Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 26, 2019
N. Madden
ForgeRock
November 22, 2018

Synthetic IV (SIV) for non-AES ciphers and MACs
draft-madden-generalised-siv-00

Abstract

This document specifies how the Synthetic Initialization Vector (SIV) block cipher mode of operation can be adapted to non-AES ciphers and message authentication codes (MACs), with block sizes and MAC tag sizes other than 128 bits. Concrete instantiations are defined using the XChaCha20 nonce-extended stream cipher combined with HMAC-SHA256.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Synthetic Initialization Vector (SIV) block cipher mode of operation [RFC5297] provides either deterministic authenticated encryption (DAE) or nonce-reuse misuse-resistant authenticated encryption (MRAE) [DAE]. It was originally specified for the combination of AES-CMAC for authenticity and AES-CTR for confidentiality. The 128-bit AES-CMAC tag is used as the 128-bit (synthetic) IV for AES-CTR. This document show how to apply SIV mode to ciphers and MACs with IV and tag lengths other than 128 bits, including where the IV and tag length may differ.

1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Motivation

Common IV-based authenticated encryption modes of operation require a unique IV (or nonce) to be provided on each call to the encryption function with the same key. If an IV is reused, either by accident or through malicious action, then some combination of the confidentiality and/or authenticity properties is usually lost. For

the popular Galois Counter Mode (GCM) [SP800-38D], NIST states that "a breach of the requirement [...] for the uniqueness of the initialization strings may compromise the security assurance almost entirely" (section 3 and Appendix A). The SIV mode of operation [RFC5297][SIV], when used with a unique nonce as part of the associated data, provides a measure of protection against nonce reuse. If a nonce is reused with SIV mode then there is no loss of authenticity, and a minimal loss of confidentiality: an attacker is able to determine only whether the exact same message has been encrypted with the same associated data and nonce using the same key.

While the SIV mode is specified as a generic composition of an IV-based encryption scheme and a pseudorandom function (PRF), most uses of the mode have concentrated on the one concrete instantiation of the mode given at the time: AES-SIV. AES-SIV is built entirely from the AES block cipher, using AES-CMAC [RFC4493] as the PRF and AES in CTR mode for confidentiality. This combination is attractive as it requires only an AES encryption operation to implement all aspects of the mode. It also has the convenient property that AES-CMAC produces a 128-bit tag, and AES-CTR requires a 128-bit IV, which allows the tag to be used directly as the (synthetic) IV.

While AES-SIV has many attractive properties, there are good reasons for extending SIV to other ciphers and PRFs. As stated in the rationale for adopting ChaCha20 and Poly1305 for IETF protocols [RFC8439], overreliance on a single cipher design, however good, may cause difficulties if a weakness is ever discovered in AES. Secondly, AES can be difficult to implement efficiently in software while avoiding timing side-channels. Finally, there is the simple fact that non-AES ciphers and PRFs exist and will continue to be used, and SIV mode is of independent interest to users of those alternative primitives.

2. The Generic SIV Construction

The generic SIV construction is defined in terms two primitive functions:

1. A PRF, F*, that takes a vector of strings as input.

2. A length-preserving IV-based encryption scheme, E.

The S2V function can be used to build F* from a PRF, F, that takes a single string input. The generalised version of this function is described in the next section.

The following constants MUST be defined for any concrete instantiation of E and F*:

IV_LEN - the length of IV/nonce expected by E, in bits.

TAG_LEN - the length of the output tag produced by F*, in bits.

PRF_KEY_LEN - the length of key required for F*, in bits.

CIPHER_KEY_LEN - the length of key required for E, in bits.

For any choice of E and F*, TAG_LEN MUST be greater than or equal to IV_LEN.

We denote the encryption operation of E as E.encrypt(key, iv, plaintext), and the decryption operation as E.decrypt(key, iv, ciphertext).

2.1. Encryption

Encryption takes as input a key, K, a plaintext P, and zero or more associated data headers to be authenticated but not encrypted. The key K MUST be as long as the sum of the key length of F* and the key length of E. For example, if F* requires a 256-bit key and E requires a 128-bit key, then K must be 384 bits long.

Encryption proceeds as follows. Firstly, keys K1 and K2 are derived from K by taking the leftmost PRF_KEY_LEN bits of K as K1, and the rightmost CIPHER_KEY_LEN bits of K as K2. Secondly, the PRF F* is applied to K1, the plaintext P, and all of the n associated data strings AD1, ..., ADn. This results in an authentication tag, T. The SIV is defined as the leftmost IV_LEN bits of T. If TAG_LEN = IV_LEN, then T is used in its entirety. The plaintext P is then encrypted using E, with K2 as the key and SIV as the IV, producing ciphertext C. The concatenation of T with C is returned as the output of the function.

In pseudocode, generalised SIV encryption is as follows:

```
SIV-ENCRYPT[F*,E](K, P, AD1, ..., ADn) {
    K1 = leftmost(K, PRF_KEY_LEN)
    K2 = rightmost(K, CIPHER_KEY_LEN)
    T = F*(K1, AD1, ..., ADn, P)
    SIV = leftmost(T, IV_LEN)
    C = E.encrypt(K2, SIV, P)

    return T || C
}
```

2.2. Decryption

Decryption takes as input a key, K, an authenticated ciphertext Z, and zero or more associated data blocks to be authenticated but not decrypted.

Keys K1 and K2 are derived as for encryption. The leftmost TAG_LEN bits of Z are taken as the tag T, while the remaining bits are the ciphertext C. As for encryption, the SIV is taken as the leftmost IV_LEN bits of T. The ciphertext C is then decrypted using E with the key K2 and SIV as the IV, producing plaintext P. The expected authentication tag T' is then computed using F* over the key K1, the associated data AD1, ..., ADn, and the plaintext P. If T' exactly matches T then the plaintext P is returned. If T' does not match T then the implementation MUST NOT return P and MUST destroy P and T' and return a failure. Authentication tag comparisons SHOULD be performed in constant time to avoid leaking the true value of T' through timing differences.

```
SIV-DECRYPT[F*,E](K, Z, AD1, ..., ADn) {
    T = leftmost(Z, TAG_LEN)
    C = rightmost(Z, len(Z) - TAG_LEN)
    K1 = leftmost(K, PRF_KEY_LEN)
    K2 = rightmost(K, CIPHER_KEY_LEN)
    SIV = leftmost(T, IV_LEN)

    P = E.decrypt(K2, SIV, C)
    T' = F*(K1, AD1, ..., ADn, P)

    if T = T' then
        return P
    else
        destroy P and T'
        return FAIL
    fi
}
```

2.3. Generalised S2V

SIV requires a PRF that takes a vector of strings as input, while most PRFs in current use are designed to only take a single string. In principle, any unambiguous encoding can be used to convert a vector of inputs into a single string, but SIV defines a particularly efficient encoding provided by the function S2V (for "string to vector") that converts a single-string PRF to a vector input PRF. S2V is defined using bitwise exclusive OR (XOR) and a doubling operation in the finite field GF(2^n) where n is the bit length of the output of the PRF. For AES-SIV, which uses AES-CMAC as the PRF,

this is GF(2^128). In this section we show how to define S2V for PRFs with different tag lengths.

Points in the finite field GF(2^n) are represented as n-bit strings a_(n-1) ... a_1 a_0, which can also be seen as binary coefficients for a polynomial f(x) = a_(n-1) * x^(n-1) + ... + a_1 * x + a_0. Multiplication is then defined as the product of two polynomials, with the remainder taken after division by a fixed polynomial. In S2V, the fixed polynomial is the lexicographically first minimum-weight primitive polynomial [SIV] (section 2). For GF(2^128), such a primitive polynomial is:

f(x) = x^128 + x^7 + x^2 + x + 1

Primitive polynomials for other fields can be found in published tables, such as [HPL-98-135]. The following polynomials are indicated for common PRF output sizes:

Field	Primitive Polynomial
GF(2^64)	f(x) = x^64 + x^4 + x^3 + x + 1
GF(2^96)	f(x) = x^96 + x^10 + x^9 + x^6 + 1
GF(2^128)	f(x) = x^128 + x^7 + x^2 + x + 1
GF(2^160)	f(x) = x^160 + x^5 + x^3 + x^2 + 1
GF(2^192)	f(x) = x^192 + x^7 + x^2 + x + 1
GF(2^224)	f(x) = x^224 + x^9 + x^8 + x^3 + 1
GF(2^256)	f(x) = x^256 + x^10 + x^5 + x^2 + 1
GF(2^384)	f(x) = x^384 + x^12 + x^3 + x^2 + 1
GF(2^512)	f(x) = x^512 + x^8 + x^5 + x^2 + 1

Doubling for S2V is defined as multiplication with the binary value 0^(n-2)10 (i.e., the number 2 represented as an n-bit binary string). The doubling operation can be efficiently implemented as a left-shift operation followed by a conditional XOR with an n-bit constant derived from the binary coefficients of the primitive polynomial. The condition being whether the most significant bit of the value being shifted off is 1. For GF(2^128), the constant is 0^(120)10000111, with one bits corresponding to x^7, x^2, x and 1 respectively. The following table lists the constants for common PRF output sizes in binary and hexadecimal form. Leading zero octets are omitted from the hexadecimal format.

Field	Doubling Constant - Binary	Doubling Constant - Hex
GF(2^64)	0^(59)11011	0x1b
GF(2^92)	0^(150)1100100001	0x0321
GF(2^128)	0^(120)10000111	0x87
GF(2^160)	0^(154)101101	0x2d
GF(2^192)	0^(184)10000111	0x87
GF(2^224)	0^(214)1100001001	0x0309
GF(2^256)	0^(245)10000100101	0x0425
GF(2^384)	0^(371)1000000001101	0x100d
GF(2^256)	0^(503)100100101	0x0125

It is recommended that the conditional XOR be performed in constant time. A constant time bit-sliced implementation is provided in Appendix A.

The S2V algorithm parameterised over a particular PRF, F, written S2V[F] is as follows, where TAG_LEN is the output size of the PRF in bits, dbl(x) is the appropriate doubling operation for TAG_LEN, and xorend is defined as in [RFC5297]. The constant <zero> is the TAG_LEN sequence of all zero bits, and <one> is TAG_LEN-1 zero bits followed by a single 1 bit. The function pad(X) pads the input to TAG_LEN bits by appending a single 1 bit followed by as many 0 bits as necessary.

```
S2V[F](K, S1, ..., Sn) {
    if n = 0 then
        return F(K, <one>)
    fi
    D = F(K, <zero>)
    for i = 1 to n-1 do
        D = dbl(D) xor F(K, Si)
    done
    if len(Sn) >= TAG_LEN then
        T = Sn xorend D
    else
        T = dbl(D) xor pad(Sn)
    fi
    return F(K, T)
}
```

2.4. AES-SIV

This section is non-normative.

The original AES-SIV mode of [RFC5297] can be seen as an instantiation of the generic SIV construction in this document, with the following parameters:

F* = S2V[AES-CMAC]

E = AES-CTR where the 31st and 63rd bits of the IV are zeroed prior to use as described in [RFC5297] section 2.5.

PRF_KEY_LEN = 128, 192 or 256 bits

CIPHER_KEY_LEN = 128, 192 or 256 bits (to match PRF_KEY_LEN).

IV_LEN = TAG_LEN = 128 bits.

3. XChaCha20-HMAC-SHA256-SIV

ChaCha20 is a stream cipher that has been adopted for use in IETF protocols by [RFC8439]. It has several attractive properties, most notably that it can be implemented efficiently in software and is relatively easy to make resistant to cache-timing side-channel attacks. As originally specified, ChaCha20 takes a 256-bit key and a 64-bit nonce, which was extended to 96-bits when adopted by the IETF. A 96-bit nonce is too small to be safely generated randomly (or pseudorandomly as in SIV) without artificially limiting the number of messages that can be encrypted with a single key. To address this problem, an extended nonce variant known as XChaCha20 [I-D.arciszewski-xchacha] has been proposed, which increase the nonce to 192-bits. This makes it an excellent choice for an SIV instantiation, providing a MRAE cipher mode alternative to AES.

In principle, any PRF that produces at least a 192-bit output could be used with XChaCha20. For concreteness, we specify the use of HMAC [RFC2104] with the SHA-256 secure hash function [RFC6234] as HMAC-SHA256 is widely implemented. The S2V function of section 2.3 is used to allow HMAC-SHA256 to take a vector of strings as input, with the primitive polynomial for GF(2^256) used for point doubling and the leftmost 192 bits of the S2V[HMAC-SHA256] tag used as the synthetic IV for XChaCha20.

The encryption and decryption procedures are as described in sections 2.1 and 2.2 above, with the following constant values:

PRF_KEY_LEN = 256 bits.

CIPHER_KEY_LEN = 256 bits.

IV_LEN = 192 bits.

TAG_LEN = 256 bits.

Test vectors for XChaCha20-HMAC-SHA256-SIV are provided in Appendix A.

4. IANA considerations

This section registers AEAD algorithms as per the registry established in [RFC5116]. As specified in [RFC5297] section 6, the interface of RFC 5116 only allows a single associated data (AD) component. When SIV is accessed via this interface, multiple AD components must be marshalled into a single string prior to calling the SIV procedures.

4.1. AEAD_XCHACHA20_SIV_HMAC_SHA256

The AEAD_XCHACHA20_SIV_HMAC_SHA256 algorithm is an instantiation of the generalised SIV mode described in Sections 2.1 and 2.2 with the XChaCha20 extended-nonce stream cipher [I-D.arciszewski-xchacha] and HMAC-SHA256 as the PRF, as described in Section 3. XChaCha20 uses a 32-bit block counter and a 512-bit block size, therefore the maximum size of plaintext that can be encrypted in a single invocation is 2^38 octets, around 256 GB. The ciphertext length is equal to the length of the plaintext plus 32 octets for the HMAC-SHA256 authentication tag (of which the leftmost 24 octets comprise the SIV).

The input and output lengths for AEAD_XCHACHA20_SIV_HMAC_SHA256 as defined by [RFC5116] are:

K_LEN is 64 octets.
P_MAX is 2^38 octets.
A_MAX is unlimited.
N_MIN is 1 octet.
N_MAX is unlimited.
C_MAX is 2^38 + 32 octets.

5. Security Considerations

The security considerations of [RFC5297] apply here.

The security proofs for SIV [DAE] require that F* (and F if constructing F* using S2V) behaves as a pseudorandom function (PRF). E must be a length-preserving semantically-secure encryption scheme.

It is RECOMMENDED that SIV mode is always used with a unique random component included as the last element of the header (associated data) to ensure semantic security. While SIV mode loses a minimal

amount of security if this component is omitted (or accidentally reused), an attacker in this case is able to determine if the same plaintext has been encrypted under the same key and with the same associated data. Depending on the application this may still be a significant loss of confidentiality. For example, a service that produces yes/no answers to questions would lose all confidentiality of its responses in this case. The misuse resistance of SIV should be considered a failsafe and not as a way to do without a nonce.

The requirement that E be length-preserving means that the ciphertext produced by SIV mode will be equal in length to the input plaintext, plus the authentication tag (which is of fixed size for any concrete instantiation of this mode). If the length of the plaintext on its own may reveal information then care should be taken to obscure this prior to encryption -- by padding to a known maximum length, for example. In the case of the yes/no answer service the English words "yes" and "no" can be distinguished purely by length, to give a simple example.

In [tightness], Chatterjee, Menezes and Sarkar show an attack on SIV within the multi-user setting. It is RECOMMENDED that concrete instantiations intended for such use define a MAC_KEY_LENGTH of at least 256 bits or describe other countermeasures.

The number of components passed to any invocation of S2V (including the plaintext) must not exceed TAG_LEN - 1. For example, a 128-bit PRF such as AES-CMAC should allow no more than 127 components. For XChaCha20-HMAC-SHA256-SIV no more than 255 components should be allowed.

6. References

6.1. Normative References

[I-D.arciszewski-xchacha]
Arciszewski, S., "XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305", draft-arciszewski-xchacha-02 (work in progress), October 2018.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

6.2. Informative References

[DAE] Rogaway, P. and T. Shrimpton, "Deterministic Authenticated-Encryption. A Provable-Security Treatment of the Key-Wrap Problem.", IACR ePrint 2006/221, August 2007.

[HPL-98-135] Seroussi, G., "Table of Low-Weight Binary Irreducible Polynomials", HPL HPL-98-135, August 1998.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

[SIV] Rogaway, P. and T. Shrimpton, "The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and Misuse-Resistant Nonce-Based Authenticated-Encryption.", August 2007.

[SP800-38D] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.", NIST Special Publication 800-38D, November 2007.

[tightness] Chatterjee, S., Menezes, A., and P. Sarkar, "Another Look at Tightness", Proceedings of SAC 2011, Lecture Notes in Computer Science 7118, August 2011.

Appendix A. Test Vectors

A.1. Nonce-Based Authenticated Encryption Example

```
Input
-----
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  ................
032  a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af  ................
048  b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf  ................

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.

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

IV:
000  40 41 42 43 44 45 46 47                          @ABCDEFG

S2V[HMAC-SHA256]
----------------
HMAC-SHA256(<zero>):
    318dcd14 73a3c69c 643eb853 e66eb357
    c5bcb67b cd96ea83 4af2a3c6 f462136f

dbl():
    631b9a28 e7478d38 c87d70a7 ccdd66af
    8b796cf7 9b2dd506 95e5478d e8c426de

HMAC-SHA256(AD1):
    8b80c006 47844e6b 54617036 b1c09145
    0ab8ad63 1e7ca653 326a8d4f e135dafb

xor:
    e89b5a2e a0c3c353 9c1c0091 7d1df7ea
    81c1c194 85517355 a78fcac2 09f1fc25

dbl():
    d136b45d 418786a7 38380122 fa3befd5
```

```
        03838329 0aa2e6ab 4f1f9584 13e3fc6f

    HMAC-SHA256(Nonce):
        7c07875c 75e0021c 6f58cbd2 052675e3
        2690107a 1f618e40 34b79efc d23d3a57

    xor:
        ad313301 346784bb 5760caf0 ff1d9a36
        25139353 15c368eb 7ba80b78 c1dec638

    xorend:
        4c616469 65732061 6e642047 656e746c
        656d656e 206f6620 74686520 636c6173
        73206f66 20273939 3a204966 20492063
        6f756c64 206f6666 65722079 6f75206f
        6e6c7920 6f6e6520 74697020 666f7220
        7468c811 55744012 f6de7b40 b985916e
        f9444076 fd7362ac 1d871f88 691de1b7
        b216

    HMAC-SHA256(final):
        28fdb5d4 d89e4860 11774606 5456a5df
        924e8f4b 0f42bc77 a7415bd0 e0430628

    XChaCha20
    ---------
    SIV:
        28fdb5d4 d89e4860 11774606 5456a5df
        924e8f4b 0f42bc77

    Block Counter:
        00000000

    XChaCha20 Subkey:
        70c5831f 36e439c1 b90e375e 2b98c3da
        ef42de2e c120e1d1 2706af76 45381de1

    XChaCha20 Nonce:
        00000000 924e8f4b 0f42bc77

    Output
    ------
    T || C:
    000  28 fd b5 d4 d8 9e 48 60 11 77 46 06 54 56 a5 df  (.....H`.wF.TV..
    016  92 4e 8f 4b 0f 42 bc 77 a7 41 5b d0 e0 43 06 28  .N.K.B.w.A[..C.(
    032  26 53 ea bf c6 ae cc 14 d0 46 aa 7e 3c 0b a2 8e  &S.......F.~<...
    048  fd 68 f3 d5 91 fc ac 6d b1 2e a2 3c f4 28 69 01  .h.....m...<.(i.
    064  3b 2b e4 83 ce 08 8a f8 2d e4 29 3a 07 e2 40 07  ;+......-.):..@.
```

```
  080  f3 7b d1 e3 78 81 a0 4b 11 5b 11 09 94 78 ae 34  .{..x..K.[...x.4
  096  75 05 43 26 8e 57 0d 1f 27 f4 da fc 5a d8 71 97  u.C&.W..'...Z.q.
  112  7f 08 b3 0b af df b5 3b 19 ef 34 2c d9 5c e7 91  .......;..4,.\..
  128  5c b4 f6 79 db 64 0d 8e c4 8a 06 b6 f3 ef 50 8c  \..y.d........P.
  144  53 30                                            S0
```

Author's Address

Neil Madden
ForgeRock
Broad Quay House
Prince Street
Bristol BS1 4DJ
United Kingdom

Email: neil.madden@forgerock.com