Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 13, 2012

D. McGrew
Cisco Systems, Inc.
K. Paterson
Royal Holloway, University of London

June 11, 2012

Authenticated Encryption with AES-CBC and HMAC-SHA
draft-mcgrew-aead-aes-cbc-hmac-sha2-00.txt

Abstract

This document specifies algorithms for authenticated encryption with associated data (AEAD) that are based on the composition of the Advanced Encryption Standard (AES) in the Cipher Block Chaining (CBC) mode of operation for encryption, and the HMAC-SHA message authentication code (MAC).

These are randomized encryption algorithms, and thus are suitable for use with applications that cannot provide distinct nonces to each invocation of the AEAD encrypt operation.

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 http://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 December 13, 2012.

Table of Contents

1. Introduction

Authenticated Encryption (AE) [BN00] is a form of encryption that, in addition to providing confidentiality for the plaintext that is encrypted, provides a way to check its integrity and authenticity. This combination of features can, when properly implemented, provide security against adversaries who have access to full decryption capabilities for ciphertexts of their choice, and access to full encryption capabilities for plaintexts of their choice. The strong form of security provided by AE is known to robust against a large class of adversaries for general purpose applications of AE, including applications such as securing network communications over untrusted networks. The strong security properties of AE stand in contrast to the known weaknesses of "encryption only" forms of encryption, see [B96][YHR04][DP09] for examples.

Authenticated encryption with Associated Data, or AEAD [R02], adds the ability to check the integrity and authenticity of some associated data (sometimes called "additional authenticated data") for which confidentiality is not required (or is not desirable). While many approaches to building AEAD schemes are known, a particularly simple, well-understood, and cryptographically strong method is to employ an "Encrypt-then-MAC" construction. This document defines new AEAD algorithms of this general type, using the interface defined in [RFC5116], based on the Advanced Encryption Standard (AES) [FIPS197] in the Cipher Block Chaining (CBC) mode of operation [SP800-38] and HMAC using the Secure Hash Algorithm (SHA) [FIPS186-2], with security levels of 128, 192, and 256 bits.

1.1. Conventions Used In This Document

We use the following notational conventions.

CBC-ENC(X,P) denotes the CBC encryption of P using the cipher with the key X

MAC(Y, M) denotes the application of the Message Authentication Code (MAC) to the message M, using the key Y

The concatenation of two octet strings A and B is denoted as A || B

len(X) denotes the number of bits in the string X, expressed as an unsigned integer in network byte order.

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 [RFC2119].

2. CBC-HMAC algorithms

This section defines CBC-HMAC, an algorithm based on the the encrypt-then-MAC method defined in Section 4.3 of [BN00]. That method constructs a randomized AEAD algorithm out of a randomized cipher, such as a block cipher mode of operation that uses a random initialization vector, and a MAC.

Section 2.1 and Section 2.2 define the CBC-HMAC encryption and decryption algorithms, without specifying the particular block cipher or hash function to be used. Section 2.4, Section 2.5, Section 2.6, and Section 2.7, define instances of CBC-HMAC that specify those details.

2.1. Encryption

We briefly recall the encryption interface defined in Section 2 of [RFC5116]. The AEAD encryption algorithm takes as input four octet strings: a secret key K, a plaintext P, associated data A, and a nonce N. An authenticated ciphertext value is provided as output. The data in the plaintext are encrypted and authenticated, and the associated data are authenticated, but not encrypted.

In CBC-HMAC, the nonce MUST be a zero-length string; a nonce is not needed and is not used (see Section 4 for further background). CBC-HMAC also has an additional parameter MIN_LEN_A, which is a nonnegative integer that indicates the smallest value of A that will be used with the particular key K. The value of MIN_LEN_A SHOULD be provided when a key is initialized, and the value MUST be unchanging across the life of the key. If the value of MIN_LEN_A is not provided, then it MUST be assumed to be zero.

The CBC-HMAC encryption process is as follows, or uses an equivalent set of steps:

1. The secondary keys MAC_KEY and ENC_KEY are generated from the input key K as follows. Each of these two keys is an octet string.

   MAC_KEY consists of the initial MAC_KEY_LEN octets of K, in order.

   ENC_KEY consists of the final ENC_KEY_LEN octets of K, in order.

   Here we denote the number of octets in the MAC_KEY as MAC_KEY_LEN, and the number of octets in ENC_KEY as ENC_KEY_LEN; the values of these parameters are specified by the AEAD

algorithms (in Section 2.4 and Section 2.5). The number of octets in the input key K is the sum of MAC_KEY_LEN and ENC_KEY_LEN. When generating the secondary keys from K, MAC_KEY and ENC_KEY MUST NOT overlap.

2. An Initialization Vector (IV) is generated randomly or pseudorandomly, as described in Section 3, for use in the cipher.

3. Prior to CBC encryption, the plaintext P is padded by appending a padding string PS to that data, to ensure that len(P || PS) is a multiple of 128, as is needed for the CBC operation. The value of PS is as follows:

```
PS = 01, if len(P) mod 128 = 120,
PS = 0202, if len(P) mod 128 = 112,
PS = 030303, if len(P) mod 128 = 104,
  ...
PS = 0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F, if len(P) mod 128 = 0.
```

   Note that padding MUST be added to the plaintext; if the number of bits in P is a multiple of 128, then 128 bits of padding will be added.

4. The plaintext and appended padding P || PS is CBC encrypted using ENC_KEY as the key, and the IV generated in the previous step. We denote the ciphertext output from this step as S, and it MUST include the IV as its prefix.

5. The octet string AL is computed as follows. If len(A) = MIN_LEN_A, then AL is the zero-length string. Otherwise, it is equal to the number of bits in A expressed as a 64-bit unsigned integer in network byte order.

6. A message authentication tag T is computed by applying HMAC [RFC2104] to the following data, in order:

   the associated data A,

   the ciphertext S computed in the previous step, and

   the octet string AL defined above.

   The string MAC_KEY is used as the MAC key. We denote the output of the MAC computed in this step as T.

7. The AEAD Ciphertext consists of the string S, with the string T appended to it. This Ciphertext is returned as the output of the AEAD encryption operation.

The encryption process can be illustrated as follows. Here P, A, and C denote the AEAD plaintext, associated data, and ciphertext, respectively.

MAC_KEY = initial MAC_KEY_LEN bytes of K

ENC_KEY = final ENC_KEY_LEN bytes of K

S = CBC-ENC(ENC_KEY, P || PS),

T = MAC(MAC_KEY, A || S || AL),

C = S || T.

2.2. Decryption

The authenticated decryption operation has four inputs: K, N, and A, as defined above, and the Ciphertext C. It has only a single output, either a plaintext value P or a special symbol FAIL that indicates that the inputs are not authentic. The authenticated decryption algorithm takes is as follows, or uses an equivalent set of steps:

1. The secondary keys MAC_KEY and ENC_KEY are generated from the input key K as in Step 1 of Section 2.1.

2. The final T_LEN octets are stripped from C. Here T_LEN denotes the number of octets in the MAC, which is a fixed parameter of the AEAD algorithm. We denote the initial octets of C as S, and denote the final T_LEN octets as T.

3. The integrity and authenticity of A and C are checked by computing HMAC with the inputs as in Step 5 of the Section 2.1. The value T, from the previous step, is compared to the HMAC output. If those values are identical, then A and C are considered valid, and processing is continued. Otherwise, all of the data used in the MAC validation are discarded, and the AEAD decryption operation returns an indication that it failed, and the operation halts.

4. The value S is decrypted, using the initial 16 octets of the ciphertext as the IV. The value ENC_KEY is used as the decryption key.

5. The padding string is removed. Note that the length of PS can be inferred from the value of the final octet of P || PS, if that value is between 00 and 0F (hexadecimal). If the final octet has a value outside that range, then all of the data used in the processing of the message is zeroized and discarded, and the AEAD

decryption operation returns an indication that it failed, and the operation halts.

6. The plaintext value is returned.

2.3. Length

The length of the ciphertext can be inferred from that of the plaintext. The number L of octets in the ciphertext is given by

L = 16 * (floor(M / 16) + 2)

where M denotes the number of octets in the plaintext, and the function floor() rounds its argument down to the nearest integer. This fact is useful to applications that need to reserve space for a ciphertext within a message or data structure.

2.4. AEAD_AES_128_CBC_HMAC_SHA_256

This algorithm is randomized and stateless. It is based on the CBC-HMAC algorithm detailed above. It uses the HMAC message authentication code [RFC2104] with the SHA-256 hash function [FIPS186-2] to provide message authentication, with the HMAC output truncated to 128 bits, corresponding to the HMAC-SHA-256-128 algorithm defined in [RFC4868]. For encryption, it uses AES in the cipher block chaining (CBC) mode of operation as defined in Section 6.2 of [SP800-38], with the padding method used by PEM, PKCS, and TLS.

The input key K is 48 octets long.

The AES CBC IV is 16 octets long. ENC_KEY_LEN is 16 octets.

The SHA-256 hash algorithm is used in HMAC. MAC_KEY_LEN is 32 octets. The HMAC-SHA-256 output is truncated to T_LEN=16 octets, by stripping off the final 16 octets. Test cases for HMAC-SHA-256 are provided in [RFC4231].

The lengths of the inputs are restricted as follows:

K_LEN is 48 octets,

P_MAX is 2^64 - 1 octets,

A_MAX is 2^64 - 1 octets,

N_MIN is zero octets,

N_MAX is 2^64 octets, and

C_MAX is 2^64 + 47 octets.

2.5. AEAD_AES_192_CBC_HMAC_SHA_384

AEAD_AES_192_CBC_HMAC_SHA_384 is based on AEAD_AES_128_CBC_HMAC_SHA_256, but with the following differences:

AES-192 is used instead of AES-128.

SHA-384 is used in HMAC instead of SHA-256.

ENC_KEY_LEN is 24 octets.

MAC_KEY_LEN is 48 octets.

The length of the input key K is 72 octets.

The HMAC-SHA-384 value is truncated to T_LEN=24 octets instead of 16 octets.

The input length restrictions are as for AEAD_AES_CBC_128_HMAC_SHA_256.

2.6. AEAD_AES_256_CBC_HMAC_SHA_512

AEAD_AES_256_CBC_HMAC_SHA_512 is based on AEAD_AES_128_CBC_HMAC_SHA_256, but with the following differences:

AES-256 is used instead of AES-128.

SHA-512 is used in HMAC instead of SHA-256.

ENC_KEY_LEN is 32 octets.

MAC_KEY_LEN is 64 octets.

The length of the input key K is 96 octets.

The HMAC-SHA-512 value is truncated to T_LEN=32 octets instead of 16 octets.

The input length restrictions are as for AEAD_AES_CBC_128_HMAC_SHA_256.

2.7. AEAD_AES_128_CBC_HMAC_SHA1

AEAD_AES_128_CBC_HMAC_SHA1 is based on AEAD_AES_128_CBC_HMAC_SHA_256, but with the following differences:

HMAC-SHA1 is used instead of HMAC-SHA-256. Test cases for HMAC-SHA1 are provided in [RFC2202].

MAC_KEY_LEN is 20 octets.

The length of the input key K is 36 octets.

The HMAC-SHA-1 value is truncated to T_LEN=12 octets instead of 16 octets. (Note that this matches the truncation used in [RFC2404].)

The input length restrictions are as for AEAD_AES_CBC_128_HMAC_SHA_256.

2.8. Summary

The parameters of the CBC-HMAC algorithms are summarized in the following table.

algorithm	ENC_KEY_LEN	MAC_KEY_LEN	T_LEN
AEAD_AES_128_CBC_HMAC_SHA_256	16	32	16
AEAD_AES_192_CBC_HMAC_SHA_384	24	48	24
AEAD_AES_256_CBC_HMAC_SHA_512	32	64	32
AEAD_AES_128_CBC_HMAC_SHA1	16	20	12

3. Randomness Requirements

Each IV MUST be unpredictable to the adversary. It MAY be chosen uniformly at random, in which case it SHOULD have min-entropy equal within one bit of ENC_KEY_LEN. Alternatively, it MAY be generated pseudorandomly, using any method that provides the same level of security as the block cipher in use. However, if a pseudorandom method is used, that method MUST NOT make use of ENC_KEY or MAC_KEY.

SP 800-90 describes suitable pseudorandom generators.

4. Rationale

The CBC-HMAC AEAD algorithms defined in this note are intended to be useful in the following applications:

- systems that have the CBC and HMAC algorithms available, but do not have dedicated AEAD algorithms such as GCM or CCM [RFC5116],

- scenarios in which AEAD is useful, but it is undesirable to have the applicaiton maintain a deterministic nonce; see Section 4 of [RFC5116] for more background,

- new systems, such as JSON Cryptography and W3C Web Crypto, which can omit unauthenticated symmetric encryption altogether by providing CBC and HMAC through an AEAD interface.

These algorithms are not intended to replace existing uses of AES-CBC and HMAC, except in those circumstances where the existing use is not sufficiently secure or sufficiently general-purpose.

The length of the associated data input A is included in the HMAC input to ensure that the encrypter and the decrypter have the same understanding of that length. Because of this, an attacker cannot trick the receiver into interpreting the initial bytes of C as the final bytes of A, or vice-versa. The MIN_LEN_A parameter is included so that it is possible to use CBC-HMAC in a way that corresponds to existing uses of CBC and HMAC. Some existing systems (such as ESP or IEEE 1619.1) make use of those algorithms, but do not include the length of the associated data in the HMAC input. These systems can be secure because the length of the associated data is fixed, or can be inferred by other means. The MIN_LEN_A parameter allows the AL field to be zero-length when both the encrypter and the decrypter know the length of the associated data field. This enables implementations of CBC-HMAC to be compatible with existing systems that do not include the length of A in the HMAC input.

The padding method used in this note is based on that of Privacy Enhanced Mail (PEM) and the Public Key Cryptography Standards (PKCS), because it is implemented in many environments.

The encrypt-then-MAC method is used because of its better security properties. It would be possible to define AEAD algorithms based on the MAC-encode-encrypt (MEE) method that is used by the Transport Layer Security (TLS) protocol [RFC5246]. That alternative would provide more code-sharing opportunities for an implementation that is co-resident with a TLS implementation. It is possible (but tricky) to implement MEE in a way that provides good security, as was shown in [PRS11]. But its negatives outweigh its positives; its security

is inadequate for some parameter choices, and it has proven to be difficult to implement in a way that resists padding oracle and related timing attacks [V02] [CHVV03] [M04] [DP10] [AP12]. For future uses of CBC and HMAC, it is better to use encrypt-then-MAC."

5. Test Cases

5.1. AEAD_AES_128_CBC_HMAC_SHA1

```
P = 41206369706865722073797374656d206d757374206e6f742062652072657175
    6972656420746f20626520736563726574c20616e64206974206d7573742062
    652061626c6520746f2066616c6c20696e746f207468652068616e6473206f66
    2074686520656e656d7920776974686f757420696e636f6e76656e69656e6365

K = 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
    20212223

MAC_KEY = 000102030405060708090a0b0c0d0e0f10111213
ENC_KEY = 1415161718191a1b1c1d1e1f20212223

A = 546865207365636f6e64207072696e6369706c65206f662041756775737465206
    4b6572636b686f666673

IV  = 1af38c2dc2b96ffdd86694092341bc04

C_0 = 1af38c2dc2b96ffdd86694092341bc04
P_1 = 41206369706865722073797374656d20
C_1 = c63aec9963f4ff33a85e564cd05f9240
P_2 = 6d757374206e6f742062652072657175
C_2 = 0a71fe98bcb339acc1d78c92b3aa6a32
P_3 = 6972656420746f206265207365637265
C_3 = 1460d2aecec43b784b3b08b830be5291
P_4 = 742c20616e64206974206d7573742062
C_4 = dc0400b8afc6cd1c8475764632a83605
P_5 = 652061626c6520746f2066616c6c2069
C_5 = 01e5319a128127ae4b0eaa9b2f97ea6d
P_6 = 6e746f207468652068616e6473206f66
C_6 = f02200d6f68c743b794ed5d6139e84c4
P_7 = 2074686520656e656d7920776974686f
C_7 = cb919ebb8d8256098b6385e41476c416
P_8 = 757420696e636f6e76656e69656e6365
C_8 = cfc85a46fac40ea450d2b4c0fd7e03dc
P_9 = 10101010101010101010101010101010
C_9 = d833c8c3d2135f0d109bd231808bb3fd

S = 1af38c2dc2b96ffdd86694092341bc04c63aec9963f4ff33a85e564cd05f9240
    0a71fe98bcb339acc1d78c92b3aa6a321460d2aecec43b784b3b08b830be5291
    dc0400b8afc6cd1c8475764632a8360501e5319a128127ae4b0eaa9b2f97ea6d
    f02200d6f68c743b794ed5d6139e84c4cb919ebb8d8256098b6385e41476c416
    cfc85a46fac40ea450d2b4c0fd7e03dcd833c8c3d2135f0d109bd231808bb3fd

T = 9aed4feb1e6d25070b9a6f07
```

```
C = 1af38c2dc2b96ffdd86694092341bc04c63aec9963f4ff33a85e564cd05f92400
    a71fe98bcb339acc1d78c92b3aa6a321460d2aecec43b784b3b08b830be5291dc
    0400b8afc6cd1c8475764632a8360501e5319a128127ae4b0eaa9b2f97ea6df02
    200d6f68c743b794ed5d6139e84c4cb919ebb8d8256098b6385e41476c416cfc8
    5a46fac40ea450d2b4c0fd7e03dcd833c8c3d2135f0d109bd231808bb3fd9aed4
    feb1e6d25070b9a6f07
```

5.2. AEAD_AES_128_CBC_HMAC_SHA256

```
K = 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
    202122232425262700010203040506 07

P = 41206369706865722073797374656d206d757374206e6f742062652072657175
    6972656420746f20626520736563726574c20616e64206974206d7573742062
    652061626c6520746f2066616c6c20696e746f207468652068616e6473206f66
    2074686520656e656d7920776974686f757420696e636f6e76656e69656e6365

A = 546865207365636f6e64207072696e6369706c65206f66204175677573746520
    4b6572636b686f666673

IV  = 1af38c2dc2b96ffdd86694092341bc04

S = 1af38c2dc2b96ffdd86694092341bc04bec9dc25654f7eee633a29d2a14becfe
    79d5b3bfd19b0676584990ee84bb4279197d7ebca389b7e5c101898eed58c34b
    df22b74683a82cf07ae4ddeb4bf5731fc00dd4a98195de6e2e36985161bbe5ec
    13067e3508162da908dede1808a97578dc896d221063c7425679fd6bcfb8ce90
    c197fd05447a2cf7f2fe02a03fdf29c675fa66ecb12e398b7f6fc3c9ae3d03ba

T = e0f20e4a7d3fa9dd5aadcd3ad982f09e

C = 1af38c2dc2b96ffdd86694092341bc04bec9dc25654f7eee633a29d2a14becfe
    79d5b3bfd19b0676584990ee84bb4279197d7ebca389b7e5c101898eed58c34b
    df22b74683a82cf07ae4ddeb4bf5731fc00dd4a98195de6e2e36985161bbe5ec
    13067e3508162da908dede1808a97578dc896d221063c7425679fd6bcfb8ce90
    c197fd05447a2cf7f2fe02a03fdf29c675fa66ecb12e398b7f6fc3c9ae3d03ba
    e0f20e4a7d3fa9dd5aadcd3ad982f09e
```

5.3. AEAD_AES_192_CBC_HMAC_SHA384

K = 000102030405060708090a0b0c0d0e0f10111213141516171819 1a1b1c1d1e1f
202122232425262700010203040506070809 0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

P = 41206369706865722073797374656d206d757374206e6f742062652072657175
6972656420746f20626520736563726 5742c20616e64206974206d7573742062
652061626c6520746f2066616c6c20696e746f207468652068616e6473206f66
2074686520656e656d7920776974686f757420696e636f6e76656e69656e6365

IV = 1af38c2dc2b96ffdd86694092341bc04

A = 546865207365636f6e64207072696e6369706c65206f66204175677573746520
4b6572636b686f666673

PS = 10101010101010101010101010101010

AL = 0000000000000150

S = 1af38c2dc2b96ffdd86694092341bc04ac57bb8225686ff568320b98ad54fa10
eea70c609d4d11d4c34435e386623d0240e9f6c0f78126678546ae2ba2d3017c
4e0ef70d7c5ec2724fecf715518bc48e9048e446077db090e135f33710a2c40d
e0ea744adeca3149a94f9be65fe3e2982ca63e89d026e39319322b417ff9ee5a
b06b71ea5894d9f0ad5089402e174a90cf65bed15f8835d3134a6302ef6cfd4d

T = 0b3948b860797cc2ba0f89d3e79d5465ed770cc8e5b8a430

C = 1af38c2dc2b96ffdd86694092341bc04ac57bb8225686ff568320b98ad54fa10
eea70c609d4d11d4c34435e386623d0240e9f6c0f78126678546ae2ba2d3017c
4e0ef70d7c5ec2724fecf715518bc48e9048e446077db090e135f33710a2c40d
e0ea744adeca3149a94f9be65fe3e2982ca63e89d026e39319322b417ff9ee5a
b06b71ea5894d9f0ad5089402e174a90cf65bed15f8835d3134a6302ef6cfd4d
0b3948b860797cc2ba0f89d3e79d5465ed770cc8e5b8a430

5.4. AEAD_AES_256_CBC_HMAC_SHA512

K = 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262700010203040506070809 0a0b0c0d0e0f1011121314151617
18191a1b1c1d1e1f202122232425262700010203040506070809 0a0b0c0d0e0f

P = 4120636970686572207379737465 6d206d757374206e6f742062652072657175
6972656420746f20626520736563726 5742c20616e64206974206d7573742062
652061626c6520746f2066616c6c20696e746f207468652068616e6473206f66
2074686520656e656d7920776974686f757420696e636f6e76656e69656e6365

IV = 1af38c2dc2b96ffdd86694092341bc04

A = 546865207365636f6e64207072696e6369706c65206f66204175677573746520
4b6572636b686f666673

PS = 10101010101010101010101010101010

AL = 0000000000000150

S = 1af38c2dc2b96ffdd86694092341bc04d39c2d2b1248eb14a7f8d1c1e8f3ff17
309d44cb0cb0dfd38695bf29f37258f74fc9ef1d05b7c71cb3c6fb04bad09105
bc605e8b9c737008a47453b9415cd7407e7314cc73f5ba432172b6da53c33553
8ba9614d79f36e393c8178ba8fb2deec0eaa4cf1eda1cf279fabd9799af60542
04c96e06eacedb231c76c33e38317882eeb55fd9e534c1e73343d8cf00ff283a

T = 2cf60bc4a50b569f0ae708a7889761b3f867c37537a8bd74c162e9b8ee859b08

C = 1af38c2dc2b96ffdd86694092341bc04d39c2d2b1248eb14a7f8d1c1e8f3ff17
309d44cb0cb0dfd38695bf29f37258f74fc9ef1d05b7c71cb3c6fb04bad09105
bc605e8b9c737008a47453b9415cd7407e7314cc73f5ba432172b6da53c33553
8ba9614d79f36e393c8178ba8fb2deec0eaa4cf1eda1cf279fabd9799af60542
04c96e06eacedb231c76c33e38317882eeb55fd9e534c1e73343d8cf00ff283a
2cf60bc4a50b569f0ae708a7889761b3f867c37537a8bd74c162e9b8ee859b08

6. Security Considerations

An earlier version of this document benefitted from some review. Comments on this version are requested and should be forwarded to the IRTF Crypto Forum Research Group (CFRG).

The algorithms defined in this document use the generic composition of CBC encryption with HMAC authentication, with the encrypt-then-MAC method defined in Section 4.3 of [BN00]. This method has sound and well-understood security properties; for details, please see that reference. Note that HMAC is a good pseudorandom function and is "strongly unforgeable", and thus meets all of the security goals of that reference.

There are security benefits to providing both confidentiality and authentication in a single atomic operation, as done in this note. This tight binding prevents subtle attacks such as the padding oracle attack.

As with any block cipher mode of operation, the security of AES-CBC degrades as the amount of data that is process increases. Each fixed key value SHOULD NOT be used to protect more than 2^64 bytes of data. This limit ensures that the AES-CBC algorithm will stay under the birthday bound, i.e. because of the limit, it is unlikely that there will be two AES plaintext inputs that are equal. (If this event occurs, information about the colliding plaintexts is leaked, so it is desirable to bound the amount of plaintext processed in order to make it unlikely.)

7. Acknowledgements

Thanks are due to Matt Miller and John Foley for their constructive feedback; special thanks to John for his generation of the test cases.

8. References

8.1. Normative References

[FIPS186-2] "FIPS 180-2: Secure Hash Standard,", Federal Information Processing Standard (FIPS) http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[FIPS197] "FIPS 197: Advanced Encryption Standard (AES)", Federal Information Processing Standard (FIPS) http://www.itl.nist.gov/fipspubs/fip197.htm.

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

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

[RFC2202] Cheng, P. and R. Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1", RFC 2202, September 1997.

[RFC2404] Madson, C. and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.

[RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, December 2005.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, May 2007.

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

8.2. Informative References

[AP12] Paterson, K. and N. AlFardan, "Plaintext-Recovery Attacks Against Datagram TLS", Network and Distributed System Security Symposium (NDSS) 2012 http://www.isg.rhul.ac.uk/~kp/dtls.pdf, 2012.

[B96] Bellovin, S., "Problem areas for the IP security protocols", Proceedings of the Sixth Usenix Unix Security Symposium https://www.cs.columbia.edu/~smb/papers/badesp.pdf, 1996.

[BN00] "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm", Proceedings of ASIACRYPT 2000, Springer-Verlag, LNCS 1976, pp. 531-545 http://www-cse.ucsd.edu/users/mihir/papers/oem.html.

[CHVV03] Vaudenay, S., Canvel, B., Hiltgen, A., and M. Vuagnoux, "Password Interception in a SSL/TLS Channel", CRYPT0 2003 http://lasecwww.epfl.ch/pub/lasec/doc/CHVV03.ps, 2003.

[DP09] Paterson, K. and D. Degabriele, "On the (in)security of IPsec in MAC-then-encrypt configurations", IEEE Symposium on Privacy and Security http://eprint.iacr.org/2007/125.pdf, 2009.

[DP10] Paterson, K. and D. Degabriele, "On the (in)security of IPsec in MAC-then-encrypt configurations.", ACM Conference on Computer and Communications Security (ACM CCS) 2010 http://www.isg.rhul.ac.uk/~kp/CCSIPsecfinal.pdf, 2010.

[M04] Moeller, B., "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures", Web Page http://www.openssl.org/~bodo/tls-cbc.txt, 2004.

[PRS11] Paterson, K., Ristenpart, T., and T. Shrimpton, "Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol", IEEE Symposium on Security and Privacy 2012 http://web.cecs.pdx.edu/~teshrim/tmAnotherLook.pdf, January 2012.

[R02] "Authenticated encryption with Associated-Data", Proceedings of the 2002 ACM Conference on Computer and Communication Security (CCS'02), pp. 98-107, ACM Press, 2002. http://www.cs.ucdavis.edu/~rogaway/papers/ad.pdf.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[SP800-38] Dworkin, M., "NIST Special Publication 800-38: Recommendation for Block Cipher Modes of Operation", U.S. National Institue of Standards and Technology http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[V02] Vaudenay, S., "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ....", EUROCRYPT 2002 http://lasecwww.epfl.ch/php_code/publications/

search.php?ref=Vau02a, 2002.

[YHR04] Yu, T., Hartman, S., and K. Raeburn, "The Perils of Unauthenticated Encryption: Kerberos Version 4", Network and Distributed Security Symposium (NDSS) 2004 http://web.mit.edu/tlyu/papers/krb4peril-ndss04.pdf, 2004.

Appendix A. CBC Encryption and Decryption

The Cipher Block Chaining (CBC) mode of operation is defined in [SP800-38]. This section recalls how that mode works, for the convenience of the reader. The following notation is used:

K denotes the key of the underlying block cipher,

The function CIPHER(K, P) denotes the encryption of the block P with the block cipher,

The function CIPHER-INV(K, C) denotes the decryption of the block C with the block cipher; this is the inverse operation of CIPHER(), and CIPHER-INV(K, CIPHER(K, P)) = P for all P and all K.

P_1, P_2, ... , P_n denotes the sequence of plaintext blocks, where each block contains exactly the number of bits that the block cipher accepts as its plaintext input,

C_0, C_1, C_2, ... , C_n denotes the sequence of ciphertext blocks, where each block contains exactly the number of bits that the block cipher accepts as its plaintext input,

P_i and C_i denote the ith blocks of the plaintext, and

IV denotes the initialization vector, which contains exactly the number of bits that the block cipher accepts as its plaintext input.

The CBC encryption operation (denoted as CBC-ENC) takes as input a sequence of n plaintext blocks and produces a sequence of n + 1 ciphertext blocks as follows:

```
IV  = random
C_i = / IV                          if i=0,
      \ CIPHER(K, P_i XOR C_{i-1})  if i=1, 2, ... , n.
```

The IV MUST be generated using a uniformly random process, or a pseudorandom process with a cryptographic strength equivalent to that of the underlying block cipher. It MUST NOT be predictable to an attacker; in particular, it MUST NOT be set to the value of any previous ciphertext blocks.

The CBC decryption operation (denoted as CBC-DEC) takes as input a sequence of m ciphertext blocks and produces a sequence of m-1 plaintext blocks as follows:

```
P_i = CIPHER-INV(K, P_1 XOR IV)     for i=1, 2, ... , n.
```

Authors' Addresses

David A. McGrew
Cisco Systems, Inc.
13600 Dulles Technology Drive
Herndon, VA 20171
US

Phone: (408) 525 8651
Email: mcgrew@cisco.com
URI: http://www.mindspring.com/~dmcgrew/dam.htm

Kenny Paterson
Royal Holloway, University of London
TW20 0EX
Egham, Surrey TW20 0EX
UK

Phone: +44 1784 414393
Email: Kenny.Paterson@rhul.ac.uk
URI: http://www.isg.rhul.ac.uk/~kp/