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

S. Arciszewski
Paragon Initiative Enterprises

April 9, 2019

XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305
draft-irtf-cfrg-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

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, misuse resistant cipher constructions come at a cost in performance as they must necessarily make two passes over the message to be encrypted. An alternative strategy can significantly reduce the chance of accidental nonce reuse 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.

Note that we're building upon uses the IETF's ChaCha20 (96-bit nonce), not Bernstein's ChaCha20 (64-bit nonce).

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 2^80 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. Instead, the block counter is replaced by the first 32 bits of the nonce.

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:

```
423b4182 fe7bb227 50420ed3 737d878a
0aa76448 7954cdf3 846acd37 7b3c58ad
77e35583 83e77c12 e0076a2d bc6cd0e5
d5e4f9a0 53a8748a 13c42ec1 dcecd326
```

HChaCha state after 20 rounds

HChaCha20 will then return only the first and last rows, in little endian, 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), with the initial block counter set to 1.

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])
    chacha20_nonce = "\x00\x00\x00\x00" + nonce[16:23]
    blk_ctr = 1
    return chacha20_encrypt(subkey, chacha20_nonce, plaintext, blk_ctr)
```

3. Security Considerations

The security of the XChaCha construction depends on both the security of the ChaCha stream cipher and the HChaCha intermediary step, for reasons explained in the XSalsa20 paper [13] (which our XChaCha construction is derived from).

Given that the 20-round ChaCha stream cipher (ChaCha20) is believed to be at least as secure as Salsa20, our only relevant security consideration involves HChaCha20.

3.1. Proving XChaCha20 to be Secure

In the XSalsa20 paper, when HSalsa20 is defined, the author states, "The indices 0, 5, 10, 15, 6, 7, 8, 9 here were not chosen arbitrarily; the choice is important for the security proof later in this paper."

In the analysis of Theorem 3.1 from the same paper (which covers generalized cascades), the author further states: "Note that this

bound is affected less by the insecurity of H than by the insecurity of S."

This means that the security of HSalsa20 affects the security of XSalsa20 less than Salsa20 does, and the same applies to any generalized cascade following a similar construction.

However, we want to be sure that HChaCha is secure even if it's less security-affecting than ChaCha (which we already believe to be secure).

In Salsa20, the indices 0, 5, 10, 15 were populated with constants, while the indices 6, 7, 8, and 9 were populated by the nonce. The security proof for HSalsa20 relies on the definition of a function Q (specified in Theorem 3.3 of the paper) that provides two critical properties:

1. Q(i) is a public computation of H(i) and S(i).

2. Q(i) is a public computation of uniform random strings from uniform random strings.

Thus, HSalsa20 uses the same indices as the public inputs (constant + nonce) for its final output.

The reliance on public computation for the security proof makes sense, and can be applied to ChaCha with a slight tweak.

ChaCha is a slightly different construction than Salsa20: The constants occupy the indices at 0, 1, 2, 3. Meanwhile, the nonce populates indices 12, 13, 14, 15.

Consequently, we can extract a public computation from ChaCha20 by selecting these indices from HChaCha20's final state as the HChaCha20 output, and the same security proof can be used.

Therefore, if the argument that makes HSalsa20 secure is valid, then it also applies to HChaCha for the corresponding output indices.

```
HSalsa20:  0,  5, 10, 15,  6,  7,  8,  9
HChaCha:   0,  1,  2,  3, 12, 13, 14, 15
```

Input and output indices for the relevant security proof

If the 20-round HChaCha (HChaCha20) is secure, and the 20-round ChaCha20 is secure, then XChaCha20 is also secure.

4. IANA Considerations

This document defines a new stream cipher ("XChaCha20", see Section 2.3) and a new AEAD cipher construction ("XChaCha20-Poly1305", see Section 2).

A new entry in the "Authenticated Encryption with Associated Data (AEAD) Parameters" registry with the name "AEAD_XCHACHA20_POLY1305" should be assigned.

Name	Reference	Number Identifier
AEAD_XCHACHA20_POLY1305	Section 2	TBD1

5. References

5.1. 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>.

5.2. 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

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

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

Note: This is for the XChaCha20 stream cipher itself, not the AEAD construction.

Counter: 1

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
```

Keystream:

```
000: 29 62 4b 4b 1b 14 0a ce 53 74 0e 40 5b 21 68 54  )bKK....St.@[!hT
010: 0f d7 d6 30 c1 f5 36 fe cd 72 2f c3 cd db a7 f4  ...0..6..r/.....
020: cc a9 8c f9 e4 7e 5e 64 d1 15 45 0f 9b 12 5b 54  .....~^d..E...[T
030: 44 9f f7 61 41 ca 62 0a 1f 9c fc ab 2a 1a 8a 25  D..aA.b.....*..%
040: 5e 76 6a 52 66 b8 78 84 61 20 ea 64 ad 99 aa 47  ^vjRf.x.a .d...G
050: 94 71 e6 3b ef cb d3 7c d1 c2 2a 22 1f e4 62 21  .q.;...|..*"..b!
060: 5c f3 2c 74 89 5b f5 05 86 3c cd dd 48 f6 29 16  \.,t.[...<..H.).
070: dc 65 21 f1 ec 50 a5 ae 08 90 3a a2 59 d9 bf 60  .e!..P....:.Y..`
080: 7c d8 02 6f ba 54 86 04 f1 b6 07 2d 91 bc 91 24  |..o.T.....-...$
090: 3a 5b 84 5f 7f d1 71 b0 2e dc 5a 0a 84 cf 28 dd  :[._..q...Z...(.
0a0: 24 11 46 bc 37 6e 3f 48 df 5e 7f ee 1d 11 04 8c  $.F.7n?H.^......
0b0: 19 0a 3d 3d eb 0f eb 64 b4 2d 9c 6f de ee 29 0f  ..==...d.-.o..).
0c0: a0 e6 ae 2c 26 c0 24 9e a8 c1 81 f7 e2 ff d1 00  ...,&.$.........
0d0: cb e5 fd 3c 4f 82 71 d6 2b 15 33 0c b8 fd cf 00  ...<O.q.+.3.....
0e0: b3 df 50 7c a8 c9 24 f7 01 7b 7e 71 2d 15 a2 eb  ..P|..$..{~q-...
0f0: 5c 50 48 44 51 e5 4e 1b 4b 99 5b d8 fd d9 45 97  \PHDQ.N.K.[...E.
100: bb 94 d7 af 0b 2c 04 df 10 ba 08 90 89 9e d9 29  .....,.........)
110: 3a 0f 55 b8 ba fa 99 92 64 03 5f 1d 4f be 7f e0  :.U.....d._.O...
120: aa fa 10 9a 62 37 20 27 e5 0e 10 cd fe cc a1 27  ....b7 '.......'
```

Ciphertext:

```
000: 7d 0a 2e 6b 7f 7c 65 a2 36 54 26 30 29 4e 06 3b  }..k.|e.6T&0)N.;
010: 7a b9 b5 55 a5 d5 14 9a a2 1e 4a e1 e4 fb ce 87  z..U......J.....
020: ec c8 e0 8a 8b 5e 35 0a be 62 2b 2f fa 61 7b 20  .....^5..b+/.a{
030: 2c fa d7 20 32 a3 03 7e 76 ff dc dc 43 76 ee 05  ,.. 2..~v...Cv..
040: 3a 19 0d 7e 46 ca 1d e0 41 44 85 03 81 b9 cb 29  :..~F...AD.....)
050: f0 51 91 53 86 b8 a7 10 b8 ac 4d 02 7b 8b 05 0f  .Q.S......M.{...
060: 7c ba 58 54 e0 28 d5 64 e4 53 b8 a9 68 82 41 73  |.XT.(.d.S..h.As
070: fc 16 48 8b 89 70 ca c8 28 f1 1a e5 3c ab d2 01  ..H..p..(...<...
080: 12 f8 71 07 df 24 ee 61 83 d2 27 4f e4 c8 b1 48  ..q..$.a..'O...H
090: 55 34 ef 2c 5f bc 1e c2 4b fc 36 63 ef aa 08 bc  U4.,_...K.6c....
0a0: 04 7d 29 d2 50 43 53 2d b8 39 1a 8a 3d 77 6b f4  .}).PCS-.9..=wk.
0b0: 37 2a 69 55 82 7c cb 0c dd 4a f4 03 a7 ce 4c 63  7*iU.|...J....Lc
0c0: d5 95 c7 5a 43 e0 45 f0 cc e1 f2 9c 8b 93 bd 65  ...ZC.E........e
0d0: af c5 97 49 22 f2 14 a4 0b 7c 40 2c db 91 ae 73  ...I"....|@,...s
0e0: c0 b6 36 15 cd ad 04 80 68 0f 16 51 5a 7a ce 9d  ..6.....h..QZz..
0f0: 39 23 64 64 32 8a 37 74 3f fc 28 f4 dd b3 24 f4  9#dd2.7t?.(...$.
100: d0 f5 bb dc 27 0c 65 b1 74 9a 6e ff f1 fb aa 09  ....'.e.t.n.....
110: 53 61 75 cc d2 9f b9 e6 05 7b 30 73 20 d3 16 83  Sau......{0s ...
120: 8a 9c 71 f7 0b 5b 59 07 a6 6f 7e a4 9a ad c4 09  ..q..[Y..o~.....
```

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
6e6c79206f6e652074697020666f7220746865206675747572652c2073756e73
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

Counter: 1

Plaintext:

```
5468652064686f6c65202870726f6e6f756e6365642022646f6c652229206973
20616c736f206b6e6f776e2061732074686520417369617469632077696c6420
646f672c2072656420646f672c20616e642077686973746c696e6720646f672e
2049742069732061626f7574207468652073697a65206f662061204765726d61
6e20736865706865726420627574206c6f6f6b73206d6f7265206c696b652061
206c6f6e672d6c656767656420666f782e205468697320686967686c7920656c
757369766520616e6420736b696c6c6564206a756d70657220697320636c6173
736966696564207769746820776f6c7665732c20636f796f7465732c206a6163
6b616c732c20616e6420666f78657320696e20746865207461786f6e6f6d6963
2066616d696c792043616e696461652e
```

Key:

```
808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f
```

IV:

```
404142434445464748494a4b4c4d4e4f5051525354555658
```

Keystream:

```
29624b4b1b140ace53740e405b2168540fd7d630c1f536fecd722fc3cddba7f4
cca98cf9e47e5e64d115450f9b125b54449ff76141ca620a1f9cfcab2a1a8a25
5e766a5266b878846120ea64ad99aa479471e63befcbd37cd1c22a221fe46221
5cf32c74895bf505863ccddd48f62916dc6521f1ec50a5ae08903aa259d9bf60
7cd8026fba548604f1b6072d91bc91243a5b845f7fd171b02edc5a0a84cf28dd
241146bc376e3f48df5e7fee1d11048c190a3d3deb0feb64b42d9c6fdeee290f
a0e6ae2c26c0249ea8c181f7e2ffd100cbe5fd3c4f8271d62b15330cb8fdcf00
b3df507ca8c924f7017b7e712d15a2eb5c50484451e54e1b4b995bd8fdd94597
bb94d7af0b2c04df10ba0890899ed9293a0f55b8bafa999264035f1d4fbe7fe0
aafa109a62372027e50e10cdfecca127
```

Ciphertext:

```
7d0a2e6b7f7c65a236542630294e063b7ab9b555a5d5149aa21e4ae1e4fbce87
ecc8e08a8b5e350abe622b2ffa617b202cfad72032a3037e76ffdcdc4376ee05
3a190d7e46ca1de04144850381b9cb29f051915386b8a710b8ac4d027b8b050f
7cba5854e028d564e453b8a968824173fc16488b8970cac828f11ae53cabd201
12f87107df24ee6183d2274fe4c8b1485534ef2c5fbc1ec24bfc3663efaa08bc
047d29d25043532db8391a8a3d776bf4372a6955827ccb0cdd4af403a7ce4c63
d595c75a43e045f0cce1f29c8b93bd65afc5974922f214a40b7c402cdb91ae73
c0b63615cdad0480680f16515a7ace9d39236464328a37743ffc28f4ddb324f4
d0f5bbdc270c65b1749a6efff1fbaa09536175ccd29fb9e6057b307320d31683
8a9c71f70b5b5907a66f7ea49aadc409
```

Author's Address

Scott Arciszewski
Paragon Initiative Enterprises
United States

Email: security@paragonie.com