Network Working Group A. Langley
Internet-Draft Google Inc
Expires: March 5, 2014 Sept 2013

ChaCha20 and Poly1305 based Cipher Suites for TLS
draft-agl-tls-chacha20poly1305-01

Abstract

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

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

Existing TLS [RFC5246] cipher suites either suffer from cryptographic weaknesses (RC4), major implementation pitfalls (CBC mode ciphers) or are difficult to effectively 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 64 output bytes. By convention, 8 of the input words consist of a 256-bit key, 4 are constants and the remaining four are a block counter. The output bytes are XORed with the plaintext to produce ciphertext.

ChaCha20 consists of 20 rounds, alternating between "column" rounds and "diagonal" rounds. Each round applies the following "quarter-round" function four times. The quarter-round function updates 4, 32-bit words (a, b, c, d) as follows:

```
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 input words are conceptually arranged in a four by four grid with the first input word in the top-left position and the forth input 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 input 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).

Finally the original 16 words of input are added to the 16 words after 20 rounds of the above processing. The sums are written out, in little-endian form, to produce the 64 bytes of output.

The first four input words are constants: (1634760805, 857760878, 2036477234, 1797285236). Input words 4 through 11 are taken from the 256-bit key by reading the bytes in little-endian order. Input words 12 and 13 are taken from an 8-byte nonce, again by reading the bytes in little-endian order. The final two input words are a block counter, with word 14 overflowing into word 15.

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 a 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 and, 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 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.

5. AEAD construction

The ChaCha20 and Poly1305 primitives are built into an AEAD [RFC5116] 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 for 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 Poly1305 key is used to calculate a tag for the following input: the concatenation of the number of bytes of additional data, the additional data itself, the number of bytes of ciphertext and the ciphertext itself. 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: the Poly1305 key is generated and the authentication tag calculated. The calculated tag is compared against the final 16 bytes of the authenticated ciphertext in constant time. If they match, the remaining ciphertext is decrypted to produce the plaintext.

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.

6. Cipher suites

The following cipher suites are defined which use the ChaCha20, Poly1305, AEAD construction:

```
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}
```

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:       00000000000000000000000000000000000000000000000000000000
           00000000
NONCE:     0000000000000000
KEYSTREAM: 76b8e0ada0f13d90405d6ae55386bd28bdd219b8a08ded1aa836efcc
           8b770dc7da41597c5157488d7724e03fb8d84a376a43b8f41518a11c
           c387b669


KEY:       00000000000000000000000000000000000000000000000000000000
           00000001
NONCE:     0000000000000000
KEYSTREAM: 4540f05a9f1fb296d7736e7b208e3c96eb4fe1834688d2604f450952
           ed432d41bbe2a0b6ea7566d2a5d1e7e20d42af2c53d792b1c43fea81
           7e9ad275


KEY:       00000000000000000000000000000000000000000000000000000000
           00000000
NONCE:     0000000000000001
KEYSTREAM: de9cba7bf3d69ef5e786dc63973f653a0b49e015adbff7134fcb7df1
           37821031e85a050278a7084527214f73efc7fa5b5277062eb7a0433e
           445f41e3


KEY:       00000000000000000000000000000000000000000000000000000000
           00000000
NONCE:     0100000000000000
KEYSTREAM: ef3fdfd6c61578fbf5cf35bd3dd33b8009631634d21e42ac33960bd1
           38e50d32111e4caf237ee53ca8ad6426194a88545ddc497a0b466e7d
           6bbdb004
```

```
KEY:       000102030405060708090a0b0c0d0e0f101112131415161718191a1b
           1c1d1e1f
NONCE:     0001020304050607
KEYSTREAM: f798a189f195e66982105ffb640bb7757f579da31602fc93ec01ac56
           f85ac3c134a4547b733b46413042c9440049176905d3be59ea1c53f1
           5916155c2be8241a38008b9a26bc35941e2444177c8ade6689de9526
           4986d95889fb60e84629c9bd9a5acb1cc118be563eb9b3a4a472f82e
           09a7e778492b562ef7130e88dfe031c79db9d4f7c7a899151b9a4750
           32b63fc385245fe054e3dd5a97a5f576fe064025d3ce042c566ab2c5
           07b138db853e3d6959660996546cc9c4a6eafdc777c040d70eaf46f7
           6dad3979e5c5360c3317166a1c894c94a371876a94df7628fe4eaaf2
           ccb27d5aaae0ad7ad0f9d4b6ad3b54098746d4524d38407a6deb
```

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: 000000000000000000000000000000000000000000000000000000000000
       0000
KEY:   746869732069732033322d62797465206b657920666f7220506f6c793133
       3035
TAG:   49ec78090e481ec6c26b33b91ccc0307


INPUT: 48656c6c6f20776f726c6421
KEY:   746869732069732033322d62797465206b657920666f7220506f6c793133
       3035
TAG:   a6f745008f81c916a20dcc74eef2b2f0
```

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

```
KEY:    e3c37ba4984da482b4f978f314b149857f4f3027470bced382ad92889e
        d4fcb6
INPUT:  1400000cbe2f24b0b1bf5276fc91a9ad
NONCE:  0000000000000000
AD:     00000000000000001603030010
OUTPUT: 46d4b8cfb0323dcad49cafe58ad009602fe190ebb314ddab20e541fdb7
```

```
        b7541c
```

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


```
KEY:   7fd1df7665397ae3f54ee182d229d9487e927cacc4b145791dcc4b61d7d4
       da18
INPUT: 0d00000000000000000000000000000000160303001010000000000000000005a
       a35eb8027a60a34f418566ea548767
```

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 n/2^102 for a 16*n byte message, even after sending 2^64 legitimate messages.

The AEAD construction is designed to meet the standard notions of privacy and authenticity. For formal definitions see Authenticated Encryption [AE].

These cipher suites require that an 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.

10. 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>.

[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>.

Author's Address

Adam Langley
Google Inc

Email: agl@google.com