(No Working Group)
Internet-Draft
Intended status: Informational
Expires: April 14, 2019

S. Arciszewski
Paragon Initiative Enterprises

October 11, 2018

XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305
draft-arciszewski-xchacha-02

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 April 14, 2019.

Table of Contents

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], Tink [8], and in Go's crypto/chacha20poly1305 [9] library.

Similarly, Google's HPolyC [10] implements XChaCha12.

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 [11], an extended-nonce Salsa20 variant used in NaCl [12].

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:

```
82413b42 27b27bfe d30e4250 8a877d73
a0f9e4d5 8a74a853 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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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://monocypher.org/manual/aead

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

[8] https://github.com/google/tink

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

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

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

[12] 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 4b 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
```

A.2. Example and Test Vector for XChaCha20

Plaintext:

```
000  54 68 65 20 64 68 6f 6c 65 20 28 70 72 6f 6e 6f  The dhole (prono
010  75 6e 63 65 64 20 22 64 6f 6c 65 22 29 20 69 73  unced "dole") is
020  20 61 6c 73 6f 20 6b 6e 6f 77 6e 20 61 73 20 74   also known as t
030  68 65 20 41 73 69 61 74 69 63 20 77 69 6c 64 20  he Asiatic wild
040  64 6f 67 2c 20 72 65 64 20 64 6f 67 2c 20 61 6e  dog, red dog, an
050  64 20 77 68 69 73 74 6c 69 6e 67 20 64 6f 67 2e  d whistling dog.
060  20 49 74 20 69 73 20 61 62 6f 75 74 20 74 68 65   It is about the
070  20 73 69 7a 65 20 6f 66 20 61 20 47 65 72 6d 61   size of a Germa
080  6e 20 73 68 65 70 68 65 72 64 20 62 75 74 20 6c  n shepherd but l
090  6f 6f 6b 73 20 6d 6f 72 65 20 6c 69 6b 65 20 61  ooks more like a
0a0  20 6c 6f 6e 67 2d 6c 65 67 67 65 64 20 66 6f 78   long-legged fox
0b0  2e 20 54 68 69 73 20 68 69 67 68 6c 79 20 65 6c  . This highly el
0c0  75 73 69 76 65 20 61 6e 64 20 73 6b 69 6c 6c 65  usive and skille
0d0  64 20 6a 75 6d 70 65 72 20 69 73 20 63 6c 61 73  d jumper is clas
0e0  73 69 66 69 65 64 20 77 69 74 68 20 77 6f 6c 76  sified with wolv
0f0  65 73 2c 20 63 6f 79 6f 74 65 73 2c 20 6a 61 63  es, coyotes, jac
100  6b 61 6c 73 2c 20 61 6e 64 20 66 6f 78 65 73 20  kals, and foxes
110  69 6e 20 74 68 65 20 74 61 78 6f 6e 6f 6d 69 63  in the taxonomic
120  20 66 61 6d 69 6c 79 20 43 61 6e 69 64 61 65 2e   family Canidae.
```

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 4b 4c 4d 4e 4f  @ABCDEFGHIJKLMNO
016  50 51 52 53 54 55 56 58                          PQRSTUVX
```

Ciphertext:

```
000  45 59 ab ba 4e 48 c1 61 02 e8 bb 2c 05 e6 94 7f  EY..NH.a...,....
010  50 a7 86 de 16 2f 9b 0b 7e 59 2a 9b 53 d0 d4 e9  P..../..~Y*.S...
020  8d 8d 64 10 d5 40 a1 a6 37 5b 26 d8 0d ac e4 fa  ..d..@..7[&.....
030  b5 23 84 c7 31 ac bf 16 a5 92 3c 0c 48 d3 57 5d  .#..1.....<.H.W]
040  4d 0d 2c 67 3b 66 6f aa 73 10 61 27 77 01 09 3a  M.,g;fo.s.a'w..:
050  6b f7 a1 58 a8 86 42 92 a4 1c 48 e3 a9 b4 c0 da  k..X..B...H.....
060  ec e0 f8 d9 8d 0d 7e 05 b3 7a 30 7b bb 66 33 31  ......~..z0{.f31
070  64 ec 9e 1b 24 ea 0d 6c 3f fd dc ec 4f 68 e7 44  d...$..l?...Oh.D
080  30 56 19 3a 03 c8 10 e1 13 44 ca 06 d8 ed 8a 2b  0V.:.....D.....+
090  fb 1e 8d 48 cf a6 bc 0e b4 e2 46 4b 74 81 42 40  ...H......FKt.B@
0a0  7c 9f 43 1a ee 76 99 60 e1 5b a8 b9 68 90 46 6e  |.C..v.`.[..h.Fn
0b0  f2 45 75 99 85 23 85 c6 61 f7 52 ce 20 f9 da 0c  .Eu..#..a.R. ...
0c0  09 ab 6b 19 df 74 e7 6a 95 96 74 46 f8 d0 fd 41  ..k..t.j..tF...A
0d0  5e 7b ee 2a 12 a1 14 c2 0e b5 29 2a e7 a3 49 ae  ^{.*......)*..I.
0e0  57 78 20 d5 52 0a 1f 3f b6 2a 17 ce 6a 7e 68 fa  Wx .R..?.*..j~h.
0f0  7c 79 11 1d 88 60 92 0b c0 48 ef 43 fe 84 48 6c  |y...`...H.C..Hl
100  cb 87 c2 5f 0a e0 45 f0 cc e1 e7 98 9a 9a a2 20  ..._..E........
110  a2 8b dd 48 27 e7 51 a2 4a 6d 5c 62 d7 90 a6 63  ...H'.Q.Jm\b...c
120  93 b9 31 11 c1 a5 5d d7 42 1a 10 18 49 74 c7 c5  ..1...].B...It..
```

A.3. Developer-Friendly Test Vectors

For the sake of usability, the above test vectors have been reproduced in a format more readily usable by implementors.

All values below are hex-encoded, as per RFC 4648 [RFC4648].

A.3.1. AEAD_XCHACHA20_POLY1305

Plaintext:

```
4c616469657320616e642047656e746c656d656e206f662074686520636c6173
73206f66202739393a204966204920636f756c64206f6666657220796f75206f
6e6c79206f6e652074697020666f722074686520667574572652c2073756e73
637265656e20776f756c642062652069742e
```

AAD:

```
50515253c0c1c2c3c4c5c6c7
```

Key:

```
808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
```

IV:

```
404142434445464748494a4b4c4d4e4f5051525354555657
```

32-bit fixed-common part:

00000000

Poly1305 Key:

7b191f80f361f099094f6f4b8fb97df847cc6873a8f2b190dd73807183f907d5

Ciphertext:

bd6d179d3e83d43b9576579493c0e939572a1700252bfaccbed2902c21396cbb
731c7f1b0b4aa6440bf3a82f4eda7e39ae64c6708c54c216cb96b72e1213b452
2f8c9ba40db5d945b11b69b982c1bb9e3f3fac2bc369488f76b2383565d3fff9
21f9664c97637da9768812f615c68b13b52e

Tag:

c0875924c1c7987947deafd8780acf49

A.3.2. XChaCha20

Plaintext:

5468652064686f6c65202870726f6e6f756e6365642022646f6c652229206973
20616c736f206b6e6f776e20617320746865204173696174696320776965c6420
646f672c2072656420646f672c20616e642077686973746c696e6720646f672e
204974206973206162f7574207468652073697a65206f662061204765726d61
6e207368657068657264206275742061c6f6f6b73206d6f7265206c696b652061
206c6f6e672d6c656767656420666f782e2054686973206867686c7920656c
757369766520616e6420736b696c6c6564206a756d70657220697320636c6173
736966696564207697468207761c6f6c7665732c20636f796f7465732c206a6163
6b616c732c20616e6420666f78657320696e207468652074617861786f6e6f6d6963
2066616d696c792043616e696461652e

Key:

808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f

IV:

404142434445464748494a4b4c4d4e4f5051525354555658

Ciphertext:

```
4559abba4e48c16102e8bb2c05e6947f50a786de162f9b0b7e592a9b53d0d4e9
8d8d6410d540a1a6375b26d80dace4fab52384c731acbf16a5923c0c48d3575d
4d0d2c673b666faa731061277701093a6bf7a158a8864292a41c48e3a9b4c0da
ece0f8d98d0d7e05b37a307bbb66333164ec9e1b24ea0d6c3ffddcec4f68e744
3056193a03c810e11344ca06d8ed8a2bfb1e8d48cfa6bc0eb4e2464b74814240
7c9f431aee769960e15ba8b96890466ef2457599852385c661f752ce20f9da0c
09ab6b19df74e76a95967446f8d0fd415e7bee2a12a114c20eb5292ae7a349ae
577820d5520a1f3fb62a17ce6a7e68fa7c79111d8860920bc048ef43fe84486c
cb87c25f0ae045f0cce1e7989a9aa220a28bdd4827e751a24a6d5c62d790a663
93b93111c1a55dd7421a10184974c7c5
```

Author's Address

Scott Arciszewski
Paragon Initiative Enterprises
United States

Email: security@paragonie.com