(No Working Group)
Internet-Draft
Intended status: Informational
Expires: March 11, 2019

S. Arciszewski
Paragon Initiative Enterprises

September 7, 2018

XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305
draft-arciszewski-xchacha-00

Abstract

The eXtended-nonce ChaCha cipher construction (XChaCha) allows for ChaCha-based ciphersuites to accept a 192-bit nonce with similar guarantees to the original construction, except with a much lower probability of nonce misuse occurring. This enables XChaCha constructions to be stateless, while retaining the same security assumptions as ChaCha.

This document defines XChaCha20, which uses HChaCha20 to convert the key and part of the nonce into a subkey, which is in turn used with the remainder of the nonce with ChaCha20 to generate a pseudorandom keystream (e.g. for message encryption).

This document also defines AEAD_XChaCha20_Poly1305, a variant of [RFC7539] that utilizes the XChaCha20 construction in place of ChaCha20.

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 https://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 March 11, 2019.

Table of Contents

1. Introduction 2
1.1. Notation and Conventions 3
2. AEAD_XChaCha20_Poly1305 3
2.1. Motivation for XChaCha20-Poly1305 3
2.2. HChaCha20 4
2.2.1. Test Vector for the HChaCha20 Block Function 4
2.3. XChaCha20 5
2.3.1. XChaCha20 Pseudocode 6
3. References 6
3.1. Normative References 6
3.2. URIs 6
Appendix A. Additional Test Vectors 7
A.1. Example and Test Vector for AEAD_XCHACHA20_POLY1305 . . . 7
Author's Address 8

1. Introduction

AEAD constructions (Authenticated Encryption with Associated Data) allow for message confidentiality to be assured even in the presence of adaptive chosen-ciphertext attacks, but they're known to be brittle to nonce-misuse conditions [1].

Several nonce misuse resistant cipher constructions have been proposed over the years, including AES-SIV ([RFC5297]), AES-GCM-SIV [2], and several CAESAR candidates [3].

However, a more straightforward strategy can prevent nonce misuse conditions in environments where a large number of messages are encrypted. Simply use a large enough nonce such that applications can generate them randomly for each message and the probability of a collision remains low.

To this end, we propose a solution that is already implemented in many software projects that extends the nonce of ChaCha20 to 192 bits and uses it to build an AEAD construction.

1.1. Notation and Conventions

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].

2. AEAD_XChaCha20_Poly1305

XChaCha20-Poly1305 is a variant of the ChaCha20-Poly1305 AEAD construction as defined in [RFC7539] that uses a 192-bit nonce instead of a 96-bit nonce.

The algorithm for XChaCha20-Poly1305 is as follows:

1. Calculate a subkey from the first 16 bytes of the nonce and the key, using HChaCha20 (Section 2.2).

2. Use the subkey and remaining 8 bytes of the nonce (prefixed with 4 NUL bytes) with AEAD_CHACHA20_POLY1305 from [RFC7539] as normal. The definition for XChaCha20 is given in Section 2.3.

XChaCha20-Poly1305 implementations already exist in WireGuard [4], libsodium [5], Monocypher [6], xsecretbox [7], and in Go's crypto/chacha20poly1305 [8] library.

Similarly, Google's HPolyC [9] implements XChaCha12-Poly1305.

2.1. Motivation for XChaCha20-Poly1305

The nonce used by the original ChaCha20-Poly1305 is too short to safely use with random strings for long-lived keys. XChaCha20-Poly1305 does not have this restriction.

By generating a subkey from a 128-bit nonce and the key, a reuse of only the latter 64 bits of the nonce isn't security-affecting, since the key (and thus, keystream) will be different. Additionally a re-use of only the first 128 bits of the nonce isn't security-affecting, as the nonce derived from the latter 64 bits is different.

Assuming a secure random number generator, random 192-bit nonces should experience a single collision (with probability 50%) after roughly 2^96 messages (approximately 7.2998163e+28). A more conservative threshold (2^-32 chance of collision) still allows for 2^64 messages to be sent under a single key.

Therefore, with XChaCha20-Poly1305, users can safely generate a random 192-bit nonce for each message and not worry about nonce-reuse vulnerabilities.

As long as ChaCha20-Poly1305 is a secure AEAD cipher and ChaCha is a secure pseudorandom function (PRF), XChaCha20-Poly1305 is secure.

2.2. HChaCha20

HChaCha20 is an intermediary step towards XChaCha20 based on the construction and security proof used to create XSalsa20 [10], an extended-nonce Salsa20 variant used in NaCl [11].

HChaCha20 is initialized the same way as the ChaCha cipher, except that HChaCha20 uses a 128-bit nonce and has no counter.

Consider the two figures below, where each non-whitespace character represents one nibble of information about the ChaCha states (all numbers little-endian):

```
                cccccccc  cccccccc  cccccccc  cccccccc
                kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
                kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
                bbbbbbbb  nnnnnnnn  nnnnnnnn  nnnnnnnn

        ChaCha20 State: c=constant k=key b=blockcount n=nonce

                cccccccc  cccccccc  cccccccc  cccccccc
                kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
                kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
                nnnnnnnn  nnnnnnnn  nnnnnnnn  nnnnnnnn

              HChaCha20 State: c=constant k=key n=nonce
```

After initialization, proceed through the ChaCha rounds as usual.

Once the 20 ChaCha rounds have been completed, the first 128 bits and last 128 bits of the ChaCha state (both little-endian) are concatenated, and this 256-bit subkey is returned.

2.2.1. Test Vector for the HChaCha20 Block Function

- Key = 00:01:02:03:04:05:06:07:08:09:0a:0b:0c:0d:0e:0f:10:11:12:13:14:15:16:17:18:19:1a:1b:1c:1d:1e:1f. The key is a sequence of octets with no particular structure before we copy it into the HChaCha state.

- Nonce = (00:00:00:09:00:00:00:4a:00:00:00:00:31:41:59:27)

After setting up the HChaCha state, it looks like this:

```
61707865 3320646e 79622d32 6b206574
03020100 07060504 0b0a0908 0f0e0d0c
13121110 17161514 1b1a1918 1f1e1d1c
09000000 4a000000 00000000 27594131
```

ChaCha state with the key setup.

After running 20 rounds (10 column rounds interleaved with 10 "diagonal rounds"), the HChaCha state looks like this:

```
82413b42 27b27bfe d30e4250 8a877d73
4864a70a f3cd5479 37cd6a84 ad583c7b
8355e377 127ce783 2d6a07e0 e5d06cbc
a0f9e4d5 8a74a853 c12ec413 26d3ecdc
```

HChaCha state after 20 rounds

HChaCha20 will then return only the first and last rows, resulting in the following 256-bit key:

```
82413b4 227b27bfe d30e4250 8a877d73
a0f9e4d 58a74a853 c12ec413 26d3ecdc
```

Resultant HChaCha20 subkey

2.3. XChaCha20

XChaCha20 can be constructed from an existing ChaCha20 implementation and HChaCha20. All one needs to do is:

1. Pass the key and the first 16 bytes of the 24-byte nonce to HChaCha20 to obtain the subkey.

2. Use the subkey and remaining 8 byte nonce with ChaCha20 as normal (prefixed by 4 NUL bytes, since [RFC7539] specifies a 12-byte nonce).

XChaCha20 is a stream cipher and offers no integrity guarantees without being combined with a MAC algorithm (e.g. Poly1305).

The same HChaCha20 subkey derivation can also be used in the context of an AEAD_ChaCha20_Poly1305 implementation to create AEAD_XChaCha20_Poly1305, as described in Section 2.

2.3.1. XChaCha20 Pseudocode

```
xchacha20_encrypt(key, nonce, plaintext):
    subkey = hchacha20(key, nonce[0:15])
    return chacha20_encrypt(subkey, nonce[16:23], plaintext)
```

3. References

3.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <https://www.rfc-editor.org/info/rfc5297>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <https://www.rfc-editor.org/info/rfc7539>.

3.2. URIs

[1] https://cryptologie.net/article/361/breaking-https-aes-gcm-or-a-part-of-it/

[2] https://eprint.iacr.org/2017/168.pdf

[3] https://competitions.cr.yp.to/caesar-submissions.html

[4] https://www.wireguard.com

[5] https://download.libsodium.org/doc/secret-key_cryptography/xchacha20-poly1305_construction.html

[6] https://github.com/LoupVaillant/Monocypher

[7] https://github.com/jedisct1/xsecretbox

[8] https://godoc.org/golang.org/x/crypto/chacha20poly1305#NewX

[9] https://github.com/google/hpolyc

[10] https://cr.yp.to/snuffle/xsalsa-20110204.pdf

[11] https://nacl.cr.yp.to

Appendix A. Additional Test Vectors

A.1. Example and Test Vector for AEAD_XCHACHA20_POLY1305

Plaintext:

```
000  4c 61 64 69 65 73 20 61 6e 64 20 47 65 6e 74 6c  Ladies and Gentl
016  65 6d 65 6e 20 6f 66 20 74 68 65 20 63 6c 61 73  emen of the clas
032  73 20 6f 66 20 27 39 39 3a 20 49 66 20 49 20 63  s of '99: If I c
048  6f 75 6c 64 20 6f 66 66 65 72 20 79 6f 75 20 6f  ould offer you o
064  6e 6c 79 20 6f 6e 65 20 74 69 70 20 66 6f 72 20  nly one tip for
080  74 68 65 20 66 75 74 75 72 65 2c 20 73 75 6e 73  the future, suns
096  63 72 65 65 6e 20 77 6f 75 6c 64 20 62 65 20 69  creen would be i
112  74 2e                                            t.
```

AAD:

```
 000  50 51 52 53 c0 c1 c2 c3 c4 c5 c6 c7              PQRS........
```

Key:

```
000  80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f  ................
016  90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f  ................
```

IV:

```
000  40 41 42 43 44 45 46 47 48 49 4a fb 4c 4d 4e 4f  @ABCDEFGHIJKLMNO
016  50 51 52 53 54 55 56 57                          PQRSTUVW
```

32-bit fixed-common part:

```
     000  00 00 00 00                                      ....
```

Poly1305 Key:

```
000  7b 19 1f 80 f3 61 f0 99 09 4f 6f 4b 8f b9 7d f8  {....a...OoK..}.
016  47 cc 68 73 a8 f2 b1 90 dd 73 80 71 83 f9 07 d5  G.hs.....s.q....
```

Ciphertext:

```
000  bd 6d 17 9d 3e 83 d4 3b 95 76 57 94 93 c0 e9 39  .m..>..;.vW....9
016  57 2a 17 00 25 2b fa cc be d2 90 2c 21 39 6c bb  W*..%+.....,!9l.
032  73 1c 7f 1b 0b 4a a6 44 0b f3 a8 2f 4e da 7e 39  s....J.D.../N.~9
048  ae 64 c6 70 8c 54 c2 16 cb 96 b7 2e 12 13 b4 52  .d.p.T.........R
064  2f 8c 9b a4 0d b5 d9 45 b1 1b 69 b9 82 c1 bb 9e  /......E..i.....
080  3f 3f ac 2b c3 69 48 8f 76 b2 38 35 65 d3 ff f9  ??.+.iH.v.85e...
096  21 f9 66 4c 97 63 7d a9 76 88 12 f6 15 c6 8b 13  !.fL.c}.v.......
112  b5 2e                                            ..
```

Tag:
c0:87:59:24:c1:c7:98:79:47:de:af:d8:78:0a:cf:49

Author's Address

Scott Arciszewski
Paragon Initiative Enterprises
United States

Email: security@paragonie.com