Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

D. McGrew
J. Foley
Cisco Systems
February 14, 2014

Authenticated Encryption with Replay prOtection (AERO)
draft-mcgrew-aero-01.txt

Abstract

This document describes Authenticated Encryption with Replay prOtection (AERO), a cryptographic technique that provides all of the essential security services needed for communication security. AERO offers several advantages over other methods: it has more compact messages, provides stronger misuse resistance, avoids the need to manage implicit state, and is simpler to use. This document defines a particular AERO algorithm as well as a registry for such algorithms.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes Authenticated Encryption with Replay prOtection (AERO), a cryptographic technique that provides all of the essential security services needed for communication security.

AERO can be considered a stateful and self-synchronizing authenticated encryption method that provides protection from replay attacks as a side benefit. Replay protection and authentication are provided through a single mechanism, in which a sequence number is encrypted along with a plaintext message. If the ciphertext message is not tampered with, then this sequence number will be recovered by the decrypter, and verified to be valid. If the ciphertext message is a replay of an earlier message, then the decrypter will detect this fact from the recovered sequence number. If the ciphertext message is tampered with, or is crafted as a forgery, this fact will be apparent to the decrypter because the value that should be a valid sequence is instead a pseudorandom value.

AERO makes use of an underlying stateless encryption algorithm. This note defines a set of AERO algorithms that are based on an underlying Wide Pseudo-Random Permutation (WPRP). AERO is not, by itself, a block cipher mode of operation.

This note is structured as follows. Section 1 introduces the basic idea of AERO, describes the conventions and notations used in this note, and provides a glossary of acronyms and variables. Section 2 provides background on the problems that AERO solves. The interface and implementation are presented in Section 3 and Section 4, respectively. The underlying cryptographic primitives are described in Section 5. Section 6 describes synchronization between encrypters and decrypters, and Section 7 describes usage. A rationale for the design details is offered in Section 8, and test considerations and test cases are presented in Section 9. IANA considerations and security considerations follow in Section 10 and Section 11, respectively.

1.1. 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].

1.2. Glossary

This section describes the symbols and acronyms used in this document.

A - Associated Data. An unencrypted value whose integrity and authenticity is to be protected.

AERO - Authenticated Encryption with Replay prOtection. A cryptographic technique that combines the three security services essential to communication security.

C - Ciphertext. An encrypted value that corresponds to a plaintext.

FAIL - a value returned by the AERO decryption operation to indicate that a ciphertext message is not valid; it could be either a replay or a forgery attempt.

K - Key. A secret key that is shared between the sender(s) and receiver(s).

M - bitmask. A bit vector used by the AERO decryption algorithm that holds information about the sequence numbers that have already been accepted. If a message with the sequence number X has been accepted and the current sequence number S > X, then M[S-X] = 1; otherwise M[S-X] = 0.

P - Plaintext. An unencrypted value whose confidentiality, integrity and authenticity is to be protected.

PMIN - minimum number of bytes of plaintext for a stateless encryption algorithm. PMIN is a parameter of the stateless encryption algorithm, which is used in the padding logic.

R - the last rejected candidate sequence number that is greater than the current sequence number S. A T-bit unsigned integer maintained by the AERO decryption algorithm.

S - the current sequence number. A T-bit unsigned integer maintained by the AERO encryption algorithm, where it represents the sequence number of the next message to be encrypted, and by the AERO decryption algorithm, where it represents the last candidate sequence number that was accepted.

T - The number of bits in the sequence number that is internal to AERO encryption and decryption.

W - Reorder tolerance parameter. A parameter used in AERO decryption (but not encryption) that determines the amount of message loss or reorder that the algorithm will tolerate. This value is 64 by default, but other values may be used as well.

V - Resynchronization parameter. A parameter that determines the amount of message loss or reorder that the AERO decryption algorithm will tolerate during resynchronization. This value is 8 by default, but other values may be used as well.

Z - the candidate sequence number. A value used in AERO decryption that may be a sequence number, or alternately may be data that resulted from the actions of an attacker.

1.3. Data types and notation

An octet string is an ordered sequence of eight-bit values. Note that these strings are not character strings, and issues such as character representation are out of scope for this note.

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

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

Conversion between unsigned integers and octet strings is done as follows. The Integer to String (I2S) algorithm takes as input an unsigned integer U, and converts it to network byte order (that is, a big-endian representation, in which the most significant octet is the initial one), and then returns the resulting octet string. The String to Integer (S2I) algorithm takes as input an octet string S, considers it as an integer in network byte order, and then converts it into the internal representation for unsigned integers used by the system.

2. Background

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.

AEAD is used in many communication security protocols, such as ESP, TLS, DTLS, IKE, and SRTP. These protocols also incorporate replay protection using an sequence number that is carried in the packet, which is authenticated because it is part of the associated data. This security service enables the receiver of a message to detect when a received message is a duplicate of a previously received message.

In order to minimize the data overhead, some protocols (such as SRTP and ESP) have the most significant part of the sequence number (that is, the high order bits) be implicit state, which is not sent on the wire. The receiver constructs an estimate of the sequence number from the data on the wire and its current state.

Authenticated Encryption with Replay prOtection (AERO) extends AEAD so that it provides a replay protection service. It also enables a receiver to correctly sequence all of the messages that it receives into the same order that they were sent. A replay protection service has a reorder tolerance parameter W; if a receiver processes a message that is more than W messages earlier than a previously processed message, then the service will reject the message even if it is not a replay.

2.1. Problems addressed by AERO

Many communication security protocols are operated in environments where bandwidth is a scarce resource, and thus they seek to minimize the data overhead, that is, the number of additional bytes that must be communicated in order to add confidentiality, authenticity, and replay protection to each message. To authenticate a message, lengthening it is unavoidable. However, traditional encryption and replay protection techniques add avoidable overhead. Encryption methods methods such as AES-CBC encryption [SP800-38] have an average overhead of 24 bytes, and many uses of CTR, GCM [GCM], and CCM [CCM] have eight bytes of overhead because a nonce is carried in each message. In addition to that overhead, replay protection typically adds its own overhead; the use of a four-byte sequence number, for instance, adds that number of bytes.

Some communication security protocol uses an implicit or partly implicit sequence number to reduce overhead. This practice complicates the use of that protocol whenever there are multiple receivers, since it is necessary to (securely) communicate the current value of the implicit part of the sequence number to a new receiver at the time that the receiver joins the session. For instance, [RFC4771] furnishes an example of the complications introduced by the need to manage implicit state.

Several cryptographic algorithms require that a distinct nonce is input into each invocation of the encryption algorithm. This requirement is difficult to satisfy if there are multiple senders, or multiple encryption units, using the same secret key. Algorithms that require nonces introduce significant complexity into the systems that use them, as is revealed by the survey of usage of and issues with nonces in Internet protocols [IV-GEN]. One of the cannons of computer security is that "input is evil", that is, data that is accepted into a security module might be manipulated by an attacker and thus should not be trusted until it has been validated. A cryptographic algorithm that accepts a nonce and requires it to be distinct introduces an opportunity for an attacker to defeat security through "evil" influence of that input.

Several cryptographic algorithms that require a distinct nonce as an input have a serious security degradation if there is nonce misuse (that is, if a system failure results in two or more identical nonces being used for the same key). CTR, CCM, and GCM all have a loss of confidentiality for each message that is associated with a misused nonce. GCM, GMAC [GCM], UMAC [RFC4418], and POLY1305 leak their authentication keys after nonce misuse. In scenarios in which it is not possible to provide high assurance that nonces will not be misused, these algorithms may not be appropriate.

Both dedicated authenticated encryption algorithms, and the traditional method of combination of separate authentication and encryption algorithms, are often complex to use. Nonces and initialization vectors, especially, are often a source of confusion for implementers and application designers.

AERO solves these problems by using a technique that combines message authentication with replay protection, thus eliminating the data overhead associated with replay protection and any need for using implicit state. AERO does not use a nonce as input, which addresses multiple issues: it reduces data overhead, it avoids all of the problems of coordinating nonces and the potential problems of nonce misuse, and it simplifies the jobs of the designers and implementers of applications and protocols. AERO can easily be used in multiple-sender scenarios, because no coordination of nonces is needed, and in

multiple-receiver scenarios, because no coordination of implicit state is needed.

3. Interface

An AERO algorithm has four operations: encryption, decryption, encryption initialization, and decryption initialization. The inputs and outputs of these algorithms are defined below in terms of octet strings. An octet string is an ordered sequence of eight-bit values.

The terms "AERO encryption" and "AERO decryption" refer to algorithms that provide authenticated encryption with replay protection. For brevity, these algorithms are simply called "encryption" and "decryption" when AERO is clear from the context.

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

3.1. Encryption Initialization

The encryption initialization operation has one input:

A secret key K, which MUST be generated in a way that is uniformly random or pseudorandom. This input is an octet string. The number of octets in K is fixed for each AERO algorithm, and is between 1 and 255.

The operation has a single output, an AERO encryption context.

3.2. Encryption

The AERO encryption operation has three inputs, each of which is an octet string:

An AERO encryption context.

A plaintext P, which contains the data to be encrypted and authenticated. The number of octets in P MAY be zero.

The associated data A, which contains the data to be authenticated, but not encrypted. The number of octets in A MAY be zero.

There is a single output:

A ciphertext C, which is an octet string that is at least as long as the plaintext.

A plaintext P and its associated data A is sometimes denoted together as (A, P), and that pair of elements is called a plaintext message. Similarly, a ciphertext C and its associated data A is denoted together as (A, C), and is called a ciphertext message.

3.3. Decryption Initialization

The decryption initialization operation has one or two inputs:

A secret key K, which MUST match the secret key used in the corresponding encryption initialization. This input is an octet string.

The reorder tolerance parameter W, a non-negative integer which indicates the amount of reorder of messages that must be tolerated. This input is optional; when it is not provided as an input, the default value of 64 SHOULD be used.

The operation has a single output, an AERO decryption context.

3.4. Decryption

The AERO decryption operation has three inputs, each of which is an octet string:

An AERO decryption context.

An octet string C.

The associated data A, which contains the data to be authenticated, but not encrypted. The length of A MAY be zero.

The output is one of these two alternatives:

An octet string P and an unsigned integer indicating the order in which P appeared in the sequence of plaintexts input to the send algorithm, or

The symbol FAIL, which indicates that either the value C was not output by the send algorithm, or that (A, C) matches a previously accepted pair of values.

3.5. Multiple senders and multiple receivers

A communications security protocol may have multiple senders (encrypters) as well as multiple receivers (decrypters). A single AERO key MAY be used by multiple senders, in which case each sender MUST use a distinct AERO encryption context.

When multiple AERO encryption contexts are in use for a single key, a receiver must use a distinct AERO decryption context to process the messages encrypted with each distinct encryption context. That is, the receiver maintains a decryption context for each sender. This matches the conventional practice of communication security protocols, in which the receiver maintains a distinct security association for each sender.

When processing a protected message, a receiver needs to select the appropriate AERO context to use in processing that message. Thus, each protected message must be associated with some indication of its sender. We call a field that contains this indication a Sender IDentifier (SID).

When there are multiple senders, the values of the associated data input to the encryption algorithm for different senders SHOULD be distinct. This practice ensures that no two messages are identical, and that fact ensures that the AERO system provides strong confidentiality. An easy way to ensure the distinctness of Associated Data values is to include an SID in those values. For example, this requirement is met if the associated data includes a message header, and that header contains a field that identifies the sender of the message. Alternatively, the value of the plaintext can be distinct for each sender, for instance, if each plaintext includes an SID.

If senders accidentally use the same SID value, the security of the session will degrade, but not catastrophically. See section Section 11.4 for more details.

3.6. AEAD interface

AERO can be used through the IETF standard interface for authenticated encryption with associated data [RFC5116]. As AERO does not use a nonce or initialization vector, N_MAX = N_MIN = 0, and the application using the AEAD interface need not provide a nonce. Note that most of the usage guidance in RFC 5116 describes stipulations and cautions on the use of the nonces, so by eliminating the need for a nonce, AERO is simplifying the use of this standard.

4. Implementation

AERO makes use of a technique that combines message authentication with replay protection, in which the sender maintains a sequence number, and the encrypted value of this sequence number is communicated to the receiver. When the receiver decrypts a message, it parses out the data that would be equal to a sequence number if the message was sent by a legitimate sender, and verifies that this data is sensible. This data is called a candidate sequence number, and is denoted as Z; the verification process is detailed below. If the data is not sensible, then the decryption algorithm rejects the message. The technique provides authenticity because Z is a pseudorandom function of both the ciphertext and associated data, and in a forgery attempt, Z will be rejected with overwhelming probability.

AERO is a stateful encryption method. It makes use of an underlying stateless encryption method; the interface to this method is defined below. This interface allows an AERO implementation to separate out the authentication and replay protection logic from the cryptographic algorithms that provide pseudorandomness. The interface also facilitates the use of different cryptographic techniques, thus allowing algorithm agility.

The stateless encryption algorithm takes as input a secret key K, a plaintext P, a T-bit unsigned integer S, and an associated data A, and returns a ciphertext C. Here K, A, P, and C are all octet strings.

The stateless decryption algorithm takes as input a secret key K, a ciphertext C, and an associated data A, and returns a plaintext P and a T-bit unsigned integer U.

The stateless encryption algorithm may not be able to encrypt plaintexts that are shorter than a certain minimum value, so plaintexts are lengthend with the "pad" routine prior to encryption, and padding is removed after decryption with the "strip" routine. We denote as PMIN the minimum number of bytes in a plaintext that the stateful encryption algorithm can accept.

Note that the stateless encryption provides only confidentiality, and not authenticity.

4.1. Contexts

An encryption context includes a T-bit unsigned integer that holds the sequence number S.

A decryption context includes a W-bit bitmask M and two T-bit unsigned integers S and R, which record the highest candidate sequence number that has been accepted, and the last candidate sequence number that has been rejected, respectively.

4.2. Encryption Initialization

The encryption initialization operation does the following:

1. S is initialized to zero.

2. The value of the secret key K is stored in the context.

4.3. Encryption

To encrypt a message (A, P), the following steps are performed:

1. The current sequence number S is read from the context. If S is greater than 2^T - 1, then the context cannot be used to encrypt any messages, so an indication of this error state is returned and the encryption processing halts. Otherwise, processing continues as follows.

2. U is set to the value of a S converted from an unsigned integer to an octet string using the Integer to String (I2S) routine described in Section 1.3.

3. If PMIN * 8 > T, then P is lengthened by appending a padding string PS to it, to ensure that L = len(P) + len(PS) + T is at least 128. The value of PS is as follows:

```
PS = 00                                if len(P) + T >= 120,
PS = 0101                              if len(P) + T = 112,
PS = 020202                            if len(P) + T = 104,
PS = 03030303                          if len(P) + T = 96,
PS = 0404040404                        if len(P) + T = 88,
PS = 050505050505                      if len(P) + T = 80,
PS = 06060606060606                    if len(P) + T = 72,
PS = 0707070707070707                  if len(P) + T = 64,
PS = 080808080808080808                if len(P) + T = 56,
PS = 09090909090909090909              if len(P) + T = 48,
PS = 0A0A0A0A0A0A0A0A0A0A0A            if len(P) + T = 40,
PS = 0B0B0B0B0B0B0B0B0B0B0B0B0B         if len(P) + T = 32,
PS = 0C0C0C0C0C0C0C0C0C0C0C0C0C0C       if len(P) + T = 24,
PS = 0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D     if len(P) + T = 16,
PS = 0E0E0E0E0E0E0E0E0E0E0E0E0E0E0E0E0E   if len(P) + T = 8.
```

Note that padding MUST be appended to the plaintext if

PMIN * 8 > T, and otherwise padding MUST NOT be used. As the parameter T is fixed for each particular key, either all of the messages processed by a particular key will use padding, or all will not.

4. The stateless encryption algorithm is applied to U, P, and A, using the secret key from the context.

5. The sequence number in the context is incremented by one.

6. The ciphertext C resulting from the stateless encryption algorithm is returned.

```
                     A      P
                +----|------|------------------+
                |    |      |                  |
                |    |      |       +------+   |
                |    |      |       |      |   |
                |    |      |       v   +----+ |
                |    |      |       S ->| +1 | |
                |    |      |       |   +----+ |
                |    |      v       v          |
                |    |   +-----+ +-----+       |     AERO
                |    |   | pad | | I2S |       |  encryption
                |    |   +-----+ +-----+       |
                |    |      |       |          |
                |    v      v       v          |
                |  +------------------+        |
            K ---->|    stateless     |        |
                |  |    encryption    |        |
                |  +------------------+        |
                |           |                  |
                +-----------|------------------+
                            v
                            C
```

Figure 1: An illustration of the AERO encryption process.

4.4. Decryption Initialization

During the decryption initialization process,

1. S is initialized to the reorder tolerance parameter W,

2. R is initialized to 2^T - 1, and

3. the bitmask M is initialized to the all-zero value.

4.5. Decryption

To decrypt an encrypted message (A, C),

1. The stateless decryption algorithm is applied to A and C, using the secret key K in the context, returning a candidate sequence number Z, which is a T-bit unsigned integer, and a plaintext Q, which is an octet string.

2. If PMIN * 8 > T, then padding is removed from Q as follows. B is set to the value of the last octet of Q, converted to an 8-bit unsigned integer. If B > (128 - T)/8, then the FAIL symbol is returned, and processing halts. Otherwise, the octet string P is set to all but the final B + 1 octets of Q.

3. Z is converted from an octet string to an unsigned integer using the S2I routine defined in Section 1.3.

4. Z is processed as follows; see Figure 3 for an illustration.

    1. If Z is between 0 and S-W, inclusive, then Z is rejected.

    2. If Z is between S-W+1 and S, inclusive, then the bitmask M is checked. If M[S-Z] = 0, then Z is accepted, M[S-Z] is set to 1, and P is returned as the plaintext. If M[S-Z] = 1, then Z is rejected.

    3. If Z is between S+1 and S+W, inclusive, then Z is accepted, S is set to Z, and P is returned as the plaintext. The bitmask is shifted by Z-S (in the direction of higher indicies).

    4. If Z is between S+W+1 and R, inclusive, then Z is rejected and R is set to Z.

    5. If Z is between R+1 and R+V, inclusive, then Z is accepted, S is set to Z, the bitmask M is set to the all-zero value, and P is returned as the plaintext.

    6. If Z is between R+V+1 and 2^T+1, inclusive, then Z is rejected and R is set to Z.

5. When Z is rejected, then AERO decryption MUST return the FAIL symbol, which indicates that either the message was a replay or a forgery attempt.

```
                         A       C
                    +------|-------|--------------+
                    |      v       v              |
                    |  +-------------+           |
                K ---->|  stateless  |           |
                    |  |  decryption |           |
                    |  +-------------+           |
                    |      |       |              |
                    |      v       v              |
                    |   +-----+ +-----+           |
                    |   |strip| | S2I |           |
                    |   +-----+ +-----+           |     AERO
                    |      |       |              |  decryption
                    |      v       v              |
                    |      P       Z              |
                    |              |              |
                    |              v              |
                    |       +-------------+       |
                    |   S-->|    check    |<--R   |
                    |   ^   +-------------+   ^   |
                    |   |    |           |    |   |
                    |  +----------+  +----------+ |
                    |  |  accept  |  |  reject  | |
                    |  +----------+  +----------+ |
                    |        |           |        |
                    +--------|-----------|--------+
                             v           v
                             P   (or)   FAIL
```

Figure 2: An illustration of the AERO decryption process.

```
            check         reject                 reject
           bitmask          and                    and
              |          set R to Z             set R to Z
   reject    |  accept      |         accept       |
     |       |    |         |           |          |
     v       v    v         v           v          v
  +---------+----+----+----------------+---+----------------+-> Z
  |         |    |    |                |   |                |
  0        S-W   S   S+W               R  R+V             2^T-1
```

Figure 3: An illustration of how the candidate sequence number Z is processed during AERO decryption. The possible values of Z are shown on a number line from 0 to 2^T-1. S and R are taken from the decryption context, the region into which Z falls determines how that value is processed. The acceptance region between R and R+V provides resynchronization as described in Section 6.

5. Stateless encryption algorithms

5.1. Wide PRP (WPRP)

An Pseudo-Random Permutation (PRP) is a keyed function that is both invertible and pseudorandom; that is, when the key is chosen at random, an attacker cannot distinguish it from an invertible function selected at random. For instance, a block cipher is a PRP with a fixed length (and narrow) input and output.

An encryption algorithm that accepts plaintexts that have varying lengths can also be a PRP. This type of algorithm is called a Wide Pseudo-Random Permutation (WPRP), because the inputs can be wider than those of a typical block cipher. For our purposes, the WPRP must also accept an associated data input. For each distinct value of the associated data, the WPRP must realize a distinct pseudorandom function. Informally, for each distinct value of A, the WPRP "looks like" a distinct PRP.

A WPRP is used as the underlying stateful encryption encryption method as follows. To encrypt an associated data A, a plaintext P, and a sequence number U with a secret key K, first U is appended to P, and the WPRP encryption is performed with the key K, the associated data A, and the plaintext Q = P || U resulting from the concatenation of P and U. The output of the WPRP encryption is returned as the ciphertext. This process is shown in Figure 4.

```
                 A     P    U
            +------|-----|----|-----+
            |      |     |    |     |
            |      |     v    v     |
            |      |   +--------+   |
            |      |   | append |   |
            |      |   +--------+   |  stateless
            |      |       |        |  encryption
            |      |       v        |
            |      |       Q        |
            |      |       |        |
            |      v       v        |
            |  +-----------------+  |
        K ---->| WPRP encryption |  |
            |  +-----------------+  |
            |           |           |
            +-----------|-----------+
                        v
                        C
```

Figure 4: Using a WPRP as a stateless encryption method in AERO.

To decrypt a ciphertext C with associated data A with a secret key K, first the WPRP decryption is applied to C, using the key K. Then the resulting output Q is split into two separate octet strings. The final T bits of the output is returned as U, and the initial len(Q) - T bits of Q are returned as P. This process is shown in Figure 5.

```
                        A       C
                    +------|-------|--------+
                    |      |       |        |
                    |      |       |        |
                    |      v       v        |
                    |  +-----------------+  |
                K ---->| WPRP decryption |  |
                    |  +-----------------+  |
                    |              |        |
                    |              v        |
                    |              Q        |  stateless
                    |              |        |  decryption
                    |              v        |
                    |          +-------+    |
                    |          | split |    |
                    |          +-------+    |
                    |           |     |     |
                    +-----------|-----|-----+
                                v     v
                                P     U
```

Figure 5: Using a WPRP as a stateless decryption method in AERO.

A WPRP has the property that each bit of its output is a pseudorandom function of each bit of both of its inputs, for both encryption and decryption. A WPRP is very well suited for use in AERO. The fact that encryption is a WPRP, combined with the use of a sequence number, ensures strong confidentiality of the plaintext. The fact that the decryption algorithm is a WPRP ensures that an attacker cannot predict (much less control) the candidate sequence number, thus ensuring strong integrity, authenticity, and replay protection.

WPRPs are used in disk-block encryption [1619.2], because they provide the best security possible in scenarios where the ciphertext cannot be any longer than the plaintext.

When a WPRP algorithm is used as the EAD algorithm in AERO, we call the resulting combination AERO-WPRP. An AERO-WPRP algorithm has very strong security properties, as described in Section 11.

5.1.1. WPRP Requirements and survey

There are many WPRP designs in the cryptographic literature, driven largely by interest in disk-block encryption. However, AERO has some requirements that are different from those of disk-block encryption. Most importantly, the WPRP encryption algorithm must be able to process plaintexts that vary in length, using a single key. (In contrast, all of the blocks in a disk typically have the same length, allowing a WPRP design to be optimized for a fixed size.)

The Advanced Encryption Standard (AES) eXtended Code Book (XCB) mode of operation, or AES-XCB [MF07], is a WPRP that can accept plaintexts with lengths of 16 bytes or higher. Internally, it roughly consists of a layer of hashing, followed by a layer of counter mode, followed by another layer of hashing, with a small amount of additional processing. It was first introduced in 2004, and a second version was introduced in 2007, and was standardized in the IEEE [1619.2]. Recent work has raised questions about the security of the second version when used in scenarios in which the lengths of the plaintexts vary, which would make it inappropriate for AERO.

Chakraborty and Sarkar presented HCH, an improved hash-couter-hash WPRP mode of operation in 2007, which reduces the amount of additional processing and improves the security bound, so that more data can be securely encrypted for a fixed key [CS07].

Sarkar introduced yet another improvement in WPRPs in 2007, presenting a mode of operation based on a hash-encrypt-hash design [S07].

5.2. Other cryptographic techniques

Using a Wide PRP in AERO provides very strong security. However, it is more computationally expensive than some other approaches. Thus, it may be useful to use other techniques that trade some well-understood reduction in security properties for lessened computational cost. We do not define any such techniques in this note, but we outline the tradeoffs that are possible.

AEAD algorithms fit into two distinct categories: online and offline. With an online encryption algorithm, the ith ciphertext block can be output before the (i+1)st plaintext block has to be read (where a block is the size of the data blocks processed by a block cipher, typically) . These algorithms make a single pass over the data, and implementations need not buffer a significant amount of state. In contrast, with an offline encryption algorithm the entire plaintext must be read in before the first block of ciphertext can be output. Such algorithms must store state equal in length to the plaintext or

ciphertext that they are processing. In a software environment, an offline algorithm is acceptable, as long it processes only plaintexts with lengths are short enough that they fit into high-speed random access memory. However, offline algorithms cannot be implemented in a hardware pipeline, such as those used in high-speed network encryption (for protocols such as 802.1AE).

A WPRP is necessarily an offline algorithm. However, an online algorithm can implement a weakened variant of a WPRP: an Online Pseudo-Random Permutation (OPRP). An online permutation is a length-preserving permutation such that the ith block of output depends only on the first i blocks of input, for all values of i. An OPRP is an keyed online permutation that a computationally limited attacker cannot distinguish from an online permutation chosen uniformly at random from the set of all online permutations. With an OPRP, an attacker will be able to recognize when two messages with common plaintext prefixes are encrypted with the same key, since the ciphertext prefixes will be identical.

There are OPRPs in the cryptographic literature that are suitable for use in AERO, such as the Pipelineable Online Encryption (POE) mode of operation [AFFFLMW14], or the McOE-G [FFLW11] family of online authenticated encryption algorithms of Fleischmann, Forler, Lucks, and Wenzel.

There are offline ciphers that are more efficient that WPRPs, such as the Synthetic Initialization Vector (SIV) mode of operation of Rogaway and Shrimpton [RFC5297]. SIV encryption makes two passes over the plaintext, first to compute a Message Authentication Code (MAC) of that plaintext, the second to encrypt the plaintext using counter mode, using the MAC as the initialization vector for the encryption process. This two-pass encryption approach can be adapted for use in AERO, by having the IV computed by the tweakable encryption of the sequence number, using the MAC as the tweak. Such a mode of operation could be computationally less costly than a WPRP, but would also lack some of the misuse-resistance properties of a WPRP.

6. Loss, reorder, and resynchronization

AERO relies on the synchronization of the sequence number S between the sender (encryption algorithm) and the receiver (decryption algorithm). If those values differ by more than W, then we say that the sender and receiver are desynchronized; if this occurs, then it is likely that the decryption algorithm will discard at least one legitimate message that it receives. Desynchronization will occur whenever W+1 or more messages are lost by the communication channel between the sender and the receiver.

When initializing an AERO decryption context, the reorder tolerance parameter W should be set to a value that is larger than the message reorder of the communication channel in use. However, values of W larger than 256 are discouraged for security reasons; see Section 11. The default value of W=64 is recommended when AERO ciphertext messages are carried over an unreliable transport protocol. When they are carried over a reliable transport protocol, W=1 is recommended.

Resynchronization occurs automatically in AERO, because of the processing of candidate sequence numbers between R+1 and R+V. If a burst of exactly J messages that are lost, where J > W+1, then the receiver will reject the first message received immediately after the burst of lost messages, and will set R to the value of the current sequence number S of the sender. The candidate sequence number of the next message received will fall into the region between R+1 and R+V, since R=S and the candidate sequence number is S+1. Thus, after a burst of J > W lost packets, resynchronization is obtained after the spurious rejection of a single packet.

Most applications can accept the induced loss of a single message in situations where J > W or more messages have been lost, since this corresponds to an increase in packet loss less than 2% when the recommended value of W=64 is used. However, an application MAY choose to cache a rejected message (A, C) and re-submit it to the AERO decryption algorithm if it appears to have been spuriously rejected during the resynchronization process, to eliminate the induced message loss. An application can do this without any need for special processing from an AERO implementation.

When there are multiple receivers, it may be the case that one of the receivers is a "late joiner" that does not receive the initial messages sent in a session, because it joined that session some time after it began. The automatic resynchronization described above will enable a late joiner to synchronize with the sender after a single spurious rejection of a valid message. If an application protocol is designed to allow a late joiner in a session, it is likely that the

application can tolerate this one spurious rejection as the cost of achieving synchronization without external coordination of state.

The one free parameter in an AERO implementation is the number T of bits in the sequence number. This parameter determines the strength of the authentication provided, as well as the number of messages that can be encrypted by a particular sender using a particular key. Securty is the main consideration in the selection of this parameter; this topic is covered in Section 11.1.1.

7. Usage and applicability

AERO addresses several outstanding issues in cryptography as it is used for communication security. It is especially well suited for scenarios where there are bandwidth constrains or high costs associated with transmission and reception, as well as scenarios in which multiple senders or multiple receivers share an encryption key. It is also suitable for use when misuse resistance is desirable.

AERO always detects replayed messages, as is expected of a replay protection service, but AERO may reject a message because it does not have all of the stored state that it needs in able to authoritatively determine that the message is not a replay. In this way, it matches the conventional practice of replay protection using sequence numbers.

The sequence number output by the decryption algorithm can be used by the application calling AERO to put the decrypted plaintexts into their correct order. Some applications will not need this information, because they can rely on information in the associated data or plaintext to get information about the ordering of messages, or because they do not need to process the messages in order.

When AERO ciphertext messages are carried above a reliable transport protocol such as TCP or SCCP, the reorder tolerance parameter W can be set to one and the resynchronization parameter V can be set to zero. This has the effect of improving the strength of the authentication that is provided.

AERO-WPRP should be used, unless it is not possible to buffer the entire plaintext or ciphertext in memory during the encryption and decryption processes, in which case an AERO-OPRP would be more appropriate. Currently, there are no OPRP algorithms that can be implemented in an efficient pipeline, as is done with other AEAD algorithms such as AES-GCM, so this is an area of ongoing work.

8. Rationale

AES-XCB was chosen as it provides suitable security and performance, and it is a standard. In the future, stateless encryption algorithms with performance or security benefits over AES-XCB may be developed, but it is essential to specify a particular algorithm in this note, in order to gain experience in the implementation and use of AERO.

It is unfortunate, but unavoidable, that the plaintext must be padded. It is unfortunate because of the slight additional overhead and implementation complexity. It is unavoidable because there is no efficient way to encrypt fewer than b bytes of plaintext using a block cipher with a b-byte block, in a way that meets the security needs of AERO. Also, there are not (yet) any dedicated encryption algorithms that are suitable.

The padding and pad-stripping operations are included in the AERO encryption and decryption algorithms, respectively, because the pad-stripping operation also incorporates an authentication check. This arrangement simplifies the stateless encryption algorithm, and also ensures that more algorithms can be used as stateless encryption methods.

Incorporating the sequence number generation inside of AERO has the advantage of putting that security-critical operation inside of the cryptographic module, which often benefits from higher assurance than other system components, including testing, as observed by Sections 5 and 6 of [IV-GEN].

9. Testing

In the typical case in which the underlying stateless encryption algorithm is deterministic, both AERO encryption and AERO decryption are deterministic. This makes it possible to use test cases for all of the operations supported by an AERO algorithm.

10. IANA Considerations

The Internet Assigned Numbers Authority (IANA) has defined the AERO algorithm registry described below. An algorithm designer MAY register an algorithm in order to facilitate its use. Additions to the AERO algorithm registry require that a specification be documented in an Internet 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 "AERO_AES_128_XCB", that starts with the string "AERO",

a positive number, and

a reference to a specification that completely defines an AERO 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].

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

11. Security considerations

There are three AERO security services: confidentiality, message authentication, and replay protection. We consider those services in that order. We use a simplistic but realistic model in which pseudorandom values are treated as random values.

A more precise analysis would define the advantage that an adversary has in distinguishing the pseudorandom output of the stateless encryption algorithm from a truly random value, then quantify an attacker's advantage at forging AERO messages or distinguishing the output of the decryption algorithm from random, and then show that the attacker's advantage against AERO is negligibly small as long as the advantage against the stateless encryption algorithm is negligibly small. However, such a detailed analysis is beyond the scope of this note.

Below we sketch out a formal security analysis for AERO, using the conventional definition of advantage as the absolute value of the difference between the probability that a distinguisher will have a true positive and the probability that it will have a false positive. AERO is secure in the following model, which captures the idea of a very strong adversary. The attacker is assumed to be able to adaptively choose plaintext messages (A, P) and ciphertext messages (A, C), and obtain the corresponding ciphertext and plaintext messages.

Assume that there is an attack that can distinguish AERO from a randomly chosen function with the same domain and range. Then we can use this attack to build a distinguisher that can tell the underlying stateless encryption function from a truly random function, as follows. Given access to an oracle that is either the stateless encryption function with a randomly chosen secret key, or is a truly random function, we use that oracle to construct an AERO instance. We then run the AERO-distinguishing attack against that AERO instance, responding to the the queries made by the attack by constructing the appropriate AERO queries. If the attack returns a guess of "AERO", then we return a guess of "not random". Otherwise, return a guess of "random". This attack on the underlying encryption functions works with essentially the same advantage as the attack on AERO.

Similarly, if there is an attack that can forge AERO messages with probability significantly greater than 1/2^T' (where T' is the effective authentication tag length defined below), then we can use that attack to build a distinguisher that can tell the underlying stateless encryption function from a truly random function, as follows. Given access to an oracle that is either the stateless

encryption function with a randomly chosen secret key, or is a truly random function, we use that oracle to construct an AERO instance. We then run the forgery attack against that AERO instance, responding to the queries made by the attack by constructing the appropriate AERO queries. If the attack against AERO succeeds in making a forgery, then we return a guess of "not random". Otherwise, return a guess of "random". This distinguishing attack works with advantage P - 1/2^T', where P is the probability that the AERO-forgery attack will succeed in a forgery attempt.

AERO is a stateful authenticated encryption method. Bellare, Kohno, and Namprempre have studied the security of stateful encrption and decryption in the context of the Secure Shell (SSH) protocol [BKN04].

11.1. Authentication and confidentiality

The confidentiality properties of AERO follow directly from that of the stateless encryption algorithm that it incorporates. When a WPRP is used, each WPRP plaintext will be distinct, because of the uniqueness of the sequence numbers. This fact ensures the indistinguishability of AERO encryption from a random function, and thus the strength of the confidentiality that it provides.

AERO decryption can detect forgery attempts in two ways: the processing of the candidate sequence number, and the strip function that removes padding from the plaintext. The candidate sequence number is a pseudorandom function of both the AERO ciphertext and the AERO associated data, as is the post-decryption padding string PS. To an attacker that cannot distinguish the pseudorandom value from a random one, the probability p that a forgery attempt will not be detected by the candidate sequence number processing is

```
                (2*W + V)
          p = ------------.
                   2^T
```

To an attacker that cannot distinguish the post-decryption value of PS from a random string, the probability ps that a forgery attempt will pass through the "strip" function undetected is

```
                (PMIN - T/8)
        ps = -------------.
                   256
```

Thus, the probability that a forgery attempt will go undetected is 1/2^T', where

$$T' = T + 8 - lg((2*W + V) - lg(PMIN - T/8))$$

and lg(X) denotes the logarithm base two of X. As revealed by the equation above, AERO provides integrity and authentication protection that is essentially equivalent to an ideal message authentication code with a tag length of T' which is slightly less than T. The following table shows T' as a function of T, using the default values of W=64 and V=8 with PMIN=16, to three significant figures. The third column shows the number of bits in the data "on the wire" required by AERO. The final column shows the delta between the number of bits on the wire and the equivalent tag size T'.

T	T'	Data size	Delta
128	121	128	7.0
120	121	128	7.0
112	112	120	8.0
104	103	112	9.0
96	94.9	104	9.1
88	86.6	96	9.4
72	70.1	80	9.9
64	61.9	72	10.1
56	53.7	64	10.3
48	45.6	56	10.4
40	37.5	48	10.5
32	29.3	40	10.7
28	25.3	32	6.7

The delta values can be considered to be the data overhead inherent in AERO, that is, the amount of additional data that must be included in a message in order to provide the replay protection, sequencing information, and self-synchronization capability. For higher authentication strengths, this overhead is roughly one byte; otherwise, it is slightly higher.

When AERO is used over a reliable transport protocol, then the reorder tolerance parameter W SHOULD be set to one, and the resynchronization parameter V SHOULD be set to zero. When this is done, the delta values are about 7 bits smaller, and for a given amount of data overhead, the probability of a successful forgery is smaller by a factor of about 1/128.

11.1.1. Authentication strength

As always, stronger authentication, and thus larger values of T', should be preferred over weaker authentication, and smaller values of T'. We recommend the use of T' = 121 (with 128 bits on the wire) whenever possible. When bandwidth is at a premium, it may be acceptable to use T' = 53.7 (64 bits on the wire).

When the plaintext is audio or video data to be converted to an analog signal and played out of an audio speaker or video monitor, it may be acceptable to use even weaker authentication, whenever the consequences of a single forgery are limited to a localized "glitch" in the audio or video output. In such cases, an authentication strength of T' = 25.3 (32 bits on the wire) may be acceptable, but users are cautioned to verify these criteria before using such weak authentication. With these parameters, the likelihood of a successful forgery is one in 32 million.

11.2. Length hiding

In some applications, it is desirable to hide the length of the plaintext from an attacker, in order to prevent the attacker from being able to make inferences about the plaintext based on those lengths. The plaintext padding scheme defined in this note is not suitable for this purpose and it MUST NOT be used as such. An application that seeks to hide plaintext lengths should instead process the plaintexts with a fragmentation and encoding scheme prior to encrypting them. Schemes of this sort are out of scope of this note.

11.3. Denial of Service (DoS)

An attacker can potentially flood a communications channel with a set of bogus ciphertext messages in order to consume the computational resources of the receiver. In this section we review the vulnerability of AERO to this type of attack.

AERO synchronization is robust even in the face of a DoS attack. An AERO sender and receiver will stay in synchronization unless either 1) the attacker succeeds in forging a message, or 2) a burst of at least B messages have been lost. The forgery likelihood is

determined by the parameters T, V, and W, and it can be set high enough that the attacker's chance of success is negligible. A DoS attack may cause bursts of lost messages, by causing congestion on communication links. But AERO recovers quickly once that loss is over.

AERO requires that all of the cryptographic processing be done during the decryption algorithm in order to detect a forgery attempt. This is in contrast to other authenticated encryption algorithms (e.g. GCM) that can avoid doing decryption processing, since the authentication check can be completed before the encryption starts. When AERO is in use on a communication environment where the data rate of the communications greatly exceeds that of AERO decryption, it may be beneficial to use additional mechanisms that allow the receiver to reject bogus messages. This could include having a fixed, unpredictable value in each message in order to foil off-path DoS attacks. To defend against on-path attackers, cryptographic techniques can be used. These techniques are beyond the scope of this document.

11.4. Multiple senders

Section 3.5 describes the sender uniqueness requirement: a set of senders sharing the same key are recommended to ensure that their associated data inputs to the encryption algorithm will be distinct, for instance by including information in the associated data that uniquely identifies each sender. If the senders do not meet this requirement, then the security will degrade somewhat. The details of this degradation depend on the details of the underlying encryption method.

When a WPRP is used in AERO and the sender uniqueness requirement is not met, an attacker can recognize when the same exact plaintext message (A, P) is encrypted by different senders with the same sequence number. Symbolically, if sender S1 and sender S2 both use the same associated data value A and plaintext value P in the ith message in sequence that they are each independently encrypting, then the resulting ciphertexts will be identical, and an attacker can infer that the plaintexts are matching. In many real-world scenarios, it is highly unlikely that distinct senders will each encrypt the same plaintext value for the same sequence number, and thus the security degradation will be tolerable. If, however, the plaintexts are either very short (such as eight or fewer octets) or very repetitive (there are only a million possible plaintexts, for instance), then the loss of confidentiality may be unacceptably high.

When the sender uniqueness requirement is not met, the receiver will be processing messages from two or more senders, while believing it

to be from a single sender. The decryption algorithm will synchronize with the highest sequence number of any sender, and reject the messages from other senders, if the highest sequence number is at least W greater than all other sequence numbers. If the sequence numbers of the senders are close, then the receiver will interleave messages from different senders. In the latter case, an attacker could potentially manipulate what the post-decryption plaintext looks like by selectively withholding messages from particular senders. Note that this property is not particular to AERO; it holds for any authenticated encryption scheme in which multiple entities are sharing the encryption key, and the receiver has no way of determining which entity encrypted a particular ciphertext message.

These security properties compare very favorably with those of conventional authenticated encryption schemes. When the sender uniqueness requirement is not met with a nonce-based encryption scheme such as counter mode, CCM, GCM, Salsa or ChaCha, then essentially all of the confidentiality guarantees are lost. For nonce-based authentication schemes such as GCM, GMAC, and UMAC, all of the authentication guarantees are lost.

11.5. Sequence number management failure

An AERO implementation might accidentally mismanage sequence numbers, due to faulty logic in the implementation, or due to external factors such as the cloning of the state of the virtual machine on which the implementation is running. If this happens, security will degrade. In order to ensure that the degradation is not catastrophic, the underlying stateless encryption algorithm should be robust against nonce misuse, in the sense of [RFC5297].

12. Acknowledgements

The authors thank Dan Wing, Stefan Lucks, and Brian Weis for feedback and discussions, and Richard Barnes, Kenny Paterson, and Robert Moskowitz for suggestions and corrections.

13. References

13.1. Normative references

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

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

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

13.2. Informative references

[1619.2] "IEEE 1619.2-2010: Standard for Wide-Block Encryption for Shared Storage Media", IEEE Computer Society https://standards.ieee.org/findstds/standard/1619.2-2010.html, 2010.

[AFFFLMW14] Abed, Fluhrer, Foley, Forler, Lucks, McGrew, and Wenzel, "Pipelineable Online Encryption", Proceedings of the 21st international conference on Fast Software Encryption (FSE 2014) .

[BKN04] Bellare, Kohno, and Namprempre, "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, 7(2), May 2004. http://homes.cs.washington.edu/~yoshi/papers/SSH/.

[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 http://www-cse.ucsd.edu/users/mihir/papers/oem.html.

[CCM] Dworkin, "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/SP800-38C.pdf.

[CS07] Chakraborty, D. and P. Sarkar, "HCH: A New Tweakable Enciphering Scheme Using the Hash-Counter-Hash Approach", IACR eprint archive http://eprint.iacr.org/2007/28, 2007.

[FFLW11] Fleischmann, Forler, Lucks, and Wenzel, "McOE: a family of almost foolproof on-line authenticated encryption schemes", Proceedings of the 19th international conference on Fast Software Encryption (FSE 2011) .

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

[GCM] McGrew, D. and J. Viega, "The Galois/Counter Mode of Operation (GCM)", Submission to NIST http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf, January 2004.

[IV-GEN] McGrew, D., "Generation of Deterministic Initialization Vectors (IVs) and Nonces", IETF Internet Draft http://www.ietf.org/id/draft-mcgrew-iv-gen-03.txt, October 2013.

[MF07] McGrew, D. and S. Fluhrer, "The Security of the Extended Codebook (XCB) Mode of Operation", 14th Workshop on Selected Areas in Cryptography (SAC 2007) http://eprint.iacr.org/2007/298, August 2007.

[R02] Rogaway, P., "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.

[RFC4418] Krovetz, T., "UMAC: Message Authentication Code using Universal Hashing", RFC 4418, March 2006.

[RFC4771] Lehtovirta, V., Naslund, M., and K. Norrman, "Integrity Transform Carrying Roll-Over Counter for the Secure Real-time Transport Protocol (SRTP)", RFC 4771, January 2007.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, October 2008.

[S07] Sarkar, P., "Improving Upon the TET Mode of Operation", IACR eprint archive http://eprint.iacr.org/2007/317, 2007.

[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.

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