         Synthetic Initialization Vector (SIV) Authenticated Encryption
                 Using the Advanced Encryption Standard (AES)

Status of This Memo

Abstract

   This memo describes SIV (Synthetic Initialization Vector), a block
   cipher mode of operation. SIV takes a key, a plaintext, and multiple
   variable-length octet strings that will be authenticated but not
   encrypted. It produces a ciphertext having the same length as the
   plaintext and a synthetic initialization vector. Depending on how it
   is used, SIV achieves either the goal of deterministic authenticated
   encryption or the goal of nonce-based, misuse-resistant authenticated
   encryption.

Table of Contents

1. Introduction

1.1. Background

   Various attacks have been described (e.g., [BADESP]) when data is
   merely privacy protected and not additionally authenticated or
   integrity protected. Therefore, combined modes of encryption and
   authentication have been developed ([RFC5116], [RFC3610], [GCM],
   [JUTLA], [OCB]). These provide conventional authenticated encryption
   when used with a nonce ("a number used once") and typically accept
   additional inputs that are authenticated but not encrypted,
   hereinafter referred to as "associated data" or AD.

   A deterministic, nonce-less, form of authenticated encryption has
   been used to protect the transportation of cryptographic keys (e.g.,
   [X9F1], [RFC3217], [RFC3394]). This is generally referred to as "Key
   Wrapping".

   This memo describes a new block cipher mode, SIV, that provides both
   nonce-based authenticated encryption as well as deterministic, nonce-
   less key wrapping. It contains a Pseudo-Random Function (PRF)
   construction called S2V and an encryption/decryption construction,
   called CTR. SIV was specified by Phillip Rogaway and Thomas
   Shrimpton in [DAE]. The underlying block cipher used herein for both
   S2V and CTR is AES with key lengths of 128 bits, 192 bits, or 256
   bits. S2V uses AES in Cipher-based Message Authentication Code
   ([CMAC]) mode, CTR uses AES in counter ([MODES]) mode.

   Associated data is data input to an authenticated-encryption mode
   that will be authenticated but not encrypted. [RFC5116] says that
   associated data can include "addresses, ports, sequence numbers,
   protocol version numbers, and other fields that indicate how the
   plaintext or ciphertext should be handled, forwarded, or processed".
   These are multiple, distinct inputs and may not be contiguous. Other
   authenticated-encryption cipher modes allow only a single associated
   data input. Such a limitation may require implementation of a
   scatter/gather form of data marshalling to combine the multiple
   components of the associated data into a single input or may require
   a pre-processing step where the associated data inputs are
   concatenated together. SIV accepts multiple variable-length octet
   strings (hereinafter referred to as a "vector of strings") as
   associated data inputs. This obviates the need for data marshalling
   or pre-processing of associated data to package it into a single
   input.

   By allowing associated data to consist of a vector of strings SIV
   also obviates the requirement to encode the length of component
   fields of the associated data when those fields have variable length.

1.2. Definitions

   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].

1.3. Motivation

1.3.1. Key Wrapping

   A key distribution protocol must protect keys it is distributing.
   This has not always been done correctly. For example, RADIUS
   [RFC2865] uses Microsoft Point-to-Point Encryption (MPPE) [RFC2548]
   to encrypt a key prior to transmission from server to client. It
   provides no integrity checking of the encrypted key. [RADKEY]
   specifies the use of [RFC3394] to wrap a key in a RADIUS request but
   because of the inability to pass associated data, a Hashed Message
   Authentication Code (HMAC) [RFC2104] is necessary to provide
   authentication of the entire request.

   SIV can be used as a drop-in replacement for any specification that
   uses [RFC3394] or [RFC3217], including the aforementioned use. It is
   a more general purpose solution as it allows for associated data to
   be specified.

1.3.2. Resistance to Nonce Misuse/Reuse

   The nonce-based authenticated encryption schemes described above are
   susceptible to reuse and/or misuse of the nonce. Depending on the
   specific scheme there are subtle and critical requirements placed on
   the nonce (see [SP800-38D]). [GCM] states that it provides
   "excellent security" if its nonce is guaranteed to be distinct but
   provides "no security" otherwise. Confidentiality guarantees are
   voided if a counter in [RFC3610] is reused. In many cases,
   guaranteeing no reuse of a nonce/counter/IV is not a problem, but in
   others it will be.

   For example, many applications obtain access to cryptographic
   functions via an application program interface to a cryptographic
   library. These libraries are typically not stateful and any nonce,
   initialization vector, or counter required by the cipher mode is
   passed to the cryptographic library by the application. Putting the
   construction of a security-critical datum outside the control of the
   encryption engine places an onerous burden on the application writer
   who may not provide the necessary cryptographic hygiene. Perhaps his
   random number generator is not very good or maybe an application
   fault causes a counter to be reset. The fragility of the cipher mode
   may result in its inadvertent misuse. Also, if one's environment is

(knowingly or unknowingly) a virtual machine, it may be possible to
roll back a virtual state machine and cause nonce reuse thereby
gutting the security of the authenticated encryption scheme (see
[VIRT]).

If the nonce is random, a requirement that it never repeat will limit
the amount of data that can be safely protected with a single key to
one block. More sensibly, a random nonce is required to "almost
always" be non-repeating, but that will drastically limit the amount
of data that can be safely protected.

SIV provides a level of resistance to nonce reuse and misuse. If the
nonce is never reused, then the usual notion of nonce-based security
of an authenticated encryption mode is achieved. If, however, the
nonce is reused, authenticity is retained and confidentiality is only
compromised to the extent that an attacker can determine that the
same plaintext (and same associated data) was protected with the same
nonce and key. See Security Considerations (Section 7).

1.3.3. Key Derivation

A PRF is frequently used as a key derivation function (e.g., [WLAN])
by passing it a key and a single string. Typically, this single
string is the concatenation of a series of smaller strings -- for
example, a label and some context to bind into the derived string.

These are usually multiple strings but are mapped to a single string
because of the way PRFs are typically defined -- two inputs: a key
and data. Such a crude mapping is inefficient because additional
data must be included -- the length of variable-length inputs must be
encoded separately -- and, depending on the PRF, memory allocation
and copying may be needed. Also, if only one or two of the inputs
changed when deriving a new key, it may still be necessary to process
all of the other constants that preceded it every time the PRF is
invoked.

When a PRF is used in this manner its input is a vector of strings
and not a single string and the PRF should handle the data as such.
The S2V ("string to vector") PRF construction accepts a vector of
inputs and provides a more natural mapping of input that does not
require additional lengths encodings and obviates the memory and
processing overhead to marshal inputs and their encoded lengths into
a single string. Constant inputs to the PRF need only be computed
once.

1.3.4. Robustness versus Performance

   SIV cannot perform at the same high throughput rates that other
   authenticated encryption schemes can (e.g., [GCM] or [OCB]) due to
   the requirement for two passes of the data, but for situations where
   performance is not a limiting factor -- e.g., control plane
   applications -- it can provide a robust alternative, especially when
   considering its resistance to nonce reuse.

1.3.5. Conservation of Cryptographic Primitives

   The cipher mode described herein can do authenticated encryption, key
   wrapping, key derivation, and serve as a generic message
   authentication algorithm. It is therefore possible to implement all
   these functions with a single tool, instead of one tool for each
   function. This is extremely attractive for devices that are memory
   and/or processor constrained and that cannot afford to implement
   multiple cryptographic primitives to accomplish these functions.

2. Specification of SIV

2.1. Notation

   SIV and S2V use the following notation:

   len(A)
      returns the number of bits in A.

   pad(X)
      indicates padding of string X, len(X) < 128, out to 128 bits by
      the concatenation of a single bit of 1 followed by as many 0 bits
      as are necessary.

   leftmost(A,n)
      the n most significant bits of A.

   rightmost(A,n)
      the n least significant bits of A.

   A || B
      means concatenation of string A with string B.

   A xor B
      is the exclusive OR operation on two equal length strings, A and
      B.

A xorend B
   where len(A) >= len(B), means xoring a string B onto the end of
   string A -- i.e., leftmost(A, len(A)-len(B)) || (rightmost(A,
   len(B)) xor B).

A bitand B
   is the logical AND operation on two equal length strings, A and B.

dbl(S)
   is the multiplication of S and 0...010 in the finite field
   represented using the primitive polynomial
   x^128 + x^7 + x^2 + x + 1. See Doubling ([Section 2.3](#)).

a^b
   indicates a string that is "b" bits, each having the value "a".

<zero>
   indicates a string that is 128 zero bits.

<one>
   indicates a string that is 127 zero bits concatenated with a
   single one bit, that is 0^127 || 1^1.

A/B
   indicates the greatest integer less than or equal to the real-
   valued quotient of A and B.

E(K,X)
   indicates AES encryption of string X using key K.

[2.2](#). Overview

SIV-AES uses AES in CMAC mode (S2V) and in counter mode (CTR). SIV-
AES takes either a 256-, 384-, or 512-bit key (which is broken up
into two equal-sized keys, one for S2V and the other for CTR), a
variable length plaintext, and multiple variable-length strings
representing associated data. Its output is a ciphertext that
comprises a synthetic initialization vector concatenated with the
encrypted plaintext.

[2.3](#). Doubling

The doubling operation on a 128-bit input string is performed using a
left-shift of the input followed by a conditional xor operation on
the result with the constant:

                   00000000 00000000 00000000 00000087

The condition under which the xor operation is performed is when the
bit being shifted off is one.

Note that this is the same operation used to generate sub-keys for
CMAC-AES.

[2.4](#). S2V

The S2V operation consists of the doubling and xoring of the outputs
of a pseudo-random function, CMAC, operating over individual strings
in the input vector: S1, S2, ... , Sn. It is bootstrapped by
performing CMAC on a 128-bit string of zeros. If the length of the
final string in the vector is greater than or equal to 128 bits, the
output of the double/xor chain is xored onto the end of the final
input string. That result is input to a final CMAC operation to
produce the output V. If the length of the final string is less than
128 bits, the output of the double/xor chain is doubled once more and
it is xored with the final string padded using the padding function
pad(X). That result is input to a final CMAC operation to produce
the output V.

S2V with key K on a vector of n inputs S1, S2, ..., Sn-1, Sn, and
len(Sn) >= 128:

```
                    +----+        +----+        +------+        +----+
                    | S1 |        | S2 | . . . | Sn-1 |        | Sn |
                    +----+        +----+        +------+        +----+
      <zero>    K      |             |             |             |
        |      |      |             |             |             V
        V      |      V             V             V   /----> xorend
    +-----+    |  +-----+        +-----+        +-----+ |       |
    | AES-|<----->| AES-|   K-->| AES-|   K--->| AES-| |       |
    | CMAC|       | CMAC|       | CMAC|        | CMAC| |       |
    +-----+       +-----+        +-----+        +-----+ |       V
        |             |             |             |    |    +-----+
        |             |             |             |    | K-->| AES-|
        |             |             |             |    |    | CMAC|
        |             |             |             |    |    +-----+
        \-> dbl -> xor -> dbl -> xor -> dbl -> xor---/       |
                                                             V
                                                          +---+
                                                          | V |
                                                          +---+
```

                                Figure 2

S2V with key K on a vector of n inputs S1, S2, ..., Sn-1, Sn, and
len(Sn) < 128:

```
                  +----+          +----+         +------+        +----------+
                  | S1 |          | S2 | . . . | Sn-1 |        | pad(Sn) |
                  +----+          +----+         +------+        +----------+
  <zero>   K       |               |               |                 |
    |      |       |               |               |                 V
    V      |       V               V               V      /------> xor
 +-----+   |   +-----+         +-----+         +-----+   |          |
 | AES-|<--->| AES-|   K-->| AES-|    K-->| AES-|   |          |
 | CMAC|      | CMAC|      | CMAC|        | CMAC|   |          |
 +-----+      +-----+         +-----+         +-----+   |          V
    |            |               |               |      |      +-----+
    |            |               |               |      |  K-->| AES-|
    |            |               |               |      |      | CMAC|
    |            |               |               |      |      +-----+
   \-> dbl -> xor -> dbl -> xor -> dbl -> xor-> dbl       |
                                                          V
                                                      +---+
                                                      | V |
                                                      +---+
```
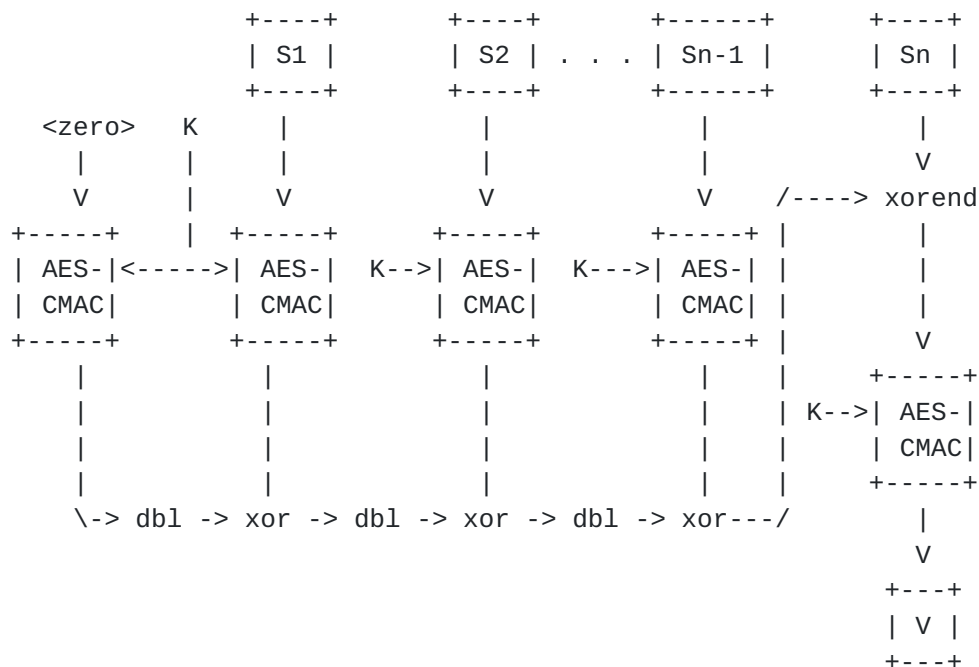
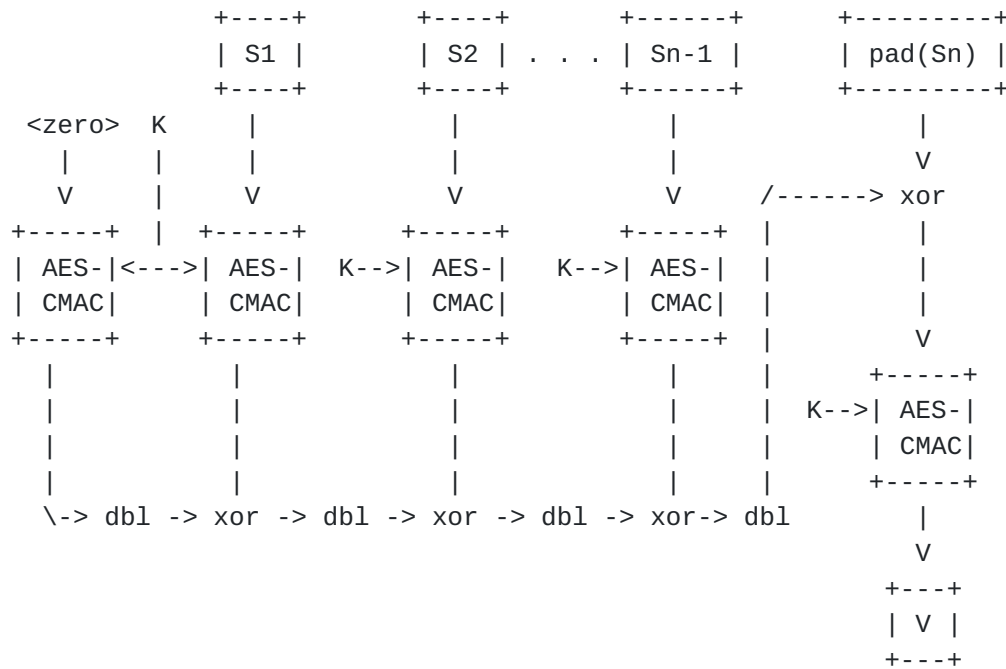   Figure 3

Algorithmically S2V can be described as:

```
   S2V(K, S1, ..., Sn) {
     if n = 0 then
       return V = AES-CMAC(K, <one>)
     fi
     D = AES-CMAC(K, <zero>)
     for i = 1 to n-1 do
       D = dbl(D) xor AES-CMAC(K, Si)
     done
     if len(Sn) >= 128 then
       T = Sn xorend D
     else
       T = dbl(D) xor pad(Sn)
     fi
     return V = AES-CMAC(K, T)
   }
```

[2.5](#). CTR

CTR is a counter mode of AES. It takes as input a plaintext P of
arbitrary length, a key K of length 128, 192, or 256 bits, and a
counter X that is 128 bits in length, and outputs Z, which represents
a concatenation of a synthetic initialization vector V and the
ciphertext C, which is the same length as the plaintext.

The ciphertext is produced by xoring the plaintext with the first
len(P) bits of the following string:

$$E(K, X) \,||\, E(K, X+1) \,||\, E(K, X+2) \,||\, ...$$

Before beginning counter mode, the 31st and 63rd bits (where the
rightmost bit is the 0th bit) of the counter are cleared. This
enables implementations that support native 32-bit (64-bit) addition
to increment the counter modulo 2^{32} (2^{64}) in a manner that cannot
be distinguished from 128-bit increments, as long as the number of
increment operations is limited by an upper bound that safely avoids
carry to occur out of the respective pre-cleared bit. More formally,
for 32-bit addition, the counter is incremented as:

    SALT=leftmost(X,96)

    n=rightmost(X,32)

    X+i = SALT || (n + i mod 2^{32}).

For 64-bit addition, the counter is incremented as:

    SALT=leftmost(X,64)

    n=rightmost(X,64)

    X+i = SALT || (n + i mod 2^{64}).

Performing 32-bit or 64-bit addition on the counter will limit the
amount of plaintext that can be safely protected by SIV-AES to 2^{39} -
128 bits or 2^{71} - 128 bits, respectively.

[2.6](#). SIV Encrypt

SIV-encrypt takes as input a key K of length 256, 384, or 512 bits,
plaintext of arbitrary length, and a vector of associated data AD[ ]
where the number of components in the vector is not greater than 126
(see [Section 7](#)). It produces output, Z, which is the concatenation
of a 128-bit synthetic initialization vector and ciphertext whose
length is equal to the length of the plaintext.

The key is split into equal halves, K1 = leftmost(K, len(K)/2) and K2
= rightmost(K, len(K)/2). K1 is used for S2V and K2 is used for CTR.

In the encryption mode, the associated data and plaintext represent
the vector of inputs to S2V, with the plaintext being the last string
in the vector. The output of S2V is a synthetic IV that represents
the initial counter to CTR.

The encryption construction of SIV is as follows:

```
 +------+ +------+   +------+                    +---+
 | AD 1 | | AD 2 |...| AD n |                    | P |
 +------+ +------+   +------+                    +---+
    |        |          |                          |
    |        |    ...    |   -----------------------|
     \       |         /  /                         |
      \      |        /  / +-----------+            |
       \     |       /  /  | K = K1||K2 |           |
        \    |      /  /   +-----------+     V
         \   |     /  /        |      |    +-----+
          \  |    /  /   K1    |      | K2 |     |
           \ |   /  /  ------/      \------>| CTR |
            \ | /  /  /              ------->|     |
             | | | |  |                 |    +-----+
             V V V V   V                 |       |
           +------------+          +--------+    V
           |    S2V     |------->|   V    |   +----+
           +------------+          +--------+   | C  |
                                       |        +----+
                                       |          |
                                   -----\         |
                                        \         |
                                         \        |
                                          V   V
                                       +-----+
                                       |  Z  |
                                       +-----+
```

where the plaintext is P, the associated data is AD1 through ADn, V
is the synthetic IV, the ciphertext is C, and Z is the output.

Figure 8

   Algorithmically, SIV Encrypt can be described as:

```
   SIV-ENCRYPT(K, P, AD1, ..., ADn) {
     K1 = leftmost(K, len(K)/2)
     K2 = rightmost(K, len(K)/2)
     V = S2V(K1, AD1, ..., ADn, P)
     Q = V bitand (1^64 || 0^1 || 1^31 || 0^1 || 1^31)
     m = (len(P) + 127)/128

     for i = 0 to m-1 do
       Xi = AES(K2, Q+i)
     done
     X = leftmost(X0 || ... || Xm-1, len(P))
     C = P xor X

     return V || C
   }
```

   where the key length used by AES in CTR and S2V is len(K)/2 and will
   each be either 128 bits, 192 bits, or 256 bits.

   The 31st and 63rd bit (where the rightmost bit is the 0th) of the
   counter are zeroed out just prior to being used by CTR for
   optimization purposes, see Section 5.

2.7. SIV Decrypt

   SIV-decrypt takes as input a key K of length 256, 384, or 512 bits,
   Z, which represents a synthetic initialization vector V concatenated
   with a ciphertext C, and a vector of associated data AD[ ] where the
   number of components in the vector is not greater than 126 (see
   Section 7). It produces either the original plaintext or the special
   symbol FAIL.

   The key is split as specified in Section 2.6

   The synthetic initialization vector acts as the initial counter to
   CTR to decrypt the ciphertext. The associated data and the output of
   CTR represent a vector of strings that is passed to S2V, with the CTR
   output being the last string in the vector. The output of S2V is
   then compared against the synthetic IV that accompanied the original
   ciphertext. If they match, the output from CTR is returned as the
   decrypted and authenticated plaintext; otherwise, the special symbol
   FAIL is returned.

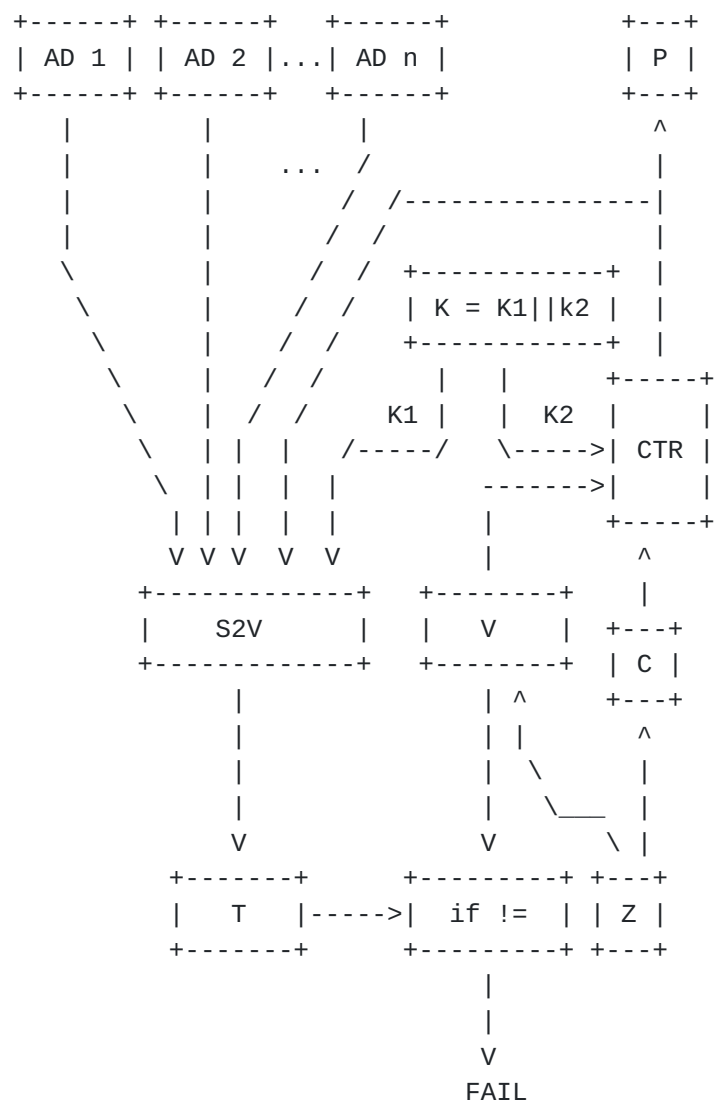The decryption construction of SIV is as follows:

```
+------+ +------+   +------+            +---+
| AD 1 | | AD 2 |...| AD n |            | P |
+------+ +------+   +------+            +---+
   |        |         |                  ^
   |        |    ...  /                   |
   |        |       / /---------------|
   |        |      / /                    |
    \       |     / /  +------------+  |
     \      |    / /   | K = K1||k2 |  |
      \     |   / /    +------------+  |
       \    |  / /        |    |    +-----+
        \   | / /      K1 |    | K2 |     |
         \  | | |  /-----/    \----->| CTR |
          \ | | | |            ------>|     |
           | | | | |             |    +-----+
           V V V  V V            |       ^
        +-------------+    +--------+    |
        |    S2V      |    |   V    | +---+
        +-------------+    +--------+ | C |
             |                 | ^    +---+
             |                 | |      ^
             |                 | \      |
             |                 | \___   |
             V                 V     \ |
        +-------+        +---------+ +---+
        |   T   |----->|  if !=  | | Z |
        +-------+        +---------+ +---+
                             |
                             |
                             V
                           FAIL
```

Figure 10

Algorithmically, SIV-Decrypt can be described as:

```
SIV-DECRYPT(K, Z, AD1, ..., ADn) {
  V = leftmost(Z, 128)
  C = rightmost(Z, len(Z)-128)
  K1 = leftmost(K, len(K)/2)
  K2 = rightmost(K, len(K)/2)
  Q = V bitand (1^64 || 0^1 || 1^31 || 0^1 || 1^31)

  m = (len(C) + 127)/128
  for i = 0 to m-1 do
    Xi = AES(K2, Q+i)
  done
  X = leftmost(X0 || ... || Xm-1, len(C))
  P = C xor X
  T = S2V(K1, AD1, ..., ADn, P)

  if T = V then
    return P
  else
    return FAIL
  fi
}
```

where the key length used by AES in CTR and S2V is len(K)/2 and will each be either 128 bits, 192 bits, or 256 bits.

The 31st and 63rd bit (where the rightmost bit is the 0th) of the counter are zeroed out just prior to being used in CTR mode for optimization purposes, see [Section 5](#).

[3](#). Nonce-Based Authenticated Encryption with SIV

SIV performs nonce-based authenticated encryption when a component of the associated data is a nonce. For purposes of interoperability the final component -- i.e., the string immediately preceding the plaintext in the vector input to S2V -- is used for the nonce. Other associated data are optional. It is up to the specific application of SIV to specify how the rest of the associated data are input.

If the nonce is random, it SHOULD be at least 128 bits in length and be harvested from a pool having at least 128 bits of entropy. A non-random source MAY also be used, for instance, a time stamp, or a counter. The definition of a nonce precludes reuse, but SIV is resistant to nonce reuse. See [Section 1.3.2](#) for a discussion on the security implications of nonce reuse.

It MAY be necessary to transport this nonce with the output generated
by S2V.

4. Deterministic Authenticated Encryption with SIV

   When the plaintext to encrypt and authenticate contains data that is
   unpredictable to an adversary -- for example, a secret key -- SIV can
   be used in a deterministic mode to perform "key wrapping". Because
   S2V allows for associated data and imposes no unnatural size
   restrictions on the data it is protecting, it is a more useful and
   general purpose solution than [RFC3394]. Protocols that use SIV for
   deterministic authenticated encryption (i.e., for more than just
   wrapping of keys) MAY define associated data inputs to SIV. It is
   not necessary to add a nonce component to the AD in this case.

5. Optimizations

   Implementations that cannot or do not wish to support addition modulo
   2^{128} can take advantage of the fact that the 31st and 63rd bits
   (where the rightmost bit is the 0th bit) in the counter are cleared
   before being used by CTR. This allows implementations that natively
   support 32-bit or 64-bit addition to increment the counter naturally.
   Of course, in this case, the amount of plaintext that can be safely
   protected by SIV is reduced by a commensurate amount -- addition
   modulo 2^{32} limits plaintext to (2^{39} - 128) bits, addition modulo
   2^{64} limits plaintext to (2^{71} - 128) bits.

   It is possible to optimize an implementation of S2V when it is being
   used as a key derivation function (KDF), for example in [WLAN]. This
   is because S2V operates on a vector of distinct strings and typically
   the data passed to a KDF contains constant strings. Depending on the
   location of variant components of the input different optimizations
   are possible. The CMACed output of intermediate and invariant
   components can be computed once and cached. This can then be doubled
   and xored with the running sum to produce the output. Or an
   intermediate value that represents the doubled and xored output of
   multiple components, up to the variant component, can be computed
   once and cached.

6. IANA Considerations

   [RFC5116] defines a uniform interface to cipher modes that provide
   nonce-based Authenticated Encryption with Associated Data (AEAD). It
   does this via a registry of AEAD algorithms.

   The Internet Assigned Numbers Authority (IANA) assigned three entries
   from the AEAD Registry for AES-SIV-CMAC-256 (15), AES-SIV-CMAC-384
   (16), and AES-SIV-CMAC-512 (17) based upon the following AEAD

algorithm definitions. [RFC5116] defines operations in octets, not
bits. Limits in this section will therefore be specified in octets.
The security analysis for each of these algorithms is in [DAE].

Unfortunately, [RFC5116] restricts AD input to a single component and
limits the benefit SIV offers for dealing in a natural fashion with
AD consisting of multiple distinct components. Therefore, when it is
required to access SIV through the interface defined in [RFC5116], it
is necessary to marshal multiple AD inputs into a single string (see
Section 1.1) prior to invoking SIV. Note that this requirement is
not unique to SIV. All cipher modes using [RFC5116] MUST similarly
marshal multiple AD inputs into a single string, and any technique
used for any other AEAD mode (e.g., a scatter/gather technique) can
be used with SIV.

[RFC5116] requires AEAD algorithm specifications to include maximal
limits to the amount of plaintext, the amount of associated data, and
the size of a nonce that the AEAD algorithm can accept.

SIV uses AES in counter mode and the security guarantees of SIV would
be lost if the counter was allowed to repeat. Since the counter is
128 bits, a limit to the amount of plaintext that can be safely
protected by a single invocation of SIV is 2^{128} blocks.

To prevent the possibility of collisions, [CMAC] recommends that no
more than 2^{48} invocations be made to CMAC with the same key. This
is not a limit on the amount of data that can be passed to CMAC,
though. There is no practical limit to the amount of data that can
be made to a single invocation of CMAC, and likewise, there is no
practical limit to the amount of associated data or nonce material
that can be passed to SIV.

A collision in the output of S2V would mean the same counter would be
used with different plaintext in counter mode. This would void the
security guarantees of SIV. The "Birthday Paradox" (see [APPCRY])
would imply that no more than 2^{64} distinct invocations to SIV be
made with the same key. It is prudent to follow the example of
[CMAC] though, and further limit the number of distinct invocations
of SIV using the same key to 2^{48}. Note that [RFC5116] does not
provide a variable to describe this limit.

The counter-space for SIV is 2^{128}. Each invocation of SIV consumes
a portion of that counter-space and the amount consumed depends on
the amount of plaintext being passed to that single invocation.
Multiple invocations of SIV with the same key can increase the
possibility of distinct invocations overlapping the counter-space.
The total amount of plaintext that can be safely protected with a

single key is, therefore, a function of the number of distinct
invocations and the amount of plaintext protected with each
invocation.

6.1. AEAD_AES_SIV_CMAC_256

The AES-SIV-CMAC-256 AEAD algorithm works as specified in Sections
2.6 and 2.7. The input and output lengths for AES-SIV-CMAC-256 as
defined by [RFC5116] are:

K_LEN is 32 octets.

P_MAX is 2^{132} octets.

A_MAX is unlimited.

N_MIN is 1 octet.

N_MAX is unlimited.

C_MAX is 2^{132} + 16 octets.

The security implications of nonce reuse and/or misuse are described
in Section 1.3.2.

6.2. AEAD_AES_SIV_CMAC_384

The AES-SIV-CMAC-384 AEAD algorithm works as specified in Sections
2.6 and 2.7. The input and output lengths for AES-SIV-CMAC-384 as
defined by [RFC5116] are:

K_LEN is 48 octets.

P_MAX is 2^{132} octets.

A_MAX is unlimited.

N_MIN is 1 octet.

N_MAX is unlimited.

C_MAX is 2^{132} + 16 octets.

The security implications of nonce reuse and/or misuse are described
in Section 1.3.2.

6.3. **AEAD_AES_SIV_CMAC_512**

   The AES-SIV-CMAC-512 AEAD algorithm works as specified in Sections
   2.6 and 2.7. The input and output lengths for AES-SIV-CMAC-512 as
   defined by [RFC5116] are:

   K_LEN is 64 octets.

   P_MAX is 2^132 octets.

   A_MAX is unlimited.

   N_MIN is 1 octet.

   N_MAX is unlimited.

   C_MAX is 2^132 + 16 octets.

   The security implications of nonce reuse and/or misuse are described
   in Section 1.3.2.

7. **Security Considerations**

   SIV provides confidentiality in the sense that the output of SIV-
   Encrypt is indistinguishable from a random string of bits. It
   provides authenticity in the sense that an attacker is unable to
   construct a string of bits that will return other than FAIL when
   input to SIV-Decrypt. A proof of the security of SIV with an "all-
   in-one" notion of security for an authenticated encryption scheme is
   provided in [DAE].

   SIV provides deterministic "key wrapping" when the plaintext contains
   data that is unpredictable to an adversary (for instance, a
   cryptographic key). Even when this key is made available to an
   attacker the output of SIV-Encrypt is indistinguishable from random
   bits. Similarly, even when this key is made available to an
   attacker, she is unable to construct a string of bits that when input
   to SIV-Decrypt will return anything other than FAIL.

   When the nonce used in the nonce-based authenticated encryption mode
   of SIV-AES is treated with the care afforded a nonce or counter in
   other conventional nonce-based authenticated encryption schemes --
   i.e., guarantee that it will never be used with the same key for two
   distinct invocations -- then SIV achieves the level of security
   described above. If, however, the nonce is reused SIV continues to
   provide the level of authenticity described above but with a slightly
   reduced amount of privacy (see Section 1.3.2).

If S2V is used as a key derivation function, the secret input MUST be
generated uniformly at random. S2V is a pseudo-random function and
is not suitable for use as a random oracle as defined in [[RANDORCL](#)].

The security bound set by the proof of security of S2V in [[DAE](#)]
depends on the number of vector-based queries made by an adversary
and the total number of all components in those queries. The
security is only proven when the number of components in each query
is limited to n-1, where n is the blocksize of the underlying pseudo-
random function. The underlying pseudo-random function used here is
based on AES whose blocksize is 128 bits. Therefore, S2V must not be
passed more than 127 components. Since SIV includes the plaintext as
a component to S2V, that limits the number of components of
associated data that can be safely passed to SIV to 126.

[8](#). Acknowledgments

Thanks to Phil Rogaway for patiently answering numerous questions on
SIV and S2V and for useful critiques of earlier versions of this
paper. Thanks also to David McGrew for numerous helpful comments and
suggestions for improving this paper. Thanks to Jouni Malinen for
reviewing this paper and producing another independent implementation
of SIV, thereby confirming the correctness of the test vectors.

[9](#). References

[9.1](#). Normative References

    [CMAC]   Dworkin, M., "Recommendation for Block Cipher Modes of
                 Operation: The CMAC Mode for Authentication", NIST
                 Special Pulication 800-38B, May 2005.

    [MODES]   Dworkin, M., "Recommendation for Block Cipher Modes of
                 Operation: Methods and Techniques", NIST Special
                 Pulication 800-38A, 2001 edition.

    [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                 Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.

    [RFC5116]   McGrew, D., "An Interface and Algorithms for
                 Authenticated Encryption", [RFC 5116](#), January 2008.

[9.2](#). Informative References

    [APPCRY]   Menezes, A., van Oorshot, P., and S. Vanstone, "Handbook
                 of Applied Cryptography", CRC Press Series on Discrete
                 Mathematics and Its Applications, 1996.

   [BADESP]   Bellovin, S., "Problem Areas for the IP Security
                Protocols", Proceedings from the 6th Usenix UNIX Security
                Symposium, July 22-25 1996.

   [RFC3610]   Whiting, D., Housley, R., and N. Ferguson, "Counter with
                CBC-MAC (CCM)", RFC 3610, September 2003.

   [DAE]   Rogaway, P. and T. Shrimpton, "Deterministic
                Authenticated Encryption, A Provable-Security Treatment
                of the Key-Wrap Problem", Advances in Cryptology --
                EUROCRYPT '06 St. Petersburg, Russia, 2006.

   [GCM]   McGrew, D. and J. Viega, "The Galois/Counter Mode of
                Operation (GCM)".

   [JUTLA]   Jutla, C., "Encryption Modes With Almost Free Message
                Integrity", Proceedings of the International Conference
                on the Theory and Application of Cryptographic
                Techniques: Advances in Cryptography.

   [OCB]   Krovetz, T. and P. Rogaway, "The OCB Authenticated
                Encryption Algorithm", Work in Progress, March 2005.

   [RADKEY]   Zorn, G., Zhang, T., Walker, J., and J. Salowey, "RADIUS
                Attributes for the Delivery of Keying Material", Work in
                Progress, April 2007.

   [RANDORCL]   Bellare, M. and P. Rogaway, "Random Oracles are
                Practical: A Paradigm for Designing Efficient
                Protocols", Proceeding of the First ACM Conference on
                Computer and Communications Security, November 1993.

   [RFC2104]   Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
                Hashing for Message Authentication", RFC 2104, February
                1997.

   [RFC2548]   Zorn, G., "Microsoft Vendor-specific RADIUS Attributes",
                RFC 2548, March 1999.

   [RFC2865]   Rigney, C., Willens, S., Rubens, A., and W. Simpson,
                "Remote Authentication Dial In User Service (RADIUS)",
                RFC 2865, June 2000.

   [RFC3217]   Housley, R., "Triple-DES and RC2 Key Wrapping", RFC 3217,
                December 2001.

   [RFC3394]   Schaad, J. and R. Housley, "Advanced Encryption Standard
               (AES) Key Wrap Algorithm", RFC 3394, September 2002.

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

   [VIRT]   Garfinkel, T. and M. Rosenblum, "When Virtual is Harder
               than Real: Security Challenges in Virtual Machine Based
               Computing Environments" In 10th Workshop on Hot Topics in
               Operating Systems, May 2005.

   [WLAN]   "Draft Standard for IEEE802.11: Wireless LAN Medium
               Access Control (MAC) and Physical Layer (PHY)
               Specification", 2007.

   [X9F1]   Dworkin, M., "Wrapping of Keys and Associated Data",
               Request for review of key wrap algorithms. Cryptology
               ePrint report 2004/340, 2004. Contents are excerpts from
               a draft standard of the Accredited Standards Committee,
               X9, entitled ANS X9.102.

[Appendix A](#). Test Vectors

   The following test vectors are for the mode defined in [Section 6.1](#).

[A.1](#). Deterministic Authenticated Encryption Example

   Input:

   Key:
           fffefdfc fbfaf9f8 f7f6f5f4 f3f2f1f0
           f0f1f2f3 f4f5f6f7 f8f9fafb fcfdfeff

   AD:
           10111213 14151617 18191a1b 1c1d1e1f
           20212223 24252627

   Plaintext:
           11223344 55667788 99aabbcc ddee

   S2V-CMAC-AES

   CMAC(zero):
           0e04dfaf c1efbf04 01405828 59bf073a

   double():
           1c09bf5f 83df7e08 0280b050 b37e0e74

   CMAC(ad):
           f1f922b7 f5193ce6 4ff80cb4 7d93f23b

   xor:
           edf09de8 76c642ee 4d78bce4 ceedfc4f

   double():
           dbe13bd0 ed8c85dc 9af179c9 9ddbf819

   pad:
           11223344 55667788 99aabbcc ddee8000

   xor:
           cac30894 b8eaf254 035bc205 40357819

   CMAC(final):
           85632d07 c6e8f37f 950acd32 0a2ecc93

```
   CTR-AES
   -------
   CTR:
           85632d07 c6e8f37f 150acd32 0a2ecc93

   E(K,CTR):
           51e218d2 c5a2ab8c 4345c4a6 23b2f08f

   ciphertext:
           40c02b96 90c4dc04 daef7f6a fe5c

   output
   ------
   IV || C:
           85632d07 c6e8f37f 950acd32 0a2ecc93
           40c02b96 90c4dc04 daef7f6a fe5c
```

A.2. Nonce-Based Authenticated Encryption Example

```
   Input:
   -----
   Key:
           7f7e7d7c 7b7a7978 77767574 73727170
           40414243 44454647 48494a4b 4c4d4e4f

   AD1:
           00112233 44556677 8899aabb ccddeeff
           deaddada deaddada ffeeddcc bbaa9988
           77665544 33221100

   AD2:
           10203040 50607080 90a0

   Nonce:
           09f91102 9d74e35b d84156c5 635688c0

   Plaintext:
           74686973 20697320 736f6d65 20706c61
           696e7465 78742074 6f20656e 63727970
           74207573 696e6720 5349562d 414553
```

```
S2V-CMAC-AES
------------
CMAC(zero):
      c8b43b59 74960e7c e6a5dd85 231e591a

double():
      916876b2 e92c1cf9 cd4bbb0a 463cb2b3

CMAC(ad1)
      3c9b689a b41102e4 80954714 1dd0d15a

xor:
      adf31e28 5d3d1e1d 4ddefc1e 5bec63e9

double():
      5be63c50 ba7a3c3a 9bbdf83c b7d8c755

CMAC(ad2)
      d98c9b0b e42cb2d7 aa98478e d11eda1b

xor:
      826aa75b 5e568eed 3125bfb2 66c61d4e

double():
      04d54eb6 bcad1dda 624b7f64 cd8c3a1b

CMAC(nonce)
      128c62a1 ce3747a8 372c1c05 a538b96d

xor:
      16592c17 729a5a72 55676361 68b48376

xorend:
      74686973 20697320 736f6d65 20706c61
      696e7465 78742074 6f20656e 63727966
      2d0c6201 f3341575 342a3745 f5c625

CMAC(final)
      7bdb6e3b 432667eb 06f4d14b ff2fbd0f
```

```
   CTR-AES
   -------
   CTR:
           7bdb6e3b 432667eb 06f4d14b 7f2fbd0f

   E(K,CTR):
           bff8665c fdd73363 550f7400 e8f9d376

   CTR+1:
           7bdb6e3b 432667eb 06f4d14b 7f2fbd10

   E(K,CTR+1):
           b2c9088e 713b8617 d8839226 d9f88159

   CTR+2
           7bdb6e3b 432667eb 06f4d14b 7f2fbd11

   E(K,CTR+2):
           9e44d827 234949bc 1b12348e bc195ec7

   ciphertext:
           cb900f2f ddbe4043 26601965 c889bf17
           dba77ceb 094fa663 b7a3f748 ba8af829
           ea64ad54 4a272e9c 485b62a3 fd5c0d

   output
   ------
   IV || C:
           7bdb6e3b 432667eb 06f4d14b ff2fbd0f
           cb900f2f ddbe4043 26601965 c889bf17
           dba77ceb 094fa663 b7a3f748 ba8af829
           ea64ad54 4a272e9c 485b62a3 fd5c0d
```

Author's Address

   Dan Harkins
   Aruba Networks

   EMail: dharkins@arubanetworks.com

Full Copyright Statement

Intellectual Property