            Authenticated Encryption with AES-CBC and HMAC-SHA
                 draft-mcgrew-aead-aes-cbc-hmac-sha2-05.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 January 5, 2015.

Copyright Notice

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 be 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] [DP07] 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.

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

1.1. History

   This subsection describes the revision history of this Internet
   Draft. It should be removed by the RFC Editor before publication as
   an RFC.

   The changes of version 05 from version 05 consist only of changes in
   Appendix A and the test cases. A variable Q was defined to make the
   legacy encoding more clear, after discussion between the authors and
   Mike Jones.

   The changes of version 02 from version 01 are:

      Added test cases for each of the five operational modes.

Added John as a coauthor.

Adds a legacy-style interface in Appendix B.

The changes of version 01 from version 00 are:

MIN_LEN_A and associated logic was eliminated.

Padding String (PS) typo corrected in Section 2.1.

Decryption Step 3 refers to the appropriate step in the encryption process.

Random IV min-entropy clarified in Section 3.

HMAC keys are now the same size as the truncated output (128 or 256 bits). Previously, the HMAC keys were the same size as the full hash output (256, 384, or 512 bits).

An algorithm based on the combination of AES-256 and HMAC-SHA-384 has been added, for compatibility with draft-burgin-kerberos-aes-cbc-hmac-sha2.

The test cases in the previous version are no longer valid, and thus have been removed. New test cases have been computed (and the authors thank John Foley for this contribution) but have not been included, pending confirmation from a second, independent implementation.

1.2. 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, 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. The key MUST be generated in a way that is uniformly random or pseudorandom; guidance on random sources is provided in [RFC4086].

In CBC-HMAC, the nonce N MUST be a zero-length string; a nonce is not needed and is not used (see Section 4 for further background).

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. 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 = 04040404                         if len(P) mod 128 = 96,
   PS = 0505050505                       if len(P) mod 128 = 88,
   PS = 060606060606                     if len(P) mod 128 = 80,
   PS = 07070707070707                   if len(P) mod 128 = 72,
   PS = 0808080808080808                 if len(P) mod 128 = 64,
   PS = 090909090909090909               if len(P) mod 128 = 56,
   PS = 0A0A0A0A0A0A0A0A0A0A             if len(P) mod 128 = 48,
   PS = 0B0B0B0B0B0B0B0B0B0B0B           if len(P) mod 128 = 40,
   PS = 0C0C0C0C0C0C0C0C0C0C0C0C         if len(P) mod 128 = 32,
   PS = 0D0D0D0D0D0D0D0D0D0D0D0D0D       if len(P) mod 128 = 24,
   PS = 0E0E0E0E0E0E0E0E0E0E0E0E0E0E     if len(P) mod 128 = 16,
   PS = 0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F   if len(P) mod 128 = 8,
   PS = 101010101010101010101010101010 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.

3. The plaintext and appended padding P || PS is CBC encrypted using
    ENC_KEY as the key, as described in Appendix A. We denote the
    ciphertext output from this step as S.

4. The octet string AL is equal to the number of bits in A expressed
    as a 64-bit unsigned integer in network byte order.

5. 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.

6. 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. As discussed above, N is an
empty string in AES-CBC and is not used below. 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 6 of Section 2.1. The
    value T, from the previous step, is compared to the HMAC output,
    using a comparison routine that takes constant time to execute.
    If those values are identical, then A and C are considered valid,
    and the processing continues. 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 CBC decrypted, as described in Appendix A, using
    the value ENC_KEY is as the decryption key.

5. The padding string is stripped from the resulting plaintext.
    Note that the length of PS can be inferred from the value of the
    final octet of P || PS, if that value is between 01 and 10

       (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; each invocation of the encrypt
   operation makes use of a random value (the IV described in
   Appendix A). It is based on the CBC-HMAC algorithm detailed above,
   and 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 the
   Advanced Encryption Standard (AES) [FIPS197] block cipher in CBC
   mode.

   The input key K is 32 octets long.

   ENC_KEY_LEN is 16 octets.

   The SHA-256 hash algorithm is used in HMAC. MAC_KEY_LEN is 16
   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 32 octets,

      P_MAX is 2^64 - 1 octets,

      A_MAX is 2^64 - 1 octets,

N_MIN and N_MAX are zero octets,

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 24 octets.

The length of the input key K is 48 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_384

AEAD_AES_256_CBC_HMAC_SHA_384 is based on
AEAD_AES_128_CBC_HMAC_SHA_256, but with the following differences:

AES-256 is used instead of AES-128.

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

ENC_KEY_LEN is 32 octets.

MAC_KEY_LEN is 24 octets.

The length of the input key K is 56 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.

2.7. 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 32 octets.

   The length of the input key K is 64 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.

2.8. 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	16	16
AEAD_AES_192_CBC_HMAC_SHA_384	24	24	24
AEAD_AES_256_CBC_HMAC_SHA_384	32	24	24
AEAD_AES_256_CBC_HMAC_SHA_512	32	32	32

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 within
   one bit of len(IV). 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.

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 application 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 algorithms in this note truncate the HMAC output to half of the
   size of the output of the underlying hash function. This size is the
   recommended minimum (see Section 5 of [RFC2104]), and this parameter
   choice has withstood the test of time.

   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 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
   very 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.

This note uses HMAC-SHA-2 because it is widely deployed, it is
mandated in newer standards, and because SHA1 is being deprecated.
It has been recently announced that the SHA-3 standard will be based
on KECCAK, but this note does not incorporate that hash function. To
do so would be to speculate on the final form of the SHA-3 standard.
In addition, while the use of KECCAK as a hash function is
straightforward, there are multiple options for its use in
authenticated encryption. The focus of this note is the definition
of AEAD algorithms based on currently used cryptographic mechanisms,
so SHA-3 is out of scope.

[5](#). Test Cases

[5.1](#). AEAD_AES_128_CBC_HMAC_SHA256

```
K =       00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

ENC_KEY = 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

P =       41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
          6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
          69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
          74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
          65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
          6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
          20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
          75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65


IV =      1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =       54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
          69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
          4b 65 72 63 6b 68 6f 66 66 73

PS =      10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

AL =      00 00 00 00 00 00 01 50
```

```
Q =        c8 0e df a3 2d df 39 d5 ef 00 c0 b4 68 83 42 79
           a2 e4 6a 1b 80 49 f7 92 f7 6b fe 54 b9 03 a9 c9
           a9 4a c9 b4 7a d2 65 5c 5f 10 f9 ae f7 14 27 e2
           fc 6f 9b 3f 39 9a 22 14 89 f1 63 62 c7 03 23 36
           09 d4 5a c6 98 64 e3 32 1c f8 29 35 ac 40 96 c8
           6e 13 33 14 c5 40 19 e8 ca 79 80 df a4 b9 cf 1b
           38 4c 48 6f 3a 54 c5 10 78 15 8e e5 d7 9d e5 9f
           bd 34 d8 48 b3 d6 95 50 a6 76 46 34 44 27 ad e5
           4b 88 51 ff b5 98 f7 f8 00 74 b9 47 3c 82 e2 db

S =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
           c8 0e df a3 2d df 39 d5 ef 00 c0 b4 68 83 42 79
           a2 e4 6a 1b 80 49 f7 92 f7 6b fe 54 b9 03 a9 c9
           a9 4a c9 b4 7a d2 65 5c 5f 10 f9 ae f7 14 27 e2
           fc 6f 9b 3f 39 9a 22 14 89 f1 63 62 c7 03 23 36
           09 d4 5a c6 98 64 e3 32 1c f8 29 35 ac 40 96 c8
           6e 13 33 14 c5 40 19 e8 ca 79 80 df a4 b9 cf 1b
           38 4c 48 6f 3a 54 c5 10 78 15 8e e5 d7 9d e5 9f
           bd 34 d8 48 b3 d6 95 50 a6 76 46 34 44 27 ad e5
           4b 88 51 ff b5 98 f7 f8 00 74 b9 47 3c 82 e2 db

T =        65 2c 3f a3 6b 0a 7c 5b 32 19 fa b3 a3 0b c1 c4

C =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
           c8 0e df a3 2d df 39 d5 ef 00 c0 b4 68 83 42 79
           a2 e4 6a 1b 80 49 f7 92 f7 6b fe 54 b9 03 a9 c9
           a9 4a c9 b4 7a d2 65 5c 5f 10 f9 ae f7 14 27 e2
           fc 6f 9b 3f 39 9a 22 14 89 f1 63 62 c7 03 23 36
           09 d4 5a c6 98 64 e3 32 1c f8 29 35 ac 40 96 c8
           6e 13 33 14 c5 40 19 e8 ca 79 80 df a4 b9 cf 1b
           38 4c 48 6f 3a 54 c5 10 78 15 8e e5 d7 9d e5 9f
           bd 34 d8 48 b3 d6 95 50 a6 76 46 34 44 27 ad e5
           4b 88 51 ff b5 98 f7 f8 00 74 b9 47 3c 82 e2 db
           65 2c 3f a3 6b 0a 7c 5b 32 19 fa b3 a3 0b c1 c4
```

5.2. AEAD_AES_192_CBC_HMAC_SHA384

```
K =        00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
           10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
           20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
          10 11 12 13 14 15 16 17

ENC_KEY = 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27
          28 29 2a 2b 2c 2d 2e 2f

P =        41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
           6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
           69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
           74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
           65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
           6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
           20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
           75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =       1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =        54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
           69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
           4b 65 72 63 6b 68 6f 66 66 73

PS =       10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

AL =       00 00 00 00 00 00 01 50
```

```
   Q =          ea 65 da 6b 59 e6 1e db 41 9b e6 2d 19 71 2a e5
                d3 03 ee b5 00 52 d0 df d6 69 7f 77 22 4c 8e db
                00 0d 27 9b dc 14 c1 07 26 54 bd 30 94 42 30 c6
                57 be d4 ca 0c 9f 4a 84 66 f2 2b 22 6d 17 46 21
                4b f8 cf c2 40 0a dd 9f 51 26 e4 79 66 3f c9 0b
                3b ed 78 7a 2f 0f fc bf 39 04 be 2a 64 1d 5c 21
                05 bf e5 91 ba e2 3b 1d 74 49 e5 32 ee f6 0a 9a
                c8 bb 6c 6b 01 d3 5d 49 78 7b cd 57 ef 48 49 27
                f2 80 ad c9 1a c0 c4 e7 9c 7b 11 ef c6 00 54 e3

   S =          1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
                ea 65 da 6b 59 e6 1e db 41 9b e6 2d 19 71 2a e5
                d3 03 ee b5 00 52 d0 df d6 69 7f 77 22 4c 8e db
                00 0d 27 9b dc 14 c1 07 26 54 bd 30 94 42 30 c6
                57 be d4 ca 0c 9f 4a 84 66 f2 2b 22 6d 17 46 21
                4b f8 cf c2 40 0a dd 9f 51 26 e4 79 66 3f c9 0b
                3b ed 78 7a 2f 0f fc bf 39 04 be 2a 64 1d 5c 21
                05 bf e5 91 ba e2 3b 1d 74 49 e5 32 ee f6 0a 9a
                c8 bb 6c 6b 01 d3 5d 49 78 7b cd 57 ef 48 49 27
                f2 80 ad c9 1a c0 c4 e7 9c 7b 11 ef c6 00 54 e3

   T =          84 90 ac 0e 58 94 9b fe 51 87 5d 73 3f 93 ac 20
                75 16 80 39 cc c7 33 d7


   C =          1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
                ea 65 da 6b 59 e6 1e db 41 9b e6 2d 19 71 2a e5
                d3 03 ee b5 00 52 d0 df d6 69 7f 77 22 4c 8e db
                00 0d 27 9b dc 14 c1 07 26 54 bd 30 94 42 30 c6
                57 be d4 ca 0c 9f 4a 84 66 f2 2b 22 6d 17 46 21
                4b f8 cf c2 40 0a dd 9f 51 26 e4 79 66 3f c9 0b
                3b ed 78 7a 2f 0f fc bf 39 04 be 2a 64 1d 5c 21
                05 bf e5 91 ba e2 3b 1d 74 49 e5 32 ee f6 0a 9a
                c8 bb 6c 6b 01 d3 5d 49 78 7b cd 57 ef 48 49 27
                f2 80 ad c9 1a c0 c4 e7 9c 7b 11 ef c6 00 54 e3
                84 90 ac 0e 58 94 9b fe 51 87 5d 73 3f 93 ac 20
                75 16 80 39 cc c7 33 d7
```

5.3.   **AEAD_AES_256_CBC_HMAC_SHA384**

```
K =         00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
            10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
            20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
            30 31 32 33 34 35 36 37

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
            10 11 12 13 14 15 16 17

ENC_KEY = 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27
            28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37

P =         41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
            6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
            69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
            74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
            65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
            6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
            20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
            75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =         54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
            69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
            4b 65 72 63 6b 68 6f 66 66 73

PS =        10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

AL =        00 00 00 00 00 00 01 50
```

```
   Q =        89 31 29 b0 f4 ee 9e b1 8d 75 ed a6 f2 aa a9 f3
              60 7c 98 c4 ba 04 44 d3 41 62 17 0d 89 61 88 4e
              58 f2 7d 4a 35 a5 e3 e3 23 4a a9 94 04 f3 27 f5
              c2 d7 8e 98 6e 57 49 85 8b 88 bc dd c2 ba 05 21
              8f 19 51 12 d6 ad 48 fa 3b 1e 89 aa 7f 20 d5 96
              68 2f 10 b3 64 8d 3b b0 c9 83 c3 18 5f 59 e3 6d
              28 f6 47 c1 c1 39 88 de 8e a0 d8 21 19 8c 15 09
              77 e2 8c a7 68 08 0b c7 8c 35 fa ed 69 d8 c0 b7
              d9 f5 06 23 21 98 a4 89 a1 a6 ae 03 a3 19 fb 30

   S =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
              89 31 29 b0 f4 ee 9e b1 8d 75 ed a6 f2 aa a9 f3
              60 7c 98 c4 ba 04 44 d3 41 62 17 0d 89 61 88 4e
              58 f2 7d 4a 35 a5 e3 e3 23 4a a9 94 04 f3 27 f5
              c2 d7 8e 98 6e 57 49 85 8b 88 bc dd c2 ba 05 21
              8f 19 51 12 d6 ad 48 fa 3b 1e 89 aa 7f 20 d5 96
              68 2f 10 b3 64 8d 3b b0 c9 83 c3 18 5f 59 e3 6d
              28 f6 47 c1 c1 39 88 de 8e a0 d8 21 19 8c 15 09
              77 e2 8c a7 68 08 0b c7 8c 35 fa ed 69 d8 c0 b7
              d9 f5 06 23 21 98 a4 89 a1 a6 ae 03 a3 19 fb 30

   T =        dd 13 1d 05 ab 34 67 dd 05 6f 8e 88 2b ad 70 63
              7f 1e 9a 54 1d 9c 23 e7

   C =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
              89 31 29 b0 f4 ee 9e b1 8d 75 ed a6 f2 aa a9 f3
              60 7c 98 c4 ba 04 44 d3 41 62 17 0d 89 61 88 4e
              58 f2 7d 4a 35 a5 e3 e3 23 4a a9 94 04 f3 27 f5
              c2 d7 8e 98 6e 57 49 85 8b 88 bc dd c2 ba 05 21
              8f 19 51 12 d6 ad 48 fa 3b 1e 89 aa 7f 20 d5 96
              68 2f 10 b3 64 8d 3b b0 c9 83 c3 18 5f 59 e3 6d
              28 f6 47 c1 c1 39 88 de 8e a0 d8 21 19 8c 15 09
              77 e2 8c a7 68 08 0b c7 8c 35 fa ed 69 d8 c0 b7
              d9 f5 06 23 21 98 a4 89 a1 a6 ae 03 a3 19 fb 30
              dd 13 1d 05 ab 34 67 dd 05 6f 8e 88 2b ad 70 63
              7f 1e 9a 54 1d 9c 23 e7
```

[5.4](#). **AEAD_AES_256_CBC_HMAC_SHA512**

```
K =         00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
            10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
            20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
            30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f

MAC_KEY = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
            10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

ENC_KEY = 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
            30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f

P =         41 20 63 69 70 68 65 72 20 73 79 73 74 65 6d 20
            6d 75 73 74 20 6e 6f 74 20 62 65 20 72 65 71 75
            69 72 65 64 20 74 6f 20 62 65 20 73 65 63 72 65
            74 2c 20 61 6e 64 20 69 74 20 6d 75 73 74 20 62
            65 20 61 62 6c 65 20 74 6f 20 66 61 6c 6c 20 69
            6e 74 6f 20 74 68 65 20 68 61 6e 64 73 20 6f 66
            20 74 68 65 20 65 6e 65 6d 79 20 77 69 74 68 6f
            75 74 20 69 6e 63 6f 6e 76 65 6e 69 65 6e 63 65

IV =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04

A =         54 68 65 20 73 65 63 6f 6e 64 20 70 72 69 6e 63
            69 70 6c 65 20 6f 66 20 41 75 67 75 73 74 65 20
            4b 65 72 63 6b 68 6f 66 66 73

PS =        10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

AL =        00 00 00 00 00 00 01 50
```

```
Q =        4a ff aa ad b7 8c 31 c5 da 4b 1b 59 0d 10 ff bd
           3d d8 d5 d3 02 42 35 26 91 2d a0 37 ec bc c7 bd
           82 2c 30 1d d6 7c 37 3b cc b5 84 ad 3e 92 79 c2
           e6 d1 2a 13 74 b7 7f 07 75 53 df 82 94 10 44 6b
           36 eb d9 70 66 29 6a e6 42 7e a7 5c 2e 08 46 a1
           1a 09 cc f5 37 0d c8 0b fe cb ad 28 c7 3f 09 b3
           a3 b7 5e 66 2a 25 94 41 0a e4 96 b2 e2 e6 60 9e
           31 e6 e0 2c c8 37 f0 53 d2 1f 37 ff 4f 51 95 0b
           be 26 38 d0 9d d7 a4 93 09 30 80 6d 07 03 b1 f6

S =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
           4a ff aa ad b7 8c 31 c5 da 4b 1b 59 0d 10 ff bd
           3d d8 d5 d3 02 42 35 26 91 2d a0 37 ec bc c7 bd
           82 2c 30 1d d6 7c 37 3b cc b5 84 ad 3e 92 79 c2
           e6 d1 2a 13 74 b7 7f 07 75 53 df 82 94 10 44 6b
           36 eb d9 70 66 29 6a e6 42 7e a7 5c 2e 08 46 a1
           1a 09 cc f5 37 0d c8 0b fe cb ad 28 c7 3f 09 b3
           a3 b7 5e 66 2a 25 94 41 0a e4 96 b2 e2 e6 60 9e
           31 e6 e0 2c c8 37 f0 53 d2 1f 37 ff 4f 51 95 0b
           be 26 38 d0 9d d7 a4 93 09 30 80 6d 07 03 b1 f6

T =        4d d3 b4 c0 88 a7 f4 5c 21 68 39 64 5b 20 12 bf
           2e 62 69 a8 c5 6a 81 6d bc 1b 26 77 61 95 5b c5


C =        1a f3 8c 2d c2 b9 6f fd d8 66 94 09 23 41 bc 04
           4a ff aa ad b7 8c 31 c5 da 4b 1b 59 0d 10 ff bd
           3d d8 d5 d3 02 42 35 26 91 2d a0 37 ec bc c7 bd
           82 2c 30 1d d6 7c 37 3b cc b5 84 ad 3e 92 79 c2
           e6 d1 2a 13 74 b7 7f 07 75 53 df 82 94 10 44 6b
           36 eb d9 70 66 29 6a e6 42 7e a7 5c 2e 08 46 a1
           1a 09 cc f5 37 0d c8 0b fe cb ad 28 c7 3f 09 b3
           a3 b7 5e 66 2a 25 94 41 0a e4 96 b2 e2 e6 60 9e
           31 e6 e0 2c c8 37 f0 53 d2 1f 37 ff 4f 51 95 0b
           be 26 38 d0 9d d7 a4 93 09 30 80 6d 07 03 b1 f6
           4d d3 b4 c0 88 a7 f4 5c 21 68 39 64 5b 20 12 bf
           2e 62 69 a8 c5 6a 81 6d bc 1b 26 77 61 95 5b c5
```

6. Security Considerations

   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.

   Implementations of the encryption and decryption algorithms should
   avoid side channels that would leak information about the secret key.
   To avoid timing channels, the processing time should be independent
   of the secret key. The Encrypt-then-MAC construction used in this
   note has some inherent strength against timing attacks because,
   during the decryption operation, the authentication check is computed
   before the plaintext padding is processed. However, the security of
   the algorithm still relies on the absence of timing channels in both
   CBC and HMAC. Additionally, comparison between the authentication
   tag T and the HMAC output should be done using a constant-time
   operation.

   During the decryption process, the inputs A and C are mapped into the
   input of the HMAC algorithm. It is essential for security that each
   possible input to the MAC algorithm corresponds unambiguously to
   exactly one pair (A, C) of possible inputs. The fact that this
   property holds can be verified as follows. The HMAC input is X = A
   || C || len(A). Let (A,C) and (A',C') denote two distinct input
   pairs, in which either 1) A != A' and C = C', 2) C != C and A = A',
   or 3) both inequalities hold. We also let X' = A' || C' || len(A').
   In cases 1 and 2, X != X' follows immediately. In case 3, if len(A)
   != len(A'), then X != X' directly. If len(A) = len(A'), then X != X
   follows from the fact that the initial len(A) bits of X and X' must
   be distinct.

   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 for his constructive feedback, Kelly
Burgin, Michael Peck, and Mike Jones for their suggestions and help,
and Jim Schaad, Rob Napier, James Manger, and David Jacobson for
their excellent review and suggestions.

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.

   [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.

[DP07]   Paterson, K. and J. Degabriele, "Attacking the IPsec Standards in Encryption-only Configurations", IEEE Symposium on Privacy and Security http://eprint.iacr.org/2007/125.pdf, 2007.

[DP10]   Paterson, K. and J. 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://www.isg.rhul.ac.uk/~kp/mee-comp.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.

[RFC4086]   Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[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
   Section 6.2 of [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, where P contains exactly b bits,

      The function CIPHER-INV(K, Q) denotes the decryption of the block
      Q with the block cipher, where Q contains exactly b bits; 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 b bits,

      Q_1, Q_2, ... , Q_n denotes the sequence of ciphertext blocks,
      where each block contains exactly b bits,

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

      IV denotes the initialization vector, which contains exactly b
      bits.

   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
        Q_i = / CIPHER(K, P_i XOR IV)   if i=0,
              \ CIPHER(K, P_i XOR Q_{i-1})   if i=1, 2, ... , n.

   The operation CBC-ENC(K, P_1 || P_2 || ... || P_n) returns the value
   IV || Q_1 || Q_2 || ... || Q_n. Note that the returned value is one
   block longer than the input value.

   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; see [RFC4086] for background on
   random sources. 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 an
   octet string whose length is a multiple of b bits, decomposes it as

IV || Q_1 || Q_2 || ... || Q_m, then produces a sequence of m
plaintext blocks as follows:

       P_i = / CIPHER-INV(K, Q_i) XOR IV   if i=1.
             \ CIPHER-INV(K, Q_i) XOR Q_{i-1}   if i=2, ... , m.

The operation CBC-DEC(K, IV || Q_1 || Q_2 || ... || Q_m) returns the
value P_1 || P_2 || ... || P_m.

Appendix B. Alternative Interface for Legacy Encoding

   In some scenarios, cryptographic data such as the ciphertext,
   initialization vector, and message authentication tag are encoded
   separately. To allow for the use of the algorithms defined in this
   document in such scenarios, this appendix describes an interface in
   which those data elements are discrete. New implementations SHOULD
   NOT use this interface, because it is incompatible with other
   authenticated encryption methods and is more complex; however, it MAY
   be useful in scenarios in which the separate encoding is already in
   use.

   The alternative interface is as follows. The inputs to the
   encryption operation the same as those defined in Section 2.1 (the
   secret key K, the plaintext P, the associated data A). The outputs
   of the encryption operation are:

      the initialization vector IV as defined in Appendix A,

      the ciphertext C, as defined in Appendix A, and

      the message authentication tag T, as defined in Section 2.1.

   The inputs to the decryption operation (in addition to K and A) are:

      the initialization vector IV as defined in Appendix A,

      the ciphertext C as defined in Appendix A, excluding the initial
      block C_0 (which is equal to the IV), and

      the message authentication tag T, as defined in Section 2.1.

   The output of the decryption operation are the same as that defined
   in Section 2.2 (either a plaintext value P or a special symbol FAIL
   that indicates that the inputs are not authentic).

   All processing other than the encoding and decoding of IV, C, and T
   is done as defined above. In particular, the IV is an output of the
   encryption operation, rather than an input.

Authors' Addresses

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

    Email: mcgrew@cisco.com
    URI: http://www.mindspring.com/~dmcgrew/dam.htm

    John Foley
    Cisco Systems
    7025-2 Kit Creek Road
    Research Triangle Park, NC 14987
    US

    Email: foleyj@cisco.com

    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/