Network Working Group
Request for Comments: 5116
Category: Standards Track
D. McGrew
Cisco Systems, Inc.
January 2008

An Interface and Algorithms for Authenticated Encryption

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document defines algorithms for Authenticated Encryption with Associated Data (AEAD), and defines a uniform interface and a registry for such algorithms. The interface and registry can be used as an application-independent set of cryptoalgorithm suites. This approach provides advantages in efficiency and security, and promotes the reuse of crypto implementations.

Table of Contents

1. Introduction

Authenticated encryption [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. Authenticated Encryption with Associated Data, or AEAD [R02], adds the ability to check the integrity and authenticity of some Associated Data (AD), also called "additional authenticated data", that is not encrypted.

1.1. Background

Many cryptographic applications require both confidentiality and message authentication. Confidentiality is a security service that ensures that data is available only to those authorized to obtain it; usually it is realized through encryption. Message authentication is the service that ensures that data has not been altered or forged by unauthorized entities; it can be achieved by using a Message Authentication Code (MAC). This service is also called data integrity. Many applications use an encryption method and a MAC together to provide both of those security services, with each algorithm using an independent key. More recently, the idea of providing both security services using a single cryptoalgorithm has become accepted. In this concept, the cipher and MAC are replaced by an Authenticated Encryption with Associated Data (AEAD) algorithm.

Several crypto algorithms that implement AEAD algorithms have been defined, including block cipher modes of operation and dedicated algorithms. Some of these algorithms have been adopted and proven useful in practice. Additionally, AEAD is close to an 'idealized' view of encryption, such as those used in the automated analysis of cryptographic protocols (see, for example, Section 2.5 of [BOYD]).

The benefits of AEAD algorithms, and this interface, are outlined in Section 1.3.

1.2. Scope

In this document, we define an AEAD algorithm as an abstraction, by specifying an interface to an AEAD and defining an IANA registry for AEAD algorithms. We populate this registry with four AEAD algorithms based on the Advanced Encryption Standard (AES) in Galois/Counter Mode [GCM] with 128- and 256-bit keys, and AES in Counter and CBC MAC Mode [CCM] with 128- and 256-bit keys.

In the following, we define the AEAD interface (Section 2), and then provide guidance on the use of AEAD algorithms (Section 3), and outline the requirements that each AEAD algorithm must meet

(Section 4). Then we define several AEAD algorithms (Section 5), and establish an IANA registry for AEAD algorithms (Section 6). Lastly, we discuss some other considerations (Section 7).

The AEAD interface specification does not address security protocol issues such as anti-replay services or access control decisions that are made on authenticated data. Instead, the specification aims to abstract the cryptography away from those issues. The interface, and the guidance about how to use it, are consistent with the recommendations from [EEM04].

1.3. Benefits

The AEAD approach enables applications that need cryptographic security services to more easily adopt those services. It benefits the application designer by allowing them to focus on important issues such as security services, canonicalization, and data marshaling, and relieving them of the need to design crypto mechanisms that meet their security goals. Importantly, the security of an AEAD algorithm can be analyzed independent from its use in a particular application. This property frees the user of the AEAD of the need to consider security aspects such as the relative order of authentication and encryption and the security of the particular combination of cipher and MAC, such as the potential loss of confidentiality through the MAC. The application designer that uses the AEAD interface need not select a particular AEAD algorithm during the design stage. Additionally, the interface to the AEAD is relatively simple, since it requires only a single key as input and requires only a single identifier to indicate the algorithm in use in a particular case.

The AEAD approach benefits the implementer of the crypto algorithms by making available optimizations that are otherwise not possible to reduce the amount of computation, the implementation cost, and/or the storage requirements. The simpler interface makes testing easier; this is a considerable benefit for a crypto algorithm implementation. By providing a uniform interface to access cryptographic services, the AEAD approach allows a single crypto implementation to more easily support multiple applications. For example, a hardware module that supports the AEAD interface can easily provide crypto acceleration to any application using that interface, even to applications that had not been designed when the module was built.

1.4. Conventions Used in This Document

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. AEAD Interface

An AEAD algorithm has two operations, authenticated encryption and authenticated decryption. The inputs and outputs of these algorithms are defined below in terms of octet strings.

An implementation MAY accept additional inputs. For example, an input could be provided to allow the user to select between different implementation strategies. However, such extensions MUST NOT affect interoperability with other implementations.

2.1. Authenticated Encryption

The authenticated encryption operation has four inputs, each of which is an octet string:

A secret key K, which MUST be generated in a way that is uniformly random or pseudorandom.

A nonce N. Each nonce provided to distinct invocations of the Authenticated Encryption operation MUST be distinct, for any particular value of the key, unless each and every nonce is zero-length. Applications that can generate distinct nonces SHOULD use the nonce formation method defined in Section 3.2, and MAY use any other method that meets the uniqueness requirement. Other applications SHOULD use zero-length nonces.

A plaintext P, which contains the data to be encrypted and authenticated.

The associated data A, which contains the data to be authenticated, but not encrypted.

There is a single output:

A ciphertext C, which is at least as long as the plaintext, or

an indication that the requested encryption operation could not be performed.

All of the inputs and outputs are variable-length octet strings, whose lengths obey the following restrictions:

The number of octets in the key K is between 1 and 255. For each AEAD algorithm, the length of K MUST be fixed.

For any particular value of the key, either 1) each nonce provided to distinct invocations of the Authenticated Encryption operation MUST be distinct, or 2) each and every nonce MUST be zero-length. If zero-length nonces are used with a particular key, then each and every nonce used with that key MUST have a length of zero. Otherwise, the number of octets in the nonce SHOULD be twelve (12). Nonces with different lengths MAY be used with a particular key. Some algorithms cannot be used with zero-length nonces, but others can; see Section 4. Applications that conform to the recommended nonce length will avoid having to construct nonces with different lengths, depending on the algorithm that is in use. This guidance helps to keep algorithm-specific logic out of applications.

The number of octets in the plaintext P MAY be zero.

The number of octets in the associated data A MAY be zero.

The number of octets in the ciphertext C MAY be zero.

This specification does not put a maximum length on the nonce, the plaintext, the ciphertext, or the additional authenticated data. However, a particular AEAD algorithm MAY further restrict the lengths of those inputs and outputs. A particular AEAD implementation MAY further restrict the lengths of its inputs and outputs. If a particular implementation of an AEAD algorithm is requested to process an input that is outside the range of admissible lengths, or an input that is outside the range of lengths supported by that implementation, it MUST return an error code and it MUST NOT output any other information. In particular, partially encrypted or partially decrypted data MUST NOT be returned.

Both confidentiality and message authentication are provided on the plaintext P. When the length of P is zero, the AEAD algorithm acts as a Message Authentication Code on the input A.

The associated data A is used to protect information that needs to be authenticated, but does not need to be kept confidential. When using an AEAD to secure a network protocol, for example, this input could include addresses, ports, sequence numbers, protocol version numbers, and other fields that indicate how the plaintext or ciphertext should be handled, forwarded, or processed. In many situations, it is desirable to authenticate these fields, though they must be left in the clear to allow the network or system to function properly. When this data is included in the input A, authentication is provided without copying the data into the plaintext.

The secret key K MUST NOT be included in any of the other inputs (N, P, and A). (This restriction does not mean that the values of those inputs must be checked to ensure that they do not include substrings that match the key; instead, it means that the key must not be explicitly copied into those inputs.)

The nonce is authenticated internally to the algorithm, and it is not necessary to include it in the AD input. The nonce MAY be included in P or A if it is convenient to the application.

The nonce MAY be stored or transported with the ciphertext, or it MAY be reconstructed immediately prior to the authenticated decryption operation. It is sufficient to provide the decryption module with enough information to allow it to construct the nonce. (For example, a system could use a nonce consisting of a sequence number in a particular format, in which case it could be inferred from the order of the ciphertexts.) Because the authenticated decryption process detects incorrect nonce values, no security failure will result if a nonce is incorrectly reconstructed and fed into an authenticated decryption operation. Any nonce reconstruction method will need to take into account the possibility of loss or reorder of ciphertexts between the encryption and decryption processes.

Applications MUST NOT assume any particular structure or formatting of the ciphertext.

2.2. Authenticated Decryption

The authenticated decryption operation has four inputs: K, N, A, and C, as defined above. It has only a single output, either a plaintext value P or a special symbol FAIL that indicates that the inputs are not authentic. A ciphertext C, a nonce N, and associated data A are authentic for key K when C is generated by the encrypt operation with inputs K, N, P, and A, for some values of N, P, and A. The authenticated decrypt operation will, with high probability, return FAIL whenever the inputs N, P, and A were crafted by a nonce-respecting adversary that does not know the secret key (assuming that the AEAD algorithm is secure).

2.3. Data Formatting

This document does not specify any particular encoding for the AEAD inputs and outputs, since the encoding does not affect the security services provided by an AEAD algorithm.

When choosing the format of application data, an application SHOULD position the ciphertext C so that it appears after any other data that is needed to construct the other inputs to the authenticated decryption operation. For instance, if the nonce and ciphertext both appear in a packet, the former value should precede the latter. This rule facilitates efficient and simple hardware implementations of AEAD algorithms.

3. Guidance on the Use of AEAD Algorithms

This section provides advice that must be followed in order to use an AEAD algorithm securely.

If an application is unable to meet the uniqueness requirement on nonce generation, then it MUST use a zero-length nonce. Randomized or stateful algorithms, which are defined below, are suitable for use with such applications. Otherwise, an application SHOULD use nonces with a length of twelve octets. Since algorithms are encouraged to support that length, applications should use that length to aid interoperability.

3.1. Requirements on Nonce Generation

It is essential for security that the nonces be constructed in a manner that respects the requirement that each nonce value be distinct for each invocation of the authenticated encryption operation, for any fixed value of the key. In this section, we call attention to some consequences of this requirement in different scenarios.

When there are multiple devices performing encryption using a single key, those devices must coordinate to ensure that the nonces are unique. A simple way to do this is to use a nonce format that contains a field that is distinct for each one of the devices, as described in Section 3.2. Note that there is no need to coordinate the details of the nonce format between the encrypter and the decrypter, as long the entire nonce is sent or stored with the ciphertext and is thus available to the decrypter. If the complete nonce is not available to the decrypter, then the decrypter will need to know how the nonce is structured so that it can reconstruct it. Applications SHOULD provide encryption engines with some freedom in choosing their nonces; for example, a nonce could contain both a counter and a field that is set by the encrypter but is not processed by the receiver. This freedom allows a set of encryption devices to more readily coordinate to ensure the distinctness of their nonces.

If a secret key will be used for a long period of time, e.g., across multiple reboots, then the nonce will need to be stored in non-volatile memory. In such cases, it is essential to use checkpointing of the nonce; that is, the current nonce value should be stored to provide the state information needed to resume encryption in case of unexpected failure. One simple way to provide a high assurance that a nonce value will not be used repeatedly is to wait until the encryption process receives confirmation from the storage process indicating that the succeeding nonce value has already been stored. Because this method may add significant latency, it may be desirable to store a nonce value that is several values ahead in the sequence. As an example, the nonce 100 could be stored, after which the nonces 1 through 99 could be used for encryption. The nonce value 200 could be stored at the same time that nonces 1 through 99 are being used, and so on.

Many problems with nonce reuse can be avoided by changing a key in a situation in which nonce coordination is difficult.

Each AEAD algorithm SHOULD describe what security degradation would result from an inadvertent reuse of a nonce value.

3.2. Recommended Nonce Formation

The following method to construct nonces is RECOMMENDED. The nonce is formatted as illustrated in Figure 1, with the initial octets consisting of a Fixed field, and the final octets consisting of a Counter field. For each fixed key, the length of each of these fields, and thus the length of the nonce, is fixed. Implementations SHOULD support 12-octet nonces in which the Counter field is four octets long.

```
    <----- variable ----> <----------- variable ----------->
   +---------------------+----------------------------------+
   |        Fixed        |             Counter              |
   +---------------------+----------------------------------+
```

Figure 1: Recommended nonce format

The Counter fields of successive nonces form a monotonically increasing sequence, when those fields are regarded as unsigned integers in network byte order. The length of the Counter field MUST remain constant for all nonces that are generated for a given encryption device. The Counter part SHOULD be equal to zero for the first nonce, and increment by one for each successive nonce that is generated. However, any particular Counter value MAY be skipped over, and left out of the sequence of values that are used, if it is convenient. For example, an application could choose to skip the initial Counter=0 value, and set the Counter field of the initial nonce to 1. Thus, at most 2^(8*C) nonces can be generated when the Counter field is C octets in length.

The Fixed field MUST remain constant for all nonces that are generated for a given encryption device. If different devices are performing encryption with a single key, then each distinct device MUST use a distinct Fixed field, to ensure the uniqueness of the nonces. Thus, at most 2^(8*F) distinct encrypters can share a key when the Fixed field is F octets in length.

3.2.1. Partially Implicit Nonces

In some cases, it is desirable to not transmit or store an entire nonce, but instead to reconstruct that value from contextual information immediately prior to decryption. As an example, ciphertexts could be stored in sequence on a disk, and the nonce for a particular ciphertext could be inferred from its location, as long as the rule for generating the nonces is known by the decrypter. We call the portion of the nonce that is stored or sent with the ciphertext the explicit part. We call the portion of the nonce that is inferred the implicit part. When part of the nonce is implicit, the following specialization of the above format is RECOMMENDED. The Fixed field is divided into two sub-fields: a Fixed-Common field and a Fixed-Distinct field. This format is shown in Figure 2. If different devices are performing encryption with a single key, then each distinct device MUST use a distinct Fixed-Distinct field. The Fixed-Common field is common to all nonces. The Fixed-Distinct field and the Counter field MUST be in the explicit part of the nonce. The Fixed-Common field MAY be in the implicit part of the nonce. These conventions ensure that the nonce is easy to reconstruct from the explicit data.

```
+-------------------+--------------------+---------------+
|    Fixed-Common   |   Fixed-Distinct   |    Counter    |
+-------------------+--------------------+---------------+
 <---- implicit ---> <------------ explicit ------------>
```

Figure 2: Partially implicit nonce format

The rationale for the partially implicit nonce format is as follows. This method of nonce construction incorporates the best known practice; it is used by both GCM Encapsulating Security Payload (ESP) [RFC4106] and CCM ESP [RFC4309], in which the Fixed field contains the Salt value and the lowest eight octets of the nonce are explicitly carried in the ESP packet. In GCM ESP, the Fixed field must be at least four octets long, so that it can contain the Salt value. In CCM ESP, the Fixed field must be at least three octets long for the same reason. This nonce generation method is also used by several counter mode variants including CTR ESP.

3.3. Construction of AEAD Inputs

If the AD input is constructed out of multiple data elements, then it is essential that it be unambiguously parseable into its constituent elements, without the use of any unauthenticated data in the parsing process. (In mathematical terms, the AD input must be an injective function of the data elements.) If an application constructs its AD input in such a way that there are two distinct sets of data elements that result in the same AD value, then an attacker could cause a receiver to accept a bogus set by substituting one set for the other. The requirement that the AD be uniquely parseable ensures that this attack is not possible. This requirement is trivially met if the AD is composed of fixed-width elements. If the AD contains a variable-length string, for example, this requirement can be met by also including the length of the string in the AD.

Similarly, if the plaintext is constructed out of multiple data elements, then it is essential that it be unambiguously parseable into its constituent elements, without using any unauthenticated data in the parsing process. Note that data included in the AD may be used when parsing the plaintext, though of course since the AD is not encrypted there is a potential loss of confidentiality when information about the plaintext is included in the AD.

3.4. Example Usage

To make use of an AEAD algorithm, an application must define how the encryption algorithm's inputs are defined in terms of application data, and how the ciphertext and nonce are conveyed. The clearest way to do this is to express each input in terms of the data that form it, then to express the application data in terms of the outputs of the AEAD encryption operation.

For example, AES-GCM ESP [RFC4106] can be expressed as follows. The AEAD inputs are

```
P = RestOfPayloadData || TFCpadding || Padding || PadLength ||
NextHeader

N = Salt || IV

A = SPI || SequenceNumber
```

where the symbol "||" denotes the concatenation operation, and the fields RestOfPayloadData, TFCpadding, Padding, PadLength, NextHeader, SPI, and SequenceNumber are as defined in [RFC4303], and the fields Salt and IV are as defined in [RFC4106]. The field RestOfPayloadData contains the plaintext data that is described by the NextHeader field, and no other data. (Recall that the PayloadData field contains both the IV and the RestOfPayloadData; see Figure 2 of [RFC4303] for an illustration.)

The format of the ESP packet can be expressed as

    ESP = SPI || SequenceNumber || IV || C

where C is the AEAD ciphertext (which in this case incorporates the authentication tag). Please note that here we have not described the use of the ESP Extended Sequence Number.

4. Requirements on AEAD Algorithm Specifications

Each AEAD algorithm MUST only accept keys with a fixed key length K_LEN, and MUST NOT require any particular data format for the keys provided as input. An algorithm that requires such structure (e.g., one with subkeys in a particular parity-check format) will need to provide it internally.

Each AEAD algorithm MUST accept any plaintext with a length between zero and P_MAX octets, inclusive, where the value P_MAX is specific to that algorithm. The value of P_MAX MUST be larger than zero, and SHOULD be at least 65,536 (2^16) octets. This size is a typical upper limit for network data packets. Other applications may use even larger values of P_MAX, so it is desirable for general-purpose algorithms to support higher values.

Each AEAD algorithm MUST accept any associated data with a length between zero and A_MAX octets, inclusive, where the value A_MAX is specific to that algorithm. The value of A_MAX MUST be larger than zero, and SHOULD be at least 65,536 (2^16) octets. Other applications may use even larger values of A_MAX, so it is desirable for general-purpose algorithms to support higher values.

Each AEAD algorithm MUST accept any nonce with a length between N_MIN and N_MAX octets, inclusive, where the values of N_MIN and N_MAX are specific to that algorithm. The values of N_MAX and N_MIN MAY be equal. Each algorithm SHOULD accept a nonce with a length of twelve (12) octets. Randomized or stateful algorithms, which are described below, MAY have an N_MAX value of zero.

An AEAD algorithm MAY structure its ciphertext output in any way; for example, the ciphertext can incorporate an authentication tag. Each algorithm SHOULD choose a structure that is amenable to efficient processing.

An Authenticated Encryption algorithm MAY incorporate or make use of a random source, e.g., for the generation of an internal initialization vector that is incorporated into the ciphertext output. An AEAD algorithm of this sort is called randomized; though note that only encryption is random, and decryption is always deterministic. A randomized algorithm MAY have a value of N_MAX that is equal to zero.

An Authenticated Encryption algorithm MAY incorporate internal state information that is maintained between invocations of the encrypt operation, e.g., to allow for the construction of distinct values that are used as internal nonces by the algorithm. An AEAD algorithm of this sort is called stateful. This method could be used by an algorithm to provide good security even when the application inputs zero-length nonces. A stateful algorithm MAY have a value of N_MAX that is equal to zero.

The specification of an AEAD algorithm MUST include the values of K_LEN, P_MAX, A_MAX, N_MIN, and N_MAX defined above. Additionally, it MUST specify the number of octets in the largest possible ciphertext, which we denote C_MAX.

Each AEAD algorithm MUST provide a description relating the length of the plaintext to that of the ciphertext. This relation MUST NOT depend on external parameters, such as an authentication strength parameter (e.g., authentication tag length). That sort of dependence would complicate the use of the algorithm by creating a situation in which the information from the AEAD registry was not sufficient to ensure interoperability.

EACH AEAD algorithm specification SHOULD describe what security degradation would result from an inadvertent reuse of a nonce value.

Each AEAD algorithm specification SHOULD provide a reference to a detailed security analysis. This document does not specify a particular security model, because several different models have been used in the literature. The security analysis SHOULD define or reference a security model.

An algorithm that is randomized or stateful, as defined above, SHOULD describe itself using those terms.

5. AEAD Algorithms

This section defines four AEAD algorithms; two are based on AES GCM, two are based on AES CCM. Each pair includes an algorithm with a key size of 128 bits and one with a key size of 256 bits.

5.1. AEAD_AES_128_GCM

The AEAD_AES_128_GCM authenticated encryption algorithm works as specified in [GCM], using AES-128 as the block cipher, by providing the key, nonce, and plaintext, and associated data to that mode of operation. An authentication tag with a length of 16 octets (128 bits) is used. The AEAD_AES_128_GCM ciphertext is formed by appending the authentication tag provided as an output to the GCM encryption operation to the ciphertext that is output by that operation. Test cases are provided in the appendix of [GCM]. The input and output lengths are as follows:

K_LEN is 16 octets,

P_MAX is 2^36 - 31 octets,

A_MAX is 2^61 - 1 octets,

N_MIN and N_MAX are both 12 octets, and

C_MAX is 2^36 - 15 octets.

An AEAD_AES_128_GCM ciphertext is exactly 16 octets longer than its corresponding plaintext.

A security analysis of GCM is available in [MV04].

5.1.1. Nonce Reuse

The inadvertent reuse of the same nonce by two invocations of the GCM encryption operation, with the same key, but with distinct plaintext values, undermines the confidentiality of the plaintexts protected in those two invocations, and undermines all of the authenticity and integrity protection provided by that key. For this reason, GCM should only be used whenever nonce uniqueness can be provided with assurance. The design feature that GCM uses to achieve minimal latency causes the vulnerabilities on the subsequent uses of the key. Note that it is acceptable to input the same nonce value multiple times to the decryption operation.

The security consequences are quite serious if an attacker observes two ciphertexts that were created using the same nonce and key values, unless the plaintext and AD values in both invocations of the encrypt operation were identical. First, a loss of confidentiality ensues because he will be able to reconstruct the bitwise exclusive-or of the two plaintext values. Second, a loss of integrity ensues because the attacker will be able to recover the internal hash key used to provide data integrity. Knowledge of this key makes subsequent forgeries trivial.

5.2. AEAD_AES_256_GCM

This algorithm is identical to AEAD_AES_128_GCM, but with the following differences:

K_LEN is 32 octets, instead of 16 octets, and

AES-256 GCM is used instead of AES-128 GCM.

5.3. AEAD_AES_128_CCM

The AEAD_AES_128_CCM authenticated encryption algorithm works as specified in [CCM], using AES-128 as the block cipher, by providing the key, nonce, associated data, and plaintext to that mode of operation. The formatting and counter generation function are as specified in Appendix A of that reference, and the values of the parameters identified in that appendix are as follows:

the nonce length n is 12,

the tag length t is 16, and

the value of q is 3.

An authentication tag with a length of 16 octets (128 bits) is used. The AEAD_AES_128_CCM ciphertext is formed by appending the authentication tag provided as an output to the CCM encryption operation to the ciphertext that is output by that operation. Test cases are provided in [CCM]. The input and output lengths are as follows:

K_LEN is 16 octets,

P_MAX is 2^24 - 1 octets,

A_MAX is 2^64 - 1 octets,

N_MIN and N_MAX are both 12 octets, and

C_MAX is 2^24 + 15 octets.

An AEAD_AES_128_CCM ciphertext is exactly 16 octets longer than its corresponding plaintext.

A security analysis of AES CCM is available in [J02].

5.3.1. Nonce Reuse

Inadvertent reuse of the same nonce by two invocations of the CCM encryption operation, with the same key, undermines the security for the messages processed with those invocations. A loss of confidentiality ensues because an adversary will be able to reconstruct the bitwise exclusive-or of the two plaintext values.

5.4. AEAD_AES_256_CCM

This algorithm is identical to AEAD_AES_128_CCM, but with the following differences:

K_LEN is 32 octets, instead of 16, and

AES-256 CCM is used instead of AES-128 CCM.

6. IANA Considerations

The Internet Assigned Numbers Authority (IANA) has defined the "AEAD Registry" described below. An algorithm designer MAY register an algorithm in order to facilitate its use. Additions to the AEAD Registry require that a specification be documented in an RFC or another permanent and readily available reference, in sufficient detail that interoperability between independent implementations is possible. Each entry in the registry contains the following elements:

a short name, such as "AEAD_AES_128_GCM", that starts with the string "AEAD",

a positive number, and

a reference to a specification that completely defines an AEAD algorithm and provides test cases that can be used to verify the correctness of an implementation.

Requests to add an entry to the registry MUST include the name and the reference. The number is assigned by IANA. These number assignments SHOULD use the smallest available positive number. Submitters SHOULD have their requests reviewed by the IRTF Crypto

Forum Research Group (CFRG) at cfrg@ietf.org. Interested applicants that are unfamiliar with IANA processes should visit http://www.iana.org.

The numbers between 32,768 (binary 1000000000000000) and 65,535 (binary 1111111111111111) inclusive, will not be assigned by IANA, and are reserved for private use; no attempt will be made to prevent multiple sites from using the same value in different (and incompatible) ways [RFC2434].

IANA has added the following entries to the AEAD Registry:

Name	Reference	Numeric Identifier
AEAD_AES_128_GCM	Section 5.1	1
AEAD_AES_256_GCM	Section 5.2	2
AEAD_AES_128_CCM	Section 5.3	3
AEAD_AES_256_CCM	Section 5.4	4

An IANA registration of an AEAD does not constitute an endorsement of that algorithm or its security.

7. Other Considerations

Directly testing a randomized AEAD encryption algorithm using test cases with fixed inputs and outputs is not possible, since the encryption process is non-deterministic. However, it is possible to test a randomized AEAD algorithm using the following technique. The authenticated decryption algorithm is deterministic, and it can be directly tested. The authenticated encryption algorithm can be tested by encrypting a plaintext, decrypting the resulting ciphertext, and comparing the original plaintext to the post-decryption plaintext. Combining both of these tests covers both the encryption and decryption algorithms.

The AEAD algorithms selected reflect those that have been already adopted by standards. It is an open question as to what other AEAD algorithms should be added. Many variations on basic algorithms are possible, each with its own advantages. While it is desirable to admit any algorithms that are found to be useful in practice, it is also desirable to limit the total number of registered algorithms. The current specification requires that a registered algorithm provide a complete specification and a set of validation data; it is hoped that these prerequisites set the admission criteria appropriately.

It may be desirable to define an AEAD algorithm that uses the generic composition with the encrypt-then-MAC method [BN00], combining a common encryption algorithm, such as CBC [MODES], with a common message authentication code, such as HMAC-SHA1 [RFC2104] or AES CMAC [CMAC]. An AEAD algorithm of this sort would reflect the best current practice, and might be more easily supported by crypto modules that lack support for other AEAD algorithms.

8. Security Considerations

This document describes authenticated encryption algorithms, and provides guidance on their use. While these algorithms make it easier, in some ways, to design a cryptographic application, it should be borne in mind that strong cryptographic security is difficult to achieve. While AEAD algorithms are quite useful, they do nothing to address the issues of key generation [RFC4086] and key management [RFC4107].

AEAD algorithms that rely on distinct nonces may be inappropriate for some applications or for some scenarios. Application designers should understand the requirements outlined in Section 3.1.

A software implementation of the AEAD encryption operation in a Virtual Machine (VM) environment could inadvertently reuse a nonce due to a "rollback" of the VM to an earlier state [GR05]. Applications are encouraged to document potential issues to help the user of the application and the VM avoid unintentional mistakes of this sort. The possibility exists that an attacker can cause a VM rollback; threats and mitigations in that scenario are an area of active research. For perspective, we note that an attacker who can trigger such a rollback may have already succeeded in subverting the security of the system, e.g., by causing an accounting error.

An IANA registration of an AEAD algorithm MUST NOT be regarded as an endorsement of its security. Furthermore, the perceived security level of an algorithm can degrade over time, due to cryptanalytic advances or to "Moore's Law", that is, the diminishing cost of computational resources over time.

9. Acknowledgments

Many reviewers provided valuable comments on earlier drafts of this document. Some fruitful discussions took place on the email list of the Crypto Forum Research Group in 2006.

10. References

10.1. Normative References

[CCM] Dworkin, M., "NIST Special Publication 800-38C: The CCM Mode for Authentication and Confidentiality", U.S. National Institute of Standards and Technology, <http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf>.

[GCM] Dworkin, M., "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.", U.S. National Institute of Standards and Technology, November 2007, <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>.

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

10.2. Informative References

[BN00] Bellare, M. and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm", Proceedings of ASIACRYPT 2000, Springer-Verlag, LNCS 1976, pp. 531-545, 2002.

[BOYD] Boyd, C. and A. Mathuria, "Protocols for Authentication and Key Establishment", Springer 2003.

[CMAC] "NIST Special Publication 800-38B", <http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf>.

[EEM04] Bellare, M., Namprempre, C., and T. Kohno, "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm", ACM Transactions on Information and System Security, <http://www-cse.ucsd.edu/users/tkohno/papers/TISSEC04/>.

[GR05] Garfinkel, T. and M. Rosenblum, "When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments", Proceedings of the 10th Workshop on Hot Topics in Operating Systems, <http://www.stanford.edu/~talg/papers/HOTOS05/virtual-harder-hotos05.pdf>.

[J02] Jonsson, J., "On the Security of CTR + CBC-MAC", Proceedings of the 9th Annual Workshop on Selected Areas on Cryptography, 2002, <http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ccm/ccm-ad1.pdf>.

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

[MV04] McGrew, D. and J. Viega, "The Security and Performance of the Galois/Counter Mode (GCM)", Proceedings of INDOCRYPT '04, December 2004, <http://eprint.iacr.org/2004/193>.

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

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

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

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

[RFC4106] Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, June 2005.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005.

Author's Address

David A. McGrew
Cisco Systems, Inc.
510 McCarthy Blvd.
Milpitas, CA 95035
US

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