

Workgroup: CoRE Working Group  
Internet-Draft:  
draft-hoeglund-core-oscore-key-limits-01  
Updates: [8613](#) (if approved)  
Published: 12 July 2021  
Intended Status: Standards Track  
Expires: 13 January 2022  
Authors: R. Höglund     M. Tiloca  
          RISE AB        RISE AB  
**Key Update for OSCORE**

## Abstract

Object Security for Constrained RESTful Environments (OSCORE) uses AEAD algorithms to ensure confidentiality and integrity of exchanged messages. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a specific key is used for encryption or decryption. This document defines how two OSCORE peers must follow these limits and what steps they must take to preserve the security of their communications. Therefore, this document updates RFC8613. Furthermore, this document specifies a lightweight method that two peers can use to update their keying material and establish a new OSCORE Security Context.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on 13 January 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

- [1. Introduction](#)
  - [1.1. Terminology](#)
- [2. AEAD Key Usage Limits in OSCORE](#)
  - [2.1. Problem Overview](#)
    - [2.1.1. Limits for 'q' and 'v'](#)
  - [2.2. Additional Information in the Security Context](#)
    - [2.2.1. Common Context](#)
    - [2.2.2. Sender Context](#)
    - [2.2.3. Recipient Context](#)
  - [2.3. OSCORE Messages Processing](#)
    - [2.3.1. Protecting a Request or a Response](#)
    - [2.3.2. Verifying a Request or a Response](#)
  - [2.4. Current methods for Rekeying OSCORE](#)
- [3. New Method for Rekeying OSCORE](#)
  - [3.1. Extensions to the OSCORE Option](#)
  - [3.2. Update Function](#)
  - [3.3. Establishment of the New OSCORE Security Context](#)
  - [3.4. Retention Policies](#)
- [4. Security Considerations](#)
- [5. IANA Considerations](#)
  - [5.1. OSCORE Flag Bits Registry](#)
- [6. References](#)
  - [6.1. Normative References](#)
  - [6.2. Informative References](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

## 1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end protection of CoAP [RFC7252] messages at the application-layer, ensuring message confidentiality and integrity, replay protection, as well as binding of response to request between a sender and a recipient.

In particular, OSCORE uses AEAD algorithms to provide confidentiality and integrity of messages exchanged between two peers. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a

specific key is used to perform encryption or decryption [[I-D.irtf-cfrg-aead-limits](#)].

Should these limits be exceeded, an adversary may break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack. The original OSCORE specification [[RFC8613](#)] does not consider such limits.

This document updates [[RFC8613](#)] and defines when a peer must stop using an OSCORE Security Context shared with another peer, due to the reached key usage limits. When this happens, the two peers have to establish a new Security Context with new keying material, in order to continue their secure communication with OSCORE.

Furthermore, this document specifies a lightweight method that the two peers can use in order to update their current keying material and establish a new OSCORE Security Context.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts related to the CoAP [[RFC7252](#)] and OSCORE [[RFC8613](#)] protocols.

## 2. AEAD Key Usage Limits in OSCORE

The following sections details how key usage limits for AEAD algorithms must be considered when using OSCORE. It covers specific limits for common AEAD algorithms used with OSCORE; necessary additions to the OSCORE Security Context, updates to the OSCORE message processing, and existing methods for rekeying OSCORE.

### 2.1. Problem Overview

The OSCORE security protocol [[RFC8613](#)] uses AEAD algorithms to provide integrity and confidentiality of messages, as exchanged between two peers sharing an OSCORE Security Context.

When processing messages with OSCORE, each peer should follow specific limits as to the number of times it uses a specific key. This applies separately to the Sender Key used to encrypt outgoing messages, and to the Recipient Key used to decrypt and verify incoming protected messages.

Exceeding these limits may allow an adversary to break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack.

The following refers to the two parameters 'q' and 'v' introduced in [\[I-D.irtf-cfrg-aead-limits\]](#), to use when deploying an AEAD algorithm.

- \*'q': this parameter has as value the number of messages protected with a specific key, i.e. the number of times the AEAD algorithm has been invoked to encrypt data with that key.

- \*'v': this parameter has as value the number of alleged forgery attempts that have been made against a specific key, i.e. the amount of failed decryptions that has been done with the AEAD algorithm for that key.

When a peer uses OSCORE:

- \*The key used to protect outgoing messages is its Sender Key, in its Sender Context.

- \*The key used to decrypt and verify incoming messages is its Recipient Key, in its Recipient Context.

Both keys are derived as part of the establishment of the OSCORE Security Context, as defined in [Section 3.2](#) of [\[RFC8613\]](#).

As mentioned above, exceeding specific limits for the 'q' or 'v' value can weaken the security properties of the AEAD algorithm used, thus compromising secure communication requirements.

Therefore, in order to preserve the security of the used AEAD algorithm, OSCORE has to observe limits for the 'q' and 'v' values, throughout the lifetime of the used AEAD keys.

#### **2.1.1. Limits for 'q' and 'v'**

Formulas for calculating the security levels as Integrity Advantage (IA) and Confidentiality Advantage (CA) probabilities, are presented in [\[I-D.irtf-cfrg-aead-limits\]](#). These formulas take as input specific values for 'q' and 'v' (see section [Section 2.1](#)) and for 'l', i.e., the maximum length of each message (in cipher blocks).

For the algorithms that can be used as AEAD Algorithm for OSCORE shows in [Figure 1](#), the key property to achieve is having IA and CA values which are no larger than  $p = 2^{-64}$ , which will ensure a safe security level for the AEAD Algorithm. This can be entailed by using the values  $q = 2^{20}$ ,  $v = 2^{20}$ , and  $l = 2^8$ , that this document recommends to use for these algorithms.

[Figure 1](#) shows the resulting IA and CA probabilities enjoyed by the considered algorithms, when taking the value of 'q', 'v' and 'l' above as input to the formulas defined in [[I-D.irtf-cfrg-aead-limits](#)].

Algorithm name	IA probability	CA probability
AEAD_AES_128_CCM	$2^{-68}$	$2^{-70}$
AEAD_AES_128_GCM	$2^{-99}$	$2^{-89}$
AEAD_AES_256_GCM	$2^{-99}$	$2^{-89}$
AEAD_CHACHA20_POLY1305	$2^{-75}$	-

Figure 1: Probabilities for algorithms based on chosen q, v and l values.

For the AEAD\_AES\_128\_CCM\_8 algorithm when used as AEAD Algorithm for OSCORE, larger IA and CA values are achieved, depending on the value of 'q', 'v' and 'l'. [Figure 2](#) shows the resulting IA and CA probabilities enjoyed by AEAD\_AES\_128\_CCM\_8, when taking different values of 'q', 'v' and 'l' as input to the formulas defined in [[I-D.irtf-cfrg-aead-limits](#)].

As shown in [Figure 2](#), it is especially possible to achieve the lowest  $IA = 2^{-54}$  and a good  $CA = 2^{-70}$  by considering the largest possible value of the (q, v, l) triplet equal to  $(2^{20}, 2^{10}, 2^8)$ , while still keeping a good security level. Note that the value of 'l' does not impact on IA, while CA displays good values for every considered value of 'l'.

When AEAD\_AES\_128\_CCM\_8 is used as AEAD Algorithm for OSCORE, this document recommends to use the triplet  $(q, v, l) = (2^{20}, 2^{10}, 2^8)$  and to never use a triplet (q, v, l) such that the resulting IA and CA probabilities are higher than  $2^{-54}$ .

'q', 'v' and 'l'	IA probability	CA probability
q=2 <sup>20</sup> , v=2 <sup>20</sup> , l=2 <sup>8</sup>	2 <sup>-44</sup>	2 <sup>-70</sup>
q=2 <sup>15</sup> , v=2 <sup>20</sup> , l=2 <sup>8</sup>	2 <sup>-44</sup>	2 <sup>-80</sup>
q=2 <sup>10</sup> , v=2 <sup>20</sup> , l=2 <sup>8</sup>	2 <sup>-44</sup>	2 <sup>-90</sup>
q=2 <sup>20</sup> , v=2 <sup>15</sup> , l=2 <sup>8</sup>	2 <sup>-49</sup>	2 <sup>-70</sup>
q=2 <sup>15</sup> , v=2 <sup>15</sup> , l=2 <sup>8</sup>	2 <sup>-49</sup>	2 <sup>-80</sup>
q=2 <sup>10</sup> , v=2 <sup>15</sup> , l=2 <sup>8</sup>	2 <sup>-49</sup>	2 <sup>-90</sup>
q=2 <sup>20</sup> , v=2 <sup>10</sup> , l=2 <sup>8</sup>	2 <sup>-54</sup>	2 <sup>-70</sup>
q=2 <sup>15</sup> , v=2 <sup>10</sup> , l=2 <sup>8</sup>	2 <sup>-54</sup>	2 <sup>-80</sup>
q=2 <sup>10</sup> , v=2 <sup>10</sup> , l=2 <sup>8</sup>	2 <sup>-54</sup>	2 <sup>-90</sup>
q=2 <sup>20</sup> , v=2 <sup>20</sup> , l=2 <sup>6</sup>	2 <sup>-44</sup>	2 <sup>-74</sup>
q=2 <sup>15</sup> , v=2 <sup>20</sup> , l=2 <sup>6</sup>	2 <sup>-44</sup>	2 <sup>-84</sup>
q=2 <sup>10</sup> , v=2 <sup>20</sup> , l=2 <sup>6</sup>	2 <sup>-44</sup>	2 <sup>-94</sup>
q=2 <sup>20</sup> , v=2 <sup>15</sup> , l=2 <sup>6</sup>	2 <sup>-49</sup>	2 <sup>-74</sup>
q=2 <sup>15</sup> , v=2 <sup>15</sup> , l=2 <sup>6</sup>	2 <sup>-49</sup>	2 <sup>-84</sup>
q=2 <sup>10</sup> , v=2 <sup>15</sup> , l=2 <sup>6</sup>	2 <sup>-49</sup>	2 <sup>-94</sup>
q=2 <sup>20</sup> , v=2 <sup>10</sup> , l=2 <sup>6</sup>	2 <sup>-54</sup>	2 <sup>-74</sup>
q=2 <sup>15</sup> , v=2 <sup>10</sup> , l=2 <sup>6</sup>	2 <sup>-54</sup>	2 <sup>-84</sup>
q=2 <sup>10</sup> , v=2 <sup>10</sup> , l=2 <sup>6</sup>	2 <sup>-54</sup>	2 <sup>-94</sup>

Figure 2: Probabilities for AEAD\_AES\_128\_CCM\_8 based on chosen q, v and l values.

## 2.2. Additional Information in the Security Context

In addition to what defined in [Section 3.1](#) of [\[RFC8613\]](#), the OSCORE Security Context MUST also include the following information.

### 2.2.1. Common Context

The Common Context is extended to include the following parameter.

\*'exp': with value the expiration time of the OSCORE Security Context, as a non-negative integer. The parameter contains a numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds, analogous to what specified for NumericDate in [Section 2](#) of [\[RFC7519\]](#).

At the time indicated in this field, a peer MUST stop using this Security Context to process any incoming or outgoing message, and is required to establish a new Security Context to continue OSCORE-protected communications with the other peer.

### 2.2.2. Sender Context

The Sender Context is extended to include the following parameters.

\*'count\_q': a non-negative integer counter, keeping track of the current 'q' value for the Sender Key. At any time, 'count\_q' has as value the number of messages that have been encrypted using the Sender Key. The value of 'count\_q' is set to 0 when establishing the Sender Context.

\*'limit\_q': a non-negative integer, which specifies the highest value that 'count\_q' is allowed to reach, before stopping using the Sender Key to process outgoing messages.

The value of 'limit\_q' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit\_q' is determined according to [Section 2.1.1](#).

### 2.2.3. Recipient Context

The Recipient Context is extended to include the following parameters.

\*'count\_v': a non-negative integer counter, keeping track of the current 'v' value for the Recipient Key. At any time, 'count\_v' has as value the number of failed decryptions occurred on incoming messages using the Recipient Key. The value of 'count\_v' is set to 0 when establishing the Recipient Context.

\*'limit\_v': a non-negative integer, which specifies the highest value that 'count\_v' is allowed to reach, before stopping using the Recipient Key to process incoming messages.

The value of 'limit\_v' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit\_v' is determined according to [Section 2.1.1](#).

## 2.3. OSCORE Messages Processing

In order to keep track of the 'q' and 'v' values and ensure that AEAD keys are not used beyond reaching their limits, the processing of OSCORE messages is extended as defined in this section.

In particular, the processing of OSCORE messages follows the steps outlined in [Section 8](#) of [RFC8613], with the additions defined below.

### 2.3.1. Protecting a Request or a Response

Before encrypting the COSE object using the Sender Key, the 'count\_q' counter MUST be incremented.

If 'count\_q' exceeds the 'limit\_q' limit, the message processing MUST be aborted. From then on, the Sender Key MUST NOT be used to encrypt further messages.

### 2.3.2. Verifying a Request or a Response

If an incoming message is detected to be a replay (see [Section 7.4](#) of [\[RFC8613\]](#)), the 'count\_v' counter MUST NOT be incremented.

If the decryption and verification of the COSE object using the Recipient Key fails, the 'count\_v' counter MUST be incremented.

After 'count\_v' has exceeded the 'limit\_v' limit, incoming messages MUST NOT be decrypted and verified using the Recipient Key, and their processing MUST be aborted.

## 2.4. Current methods for Rekeying OSCORE

Before the limit of 'q' or 'v' has been reached for an OSCORE Security Context, the two peers have to establish a new OSCORE Security Context, in order to continue using OSCORE for secure communication.

In practice, the two peers have to establish new Sender and Recipient Keys, as the keys actually used by the AEAD algorithm. When this happens, both peers reset their 'count\_q' and 'count\_v' values to 0 (see [Section 2.2](#)).

Other specifications define a number of ways to accomplish this.

\*The two peers can run the procedure defined in Appendix B.2 of [\[RFC8613\]](#). That is, the two peers exchange three or four messages, protected with temporary Security Contexts adding randomness to the ID Context.

As a result, the two peers establish a new OSCORE Security Context with new ID Context, Sender Key and Recipient Key, while keeping the same OSCORE Master Secret and OSCORE Master Salt from the old OSCORE Security Context.

This procedure does not require any additional components to what OSCORE already provides, and it does not provide perfect forward secrecy.



\*The two peers can run the OSCORE profile [[I-D.ietf-ace-oscore-profile](#)] of the Authentication and Authorization for Constrained Environments (ACE) Framework [[I-D.ietf-ace-oauth-authz](#)].

When a CoAP client uploads an Access Token to a CoAP server as an access credential, the two peers also exchange two nonces. Then, the two peers use the two nonces together with information provided by the ACE Authorization Server that issued the Access Token, in order to derive an OSCORE Security Context.

This procedure does not provide perfect forward secrecy.

\*The two peers can run the EDHOC key exchange protocol based on Diffie-Hellman and defined in [[I-D.ietf-lake-edhoc](#)], in order to establish a pseudo-random key in a mutually authenticated way.

Then, the two peers can use the established pseudo-random key to derive external application keys. This allows the two peers to securely derive especially an OSCORE Master Secret and an OSCORE Master Salt, from which an OSCORE Security Context can be established.

This procedure additionally provides perfect forward secrecy.

\*If one peer is acting as LwM2M Client and the other peer as LwM2M Server, according to the OMA Lightweight Machine to Machine Core specification [[LwM2M](#)], then the LwM2M Client peer may take the initiative to bootstrap again with the LwM2M Bootstrap Server, and receive again an OSCORE Security Context. Alternatively, the LwM2M Server can instruct the LwM2M Client to initiate this procedure.

If the OSCORE Security Context information on the LwM2M Bootstrap Server has been updated, the LwM2M Client will thus receive a fresh OSCORE Security Context to use with the LwM2M Server.

Manually updating the OSCORE Security Context at the two peers should be a last resort option, and it might often be not practical or feasible.

Even when any of the alternatives mentioned above is available, it is RECOMMENDED that two OSCORE peers update their Security Context by using the procedure defined in [Section 3](#) of this document.

It is RECOMMENDED that the peer initiating the rekeying procedure starts it before reaching the 'q' or 'v' limits. Otherwise, the AEAD keys possibly to be used during the rekeying procedure itself may already be or become invalid before the rekeying is completed, which may prevent a successful establishment of the new OSCORE Security Context altogether.

### 3. New Method for Rekeying OSCORE

This section defines a lightweight method that two OSCORE peers can use to update their keying material and establish a new OSCORE Security Context.

The key update procedure relies on the support function `updateCtx()` defined in [Section 3.2](#) and the message exchange defined in [Section 3.3](#). The following properties are fulfilled.

- \*The procedure can be initiated by either peer. In particular, the client or the server may start the procedure by sending the first rekeying message.
- \*The new OSCORE Security Context enjoys Perfect Forward Secrecy.
- \*The same ID Context value used in the old OSCORE Security Context is preserved in the new Security Context. Furthermore, the ID Context value never changes throughout the procedure.
- \*The procedure is robust against a peer rebooting, and it especially avoids the reuse of AEAD (nonce, key) pairs.
- \*The procedure completes in one round trip. The two peers achieve mutual proof-of-possession in the following exchange, which is protected with the newly established OSCORE Security Context.

#### 3.1. Extensions to the OSCORE Option

In order to support the message exchange for establishing a new OSCORE Security Context as defined in [Section 3.3](#), this document extends the use of the OSCORE option originally defined in [\[RFC8613\]](#) as follows.

- \*This document defines the usage of the seventh least significant bit, called "Extension-1 Flag", in the first byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in [Section 5.1](#).

When the Extension-1 Flag is set to 1, the second byte of the OSCORE option MUST include the set of OSCORE flag bits 8-15.

- \*This document defines the usage of the first least significant bit "ID Detail Flag", 'd', in the second byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in [Section 5.1](#).

When it is set to 1, the compressed COSE object contains an 'id detail', to be used for the key update procedure defined in [Section 3.3](#). In particular, the 1 byte following 'kid context'

(if any) encodes the length  $x$  of 'id detail', and the following  $x$  bytes encode 'id detail'.

\*The second-to-eighth least significant bits in the second byte of the OSCORE option containing the OSCORE flag bits are reserved for future use. These bits SHALL be set to zero when not in use. According to this specification, if any of these bits are set to 1, the message is considered to be malformed and decompression fails as specified in item 2 of [Section 8.2](#) of [\[RFC8613\]](#).

[Figure 3](#) shows the OSCORE option value including also 'id detail'.

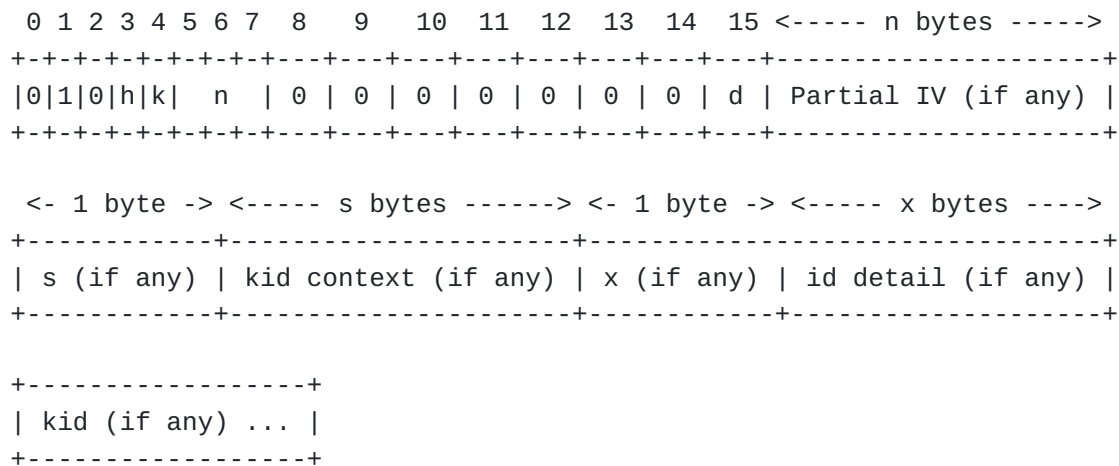


Figure 3: The OSCORE option value, including 'id detail'

### 3.2. Update Function

The `updateCtx()` function shown in [Figure 4](#) takes as input a nonce  $N$  as well as an OSCORE Security Context `CTX_IN`, and returns a new OSCORE Security Context `CTX_OUT`.

First, the `updateCtx()` function derives the new values of the Master Secret and Master Salt for `CTX_OUT` as follows.

\*If the two peers established their original Security Context as a result of running the EDHOC protocol [\[I-D.ietf-lake-edhoc\]](#), the EDHOC key `PRK_4x3m` is updated using the `EDHOC-KeyUpdate()` function, which takes  $N$  as input. Then, the `EDHOC-Exporter()` function is used to derive the new values for the Master Secret and Master Salt. The original Security Context is the Security Context a peer had before the key update procedure was used for the first time.

\*If the two peers established their original Security Context through other means, the new Master Secret is derived through an `HKDF-Expand()` step, which takes as input  $N$  as well as the Master

Secret value from the Security Context CTX\_IN. Instead, the new Master Salt takes N as value.

In either case, the derivation of new values follows the same approach used in TLS 1.3, which is also based on HKDF-Expand (see [Section 7.1](#) of [[RFC8446](#)]) and used for computing new keying material in case of key update (see [Section 4.6.3](#) of [[RFC8446](#)]).

After that, the new Master Secret and Master Salt parameters are used to derive a new Security Context CTX\_OUT as per [Section 3.2](#) of [[RFC8613](#)]. Any other parameter required for the derivation takes the same value as in the Security Context CTX\_IN. Finally, the function returns the newly derived Security Context CTX\_OUT.

```

updateCtx( N, CTX_IN ) {

    CTX_OUT          // The new Security Context
    MSECRET_NEW      // The new Master Secret
    MSALT_NEW        // The new Master Salt

    if <the original Security Context was established through EDHOC> {

        EDHOC-KeyUpdate( N )
        // This results in updating the key PRK_4x3m of the EDHOC session,
        // i.e., PRK_4x3m = Extract( N, PRK_4x3m )

        MSECRET_NEW = EDHOC-Exporter( "OSCORE Master Secret", key_length )
            = EDHOC-KDF( PRK_4x3m, TH_4, "OSCORE Master Secret", key_length )

        MSALT_NEW = EDHOC-Exporter( "OSCORE Master Salt", salt_length )
            = EDHOC-KDF( PRK_4x3m, TH_4, "OSCORE Master Salt", salt_length )

    }
    else {
        Master Secret Length = < Size of CTX_IN.MasterSecret in bytes >

        MSECRET_NEW = HKDF-Expand-Label(CTX_IN.MasterSecret, Label,
                                         N, Master Secret Length)
            = HKDF-Expand(CTX_IN.MasterSecret, HkdfLabel,
                         Master Secret Length)

        MSALT_NEW = N;
    }

    < Derive CTX_OUT using MSECRET_NEW and MSALT_NEW,
      together with other parameters from CTX_IN >

    Return CTX_OUT;

}

```

Where HkdfLabel is defined as

```

struct {
    uint16 length = Length;
    opaque label<7..255> = "oscore " + Label;
    opaque context<0..255> = Context;
} HkdfLabel;

```

Figure 4: Function for deriving a new OSCORE Security Context

### 3.3. Establishment of the New OSCORE Security Context

This section defines the actual message exchange performed by two peers to update their OSCORE keying material. Before starting the key update procedure, the two peers share the OSCORE Security Context CTX\_OLD. Once completed the key update procedure, the two peers agree on a newly established OSCORE Security Context CTX\_NEW.

In particular, each peer contributes by generating a fresh value R1 or R2, and providing it to the other peer. Their concatenation R1|R2 is used as input N by the updateCtx() function, in order to derive the new OSCORE Security Context CTX\_NEW.

Both in the client- and server-initiated version of the procedure:

- \*The initiator always offers the fresh value R1.
- \*The responder always offers the fresh value R2.
- \*The responder is always the first one deriving the new OSCORE Security Context CTX\_NEW.
- \*The initiator is always the first one achieving key confirmation, hence able to safely discard the old OSCORE Security Context CTX\_OLD.
- \*Both the initiator and the responder use the same respective OSCORE Sender ID and Recipient ID. Also, they both preserve and use the same OSCORE ID Context from CTX\_OLD.

[Figure 5](#) shows the key update procedure with the client acting as initiator.

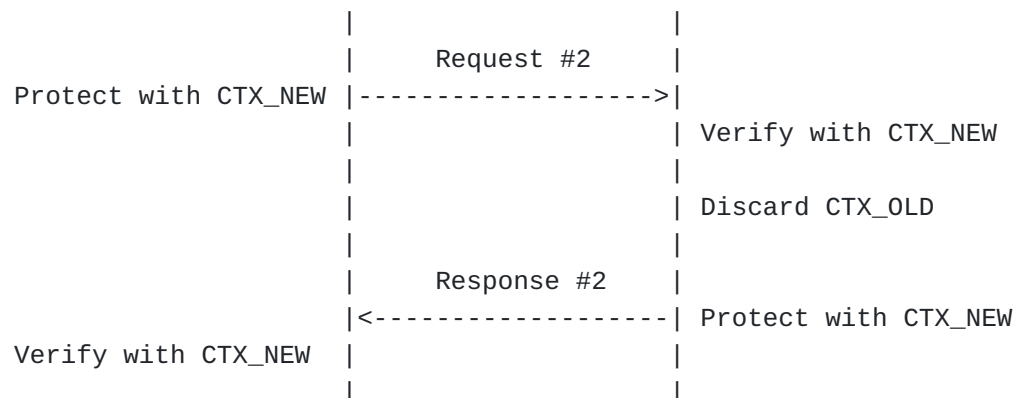
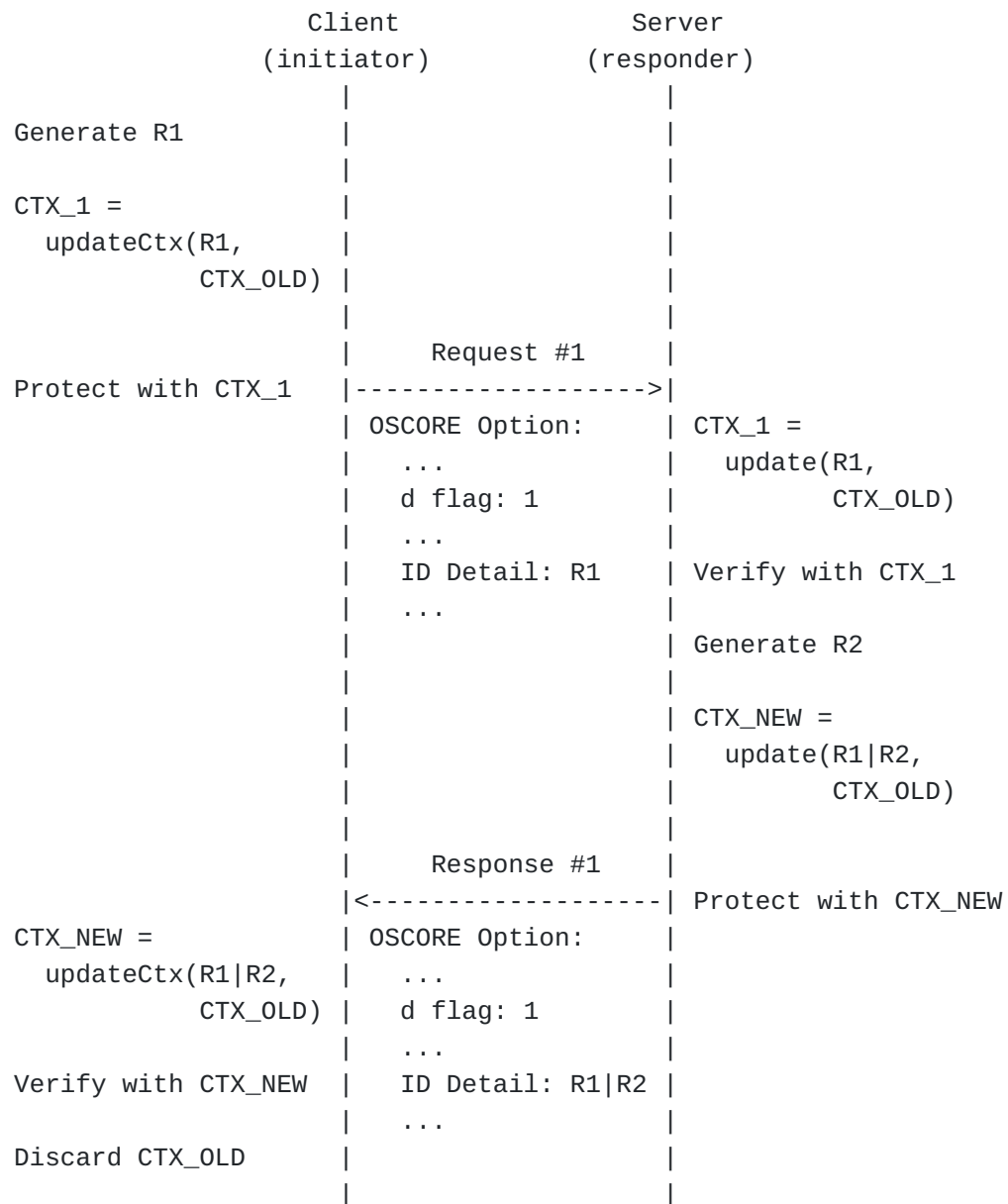


Figure 5: Procedure for Establishing a New Security Context (Client-initiated)

First, the client generates a random value  $R_1$  and uses it together with the old Security Context  $CTX\_OLD$  to derive a temporary Security Context  $CTX\_1$ . Then, the client sends a request to the server, protected with the Security Context  $CTX\_1$ . In particular, the request has the 'd' flag bit set to 1 and specifies  $R_1$  as 'id detail' (see [Section 3.1](#)).

Upon receiving the request, the server derives the temporary Security Context  $CTX\_1$ , by using the value  $R_1$  retrieved from the request and the old Security Context  $CTX\_OLD$ . Then, the server verifies the request by using the Security Context  $CTX\_1$ .

After that, the server generates a random value  $R_2$  and uses the concatenation  $R_1|R_2$  together with the old Security Context  $CTX\_OLD$  to derive the new Security Context  $CTX\_NEW$ . Then, the server, sends a response to the client, protected with the new Security Context  $CTX\_NEW$ . In particular, the response has the 'd' flag bit set to 1 and specifies  $R_1|R_2$  as 'id detail'.

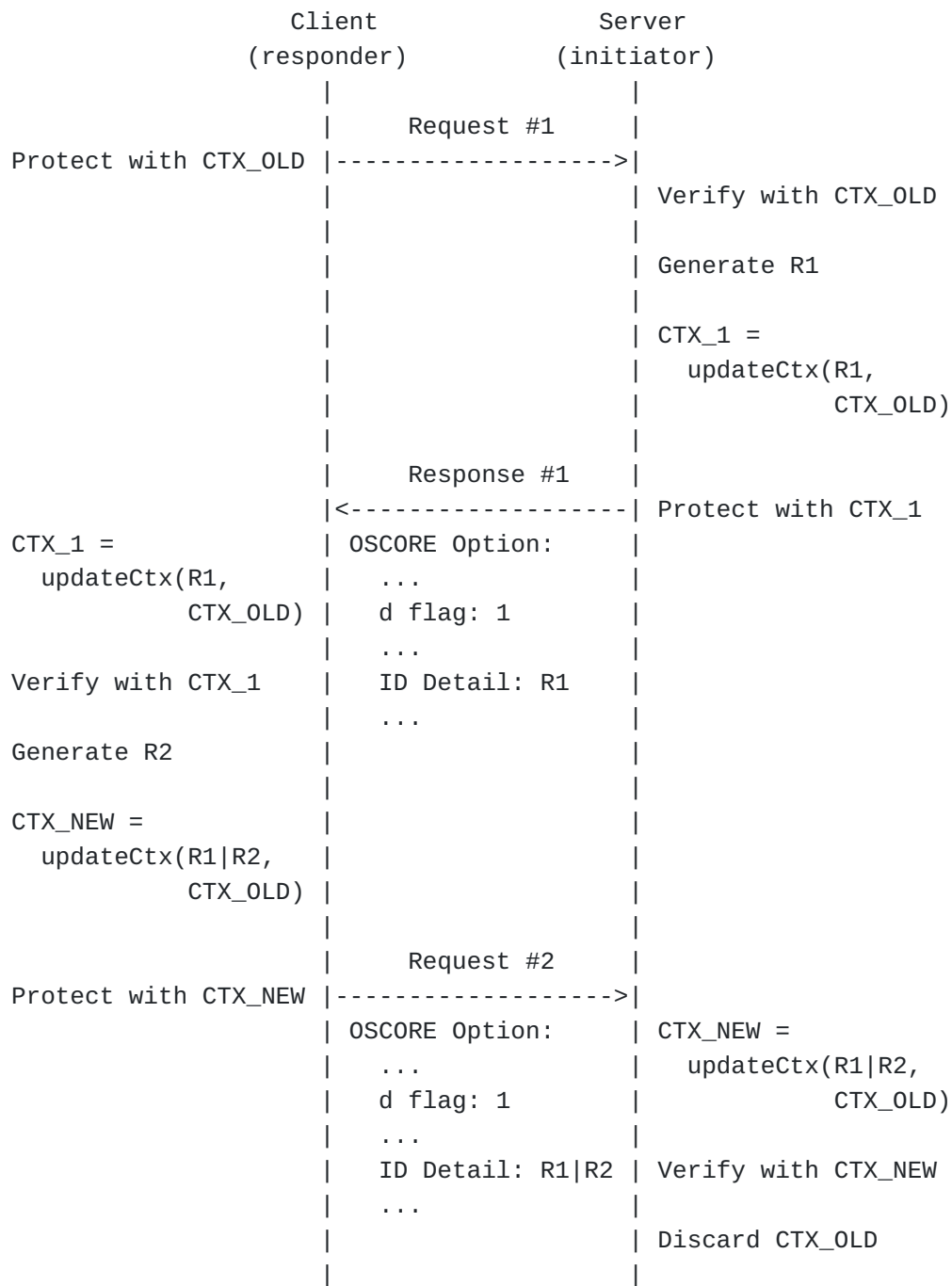
Upon receiving the response, the client retrieves the value  $R_1|R_2$  from the response, and verifies that  $R_1$  coincides with the value  $R_1$  specified in the previously sent request. If so, the client derives the new Security Context  $CTX\_NEW$ , by using the value  $R_1|R_2$  retrieved from the response and the old Security Context  $CTX\_OLD$ . Finally, the client verifies the response by using the Security Context  $CTX\_NEW$  and deletes the old Security Context  $CTX\_OLD$ .

After that, the client can send a new request protected with the new Security Context  $CTX\_NEW$ . When successfully verifying the request using the Security Context  $CTX\_NEW$ , the server deletes the old Security Context  $CTX\_OLD$  and can reply with a response protected with the new Security Context  $CTX\_NEW$ .

From then on, the two peers can protect their message exchanges by using the new Security Context  $CTX\_NEW$ .

[Figure 6](#) shows the key update procedure with the server acting as initiator.





// The actual key update ends here

// The two peers can use the new Security Context CTX\_NEW

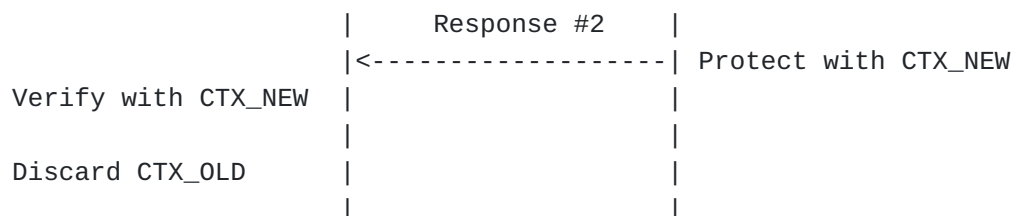


Figure 6: Procedure for Establishing a New Security Context (Server-initiated)

First, the client sends a normal request to the server, protected with the old Security Context CTX\_OLD.

Upon receiving the request and after having verified it with the old Security Context CTX\_OLD as usual, the server generates a random value R1 and uses it together with the old Security Context CTX\_OLD to derive a temporary Security Context CTX\_1. Then, the server sends a response to the client, protected with the Security Context CTX\_1. In particular, the request has the 'd' flag bit set to 1 and specifies R1 as 'id detail' (see [Section 3.1](#)).

Upon receiving the response, the client derives the temporary Security Context CTX\_1, by using the value R1 retrieved from the response and the old Security Context CTX\_OLD. Then, the client verifies the response by using the Security Context CTX\_1.

After that, the client generates a random value R2 and uses the concatenation R1|R2 together with the old Security Context CTX\_OLD to derive the new Security Context CTX\_NEW. Then, the client, sends a request to the server, protected with the new Security Context CTX\_NEW. In particular, the request has the 'd' flag bit set to 1 and specifies R1|R2 as 'id detail'.

Upon receiving the request, the server retrieves the value R1|R2 from the request, and verifies that R1 coincides with the value R1 specified in the previously sent response. If so, the server derives the new Security Context CTX\_NEW, by using the value R1|R2 retrieved from the request and the old Security Context CTX\_OLD. Finally, the server verifies the request by using the Security Context CTX\_NEW and deletes the old Security Context CTX\_OLD.

After that, the client can send a response protected with the new Security Context CTX\_NEW. When successfully verifying the response using the Security Context CTX\_NEW, the client deletes the old Security Context CTX\_OLD.

From then on, the two peers can protect their message exchanges by using the new Security Context CTX\_NEW.

### 3.4. Retention Policies

Applications MAY define policies that allows a peer to also temporarily keep the old Security Context CTX\_OLD, rather than simply overwriting it to become CTX\_NEW. This allows the peer to decrypt late, still on-the-fly incoming messages protected with CTX\_OLD.

When enforcing such policies, the following applies.

\*Outgoing messages MUST be protected by using only CTX\_NEW.

\*Incoming messages MUST first be attempted to decrypt by using CTX\_NEW. If decryption fails, a second attempt can use CTX\_OLD.

\*When an amount of time defined by the policy has elapsed since the establishment of CTX\_NEW, the peer deletes CTX\_OLD.

#### 4. Security Considerations

This document mainly covers security considerations about using AEAD keys in OSCORE and their usage limits, in addition to the security considerations of [\[RFC8613\]](#).

Depending on the specific rekeying procedure used to establish a new OSCORE Security Context, the related security considerations also apply.

TODO: Add more considerations.

#### 5. IANA Considerations

This document has the following actions for IANA.

##### 5.1. OSCORE Flag Bits Registry

IANA is asked to add the following value entries to the "OSCORE Flag Bits" subregistry as part of the "CoRE Parameters" registry.

Bit Position	Name	Description	Reference
1	Extension-1 Flag	Set to 1 if the OSCORE Option specifies a second byte of OSCORE flag bits	[This Document]
15	ID Detail Flag	Set to 1 if the compressed COSE object contains 'id detail'	[This Document]

#### 6. References

##### 6.1. Normative References

[\[RFC2119\]](#)

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

## 6.2. Informative References

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-43, 10 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-43.txt>>.

[I-D.ietf-ace-oscore-profile] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

[I-D.ietf-lake-edhoc] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-07, 24 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-07.txt>>.

[I-D.irtf-cfrg-aead-limits] Günther, F., Thomson, M., and C. A. Wood, "Usage Limits on AEAD Algorithms", Work in Progress, Internet-Draft, draft-irtf-cfrg-aead-limits-02,

22 February 2021, <<https://www.ietf.org/archive/id/draft-irtf-cfrg-aead-limits-02.txt>>.

- [LwM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Core, Approved Version 1.2, OMA-TS-LightweightM2M\_Core-V1\_2-20201110-A", November 2020, <[http://www.openmobilealliance.org/release/LightweightM2M/V1\\_2-20201110-A/OMA-TS-LightweightM2M\\_Core-V1\\_2-20201110-A.pdf](http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf)>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## Acknowledgments

The authors sincerely thank Christian Amsuess, John Mattsson and Goeran Selander for the initial discussions that allowed shaping this document.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

## Authors' Addresses

Rikard Höglund  
RISE AB  
Isafjordsgatan 22  
SE-16440 Stockholm Kista  
Sweden

Email: [rikard.hoglund@ri.se](mailto:rikard.hoglund@ri.se)

Marco Tiloca  
RISE AB  
Isafjordsgatan 22  
SE-16440 Stockholm Kista  
Sweden

Email: [marco.tiloca@ri.se](mailto:marco.tiloca@ri.se)