

Network Working Group
Internet-Draft
Expires: December 15, 2006

N. Modadugu
Stanford University
E. Rescorla
Network Resonance
June 13, 2006

AES Counter Mode Cipher Suites for TLS and DTLS
draft-ietf-tls-ctr-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on December 15, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document describes the use of the Advanced Encryption Standard (AES) Counter Mode for use as a Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) confidentiality mechanism.

Table of Contents

1.	Introduction	3
1.1.	Conventions Used In This Document	3
2.	Terminology	3
3.	Encrypting Records with AES Counter Mode	4
3.1.	TLS	4
3.1.1.	Encryption	4
3.1.2.	Decryption	5
3.1.3.	Counter Block Construction	5
3.2.	DTLS	6
3.3.	Padding	7
3.4.	Session Resumption	7
4.	Design Rationale	7
5.	Security Considerations	7
5.1.	Maximum Key Lifetime	8
6.	IANA Considerations	8
7.	Normative References	8
	Authors' Addresses	9
	Intellectual Property and Copyright Statements	10

1. Introduction

Transport Layer Security [3] provides channel-oriented security for application layer protocols. In TLS, cryptographic algorithms are specified in "Cipher Suites, which consist of a group of algorithms to be used together."

Cipher suites supported by TLS are divided into stream and block ciphers. Counter mode ciphers behave like stream ciphers, but are constructed based on a block cipher primitive (that is, counter mode operation of a block cipher results in a stream cipher.) This specification is limited to discussion of the operation of AES in counter mode (AES-CTR.)

Counter mode ciphers (CTR) offer a number of attractive features over other block cipher modes and stream ciphers such as RC4:

Low Bandwidth: AES-CTR provides a saving of 17-32 bytes per record compared to AES-CBC as used in TLS 1.1 and DTLS. 16 bytes are saved from not having to transmit an explicit IV, and another 1-16 bytes are saved from the absence of the padding block.

Random Access: AES-CTR is capable of random access within the key stream. For DTLS, this implies that records can be processed out of order without dependency on packet arrival order, and also without keystream buffering.

Parallelizable: As a consequence of AES-CTR supporting random access within the key stream, making the cipher amenable to parallelizing and pipelining in hardware.

Multiple mode support: AES-CTR support in TLS/DTLS allows for implementator to support both a stream (CTR) and block (CBC) cipher through the implementation of a single symmetric algorithm.

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 [1].

2. Terminology

This document reuses some terminology introduced in [2] and [3]. The term 'counter block' has the same meaning as used in [2]. However, the term 'IV' in this document, holds the meaning defined in [3].

3. Encrypting Records with AES Counter Mode

AES-CTR is functionally equivalent to a stream cipher; it generates a pseudo-random cipher stream that is XORed into the plaintext to form ciphertext.

The cipher stream is generated by applying the AES encrypt operation on a sequence of 128-bit counter blocks. Counter blocks, in turn, are generated based on record sequence numbers (in the case of TLS), or a combination of record sequence and epoch numbers (in the case of DTLS.)

It should be noted that although the client and server use the same sequence number space, they use different write keys and counter blocks.

There is one important constraint on the use of counter mode ciphers: for a given key, a counter block value **MUST** never be used more than once.

This constraint is required because a given key and counter block value completely specify a portion of the cipher stream. Hence, a particular counter block value when used (with a given key) to generate more than one ciphertext leaks information about the corresponding plaintexts. For a detailed explanation, see Section 7 of [2].

Given this constraint, the challenge then is in the design of the counter block. We describe the construction of the counter block in the following sections.

TLS/DTLS records encrypted with AES-CTR mode use a CipherSpec.cipher_type of GenericStreamCipher (Section 6.2.3 of [3]).

3.1. TLS

AES counter mode requires the encryptor and decryptor to share a per-record unique counter block. As previously stated, a given counter block **MUST** never be used more than once with the same key. The following description of AES-CTR mode has been adapted from [2].

3.1.1. Encryption

To encrypt a payload with AES-CTR, the encryptor sequentially partitions the plaintext (PT) into 128-bit blocks. The final PT block **MAY** be less than 128-bits. This partitioning is denoted as:

$$PT = PT[1] \ PT[2] \ \dots \ PT[n]$$

In order to encrypt, each PT block is XORed with a block of the key stream to generate the ciphertext (CT.) The keystream is generated via the AES encryption of each counter block value, with each encryption operation producing 128-bits of key stream.

The encryption operation is performed as follows:

```
FOR i := 1 to n-1 DO
  CT[i] := PT[i] XOR AES(CtrBlk)
  CtrBlk := CtrBlk + 1
END
CT[n] := PT[n] XOR TRUNC(AES(CtrBlk))
```

The AES() function performs AES encryption with the fresh key.

The TRUNC() function truncates the output of the AES encrypt operation to the same length as the final plaintext block, returning the leftmost bits.

3.1.2. Decryption

Decryption is similar to encryption. The decryption of n ciphertext blocks is performed as follows:

```
FOR i := 1 to n-1 DO
  PT[i] := CT[i] XOR AES(CtrBlk)
  CtrBlk := CtrBlk + 1
END
PT[n] := CT[n] XOR TRUNC(AES(CtrBlk))
```

The AES() and TRUNC() operate identically as in the case of encryption.

3.1.3. Counter Block Construction

To construct the counter block, the leftmost 48-bits of the counter block are set to the rightmost 48-bits of the client_write_IV (for the half-duplex stream originated by the client) or the rightmost 48-bits of the server_write_IV (for the half-duplex stream originated by the server.) The following 64-bits of the counter block are set to record sequence number, and the remaining 16-bits function as the block counter. The block counter is a 16-bit unsigned integer in network byte order (i.e. big-endian). The block counter is initially set to one, and is incremented by one to generate subsequent counter blocks, each resulting in another 128-bits of key stream.

The structure of the counter block is depicted below:

```
struct {  
    case client:  
        uint48 client_write_IV; // low order 48-bits  
    case server:  
        uint48 server_write_IV; // low order 48-bits  
    uint64 seq_num;  
    uint16 blk_ctr;  
} CtrBlk;
```

The seq_num and blk_ctr fields of the counter block are initialized for each record processed, while the IV is initialized immediately after a key calculation is made (key calculations are made whenever a TLS/DTLS handshake, either full or abbreviated, is executed.) seq_num is set to the sequence number of the record, and blk_ctr is initialized to 1.

Note that the block counter does not overflow since the maximum size of input to the record payload protection layer in TLS or DTLS (TLSCompressed.length) is $2^{14} + 1024$ octets, and 16 bits of blk_ctr allow the generation of 2^{20} octets (2^{16} AES blocks) of keying material per record.

Note that for TLS, no part of the counter block need be transmitted, since the client_write_IV and server_write_IV are derived during the key calculation phase, and the record sequence number is implicit.

[3.2.](#) DTLS

The operation of AES-CTR in DTLS is the same as in TLS, with the only difference being the inclusion of the epoch in the counter block. The counter block is constructed as follows for DTLS:

```
struct {  
    case client:  
        uint48 client_write_IV; // low order 48-bits  
    case server:  
        uint48 server_write_IV; // low order 48-bits  
    uint16 epoch;  
    uint48 seq_num;  
    uint16 blk_ctr;  
} CtrBlk;
```

For decryption, the epoch and seq_num fields are initialized based on the corresponding values in a received record.

3.3. Padding

Stream ciphers in TLS and DTLS do not require plaintext padding.

3.4. Session Resumption

TLS supports session resumption via caching of session ID's and connection parameters on both client and server. While resumed sessions use the same master secret that was originally negotiated, a resumed session uses new keys that are derived, in part, using fresh client_random and server_random parameters. As a result resumed sessions do not use the same encryption keys or IV's as the original session.

4. Design Rationale

An alternate design for the construction of the counter block would be the use of an explicit 'record tag' (as a substitute for the implicit record sequence number) that could potentially be generated via an LFSR. Such a design, however, suffers a major drawback when used in the TLS or DTLS protocol, without offering any significant benefit: in both TLS and DTLS inclusion of such a tag would incur a bandwidth cost.

5. Security Considerations

The security considerations for the use of AES-CTR in TLS/DTLS are specified below. The below text is based heavily on that for AES-CTR in IPsec [2].

- o Counter blocks must not be used more than once with a given key. Doing so allows a passive attacker to determine the XOR of the affected plain text blocks. Extracting two plaintexts from their XOR is a relatively straightforward operation. Because the counter block is derived from the per-record sequence, this means that sequence numbers MUST never be re-used with different data. Note, however, that retransmitting the same record in DTLS is safe.
- o AES-CTR can be used in pre-shared key mode, since session keys and not pre-shared keys are used for ciphering. Also, since separate read and write keys are generated, counter blocks generated by client and server can safely overlap.
- o As with other stream ciphers, data forgery is trivial if no message integrity mechanism is employed. This threat is of no concern in TLS/DTLS since all ciphersuites that support encryption also employ message integrity.

5.1. Maximum Key Lifetime

TLS/DTLS sessions employing AES-CTR MUST be renegotiated before sequence numbers repeat. In the case of TLS, this implies a maximum of 2^{64} records per session, while for DTLS the maximum is 2^{48} (with the remaining bits reserved for epoch.)

6. IANA Considerations

IANA has assigned the following values for AES-CTR mode ciphers:

```
CipherSuite TLS_RSA_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_DSS_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_RSA_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DHE_DSS_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DHE_RSA_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_anon_WITH_AES_128_CTR_SHA = { 0xxx, 0xxx };
```

```
CipherSuite TLS_RSA_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_DSS_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_RSA_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DHE_DSS_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DHE_RSA_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
CipherSuite TLS_DH_anon_WITH_AES_256_CTR_SHA = { 0xxx, 0xxx };
```

7. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [2] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", [RFC 3686](#), January 2004.
- [3] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", [RFC 4346](#), April 2006.

Authors' Addresses

Nagendra Modadugu
Stanford University
353 Serra Mall
Stanford, CA 94305
USA

Email: nagendra@cs.stanford.edu

Eric Rescorla
Network Resonance
2483 E. Bayshore Rd., #212
Palo Alto, CA 94303
USA

Email: ekr@networkresonance.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

