027]), and fall back to the commonly used NIST P-256 (secp256r1) curve [RFC4492]. In

addition, clients SHOULD send an `ec_point_formats` extension with a single element, "uncompressed".

### 3.4. Public Key Length

Because Diffie-Hellman keys of 1024 bits are estimated to be roughly equivalent to 80-bit symmetric keys, it is better to use longer keys for the "DH" family of cipher suites. Unfortunately, some existing software cannot handle (or cannot easily handle) key lengths greater than 1024 bits. The most common workaround for these systems is to prefer the "ECDHE" family of cipher suites instead of the "DH" family, then use longer keys. Key lengths of at least 2048 bits are RECOMMENDED, since they are estimated to be roughly equivalent to 112-bit symmetric keys and might be sufficient for at least the next 10 years. In addition to 2048-bit server certificates, the use of SHA-256 fingerprints is RECOMMENDED (see [CAB-Baseline] for more details).

Note: The foregoing recommendations are preliminary and will likely be corrected and enhanced in a future version of this document.

### 3.5. Compression

Implementations and deployments SHOULD disable TLS-level compression ([RFC5246], Sec. 6.2.2).

### 3.6. Session Resumption

If TLS session resumption is used, care ought to be taken to do so safely. In particular, the resumption information (either session IDs [RFC5246] or session tickets [RFC5077]) needs to be authenticated and encrypted to prevent modification or eavesdropping by an attacker. For session tickets, a strong cipher suite SHOULD be used when encrypting the ticket (as least as strong as the main TLS cipher suite); ticket keys MUST be changed regularly, e.g. once every week, so as not to negate the effect of forward secrecy. Session ticket validity SHOULD be limited to a reasonable duration (e.g. 1 day), so as not to negate the benefits of forward secrecy.

## 4. Detailed Guidelines

The following sections provide more detailed information about the recommendations listed above.



#### 4.1. Cipher Suite Negotiation Details

Clients SHOULD include `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` as the first proposal to any server, unless they have prior knowledge that the server cannot respond to a TLS 1.2 `client_hello` message.

Servers SHOULD prefer this cipher suite (or a similar but stronger one) whenever it is proposed, even if it is not the first proposal.

Both clients and servers SHOULD include the "Supported Elliptic Curves" extension [RFC4492].

Clients are of course free to offer stronger cipher suites, e.g. using AES-256; when they do, the server SHOULD prefer the stronger cipher suite unless there are compelling reasons (e.g., seriously degraded performance) to choose otherwise.

Note that other profiles of TLS 1.2 exist that use different cipher suites. For example, [RFC6460] defines a profile that uses the `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` and `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` cipher suites.

This document is not an application profile standard, in the sense of Sec. 9 of [RFC5246]. As a result, clients and servers are still required to support the TLS mandatory cipher suite, `TLS_RSA_WITH_AES_128_CBC_SHA`.

#### 4.2. Alternative Cipher Suites

Elliptic Curves Cryptography is not universally deployed for several reasons, including its complexity compared to modular arithmetic and longstanding IPR concerns. On the other hand, there are two related issues hindering effective use of modular Diffie-Hellman cipher suites in TLS:

- o There are no protocol mechanisms to negotiate the DH groups or parameter lengths supported by client and server.
- o There are widely deployed client implementations that reject received DH parameters, if they are longer than 1024 bits.

We note that with DHE and ECDHE cipher suites, the TLS master key only depends on the Diffie Hellman parameters and not on the strength of the RSA certificate; moreover, 1024 bits DH parameters are generally considered insufficient at this time.

Because of the above, we recommend using (in priority order):

1. Elliptic Curve DHE with negotiated parameters [RFC5289]
2. TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 [RFC5288], with 2048-bit Diffie-Hellman parameters
3. The same cipher suite, with 1024-bit parameters.

With modular ephemeral DH, deployers SHOULD carefully evaluate interoperability vs. security considerations when configuring their TLS endpoints.

## 5. IANA Considerations

This document requests no actions of IANA.

## 6. Security Considerations

### 6.1. AES-GCM

Please refer to [RFC5246], Sec. 11 for general security considerations when using TLS 1.2, and to [RFC5288], Sec. 6 for security considerations that apply specifically to AES-GCM when used with TLS.

### 6.2. Forward Secrecy

Forward secrecy (also often called Perfect Forward Secrecy or "PFS") is a defense against an attacker who records encrypted conversations where the session keys are only encrypted with the communicating parties' long-term keys. Should the attacker be able to obtain these long-term keys at some point later in the future, he will be able to decrypt the session keys and thus the entire conversation. In the context of TLS and DTLS, such compromise of long-term keys is not entirely implausible. It can happen, for example, due to:

- o A client or server being attacked by some other attack vector, and the private key retrieved.
- o A long-term key retrieved from a device that has been sold or otherwise decommissioned without prior wiping.
- o A long-term key used on a device as a default key [Heninger2012].
- o A key generated by a Trusted Third Party like a CA, and later retrieved from it either by extortion or compromise [Soghoian2011].

- o A cryptographic break-through, or the use of asymmetric keys with insufficient length [Kleijnung2010].

PFS ensures in such cases that the session keys cannot be determined even by an attacker who obtains the long-term keys some time after the conversation. It also protects against an attacker who is in possession of the long-term keys, but remains passive during the conversation.

PFS is generally achieved by using the Diffie-Hellman scheme to derive session keys. The Diffie-Hellman scheme has both parties maintain private secrets and send parameters over the network as modular powers over certain cyclic groups. The properties of the so-called Discrete Logarithm Problem (DLP) allow to derive the session keys without an eavesdropper being able to do so. There is currently no known attack against DLP if sufficiently large parameters are chosen.

Unfortunately, many TLS/DTLS cipher suites were defined that do not enable PFS, e.g. `TLS_RSA_WITH_AES_256_CBC_SHA256`. We thus advocate strict use of PFS-only ciphers.

## 7. Acknowledgements

We would like to thank Stephen Farrell, Simon Josefsson, Yoav Nir, Kenny Paterson, Patrick Pelletier, and Rich Salz for their review. Thanks to Brian Smith whose "browser cipher suites" page is a great resource. Finally, thanks to all others who commented on the TLS and other lists and are not mentioned here by name.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, August 2008.

- [RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, August 2008.
- [RFC6176] Turner, S. and T. Polk, "Prohibiting Secure Sockets Layer (SSL) Version 2.0", RFC 6176, March 2011.
- [RFC7027] Merkle, J. and M. Lochter, "Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)", RFC 7027, October 2013.

## 8.2. Informative References

- [CAB-Baseline]  
"Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates Version 1.1.6", 2013,  
<<https://www.cabforum.org/documents.html>>.
- [Heninger2012]  
Heninger, N., Durumeric, Z., Wustrow, E., and J. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", Usenix Security Symposium 2012, 2012.
- [I-D.popov-tls-prohibiting-rc4]  
Popov, A., "Prohibiting RC4 Cipher Suites", draft-popov-tls-prohibiting-rc4-01 (work in progress), October 2013.
- [I-D.sheffer-uta-tls-attacks]  
Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Current Attacks on TLS and DTLS", draft-sheffer-uta-tls-attacks-00 (work in progress), February 2014.
- [Kleinjung2010]  
Kleinjung, T., "Factorization of a 768-Bit RSA Modulus", CRYPTO 10, 2010.
- [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.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.
- [RFC6101] Freier, A., Karlton, P., and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, August 2011.
- [RFC6460] Salter, M. and R. Housley, "Suite B Profile for Transport Layer Security (TLS)", RFC 6460, January 2012.
- [Soghoian2011] Soghoian, C. and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against SSL.", Proc. 15th Int. Conf. Financial Cryptography and Data Security , 2011.

#### Appendix A. Appendix: Change Log

Note to RFC Editor: please remove this section before publication.

##### A.1. -02

- o Reorganized the content to focus on recommendations.
- o Moved description of attacks to a separate document (draft-sheffer-uta-tls-attacks).
- o Strengthened recommendations regarding session resumption.

##### A.2. -01

- o Clarified our motivation in the introduction.
- o Added a section justifying the need for PFS.
- o Added recommendations for RSA and DH parameter lengths. Moved from DHE to ECDHE, with a discussion on whether/when DHE is appropriate.
- o Recommendation to avoid fallback to SSLv3.
- o Initial information about browser support - more still needed!
- o More clarity on compression.
- o Client can offer stronger cipher suites.
- o Discussion of the regular TLS mandatory cipher suite.

A.3. -00

- o Initial version.

Authors' Addresses

Yaron Sheffer  
Porticor  
29 HaHarash St.  
Hod HaSharon 4501303  
Israel

Email: yaronf.ietf@gmail.com

Ralph Holz  
Technische Universitaet Muenchen  
Boltzmannstr. 3  
Garching 85748  
Germany

Email: holz@net.in.tum.de

Peter Saint-Andre  
&yet

Email: ietf@stpeter.im