The AEM Authenticated-Encryption Mode

(Specification 1.3 — November 6, 2003)

Phillip Rogaway

1 Overview

This note specifies AEM, a mode of operation giving authenticated encryption. AEM is a refinement to Rogaway, Bellare, and Black’s OCB mode [10], while OCB was, in turn, a refinement to Jutla’s IAPM [5]. AEM is also a successor to the work of Gligor and Donescu’s [4] and to the broader line of research that has defined and investigated authenticated encryption [1, 2, 6–8]. The acronym AEM stands for authenticated-encryption mode and advanced encryption mode. Prominent characteristics of AEM are:

1. AEM is a mode of operation parameterized by an $n$-bit block cipher $E$ and a tag length $\tau \in [0..n]$.
2. Encryption and decryption depend on an $n$-bit nonce $N$, which must be selected as a new value for each encryption. The nonce need not be random or secret.
3. AEM allows an arbitrary header $H$ to be specified when one encrypts or decrypts a string.
4. The message $M$ and header $H$ can have any bit length, and the ciphertext $C$ one gets by encrypting $M$ in the presence of $H$ will always have $\tau$ bits more than $M$.
5. AEM encryption protects the privacy and authenticity of $M$ and the authenticity of $H$ and $N$.
6. AEM uses $\|M\| + \|H\| + 2$ block-cipher calls, where $\|\cdot\|$ is the length of the specified string measured in $n$-bit blocks.
7. If the header $H$ is fixed during a session then, after preprocessing, there is effectively no cost to have $H$ authenticated—the mode will use $\|M\| + 2$ block-cipher calls regardless of $\|H\|$.
8. AEM uses a single key $K$ for the underlying block cipher, and all block-cipher calls are keyed by $K$.
9. AEM is on-line: one need not know the length of $H$ or $M$ to proceed with encryption, and one need not know the length of $H$ or $C$ to proceed with decryption.
10. AEM is parallelizable: the bulk of its block-cipher calls may be performed simultaneously.
11. The main computational work beyond the block-cipher calls consists of one doubling operation and three xors for each $n$-bit block. Doubling consists of one shift and one conditional xor.
12. If the header $H$ is empty, no key setup is necessary or useful for AEM. If the header $H$ is nonempty, key setup is a single block-cipher call.
13. AEM enjoys provable security. One must assume that the block cipher $E$ is secure in the customary (strong PRP) sense. Security falls off in $\sigma^2/2^n$ where $\sigma$ is the total number of blocks one acts on.

Like all authenticated-encryption modes discussed in the literature, AEM becomes completely insecure if one acts on a total number of blocks approaching $\sigma = 2^{n/2}$. Care must be taken to re-key well before then. Security is also sacrificed if a nonce is re-used.

Two major factors differentiate AEM from its predecessor OCB: the first is the presence of the header $H$, an issue discussed in [8], and the second is the simpler way in which AEM computes offsets, which no longer involves counting the number-of-trailing-zero bits in a counter or doing any other non-constant-time
calculation for each \( n \)-bit block. The theory underlying AEM is described in Rogaway [9] and the body of work on which that builds. The current note gives only a specification—no security definitions or proofs.

2 Notation

By a *string* we mean a finite sequence of zeros and ones. The length of a string \( X \) (in bits) is written \(|X|\). The string of length 0 is called the *empty string* and is denoted \( \varepsilon \). By \( 0^i \) we mean the string of \( i \) zero-bits and by \( 1^j \) we mean the string of \( j \) one-bits. The set of all strings is denoted \( \{0, 1\}^* \) and the set of all strings of length \( n \) is denoted \( \{0, 1\}^n \). If \( X \) and \( Y \) are strings then \( XY \) or \( X \parallel Y \) is their concatenation. If \( X \) is a string of length \( n \) and \( X \in [0..n] \) then we write \( X \) [first \( \tau \) bits] for the first \( \tau \) bits of \( X \). If \( X \) and \( Y \) are strings of equal length then \( X \oplus Y \) denotes their bitwise xor, while if \(|X| < |Y|\) then \( X \oplus Y = X \ominus Y \) [first \( |X| \) bits]. If \( X \) is a string of length at most \( n \) and \( n \) is understood, then \( X0^n \equiv X0^{|X| \text{ mod } n} \) if \( X \neq \varepsilon \) and \( X0^n = 0^n \) if \( X = \varepsilon \). If \( X \) is a string of length at most \( 2^n - 1 \) and \( n \) is understood then by \( \text{len}(X) \) we mean the \( n \)-bit that encodes the length of \( X \), in binary, most-significant-bit first and least-significant-bit last.

Let \( X \) be a 128-bit string. We define what it means to *double* \( X \), or multiply \( X \) by two, giving a 128-bit string we denote by \( 2X \). This is defined as \( 2X = X \ll 1 \) if the first bit of \( X \) is a 0, and \( 2X = (X \ll 1) \oplus 0^{128}1^41^3 \) if the first bit of \( X \) is a 1. Here \( X \ll 1 \) means the left shift of \( X \) by one position: if \( X = X_1 \cdots X_{128} \) then \( X \ll 1 = X_2 \cdots X_{128}0 \). We similarly define what it means to multiply \( X \) by three and five, defining \( 3X = 2X \oplus X \) and \( 5X = (2(2X)) \oplus X \). We emphasize that \( 2X \), \( 3X \), and \( 5X \) should not be confused with multiplication in the integers: the semantics and the mechanics of computing these values is very different from integer multiplication.

An \( n \)-bit block cipher is a function \( E: \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n \) where \( n \geq 1 \) is a number (the block length) and \( \mathcal{K} \) is a finite nonempty set (the keys) and \( E(K, \cdot) = E_K(\cdot) \) is a permutation for all \( K \in \mathcal{K} \). We write \( E_K^{-1}(Y) \) for the string \( X \) such that \( E_K(X) = Y \).

3 Definition of AEM

**Parameters.** AEM is parameterized by an \( n \)-bit block cipher \( E \) and a tag length \( \tau \in [0..n] \). The parameters \( E \) and \( \tau \) must be fixed for a given session and would typically be fixed for a given application. If parameters are negotiated between communicating parties at the beginning of a session then they must be negotiated in an authenticated way. This specification assumes the use of a block cipher having block length of \( n = 128 \) bits. Although this specification could easily be extended to allow other values, block lengths of \( n = 64 \) bits would have to be disparaged due to the \( \sigma^2/2^n \) security degradation discussed in Section 1, while block lengths other than \( n = 64 \) and \( n = 128 \) have little practical value. We expect that \( E \) will usually be AES (meaning AES128, AES192, or AES256). To be more explicit in specifying parameters one may write AEM-\( E \) or AEM[\( E \)] or AEM[\( E, \tau \)].

**Encryption interface.** Once the parameters \( E \) and \( \tau \) have been fixed, AEM provides a method to *encrypt* and a method to *decrypt*. To encrypt, AEM.Encrypt takes

- a key \( K \) (a random string drawn from the key space of the underlying block cipher),
- a nonce \( N \) (an \( n \)-bit string that must not be repeated, during all encryption requests, during a session),
- a header \( H \) (an arbitrary string), and
- a message \( M \) (an arbitrary string)

and produces as output

- a ciphertext (having \( \tau \) bits more than \( M \)).

Ciphertext \( \mathcal{C} = \text{AEM.Encrypt}_K^H(M) \) protects the privacy of \( M \) and the authenticity of \( M, N, \) and \( H \). If one does not need the header \( H \) let it be the empty string \( H = \varepsilon \). Note that a small amount of material may be authenticated by placing it in \( N \) since all of \( N \) is authenticated, too.

**Decryption interface.** To decrypt, AEM.Decrypt takes

- a key \( K \) (a random string drawn from the key space of the underlying block cipher),
- a nonce $N$ (an $n$-bit string)
- a header $H$ (an arbitrary string), and
- a ciphertext $C$ (an arbitrary string)
and produces as output either
- a message $M$ (having $\tau$ bits fewer than $C$) or
- the distinguished symbol INVALID (indicating that $(N, H, C)$ is invalid with respect to $K$).
The message $M = \text{AEM}.\text{Decrypt}^N_H(C)$ is always returned if $C = \text{AEM}.\text{Encrypt}^N_H(M)$. The symbol INVALID is usually returned as $\text{AEM}.\text{Decrypt}^N_H(C)$ if $C$ was produced in a manner other than by setting $C = \text{AEM}.\text{Encrypt}^N_H(M)$ for some $M$. It is beyond the scope of this note to explain the scientific meaning of the last sentence; see a work such as [9] for that.

**Specification.** Our definition of AEM.Encrypt and AEM.Decrypt are given in Figure 1. The definitions make use of functions $\text{OCB1.Encrypt}$, $\text{OCB1.Decrypt}$, and $\text{PMAC1}$, which are defined in the same figure. The instruction “Parse $C$ into $C || T$” means to let $C$ be the first $|C| - \tau$ bits of $C$ and to let $T$ be the last $\tau$ bits of $C$. The instruction “Parse $M$ into $M[1] \cdots M[m]$” means to let $m = \max\{1, |M|/n\}$ and to let $M[1], \ldots, M[m]$ be the unique strings such that $M[1] \cdots M[m] = M$ and $|M[1]| = \cdots = |M[m-1]| = n$. The instruction “Parse $C$ into $C[1] \cdots C[m] T$” means to let $C$ be the first $|C| - \tau$ bits of $C$ and to let $T$ be the last $\tau$ bits of $C$ and to let $m = \max\{1, |C|/n\}$ and to let $C[1], \ldots, C[m]$ be the unique strings such that $C[1] \cdots C[m] = C$ and $|C[1]| = \cdots = |C[m-1]| = n$. Illustrations of $\text{OCB1.Encrypt}$ and $\text{PMAC1}$ are given in Figures 2 and 3.

**Acknowledgments**

This research was supported by NSF 0208842 and by a gift from Cisco System. Thanks to the NSF (particularly Carl Landwehr) and to Cisco (particularly David McGrew) for their kind support of my research.

**References**


Algorithm AEM.Encrypt_N^H_K (M)
100 \( \ell \leftarrow \text{OCB1.Encrypt}_K^N(M) \)
101 Parse \( \ell \) into \( C \parallel T \)
102 \( \text{if } \text{if } H \neq \varepsilon \text{ then } T \leftarrow T \oplus \text{PMAC1}_K(H) \)
103 \( \text{return } C \parallel T \)

Algorithm OCB1.Encrypt_N^K (M)
300 Parse \( M \) into \( M[1] \cdots M[m] \)
301 \( \Delta \leftarrow E_K(N) \)
302 \( \Sigma \leftarrow 0^n \)
303 \( \text{for } i \leftarrow 1 \text{ to } m - 1 \text{ do} \)
304 \( \Delta \leftarrow 2 \Delta \)
305 \( C[i] \leftarrow E_K(M[i] \oplus \Delta) \oplus \Delta \)
306 \( \Sigma \leftarrow \Sigma \oplus M[i] \)
307 \( \Delta \leftarrow 2 \Delta \)
308 \( \text{Pad } \leftarrow E_K(\text{len}(M[m]) \oplus \Delta) \)
309 \( C[m] \leftarrow M[m] \oplus \text{Pad} \)
310 \( C \leftarrow C[1] \cdots C[m] \)
311 \( \Sigma \leftarrow \Sigma \oplus C[m]0^n \oplus \text{Pad} \)
312 \( \Delta \leftarrow 3 \Delta \)
313 \( \text{Tag } \leftarrow E_K(\Sigma \oplus \Delta) \)
314 \( T \leftarrow \text{Tag [first } \tau \text{ bits]} \)
315 \( \text{return } \ell \leftarrow C \parallel T \)

Algorithm PMAC1_K (M)
500 Parse \( M \) into \( M[1] \cdots M[m] \)
501 \( \Theta \leftarrow 5 E_K(0^n) \)
502 \( \Sigma \leftarrow 0^n \)
503 \( \text{for } i \leftarrow 1 \text{ to } m - 1 \text{ do} \)
504 \( \Theta \leftarrow 2 \Theta \)
505 \( Y \leftarrow E_K(M[i] \oplus \Theta) \)
506 \( \Sigma \leftarrow \Sigma \oplus Y \)
507 \( \Theta \leftarrow 2 \Theta \)
508 \( \text{if } |M[m]| = n \text{ then } \Theta \leftarrow 3 \Theta, \quad \Sigma \leftarrow \Sigma \oplus M[m] \)
509 \( \text{else } \Theta \leftarrow 5 \Theta, \quad \Sigma \leftarrow \Sigma \oplus M[m]10^n \)
510 \( \text{Tag } \leftarrow E_K(\Sigma \oplus \Theta) \)
511 \( T \leftarrow \text{Tag [first } \tau \text{ bits]} \)
512 \( \text{return } T \)

Algorithm AEM.Decrypt_N^H_K (\ell)
200 \( \text{if } |\ell| < \tau \text{ then return INVALID} \)
201 Parse \( \ell \) into \( C \parallel T \)
202 \( \text{if } H \neq \varepsilon \text{ then } T \leftarrow T \oplus \text{PMAC1}_K(H) \)
203 \( M \leftarrow \text{OCB1.Decrypt}_K^N(C \parallel T) \)
204 \( \text{return } M \)

Algorithm OCB1.Decrypt_N^K (\ell)
400 Parse \( \ell \) into \( C[1] \cdots C[m] T \)
401 \( \Delta \leftarrow E_K(N) \)
402 \( \Sigma \leftarrow 0^n \)
403 \( \text{for } i \leftarrow 1 \text{ to } m - 1 \text{ do} \)
404 \( \Delta \leftarrow 2 \Delta \)
405 \( M[i] \leftarrow E_K^{-1}(C[i] \oplus \Delta) \oplus \Delta \)
406 \( \Sigma \leftarrow \Sigma \oplus M[i] \)
407 \( \Delta \leftarrow 2 \Delta \)
408 \( \text{Pad } \leftarrow E_K(\text{len}(C[m]) \oplus \Delta) \)
409 \( M[m] \leftarrow C[m] \oplus \text{Pad} \)
410 \( M \leftarrow M[1] \cdots M[m] \)
411 \( \Delta \leftarrow 3 \Delta \)
412 \( \text{Tag } \leftarrow E_K(\Sigma \oplus \Delta) \)
413 \( T' \leftarrow \text{Tag [first } \tau \text{ bits]} \)
414 \( \text{if } T = T' \text{ then return } M \)
415 \( \text{else return INVALID} \)

Figure 1: The AEM mode of operation. The plaintext is \( M \), the ciphertext is \( \ell = C \parallel T \), the key is \( K \), and the header is \( H \). The underlying block cipher is \( E \).
Figure 2: An illustration of OCB1 encryption acting on a message of four blocks. OCB1 is the core of AEM. Read the diagram in vertical strips, from top-to-bottom and from left-to-right.

Figure 3: An illustration of PMAC1 authentication of a four-block message. On the left is the case where the final block is a full block, and on the right is the case where the final block is a partial block. Read each picture in vertical strips, from top-to-bottom and from left-to-right.