CoRE Working Group
Internet-Draft
Updates: 8613 (if approved)
Intended status: Standards Track
Expires: 8 January 2026

R. H旦glund
M. Tiloca
RISE AB

7 July 2025

Key Update for OSCORE (KUDOS)
draft-ietf-core-oscore-key-update-11

Abstract

Communications with the Constrained Application Protocol (CoAP) can be protected end-to-end at the application-layer by using the security protocol Object Security for Constrained RESTful Environments (OSCORE). Under some circumstances, two CoAP endpoints need to update their OSCORE keying material before communications can securely continue, e.g., due to approaching key usage limits. This document defines Key Update for OSCORE (KUDOS), a lightweight key update procedure that two CoAP endpoints can use to update their OSCORE keying material by establishing a new OSCORE Security Context. Accordingly, this document updates the use of the OSCORE flag bits in the CoAP OSCORE Option as well as the protection of CoAP response messages with OSCORE. Also, it deprecates the key update procedure specified in Appendix B.2 of RFC 8613. Therefore, this document updates RFC 8613.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/oscore-key-update.

Status of This Memo

This Internet-Draft will expire on 8 January 2026.

Table of Contents

1. Introduction

The security protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end protection at the application-layer for messages exchanged with the Constrained Application Protocol (CoAP) [RFC7252]. In particular, OSCORE ensures message confidentiality and integrity, replay protection, and binding of response to request between a sender and a recipient.

Under some circumstances, two CoAP endpoints using OSCORE may need to update their shared keying material in order to ensure the security of their communications. Among other reasons, approaching key usage

limits [I-D.irtf-cfrg-aead-limits][I-D.ietf-core-oscore-key-limits] requires updating the OSCORE keying material before communications can securely continue.

This document defines Key Update for OSCORE (KUDOS), a lightweight key update procedure that two CoAP endpoints can use to update their OSCORE keying material by establishing a new OSCORE Security Context.

Accordingly, this document updates [RFC8613] as follows:

* With reference to the "OSCORE Flag Bits" registry defined in Section 13.7 of [RFC8613] as part of the "Constrained RESTful Environments (CoRE) Parameters" registry group, it updates the entries with Bit Position 0 and 1 (see Section 6), both of which were originally marked as "Reserved".

  In particular, it defines and registers the usage of the OSCORE flag bit with Bit Position 0, as the one intended to expand the space for the OSCORE flag bits in the OSCORE Option (see Section 4.1). Also, it marks the bit with Bit Position of 1 as "Unassigned".

* It updates the protection of CoAP responses with OSCORE that was originally specified in Section 8.3 of [RFC8613], as defined in Section 3 of this document.

* It deprecates the key update procedure specified in Appendix B.2 of [RFC8613], as intended to be superseded by KUDOS.

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 CoAP [RFC7252], Observe [RFC7641], Concise Binary Object Representation (CBOR) [RFC8949], OSCORE [RFC8613], and Ephemeral Diffie-Hellman Over COSE (EDHOC) [RFC9528].

This document additionally defines the following terminology.

* FS mode: the KUDOS execution mode that achieves forward secrecy (see Section 4.3).

* No-FS mode: the KUDOS execution mode that does not achieve forward secrecy (see Section 4.4).

* Equilibrium: The situation where a peer has no execution of KUDOS currently ongoing with the other KUDOS peer for updating one of their OSCORE Security Contexts.

2. Current Methods for Rekeying OSCORE

Two peers communicating using OSCORE may choose to renew their shared keying information by establishing a new OSCORE Security Context for a variety of reasons. A particular reason is the approaching of limits that have been set for safe key usage [I-D.ietf-core-oscore-key-limits]. Practically, when the relevant limits are 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. That is, the two peers have to establish new Sender and Recipient Keys, as the keys actually used by the AEAD algorithm.

In addition to the approaching of key usage limits, there may be other reasons for a peer to initiate a key update procedure. These include: the OSCORE Security Context approaching its expiration time; application policies prescribing a regular key rollover; approaching the exhaustion of the Sender Sequence Number space in the OSCORE Sender Context.

It is RECOMMENDED that the peer initiating the key update procedure starts it with some margin, i.e., well before actually experiencing the trigger event that forces to perform a key update (e.g., the OSCORE Security Context expiration or the exhaustion of the Sender Sequence Number space). If the rekeying is not initiated ahead of these events, it may become practically impossible to perform a key update with certain methods and/or without aborting ongoing message exchanges.

Other specifications define a number of ways for rekeying OSCORE, which are summarized below.

* The two peers can run the procedure defined in Appendix B.2 of [RFC8613]. That is, the two peers exchange three or four messages that are protected with temporary Security Contexts, thus adding randomness to the OSCORE 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 forward secrecy.

The procedure defined in Appendix B.2 of [RFC8613] is used in 6TiSCH networks [RFC7554][RFC8180] when handling failure events. That is, a node acting as Join Registrar/Coordinator (JRC) assists new devices, namely "pledges", to securely join the network as per the Constrained Join Protocol [RFC9031]. In particular, a pledge exchanges OSCORE-protected messages with the JRC, from which it obtains a short identifier, link-layer keying material and other configuration parameters. As per Section 8.3.3 of [RFC9031], a JRC that experiences a failure event may likely lose information about joined nodes, including their assigned identifiers. Then, the reinitialized JRC can establish a new OSCORE Security Context with each pledge, through the procedure defined in Appendix B.2 of [RFC8613].

* The two peers can use the OSCORE profile [RFC9203] of the Authentication and Authorization for Constrained Environments (ACE) Framework [RFC9200].

  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 forward secrecy.

* The two peers can run the EDHOC key exchange protocol based on Diffie-Hellman and defined in [RFC9528], in order to establish a shared pseudo-random secret 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 an OSCORE Master Secret and an OSCORE Master Salt, from which an OSCORE Security Context can be established.

  This procedure additionally provides forward secrecy.

  EDHOC also specifies an optional function, namely EDHOC_KeyUpdate, to perform a key update in a more efficient way than re-running EDHOC. The two communicating peers call EDHOC_KeyUpdate with equivalent input, which results in the derivation of a new shared pseudo-random secret key. Usage of EDHOC_KeyUpdate preserves forward secrecy.

Note that EDHOC may be run standalone or as part of other workflows, such as when using the EDHOC and OSCORE profile of ACE [I-D.ietf-ace-edhoc-oscore-profile].

* 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.

  The LwM2M Client, the LwM2M Server, and the LwM2M Bootstrap server are also required to use the procedure defined in Appendix B.2 of [RFC8613] and overviewed above, when they use a certain OSCORE Security Context for the first time [LwM2M-Transport].

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 KUDOS procedure as defined in Section 4 of this document.

3. Updated Protection of Responses with OSCORE

The protection of CoAP responses with OSCORE is updated, by adding the following text at the end of step 3 of Section 8.3 of [RFC8613] as well as before the text "Then, go to 4." of Section 8.3.1 of [RFC8613].

> If the server is using a different Security Context for the response compared to what was used to verify the request (e.g., due to an occurred key update), then the server MUST take the second alternative. That is, the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response.
>
> This prevents the server from using the same AEAD (key, nonce) pair for two responses, protected with different OSCORE Security Contexts.

| An exception is the procedure in Appendix B.2 of [RFC8613], which
| is secure although not complying with the above. The reason is
| that, in that procedure, the server uses the new OSCORE Security
| Context only and solely to protect the outgoing response (response
| #1), and no other message is protected with that OSCORE Security
| Context. Other procedures where that holds would also remain
| secure.

4. Key Update for OSCORE (KUDOS)

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

Hereafter, this document refers to two specific peers that run KUDOS to update specifically one OSCORE Security Context that they share with each other.

KUDOS relies on the OSCORE Option defined in [RFC8613] and extended as defined in Section 4.1, as well as on the support function updateCtx() defined in Section 4.2.

In order to run KUDOS, two peers exchange OSCORE-protected CoAP messages. The key update procedure is described in detail in Section 4.3, with particular reference to the stateful FS mode providing forward secrecy. The possible use of the stateless no-FS mode is described in Section 4.4, as intended to peers that are not able to write in non-volatile memory. Two peers MUST run KUDOS in FS mode if they are both capable to do so.

The key update procedure has the following properties:

* It can be initiated by either peer.

* The new OSCORE Security Context enjoys forward secrecy, unless the no-FS mode is used (see Section 4.4).

* The same ID Context value used in the old OSCORE Security Context (if set) is preserved in the new OSCORE Security Context.

* The same Sender and Recipient IDs used in the old OSCORE Security Context are preserved in the new OSCORE Security Context.

* It is robust against a peer rebooting and loss of state, avoiding the reuse of AEAD (nonce, key) pairs also in such cases.

* It typically completes in one round trip by exchanging two OSCORE-protected CoAP messages. The two peers achieve mutual key confirmation in a following exchange, which is protected with the newly established OSCORE Security Context.

4.1. Extensions to the OSCORE Option

This document extends the use of the OSCORE Option originally defined in [RFC8613] as follows:

* This document defines the usage of the eight least significant bit, called "Extension-1 Flag", in the first byte of the OSCORE Option containing the OSCORE flag bits. The registration of this flag bit in the "OSCORE Flag Bits" registry is specified in Section 6.1.

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

* This document defines the usage of the least significant bit "Nonce Flag", 'd', in the second byte of the OSCORE Option containing the OSCORE flag bits 8-15. The registration of this flag bit in the "OSCORE Flag Bits" registry is specified in Section 6.1.

  When it is set to 1, the compressed COSE object contains a field 'x' and a field 'nonce', to be used for the steps defined in Section 4.3. In particular, the 1 byte 'x' following 'kid context' (if any) includes the size of the following field 'nonce' as well as a number of signaling bits that indicate the specific behavior to adopt during the KUDOS execution.

  Hereafter, a message is referred to as a "KUDOS message", if and only if the second byte of flags is present and the 'd' bit is set to 1. If that is not the case, the message is referred to as a "non KUDOS message".

  The encoding of the 'x' field is as follows:

  - The four least significant bits encode the 'nonce' size in bytes minus 1, namely 'm'.

  - The fifth least significant bit is the "No Forward Secrecy" 'p' bit. The sender peer indicates its wish to run KUDOS in FS mode or in no-FS mode, by setting the 'p' bit to 0 or 1, respectively.

This makes KUDOS possible to run also for peers that cannot support the FS mode. At the same time, two peers MUST run KUDOS in FS mode if they are both capable to do so, as per Section 4.3. The execution of KUDOS in no-FS mode is defined in Section 4.4.

- The sixth least significant bit is the "Preserve Observations" ’b’ bit. The sender peer indicates its wish to preserve or terminate the ongoing observations with the other peer beyond the KUDOS execution, by setting the ’b’ bit to 1 or 0, respectively. The related processing is defined in Section 4.5.

- The seventh least significant bit is the ’z’ bit, which has the following meaning:

  - If the bit ’z’ is set to 0, the present message is a "divergent" KUDOS message, i.e., it is protected with a temporary OSCORE Security Context and indicates that the sender peer is moving away from "equilibrium".

    That is, the sender peer is offering its own nonce in the ’nonce’ field of the message and is waiting to receive the other peer’s nonce.

  - If the bit ’z’ is set to 1, the present message is a "convergent" KUDOS message, i.e., it is protected with the newly established OSCORE Security Context and indicates that the sender peer is attempting to return to "equilibrium".

    That is, the sender peer is offering its own nonce in the ’nonce’ field of the message, has received the other peer’s nonce, and is going to wait for key confirmation (as a pre-condition to return to equilibrium).

- The eight least significant bit is reserved for future use. This bit SHALL be set to 0 when not in use. According to this specification, if this bit is set to 1, the following applies:

  - If the message is a request, it is considered to be malformed and decompression fails as specified in item 2 of Section 8.2 of [RFC8613].

  - If the message is a response, it is considered to be malformed, decompression fails as specified in item 2 of Section 8.4 of [RFC8613], and the client SHALL discard the response as specified in item 8 of Section 8.4 of [RFC8613].

Figure 1 shows the extended OSCORE Option value, with the presence of the 'x' and 'nonce' fields.

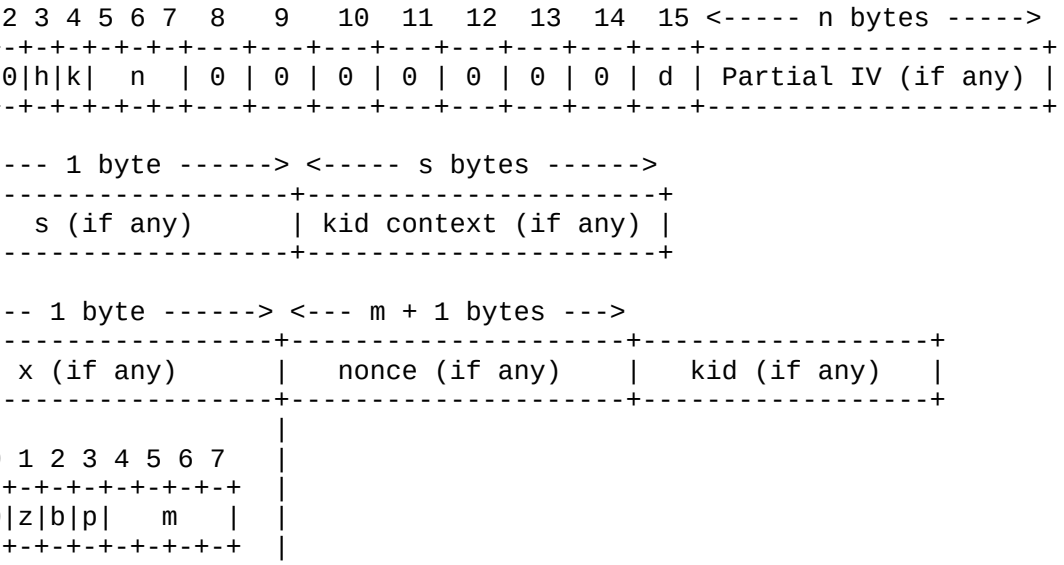


Figure 1: The Extended OSCORE Option Value

4.2. Function for Security Context Update

This section defines the updateCtx() function shown in Figure 2, which takes as input the three parameters input1, input2, and CTX_IN.

In particular, input1 and input2 are built from the 'x' and 'nonce' fields transported in the OSCORE Option value of the exchanged KUDOS messages (see Section 4.1), while CTX_IN is the OSCORE Security Context to update through the KUDOS execution. The function returns a new OSCORE Security Context CTX_OUT.

```
function updateCtx(input1, input2, CTX_IN):

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

  // Define the label for the key update
  Label = "key update"

  // Create CBOR byte strings from input1 and input2
  input1_cbor = create_cbor_bstr(input1)
```

```
  input2_cbor = create_cbor_bstr(input2)

  // Concatenate the CBOR-encoded input1 and input2
  // according to their lexicographic order
  X_N = is_lexicographically_shorter(input1_cbor, input2_cbor) ?
        (input1_cbor | input2_cbor) : (input2_cbor | input1_cbor)

  // Set the new Master Salt to X_N
  MSALT_NEW = X_N

  // Determine the length in bytes of the current Master Secret
  oscore_key_length = length(CTX_IN.MasterSecret)

  // Create the new Master Secret using KUDOS_Expand_Label
  MSECRET_NEW = KUDOS_Expand_Label(CTX_IN.MasterSecret, Label,
                                   X_N, oscore_key_length)

  // Derive the new Security Context CTX_OUT, using
  // the new Master Secret, the new Master Salt,
  // and other parameters from CTX_IN
  CTX_OUT = derive_security_context(MSECRET_NEW, MSALT_NEW, CTX_IN)

  // Return the new Security Context
  return CTX_OUT

=== === === === === === === === === === === === === === === === === ===

 function KUDOS_Expand_Label(master_secret, Label, X_N, key_length):

  struct {
      uint16 length = key_length;
      opaque label<7..255> = "oscore " + Label;
      opaque context<0..255> = X_N;
  } ExpandLabel;

  return KUDOS_Expand(master_secret, ExpandLabel, key_length)
```

Figure 2: Functions for Deriving a new OSCORE Security Context

The updateCtx() function builds the two CBOR byte strings input1_cbor and input2_cbor, whose values are the input parameter input1 and input2, respectively. Then, it builds X_N as the byte concatenation of input1_cbor and input2_cbor.

In order to be agnostic of the order according to which the nonce values were exchanged, the binary representations of input1_cbor and input2_cbor are sorted in lexicographical order before they are concatenated. That is, with | denoting byte concatenation, X_N MUST

take input1_cbor | input2_cbor if input1_cbor comes before input2_cbor in lexicographical order, or input2_cbor | input1_cbor otherwise.

After that, the updateCtx() function derives the new values of the Master Salt and Master Secret for the Security Context CTX_OUT. In particular, the new Master Salt takes X_N as value. Instead, the new Master Secret is derived through a KUDOS-Expand step that is invoked through the KUDOS_Expand_Label() function. The latter takes as input the Master Secret value from the Security Context CTX_IN, the literal string "key update", X_N, and the length of the Master Secret in bytes.

The definition of KUDOS-Expand depends on the key derivation function used for OSCORE by the two peers, as specified in CTX_IN. If the key derivation function is an HKDF Algorithm (see Section 3.1 of [RFC8613]), then KUDOS-Expand is mapped to HKDF-Expand [RFC5869], as shown below. In particular, the hash algorithm is the same one used by the HKDF Algorithm specified in CTX_IN.

```
KUDOS-Expand(CTX_IN.MasterSecret, ExpandLabel, key_length) =
   HKDF-Expand(CTX_IN.MasterSecret, ExpandLabel, key_length)
```

If a future specification updates [RFC8613] by admitting different key derivation functions than HKDF Algorithms (e.g., KMAC as based on the SHAKE128 or SHAKE256 hash functions), that specification has to also update the present document in order to define the mapping between such key derivation functions and KUDOS-Expand.

When an HKDF Algorithm is used, 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 is used for computing new keying material in case of key update (see Section 4.6.3 of [RFC8446]).

After that, the newly computed Master Secret and Master Salt are used to derive a new Security Context CTX_OUT, as per Section 3.2 of [RFC8613]. Any other parameter required for that derivation takes the same value as in the Security Context CTX_IN.

Note that the following holds for the newly derived CTX_OUT:

* In its Sender Context, the Sender Sequence Number is initialized to 0 as per Section 3.2.2 of [RFC8613].

* If the peer that has derived CTX_OUT supports CoAP Observe [RFC7641], the Notification Number used for the replay protection of Observe notifications (see Section 7.4.1 of [RFC8613]) is left as not initialized.

Finally, the updateCtx() function returns the newly derived Security Context CTX_OUT.

Note that, thanks to the input parameters input1 and input2 provided to the updateCtx() function, the derivation of CTX_OUT also considers as input the information from the 'x' field conveyed in the OSCORE Option value of the exchanged KUDOS messages. In turn, this ensures that a successfully completed KUDOS execution has occurred as intended by the two peers.

4.3. Key Update

When using KUDOS as described in this section, forward secrecy is achieved for the new OSCORE keying material that results from the KUDOS execution, as long as the original OSCORE keying material was also established with forward secrecy. For peers that are unable to store information to persistent memory, Section 4.4 provides an alternative approach that does not achieve forward secrecy but allows also such very constrained peers to perform a key update.

A peer can run KUDOS for active rekeying at any time, or for a variety of more compelling reasons. These include the (approaching) expiration of the OSCORE Security Context, approaching the limits for the corresponding key usage [I-D.ietf-core-oscore-key-limits], the enforcement of application policies, and the (approaching) exhaustion of the OSCORE Sender Sequence Number space.

The expiration time of an OSCORE Security Context and the key usage limits are hard limits. Once reached them, a peer MUST stop using the keying material in the OSCORE Security Context for exchanging application data with the other peer and has to perform a rekeying before resuming secure communication.

Before starting KUDOS, the two peers share the OSCORE Security Context CTX_OLD. In particular, CTX_OLD is the most recent OSCORE Security Context that a peer has with the other peer, before initiating the KUDOS procedure or upon having received and successfully verified a divergent KUDOS message. During an execution of KUDOS, a temporary OSCORE Security Context CTX_TEMP is also derived.

Once successfully completed a KUDOS execution, the two peers agree on a newly established OSCORE Security Context CTX_NEW that replaces CTX_OLD. In particular, CTX_NEW is the most recent OSCORE Security Context that a peer has with the other peer, before sending a convergent KUDOS message to the other peer or upon having received and successfully verified a convergent KUDOS message from that other peer. CTX_OLD can be safely deleted upon receiving key confirmation from the other peer, i.e., upon gaining knowledge that the other peer also has CTX_NEW.

The following specifically defines how KUDOS is run in its stateful FS mode achieving forward secrecy. That is, in the OSCORE Option value of all the exchanged KUDOS messages, the "No Forward Secrecy" 'p' bit is set to 0.

In order to run KUDOS in FS mode, both peers have to be able to write in non-volatile memory. From the newly derived Security Context CTX_NEW, the peers MUST store to non-volatile memory the immutable parts of the OSCORE Security Context as specified in Section 3.1 of [RFC8613], with the possible exception of the Common IV, Sender Key, and Recipient Key that can be derived again when needed, as specified in Section 3.2.1 of [RFC8613]. If either or both peers are unable to write in non-volatile memory, the two peers have to run KUDOS in its stateless no-FS mode (see Section 4.4).

4.3.1. Nonces and X Bytes

When running KUDOS, each peer contributes by generating a nonce value N1 or N2 and providing it to the other peer. The size of the nonces N1 and N2 is application specific and the use of 8 byte nonce values is RECOMMENDED. The nonces N1 and N2 MUST be random values, with the possible exception described later in Section 4.4.1. Note that a good amount of randomness is important for the nonce generation. [RFC4086] provides guidance on the generation of random values.

In the following, X1 and X2 denote the value of the 'x' field specified in the OSCORE Option value of a KUDOS message. From one peer's point of view, X1 and N1 are generated by that peer, while X2 and N2 are obtained from the other peer.

In a KUDOS message, the sender peer sets the X1 value based on: the length of N1 in bytes, the specific settings that the peer wishes to run KUDOS with, and the message being divergent or convergent. During the same KUDOS execution, all the KUDOS messages sent by a peer MUST have the same value of the bit 'b' in the 'x' field conveying X1.

N1, N2, X1, and X2 are used by the peers to build the 'input1' and 'input2' values provided as input to the updateCtx() function, in order to derive an OSCORE Security Context (see Section 4.2). As for any newly derived OSCORE Security Context, the Sender Sequence Number and the Replay Window are re-initialized accordingly (see Section 3.2.2 of [RFC8613]).

Specifically, the input to updateCtx() is built as follows, where | denotes byte concatenation:

* When deriving CTX_TEMP to protect a divergent outgoing message, input1 is X1 | N1 and input2 is 0x.

* When deriving CTX_TEMP to unprotect a divergent incoming message, input1 is X2 | N2 and input2 is 0x.

* When deriving CTX_NEW to protect or unprotect a convergent message, input1 is X1 | N1 and input2 is X2 | N2.

For any divergent KUDOS message, the sender peer's (X, nonce) are included in the 'x' and 'nonce' field of the OSCORE Option value, respectively. Also, both peers use those as input to updateCtx() for deriving CTX_TEMP, which is used to protect and unprotect the KUDOS message.

For any convergent KUDOS message, the sender peer's (X, nonce) are included in the 'x' and 'nonce' field of the OSCORE Option value, respectively. Both the sender peer's (X, nonce) and the recipient peer's (X, nonce) are used as input to updateCtx() for deriving CTX_NEW, which is used to protect and unprotect the KUDOS message.

A pair (X, nonce) offered by a peer is bound to CTX_OLD, and is reused as much as possible during the same KUDOS execution. A peer generates its (X, nonce) pair before invoking updateCtx(), if the peer does not have one such pair already associated with the CTX_OLD to use as input CTX_IN for updateCtx(). The peer associates the newly generated pair with CTX_OLD before entering updateCtx().

4.3.2. KUDOS States

A peer performs a KUDOS execution according to the state machine specified in Section 4.3.3, where the following states are considered.

The peer can be in three possible states: IDLE, BUSY, and PENDING.

Normally, the peer is in the IDLE state, i.e., in "equilibrium". The peer starts a KUDOS execution upon entering the BUSY state from a state different than BUSY. The peer succesfully completes a KUDOS execution by entering the IDLE state, at which point the peer has the OSCORE Security Context CTX_NEW and has achieved key confirmation.

The sending of a KUDOS message is per the KUDOS state machine and is based on the perception that the sender peer has about the state of the other peer.

The processing of a received KUDOS message is per the KUDOS state machine and is based on the local status of the recipient peer. Moving to a state due to a received KUDOS message occurs only in case of successful decryption and verification of the message with OSCORE.

In its local status, a peer tracks its current KUDOS state by means of the bits (c_tx, c_rx) as follows:

* (00) IDLE - The peer is not running KUDOS.

* BUSY - The peer is running KUDOS and:

  - (01) has not offered a nonce, but has received the nonce from the other peer; or

  - (10) has offered a nonce, but has not received the nonce from the other peer.

* (11) PENDING: the peer is running KUDOS, has offered its nonce, has received the nonce from the other peer, and is waiting for key confirmation.

While in the *BUSY* or the *PENDING* state, the peer MUST NOT send non KUDOS messages.

4.3.3. KUDOS State Machine

From a peer’s point of view, KUDOS states evolve as follows.

4.3.3.1. Startup

At startup, the peer enters a *Pre-IDLE* stage.

4.3.3.2. Pre-IDLE Stage

Upon entering the *Pre-IDLE* stage, perform the following steps:

1. If the peer has any CTX_TEMP Security Contexts, delete them.

2. If the peer has both an old and a new OSCORE Security Context:

    * Delete the (X, nonce) pair associated with the old OSCORE Security Context.

    * Delete the old OSCORE Security Context, or retain it only for processing late incoming messages as allowed by retention policies (see Section 4.6).

3. Move to *IDLE*.

4.3.3.3. IDLE

While in *IDLE*, the following applies:

* Upon receiving a divergent message, move to *BUSY*.

* Upon sending a divergent message, move to *BUSY*.

* Upon receiving a convergent message:

    1. Ignore the message for the sake of KUDOS processing, but process it as a CoAP message.

    2. Stay in *IDLE*.

4.3.3.4. BUSY

Upon entering *BUSY* due to receiving a divergent message:

1. Send a convergent message.

2. Move to *PENDING*.

While in *BUSY*, the following applies:

* Upon receiving a divergent message:

    1. Send a convergent message.

    2. Move to *PENDING*.

* Upon receiving a convergent message:

    1. Achieve key confirmation.

    2. Move to the *Pre-IDLE* stage.

* Upon sending a divergent message:

  - If, as in most cases, CTX_TEMP is usable to protect the intended divergent message:

    1. Send the message protected with CTX_TEMP.

    2. Stay in *BUSY*.

  - Otherwise, perform the following steps (e.g., this happens upon eventually exhausting the Sender Sequence Number space of CTX_TEMP):

    1. Delete CTX_TEMP.

    2. Delete the (X, nonce) pair associated with the Security Context CTX_IN that was used to generate the CTX_TEMP deleted at the previous step.

    3. Generate a new (X, nonce) pair and associate it with CTX_IN.

    4. Generate a new CTX_TEMP from CTX_IN.

    5. Send the message protected with the CTX_TEMP generated at the previous step.

    6. Stay in *BUSY*.

4.3.3.5. Pending

While in *PENDING*, the following applies:

* Upon needing to send a message (e.g., the application wants to send a request):

  1. Send the message as a convergent message.

  2. Stay in *PENDING*.

* Upon receiving a non KUDOS message protected with the latest CTX_NEW:

  1. Achieve key confirmation.

  2. Move to the *Pre-IDLE* stage.

* Upon receiving a convergent message:

1. Achieve key confirmation.

2. Move to the *Pre-IDLE* stage.

* Upon receiving a divergent message:

  - In case of successful decryption and verification of the message using a CTX_TEMP derived from CTX_OLD:

    1. Delete CTX_NEW.

    2. Delete the pair (X, nonce) associated with the Security Context CTX_IN that was used to generate the CTX_NEW deleted at the previous step.

    3. Abort the ongoing KUDOS execution.

    4. Move to *BUSY* and enter it consistently with the reception of a divergent message.

  - Otherwise, in case of successful decryption and verification of the message using a CTX_TEMP derived from CTX_NEW:

    1. Delete the oldest CTX_TEMP.

    2. Delete the Security Context that was used as CTX_IN to generate the CTX_TEMP deleted at the previous step.

    3. CTX_NEW becomes the oldest Security Context. From this point on, that Security Context is what this KUDOS execution refers to as CTX_OLD.

    4. Abort the ongoing KUDOS execution.

    5. Move to *BUSY* and enter it consistently with the reception of a divergent message.

4.3.4. Optimization upon Receiving a Divergent Message while in PENDING

When a peer is in the *PENDING* state and receives a divergent message, an optimization can be applied to avoid unnecessary state transitions and cryptographic derivations. It builds on comparing the just received divergent message MSG_A with a previously received divergent message MSG_B that originally caused the latest transition to *PENDING* or *BUSY*. Normally the reception of MSG_A would move the peer to *BUSY* state.

If the two messages MSG_A and MSG_B contain the same X byte and Nonce from the other peer, then the peer stays in *PENDING*. Otherwise, the peer moves to *BUSY* upon reception of the divergent message MSG_\A, as normal.

If upon reception of MSG_A, CTX_NEW is not usable to protect outgoing messages (e.g., this happens upon eventually exhausting the Sender Sequence Number values of CTX_TEMP), the peer moves to *BUSY*, as normal.

This optimization avoids repeated cryptographic operations and redundant transitions in the state machine when divergent messages originate from the same peer and carry identical X byte and Nonce.

To determine whether message MSG_A and MSG_B are equivalent, the peer MUST:

1. Decompose the Master Salt from the current CTX_NEW into its CBOR byte string components, as described in Section 4.2. Then identify:

   * The component containing the peer's own X and Nonce, i.e., the (X, nonce) pair associated with the Security Context CTX_IN that was used to generate the CTX_TEMP used to unprotect MSG_A.

   * The component containing the other peer's X and Nonce that was used in the divergent message MSG_B, i.e., the result of removing from the Master Salt the component above related to this peer.

2. Extract the X and Nonce values from the latest received divergent message MSG_A.

3. There is a match if the X and Nonce from message MSG_A are the same as those from message MSG_B identified above.

4.3.5. Handling of OSCORE Security Contexts

A peer completes the key update procedure when it has derived the new OSCORE Security Context CTX_NEW and achieved key confirmation, and thus has moved back to the IDLE state.

Once the peer has successfully derived CTX_NEW, the peer MUST use CTX_NEW to protect outgoing non KUDOS messages and MUST NOT use the originally shared OSCORE Security Context CTX_OLD for protecting outgoing messages.

The peer MUST terminate all the ongoing observations [RFC7641] that it has with the other peer as protected with the old Security Context CTX_OLD, unless the two peers have explicitly agreed otherwise as defined in Section 4.5.

More specifically, if either or both peers indicate the wish to cancel their observations, those will all be cancelled following a successful KUDOS execution.

Note that, even in case a peer has no fundamental reason to update its OSCORE keying material, running KUDOS can be intentionally exploited as a more efficient way to terminate all the ongoing observations with the other peer, compared to sending one cancellation request per observation (see Section 3.6 of [RFC7641]).

4.3.6. KUDOS Messages as CoAP Requests or Responses

If a KUDOS message is a CoAP request, then it can target two different types of resources at the recipient CoAP server:

* The well-known KUDOS resource at /.well-known/kudos (see Section 6.4), or an alternative KUDOS resource with resource type "core.kudos" (see Section 4.7.3 and Section 6.5).

  In such a case, no application processing is expected at the CoAP server and the plain CoAP request composed before OSCORE protection SHOULD NOT include an application payload.

* A non-KUDOS resource, i.e., an actual application resource that a CoAP request can target in order to trigger application processing at the CoAP server.

  In such a case, the plain CoAP request composed before OSCORE protection can include an application payload, if admitted by the request method.

In either case, the link to the target resource can be accompanied by the "osc" target attribute to indicate that the resource is only accessible using OSCORE (see Section 9 of [RFC8613]).

Similarly, any CoAP response can also be a KUDOS message. If the corresponding CoAP request has targeted a KUDOS resource, then the plain CoAP response composed before OSCORE protection SHOULD NOT include an application payload. Otherwise, an application payload MAY be included.

4.3.7. Avoiding In-Transit Requests During a Key Update

Before sending a KUDOS message, a peer MUST ensure that it has no outstanding interactions with the other peer (see Section 4.7 of [RFC7252]), with the exception of ongoing observations [RFC7641].

If any such outstanding interactions are found, the peer MUST NOT initiate or continue the KUDOS execution, before either:

* having all those outstanding interactions cleared; or

* freeing up the Token values used with those outstanding interactions, with the exception of ongoing observations with the other peer.

Later on, this prevents a non KUDOS response protected with the new Security Context CTX_NEW from cryptographically matching with both the corresponding request also protected with CTX_NEW and with an older request protected with CTX_OLD, in case the two requests were protected using the same OSCORE Partial IV.

During an ongoing KUDOS execution, a peer MUST NOT send a non KUDOS message to the other peer, until having aborted or successfully completed the key update procedure on its side. Note that, without the constraint above, this could otherwise be possible if a client running KUDOS uses a value of NSTART greater than 1 (see Section 4.7 of [RFC7252]).

4.4. Key Update Admitting no Forward Secrecy

The FS mode of the KUDOS procedure defined in Section 4.3 ensures forward secrecy of the OSCORE keying material. However, it requires peers running KUDOS to preserve their state (e.g., across a device reboot occurred in an unprepared way), by writing information such as data from the newly derived OSCORE Security Context CTX_NEW in non-volatile memory.

This can be problematic for devices that cannot dynamically write information to non-volatile memory. For example, some devices may support only a single writing in persistent memory when initial keying material is provided (e.g., at manufacturing or commissioning time), but no further writing after that. Therefore, these devices cannot perform a stateful key update procedure and thus are not capable to run KUDOS in FS mode to achieve forward secrecy.

In order to address these limitations, KUDOS can be run in its stateless no-FS mode, as defined in the following. This allows two peers to accomplish the same results as when running KUDOS in FS mode

(see Section 4.3), with the difference that forward secrecy is not achieved and no state information is required to be dynamically written in non-volatile memory.

From a practical point of view, the two modes differ in which exact OSCORE Master Secret and Master Salt are used as part of the OSCORE Security Context CTX_IN that is provided as input to the updateCtx() function (see Section 4.2).

If either or both peers are unable to write in non-volatile memory, then the two peers have to run KUDOS in no-FS mode.

4.4.1. Handling and Use of Keying Material

In the following, a device is denoted as "CAPABLE" if it is able to store information in non-volatile memory (e.g., on disk), beyond a one-time-only writing occurring at manufacturing or (re-)commissioning time. If that is not the case, the device is denoted as "non-CAPABLE".

The following terms are used to refer to OSCORE keying material.

* Bootstrap Master Secret and Bootstrap Master Salt. If pre-provisioned during manufacturing or (re-)commissioning, these OSCORE Master Secret and Master Salt are initially stored on disk and are never going to be overwritten by the device.

* Latest Master Secret and Latest Master Salt. These OSCORE Master Secret and Master Salt can be dynamically updated by the device. In case of reboot, they are lost unless they have been stored on disk.

Note that:

* A peer running KUDOS can have none of the pairs above associated with another peer, only one, or both.

* If a peer has neither of the pairs above associated with another peer, then the peer cannot run KUDOS in any mode with that other peer.

* If a peer has only one of the pairs above associated with another peer, then the peer can attempt to run KUDOS with that other peer, but the procedure might fail depending on the other peer’s capabilities. In particular:

- In order to run KUDOS in FS mode, a peer must be a CAPABLE device. It follows that two peers have to both be CAPABLE devices in order to be able to run KUDOS in FS mode with one another.

- In order to run KUDOS in no-FS mode, a peer must have Bootstrap Master Secret and Bootstrap Master Salt available as stored on disk.

* A peer that is a non-CAPABLE device MUST support the no-FS mode, with the exception described in Section 4.4.3 for non-CAPABLE devices that lack Bootstrap Master Secret and Bootstrap Master Salt.

* A peer that is a CAPABLE device MUST support the FS mode and the no-FS mode.

* As an exception to the nonces being generated as random values (see Section 4.3.1), a peer that is a CAPABLE device MAY use a value obtained from a monotonically incremented counter as nonce. Related privacy implications are described in Section 5.

  In such a case, the peer MUST enforce measures to ensure freshness of the nonce values. For example, the peer can use the same procedure described in Appendix B.1.1 of [RFC8613] for handling the OSCORE Sender Sequence Number values. These measures require to regularly store the used counter values in non-volatile memory, which makes non-CAPABLE devices unable to safely use counter values as nonce values.

As a general rule, once successfully generated a new OSCORE Security Context CTX (e.g., CTX is the CTX_NEW resulting from a KUDOS execution, or it has been established through the EDHOC protocol [RFC9528]), a peer considers the Master Secret and Master Salt of CTX as Latest Master Secret and Latest Master Salt. After that:

* If the peer is a CAPABLE device, it MUST store Latest Master Secret and Latest Master Salt on disk, with the exception of possible temporary OSCORE Security Contexts used during a key update procedure, such as CTX_TEMP used during the KUDOS execution. That is, the OSCORE Master Secret and Master Salt from such temporary Security Contexts are not stored on disk.

* The peer MUST store Latest Master Secret and Latest Master Salt in volatile memory, thus making them available to OSCORE message processing and possible key update procedures.

Following a state loss (e.g., due to a reboot occurred in an unprepared way), a device MUST complete a successful KUDOS execution before performing an exchange of OSCORE-protected application data with another peer, unless:

* The device is CAPABLE and implements a functionality for safely reusing old keying material, such as that described in Appendix B.1 of [RFC8613]; or

* The device is exchanging OSCORE-protected data within a KUDOS message, as described in Section 4.3.6. In such a case, the plain CoAP message composed before OSCORE protection can include an application payload, if admitted.

4.4.2. Selection of KUDOS Mode

The following refers to CTX_BOOTSTRAP as to the OSCORE Security Context where the OSCORE Master Secret is the Bootstrap Master Secret and the Master Salt is the Bootstrap Master Salt Section 4.4.1.

During a KUDOS execution, the two peers agree on whether to perform the key update procedure in FS mode or no-FS mode, by leveraging the "No Forward Secrecy" bit, ’p’, in the ’x’ byte of the OSCORE Option value of the KUDOS messages (see Section 4.1).

The ’p’ bit practically determines what OSCORE Security Context CTX_IN to use as input to updateCtx() during the KUDOS execution, consistently with the indicated mode. That is:

* If the ’p’ bit is set to 0 (FS mode), the updateCtx() function used to derive CTX_TEMP or CTX_NEW considers CTX_IN to be CTX_OLD, i.e., the current OSCORE Security Context shared with the other peer as is. In particular, CTX_OLD includes Latest Master Secret as OSCORE Master Secret and Latest Master Salt as OSCORE Master Salt.

* If the ’p’ bit is set to 1 (no-FS mode), the updateCtx() function used to derive CTX_TEMP or CTX_NEW considers CTX_IN to be CTX_BOOTSTRAP. Thus, every execution of KUDOS in no-FS mode between these two peers considers the same CTX_BOOTSTRAP, i.e., the same CTX_IN as input to the updateCtx() function, hence the impossibility to achieve forward secrecy.

  In particular, updateCtx() will take CTX_BOOTSTRAP as input when creating every OSCORE Security Context for protecting or unprotecting a KUDOS message where the ’p’ bit is set to 1.

If at least one KUDOS message in a successful KUDOS execution had the 'p' bit set to 1, then that KUDOS execution was run in no-FS mode.

When a peer moves to the *Pre-IDLE* stage after having succesfully completed a KUDOS execution in no-FS mode, then the peer MUST additionally perform the following Step A before Step 1 in Section 4.3.3.2:

A. Delete CTX_BOOTSTRAP.

A peer determines to run KUDOS either in FS mode or in no-FS mode with another peer as follows.

* If a peer A is a non-CAPABLE device, it MUST run KUDOS only in no-FS mode. That is, when sending a KUDOS message, it MUST set to 1 the 'p' bit of the 'x' byte in the OSCORE Option value. Note that, if the peer A lacks a Bootstrap Master Secret and Bootstrap Master Salt to use with the other peer B, it can still run KUDOS in FS mode according to what is defined in Section 4.4.3.

* If a peer A is a CAPABLE device, it SHOULD run KUDOS only in FS mode. That is, when sending a KUDOS message, it SHOULD set to 0 the 'p' bit of the 'x' byte in the OSCORE Option value. An exception applies in the following cases.

  - The peer A is running KUDOS with another peer B, which A has learned to be a non-CAPABLE device (and hence not able to run KUDOS in FS mode).

    Note that, if the peer A is a CAPABLE device, it is able to store such information about the other peer B on disk and it MUST do so. From then on, the peer A will perform every execution of KUDOS with the peer B in no-FS mode, including after a possible reboot.

  - While the peer A is running KUDOS with another peer B without knowing its capabilities, the peer A receives a KUDOS message where the 'p' bit of the 'x' byte in the OSCORE Option value is set to 1.

* If a peer A is a CAPABLE device and has learned that another peer B is also a CAPABLE device (and hence able to run KUDOS in FS mode), then the peer A MUST NOT run KUDOS with the peer B in no-FS mode. If the peer A receives a KUDOS message from the peer B where the 'p' bit of the 'x' byte in the OSCORE Option value is set to 1, then the peer A MUST ignore the message for the sake of KUDOS processing, but processes it as a CoAP message.

Note that, if the peer A is a CAPABLE device, it is able to remember that the peer B is also a CAPABLE device and thus to store such information on disk, which it MUST do. This ensures that the peer A will perform every execution of KUDOS with the peer B in FS mode. This also prevents a possible downgrading attack aimed at making A believe that B is a non-CAPABLE device, hence at making A run KUDOS in no-FS mode although the FS mode can actually be used by both peers.

4.4.3. Non-CAPABLE Devices Operating in FS Mode

Devices may not be pre-provisioned with Bootstrap material, for instance due to storage limitations of their persistent memory or to fulfill particular use cases. Bootstrap material specifically consists in the Bootstrap Master Secret and Bootstrap Master Salt, while Latest material specifically consists in the Latest Master Secret and Latest Master Salt as defined in Section 4.4.1.

Normally, a non-CAPABLE device always uses KUDOS in no-FS mode. An exception is possible, if the Bootstrap material is dynamically installed at that device through an in-band process between that device and the peer device. In such a case, it is possible for that device to run KUDOS in FS mode with the peer device.

Note that, under the assumption that peer A does not have any Bootstrap material with another peer B, peer A cannot use the no-FS mode with peer B, even though peer A is a non-CAPABLE device. Thus, allowing peer A to use KUDOS in FS mode ensures that peer A can perform a key update using KUDOS at all.

In the situation outlined above, the following describes how a non-CAPABLE device, namely peer A, runs KUDOS in FS mode with another peer B:

* Peer A is not provisioned with Bootstrap material associated with peer B at the time of manufacturing or commissioning.

* Peer A establishes OSCORE keying material associated with peer B through an in-band procedure run with peer B. Then, peer A considers that keying material as the Latest material with peer B and stores it only in volatile memory.

  An example of such an in-band procedure is the EDHOC and OSCORE profile of ACE [I-D.ietf-ace-edhoc-oscore-profile], according to which the two peers run the EDHOC protocol [RFC9528] for establishing an OSCORE Security Context to associate with access rights. This in-band procedure may occur multiple times over the device’s lifetime.

* Peer A runs KUDOS in FS mode with peer B, thereby achieving forward secrecy for subsequent key update epochs, as long as the OSCORE keying material was originally established with forward secrecy. Peer A stores each newly derived Security Context in volatile memory.

As long as peer A does not reboot, executions of KUDOS rely on the Latest material stored in volatile memory. If peer A reboots, no OSCORE keying material associated with the peer B will be retained, as peer A is non-CAPABLE and therefore stores it only in volatile memory. Consequently, peer A must first establish new OSCORE keying material to use as Latest material with peer B, before running KUDOS again with peer B. This can be accomplished by running again the in-band procedure mentioned above.

4.5. Preserving Observations Across Key Updates

As defined in Section 4.3, once a peer has successfully completed the KUDOS execution and derived the new OSCORE Security Context CTX_NEW, that peer normally terminates all the ongoing observations that it has with the other peer [RFC7641] and that are protected with the old OSCORE Security Context CTX_OLD.

This section describes a method that the two peers can use to safely preserve the ongoing observations that they have with one another, beyond the completion of a KUDOS execution. In particular, this method ensures that an Observe notification can never successfully cryptographically match against the Observe requests of two different observations, e.g., against an Observe request protected with CTX_OLD and an Observe request protected with CTX_NEW.

The actual preservation of ongoing observations has to be agreed by the two peers at each execution of KUDOS that they run with one another, as defined in Section 4.5.1. If, at the end of a KUDOS execution, the two peers have not agreed on that, they MUST terminate the ongoing observations that they have with one another, just as defined in Section 4.3.5.

4.5.1. Management of Observations

As per Section 3.1 of [RFC7641], a client can register its interest in observing a resource at a server, by sending a registration request including the Observe Option with value 0.

If the server registers the observation as ongoing, the server sends back a successful response also including the Observe Option, hence confirming that an entry has been successfully added for that client.

If the client receives back the successful response above from the server, then the client also registers the observation as ongoing.

In case the client can ever consider to preserve ongoing observations beyond a key update as defined below, then the client MUST NOT simply forget about an ongoing observation if not interested in it anymore. Instead, the client MUST send an explicit cancellation request to the server, i.e., a request including the Observe Option with value 1 (see Section 3.6 of [RFC7641]). After sending this cancellation request, if the client does not receive back a response confirming that the observation has been terminated, the client MUST NOT consider the observation terminated. The client MAY try again to terminate the observation by sending a new cancellation request.

In case a peer A performs a KUDOS execution with another peer B and A has ongoing observations with B that it is interested to preserve beyond the key update, then A can explicitly indicate its interest to do so. To this end, the peer A sets to 1 the bit "Preserve Observations", ’b’, in the ’x’ byte of the OSCORE Option value (see Section 4.1), in the KUDOS message that it sends to the other peer B.

If a peer receives a KUDOS message with the bit ’b’ set to 0, then the peer MUST set to 0 the bit ’b’ in the KUDOS message that it sends as follow-up, regardless of its wish to preserve ongoing observations with the other peer.

If a peer has sent a KUDOS message with the bit ’b’ set to 0, the peer MUST ignore the bit ’b’ in the follow-up KUDOS message that it receives from the other peer.

Note that during the same KUDOS execution, all the KUDOS messages sent by a peer MUST have the same value in the bit ’b’ for preserving ongoing observations.

After successfully completing the KUDOS execution (i.e., after having successfully derived the new OSCORE Security Context CTX_NEW), both peers have expressed their interest in preserving their common ongoing observations if and only if the bit ’b’ was set to 1 in all the exchanged KUDOS messages. In such a case, each peer X performs the following actions:

1. The peer X considers all the still ongoing observations that it has with the other peer, such that X acts as client in those observations. If there are no such observations, the peer X takes no further actions. Otherwise, it moves to Step 2.

2. The peer X considers all the OSCORE Partial IV values used in the Observe registration requests associated with any of the still ongoing observations determined at Step 1.

3. The peer X determines the value PIV* as the highest OSCORE Partial IV value among those considered at Step 2.

4. In the Sender Context of the OSCORE Security Context shared with the other peer, the peer X sets its own Sender Sequence Number to (PIV* + 1), rather than to 0.

As a result, each peer X will "jump" beyond the OSCORE Partial IV (PIV) values that are occupied and in use for ongoing observations with the other peer where X acts as client.

Note that, each time it runs KUDOS, a peer must determine if it wishes to preserve ongoing observations with the other peer or not, before sending a KUDOS message.

To this end, the peer should also assess the new value that PIV* would take after a successful completion of KUDOS, in case ongoing observations with the other peer are going to be preserved. If the peer considers such a new value of PIV* to be too close to or equal to the maximum possible value admitted for the OSCORE Partial IV, then the peer may choose to run KUDOS with no intention to preserve its ongoing observations with the other peer, in order to "start over" from a fresh, entirely unused Sender Sequence Number space.

Application policies can further influence whether attempting to preserve observations beyond a key update is appropriate or not.

4.6. Retention Policies

Applications MAY define policies that allow a peer to temporarily keep the old Security Context CTX_OLD beyond having established the new Security Context CTX_NEW and having achieved key confirmation, rather than simply deleting CTX_OLD. This allows the peer to decrypt late, still on-the-fly incoming non KUDOS messages that are protected with CTX_OLD.

When enforcing such policies, the following applies:

* Outgoing non KUDOS messages MUST be protected only by using CTX_NEW.

* Incoming non KUDOS messages MUST first be attempted to decrypt by using CTX_NEW. If decryption fails, a second attempt can use CTX_OLD.

* KUDOS messages MUST NOT be protected or unprotected by using CTX_OLD.

* When an amount of time defined by the policy has elapsed since the establishment of CTX_NEW, the peer deletes CTX_OLD.

A peer MUST NOT retain CTX_OLD beyond the establishment of CTX_NEW and the achievement of key confirmation, if any of the following conditions holds:

* CTX_OLD is expired.

* Limits set for safe key usage have been reached for the Recipient Key of the Recipient Context of CTX_OLD (see [I-D.ietf-core-oscore-key-limits]).

4.7. Discussion

KUDOS is intended to deprecate and replace the procedure defined in Appendix B.2 of [RFC8613], as fundamentally achieving the same goal while displaying a number of improvements and advantages.

In particular, it is especially convenient for the handling of failure events concerning the JRC node in 6TiSCH networks (see Section 2). That is, among its intrinsic advantages compared to the procedure defined in Appendix B.2 of [RFC8613], KUDOS preserves the same ID Context value when establishing a new OSCORE Security Context.

Since the JRC uses ID Context values as identifiers of network nodes, namely "pledge identifiers", the above implies that the JRC does not have to perform anymore a mapping between a new, different ID Context value and a certain pledge identifier (see Section 8.3.3 of [RFC9031]). It follows that pledge identifiers can remain constant once assigned and thus ID Context values used as pledge identifiers can be employed in the long-term as originally intended.

4.7.1. Communication Overhead

Each KUDOS messages results in communication overhead. This is determined by the following, additional information conveyed in the OSCORE Option (see Section 4.1).

* The second byte of the OSCORE Option value.

* The 1-byte ’x’ field of the OSCORE Option value.

* The nonce conveyed in the 'nonce' field of the OSCORE Option value. Its size ranges from 1 to 16 bytes, is typically of 8 bytes, and is indicated in the 'x' field.

* The 'Partial IV' field of the OSCORE Option value, in a CoAP response message that is a KUDOS message.

  This takes into account the fact that OSCORE-protected CoAP response messages normally do not include the 'Partial IV' field, but they have to when they are KUDOS messages (see Section 3).

* The first byte of the OSCORE Option value (i.e., the first OSCORE flag byte), in a CoAP response message that is a KUDOS message.

  This takes into account the fact that OSCORE-protected CoAP response messages normally convey an OSCORE Option that only consists of the all zero (0x00) flag byte. In turn, this results in the OSCORE Option being encoded as with empty value (see Section 2 of [RFC8613]).

* The possible presence of the 1-byte 'Option Length (extended)' field in the OSCORE Option (see Section 3.1 of [RFC7252]). This is the case where the length of the OSCORE Option value is between 13 and 255 bytes (see Section 2 of [RFC8613]).

The results shown in figure Table 1 are the minimum, typical, and maximum communication overhead in bytes introduced by KUDOS, when considering a nonce with size 1, 8, and 16 bytes. All the indicated values are in bytes.

The overhead is calculated considering a scenario where a CoAP request serves as the divergent message and a following, corresponding CoAP response serves as the convergent message.

In particular, the results build on the following assumptions.

* Both messages of the same KUDOS execution use nonces of the same size and do not include the 'kid context' field in the OSCORE Option value.

* When included in the OSCORE Option value, the 'Partial IV' field has size 1 byte.

* CoAP request messages include the 'kid' field with size 1 byte in the OSCORE Option value.

* CoAP response messages do not include the 'kid' field in the OSCORE Option value.

Nonce size	Request KUDOS message	Response KUDOS message	Total
1	3	5	8
8	11	12	23
16	19	21	40

Table 1: Communication Overhead (Bytes)

4.7.2. Resource Type core.kudos

The "core.kudos" resource type registered in Section 6.5 is defined to ensure a means for clients to send KUDOS requests without incurring any side effects. Specifically, a resource of this type does not pertain to any real application, which ensures that no application-level actions are triggered as a result of the KUDOS request.

This allows clients to issue KUDOS requests when they do not include any actionable application payload in the plain CoAP request composed before OSCORE protection, or when no application-layer processing is intended to occur on the server.

4.7.3. Well-Known KUDOS Resource

If a client wishes to run the KUDOS procedure without triggering any application processing on the server, then a request sent as a KUDOS message can target a KUDOS resource with resource type "core.kudos" (see Section 4.7.2), e.g., at the Uri-Path "/.well-known/kudos" (see Section 6.4). An alternative KUDOS resource can be discovered, e.g., by using a resource directory [RFC9176], by using the resource type "core.kudos" as filter criterion.

4.7.4. Rekeying when Using SCHC with OSCORE

In the interest of rekeying, the following points must be taken into account when using the Static Context Header Compression and fragmentation (SCHC) framework [RFC8724] for compressing CoAP messages protected with OSCORE, as defined in [RFC8824].

The SCHC compression of the ’Partial IV’ field in the OSCORE Option value has implications for the frequency of rekeying. That is, if the ’Partial IV’ field is compressed, the communicating peers must perform rekeying more often, as the available Sender Sequence Number

space that is used for the Partial IV becomes effectively smaller due to the compression. For instance, if only 3 bits of the Partial IV are sent, then the maximum Partial IV that can be used before having to rekey is only 2^3 - 1 = 7.

Furthermore, any time the SCHC context Rules are updated on an OSCORE endpoint, that endpoint must perform a rekeying (see Section 9 of [RFC8824]).

That is, the use of SCHC plays a role in triggering KUDOS executions and in affecting their cadence. Hence, the employed SCHC Rules and their update policies should ensure that the KUDOS executions occurring as their side effect do not significantly impair the gain expected from message compression.

4.7.5. Combining KUDOS with Access Control

Resource where messages can be sent at the server might be following the enforcement of access control means on the request. For example, when combining KUDOS with the EDHOC and OSCORE profile of ACE [I-D.ietf-ace-edhoc-oscore-profile], certain considerations must be taken into account to ensure proper access control behavior:

* A KUDOS request that targets a non-KUDOS resource MUST trigger standard ACE-based access control checks.

* A KUDOS request that targets a KUDOS resource MUST NOT trigger ACE-based access control.

To support this, the path of any KUDOS resource can be included in the ACE access control exclusion list (i.e., the "do not enforce access control" list). The same principles have to be applied if other means are used to enforce access control.

In some deployment scenarios, an ACE Access Token may be bound to both CTX_OLD and CTX_NEW, allowing it to be valid and still usable after the execution of a KUDOS procedure.

It is important to note that KUDOS is not compatible with the OSCORE profile of ACE [RFC9203], this is because KUDOS changes the OSCORE Master Secret, which is used as proof-of-possession key in that profile. However, as described above, KUDOS is compatible with the EDHOC and OSCORE profile of ACE [I-D.ietf-ace-edhoc-oscore-profile].

4.8. Signaling KUDOS support in EDHOC

The EDHOC protocol defines the transport of additional External Authorization Data (EAD) within an optional EAD field of the EDHOC messages (see Section 3.8 of [RFC9528]). An EAD field is composed of one or multiple EAD items, each of which specifies an identifying 'ead_label' encoded as a CBOR integer and an optional 'ead_value' encoded as a CBOR byte string.

This document defines a new EDHOC EAD item KUDOS_EAD and registers its 'ead_label' in Section 6.3. By including this EAD item in an outgoing EDHOC message, a sender peer can indicate whether it supports KUDOS and in which modes, as well as query the other peer about its support. Note that peers do not have to use this EDHOC EAD item to be able to run KUDOS with each other, irrespective of the modes that they support. A KUDOS peer MUST only use the EDHOC EAD item KUDOS_EAD as non-critical. That is, when included in an EDHOC message, its 'ead_label' MUST be used with positive sign. The possible values of the 'ead_value' are as follows:

Name	Value	Description
ASK	h'' (0x40)	Used only in EDHOC message_1. It asks the recipient peer to specify in EDHOC message_2 whether it supports KUDOS.
NONE	h'00' (0x4100)	Used only in EDHOC message_2 and message_3. It specifies that the sender peer does not support KUDOS.
FULL	h'01' (0x4101)	Used only in EDHOC message_2 and message_3. It specifies that the sender peer supports KUDOS in FS mode and no-FS mode.
PART	h'02' (0x4102)	Used only in EDHOC message_2 and message_3. It specifies that the sender peer supports KUDOS in no-FS mode only.

Table 2: Values for the EDHOC EAD Item KUDOS_EAD

When the KUDOS_EAD item is included in EDHOC message_1 with 'ead_value' ASK, a recipient peer that supports the KUDOS_EAD item MUST specify whether it supports KUDOS in EDHOC message_2.

When the KUDOS_EAD item is not included in EDHOC message_1 with 'ead_value' ASK, a recipient peer that supports the KUDOS_EAD item MAY still specify whether it supports KUDOS in EDHOC message_2.

When the KUDOS_EAD item is included in EDHOC message_2 with 'ead_value' FULL or PART, a recipient peer that supports the KUDOS_EAD item SHOULD specify whether it supports KUDOS in EDHOC message_3. An exception applies in case, based on application policies or other context information, the recipient peer that receives EDHOC message_2 already knows that the sender peer is supposed to have such knowledge.

When the KUDOS_EAD item is included in EDHOC message_2 with 'ead_value' NONE, a recipient peer that supports the KUDOS_EAD item MUST NOT specify whether it supports KUDOS in EDHOC message_3.

In the following cases, the recipient peer silently ignores the KUDOS_EAD item specified in the received EDHOC message and does not include a KUDOS_EAD item in the next EDHOC message it sends (if any).

* The recipient peer does not support the KUDOS_EAD item.

* The KUDOS_EAD item is included in EDHOC message_1 with 'ead_value' different than ASK

* The KUDOS_EAD item is included in EDHOC message_2 or message_3 with 'ead_value' ASK.

* The KUDOS_EAD item is included in EDHOC message_4.

That is, by specifying 'ead_value' ASK in EDHOC message_1, a peer A can indicate to the other peer B that it wishes to know if B supports KUDOS and in what mode(s). In the following EDHOC message_2, the peer B indicates whether it supports KUDOS and in what mode(s), by specifying either NONE, FULL, or PART as 'ead_value'. Specifying the 'ead_value' FULL or PART in EDHOC message_2 also asks A to indicate whether it supports KUDOS in EDHOC message_3.

To further illustrate the signaling process defined above, the following presents two examples of EDHOC execution where only the new KUDOS_EAD item is shown when present, assuming that no other EAD items are used by the two peers.

In the example shown in Figure 3, the EDHOC Initiator asks the EDHOC Responder about its support for KUDOS ('ead_value' = ASK). In EDHOC message_2, the Responder indicates that it supports both the FS and no-FS mode of KUDOS ('ead_value' = FULL). Finally, in EDHOC message_3, the Initiator indicates that it also supports both the FS

and no-FS mode of KUDOS ('ead_value' = FULL). After the EDHOC execution has been successfully completed, both peers are aware that they both support KUDOS, in the FS and no-FS modes.

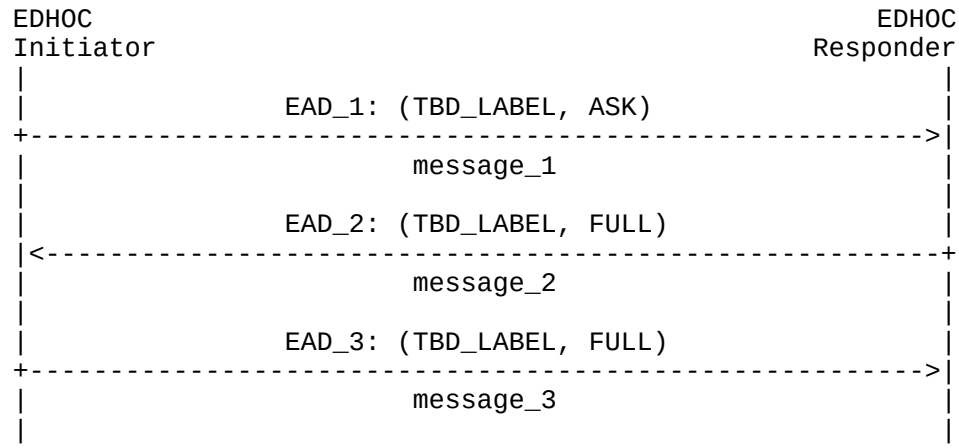


Figure 3: Example of EDHOC Execution with Signaling of Support for KUDOS (Both Peers Support KUDOS)

In the example shown in Figure 4, the EDHOC Initiator asks the EDHOC Responder about its support for KUDOS ('ead_value' = ASK). In EDHOC message_2, the Responder indicates that it does not support KUDOS at all ('ead_value' = NONE). Finally, in EDHOC message_3, the Initiator does not include the KUDOS_EAD item, since it already knows that using KUDOS with the other peer will not be possible. After the EDHOC execution has been successfully completed, the Initiator is aware that the Responder does not support KUDOS, which the two peers are not going to use with each other.

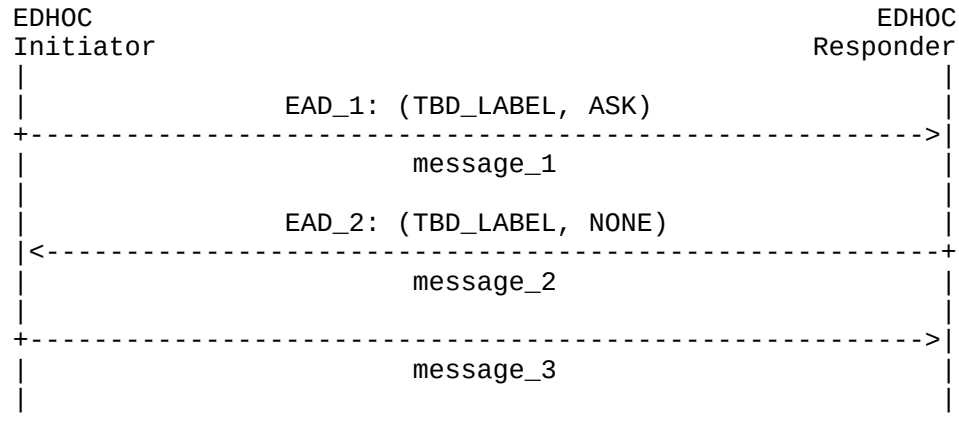


Figure 4: Example of EDHOC Execution with Signaling of Support for KUDOS (the EDHOC Responder Does Not Support KUDOS)

5. Security Considerations

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

As mentioned in Section 4.3.1, it is RECOMMENDED that the size for the nonces N1 and N2 is 8 bytes. Applications need to set the size of each nonce such that the probability of its value being repeated is negligible throughout executions of KUDOS that aim to update a given OSCORE Security Context, even in case of loss of state (e.g., due to a reboot occurred in an unprepared way).

Considering the birthday paradox and a nonce size of 8 bytes, the average collision for each nonce will happen after the generation of 2^32 (X, nonce) pairs generated by a given peer (see Section 4.3.1), which is considerably more than the number of such pairs that a peer is expected to generate throughout the update of a given OSCORE Security Context using KUDOS (in fact, that number is expected to typically be 1). Yet, determining the appropriate nonce size also ought to take into account the possible use of KUDOS in no-FS mode (see Section 4.4), in which case every execution in no-FS mode between two given peers considers the same CTX_BOOTSTRAP as the OSCORE Security Context to update (see Section 4.4.2), hence raising the chances of reusing a nonce.

Overall, the size of the nonces N1 and N2 should be set such that the security level is harmonized with other components of the deployment. Considering the constraints of embedded implementations, there might be a need for allowing N1 and N2 values that have a size smaller than the recommended one. This is acceptable, provided that safety, reliability, and robustness within the system can still be assured. That is, if nonces with a smaller size are used and thus a collision will occur on average after fewer KUDOS executions that aim to update a given OSCORE Security Context, care must be taken to ensure that this does not pose significant problems, e.g., considering as benchmark a constrained server operating at a capacity of one request per second.

The nonces exchanged in the KUDOS messages are sent in the clear, so using random nonces is preferable for maintaining privacy. Instead, if counter values are used (see Section 4.4.1), this can leak information such as the frequency according to which two peers perform a key update.

As discussed in [Symmetric-Security], key update methods built on symmetric key exchange have weaker security properties compared to methods built on ephemeral Diffie-Hellman key exchange. In fact, while the two approaches can co-exist, rekeying with symmetric key exchange is not intended as a substitute for ephemeral Diffie-Hellman key exchange. Peers should periodically perform a key update based on ephemeral Diffie-Hellman key exchange (e.g., by running the EDHOC protocol [RFC9528]). The cadence of such periodic key updates should be determined based on how much the two peers and their network environment are constrained, as well as on the maximum amount of time and of exchanged data that are acceptable between two such consecutive key updates.

6. IANA Considerations

This document has the following actions for IANA.

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification and delete this paragraph.

6.1. OSCORE Flag Bits Registry

IANA is asked to add the following entries to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Bit Position	Name	Description	Reference
0 (suggested)	Extension-1 Flag	Set to 1 if the OSCORE Option specifies a second byte, which includes the OSCORE flag bits 8-15	[RFC-XXXX]
8	Extension-2 Flag	Set to 1 if the OSCORE Option specifies a third byte, which includes the OSCORE flag bits 16-23	[RFC-XXXX]
15	Nonce Flag	Set to 1 if nonce is present in the compressed COSE object	[RFC-XXXX]
16	Extension-3 Flag	Set to 1 if the OSCORE Option specifies a	[RFC-XXXX]

		fourth byte, which includes the OSCORE flag bits 24-31	
24	Extension-4 Flag	Set to 1 if the OSCORE Option specifies a fifth byte, which includes the OSCORE flag bits 32-39	[RFC-XXXX]
32	Extension-5 Flag	Set to 1 if the OSCORE Option specifies a sixth byte, which includes the OSCORE flag bits 40-47	[RFC-XXXX]
40	Extension-6 Flag	Set to 1 if the OSCORE Option specifies a seventh byte, which includes the OSCORE flag bits 48-55	[RFC-XXXX]
48	Extension-7 Flag	Set to 1 if the OSCORE Option specifies an eighth byte, which includes the OSCORE flag bits 56-63	[RFC-XXXX]

Table 3: Registrations in the OSCORE Flag Bits Registry

In the same registry, IANA is asked to mark as 'Unassigned' the entry with Bit Position of 1, i.e., to update the entry as follows.

Bit Position	Name	Description	Reference
1	Unassigned		

Table 4: Update in the OSCORE Flag Bits Registry

6.2. CoAP Option Numbers Registry

IANA is asked to add this document as a reference for the OSCORE Option in the "CoAP Option Numbers" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

6.3. EDHOC External Authorization Data Registry

IANA is asked to add the following entry to the "EDHOC External Authorization Data" registry defined in Section 10.5 of [RFC9528] within the "Ephemeral Diffie-Hellman Over COSE (EDHOC)" registry group.

Name	Label	Description	Reference
KUDOS_EAD	TBD1	Indicates whether this peer supports KUDOS and in which mode(s)	[RFC-XXXX]

Table 5: Registrations in the EDHOC External Authorization Data Registry

6.4. The Well-Known URI Registry

IANA is asked to add the ’kudos’ well-known URI to the Well-Known URIs registry as defined by [RFC8615].

* URI suffix: kudos
* Change controller: IETF
* Specification document(s): [RFC-XXXX]
* Related information: None

6.5. Resource Type (rt=) Link Target Attribute Values Registry

IANA is requested to add the resource type "core.kudos" to the "Resource Type (rt=) Link Target Attribute Values" registry under the registry group "Constrained RESTful Environments (CoRE) Parameters".

* Value: "core.kudos"
* Description: KUDOS resource.
* Reference: [RFC-XXXX]

7. References

7.1. 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/rfc/rfc2119>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[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/rfc/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

[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/rfc/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/rfc/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9528] Selander, G., Preu Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

7.2. Informative References

[I-D.ietf-ace-edhoc-oscore-profile]
Selander, G., Mattsson, J. P., Tiloca, M., and R. Hglund, "Ephemeral Diffie-Hellman Over COSE (EDHOC) and Object Security for Constrained Environments (OSCORE) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-edhoc-oscore-profile-08, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-edhoc-oscore-profile-08>.

[I-D.ietf-core-oscore-key-limits]
Hglund, R. and M. Tiloca, "Key Usage Limits for OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-key-limits-04, 8 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-limits-04>.

[I-D.irtf-cfrg-aead-limits]
Gnther, F., Thomson, M., and C. A. Wood, "Usage Limits on AEAD Algorithms", Work in Progress, Internet-Draft, draft-irtf-cfrg-aead-limits-10, 8 April 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aead-limits-10>.

[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>.

[LwM2M-Transport]
Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Transport Bindings, Approved Version 1.2, OMA-TS-LightweightM2M_Transport-V1_2-20201110-A", November 2020, <http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Transport-V1_2-20201110-A.pdf>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement", RFC 7554, DOI 10.17487/RFC7554, May 2015, <https://www.rfc-editor.org/rfc/rfc7554>.

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/rfc/rfc7967>.

[RFC8180] Vilajosana, X., Ed., Pister, K., and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration", BCP 210, RFC 8180, DOI 10.17487/RFC8180, May 2017, <https://www.rfc-editor.org/rfc/rfc8180>.

[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/rfc/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/rfc/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/rfc/rfc8824>.

[RFC9031] Vuini, M., Ed., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", RFC 9031, DOI 10.17487/RFC9031, May 2021, <https://www.rfc-editor.org/rfc/rfc9031>.

[RFC9176] Amsss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

[RFC9203] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <https://www.rfc-editor.org/rfc/rfc9203>.

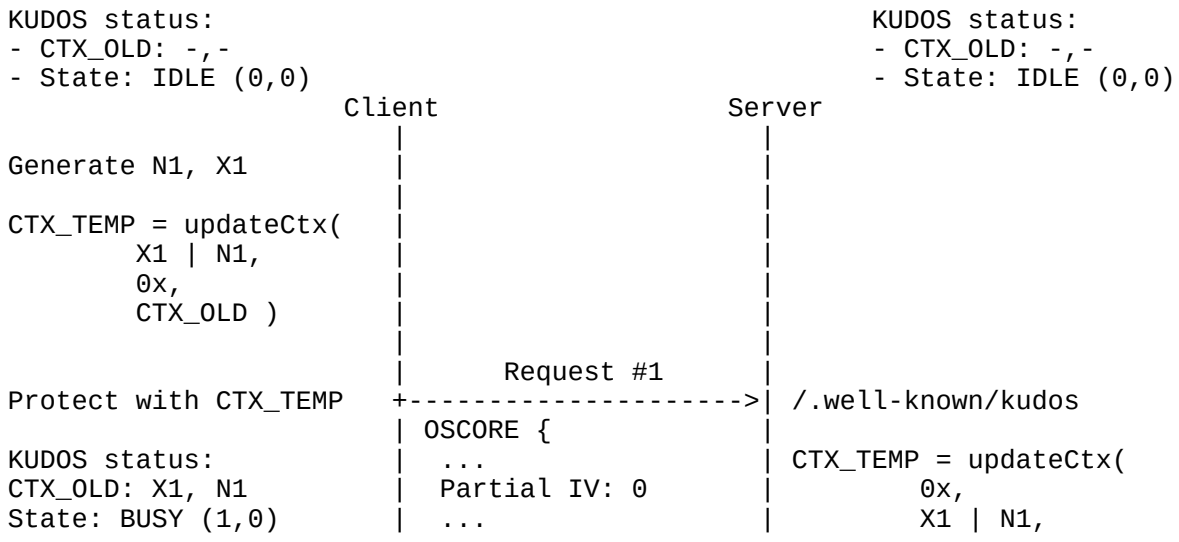
[Symmetric-Security] Mattsson, J. P., "Security of Symmetric Ratchets and Key Chains - Implications for Protocols like TLS 1.3, Signal, and PQ3", February 2024, <https://eprint.iacr.org/2024/220>.

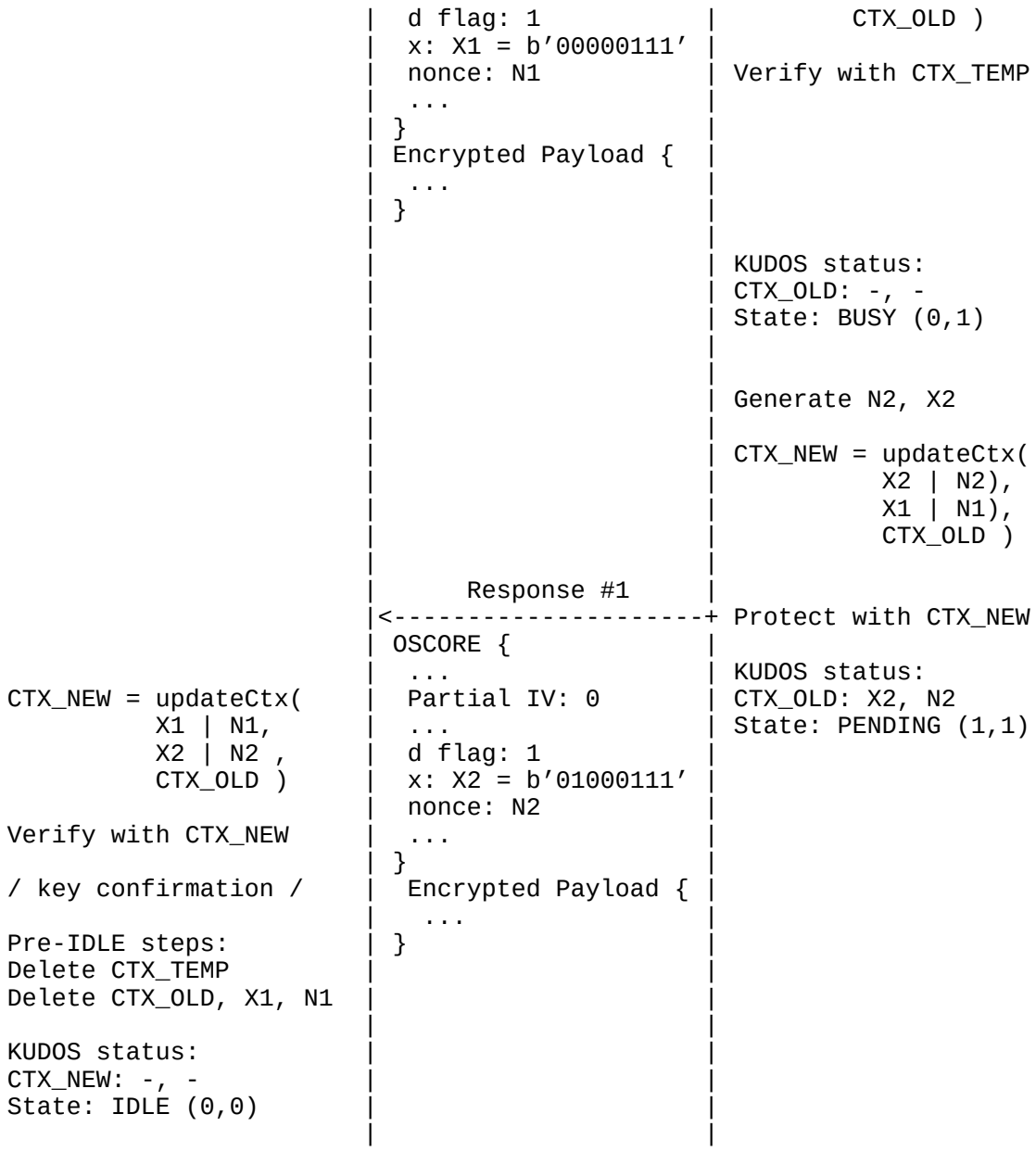
Appendix A. Examples

The following sections show two examples of KUDOS being executed and successfully completing.

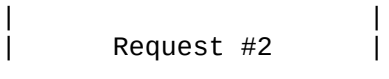
A.1. Successful KUDOS Execution Initiated with a Request Message

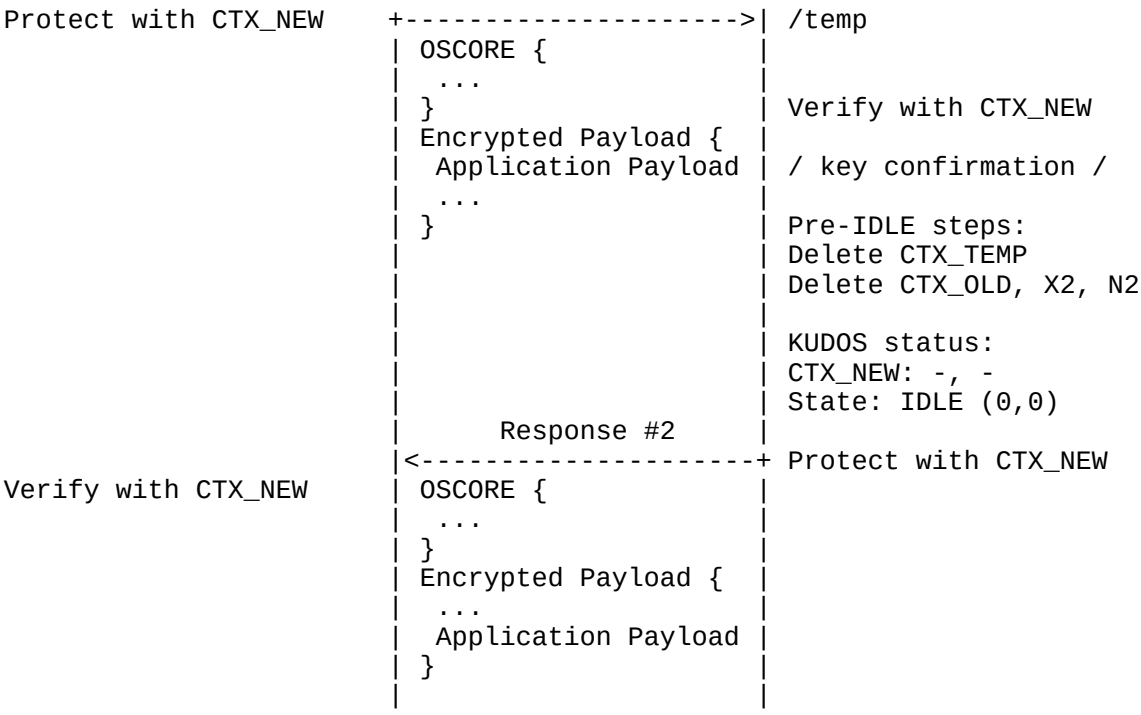
The following shows a succesful execution of KUDOS where KUDOS is started by the client sending a divergent KUDOS message as a CoAP request.





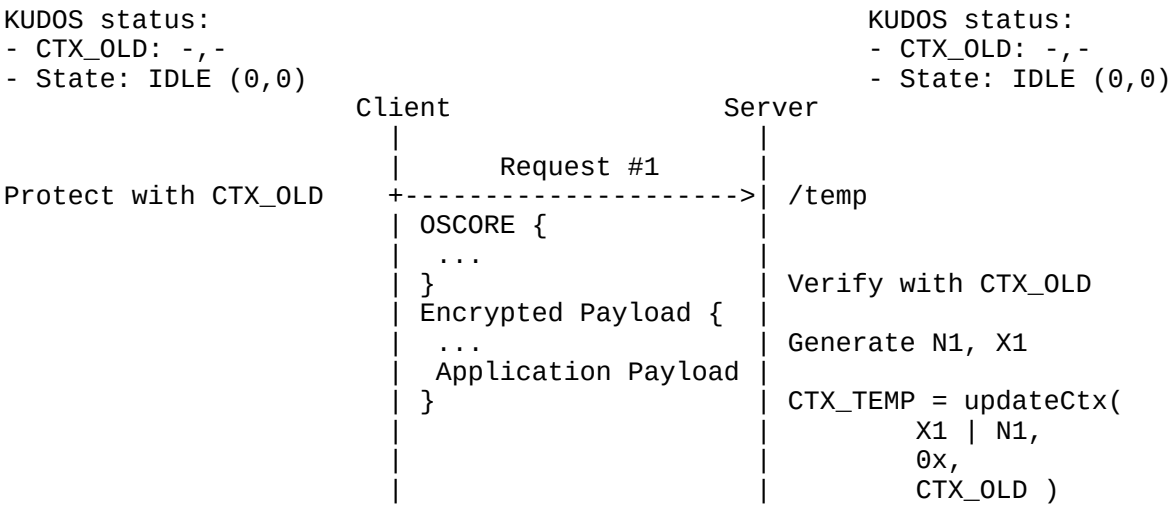
The actual key update process ends here.
The two peers can use the new Security Context CTX_NEW.

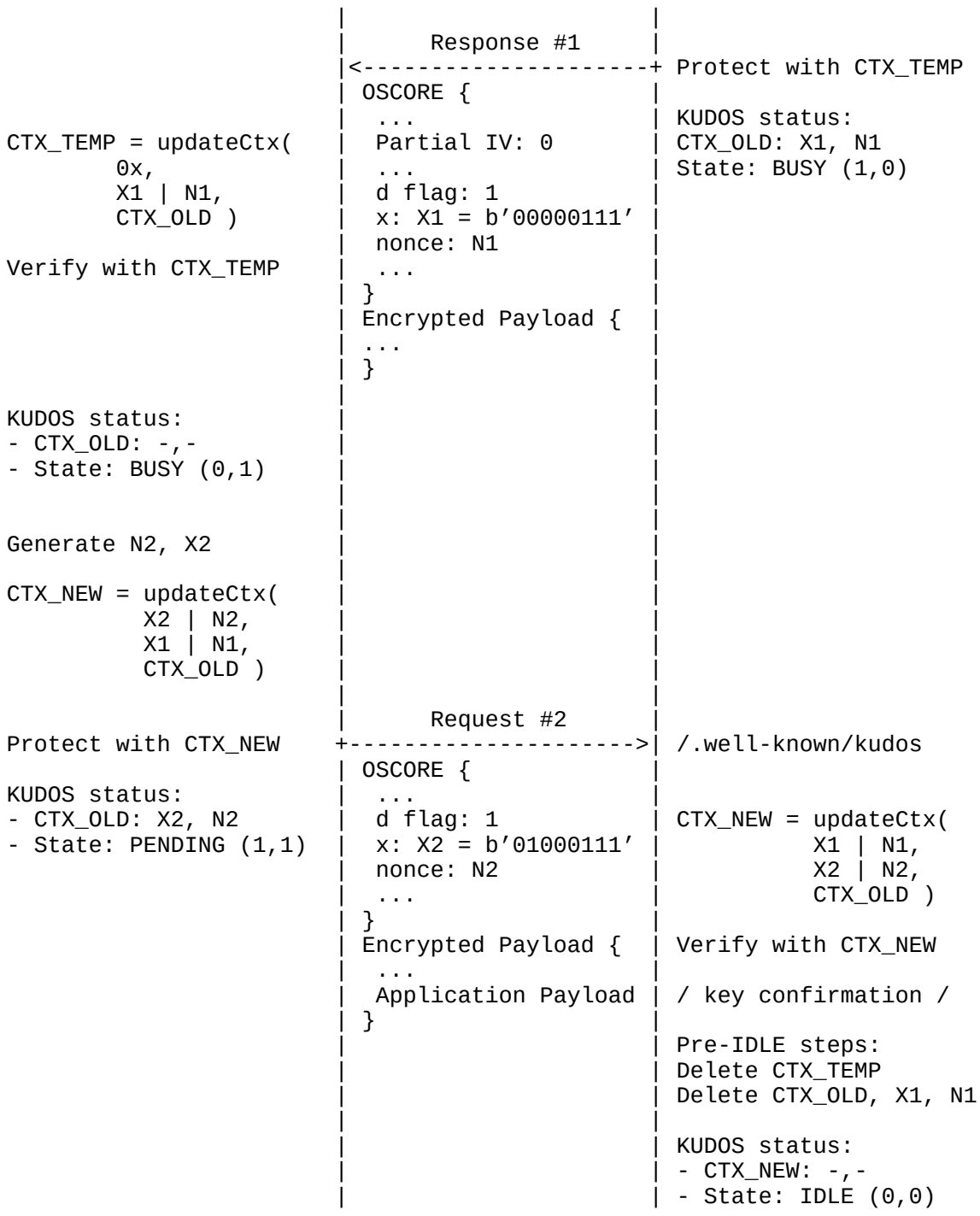


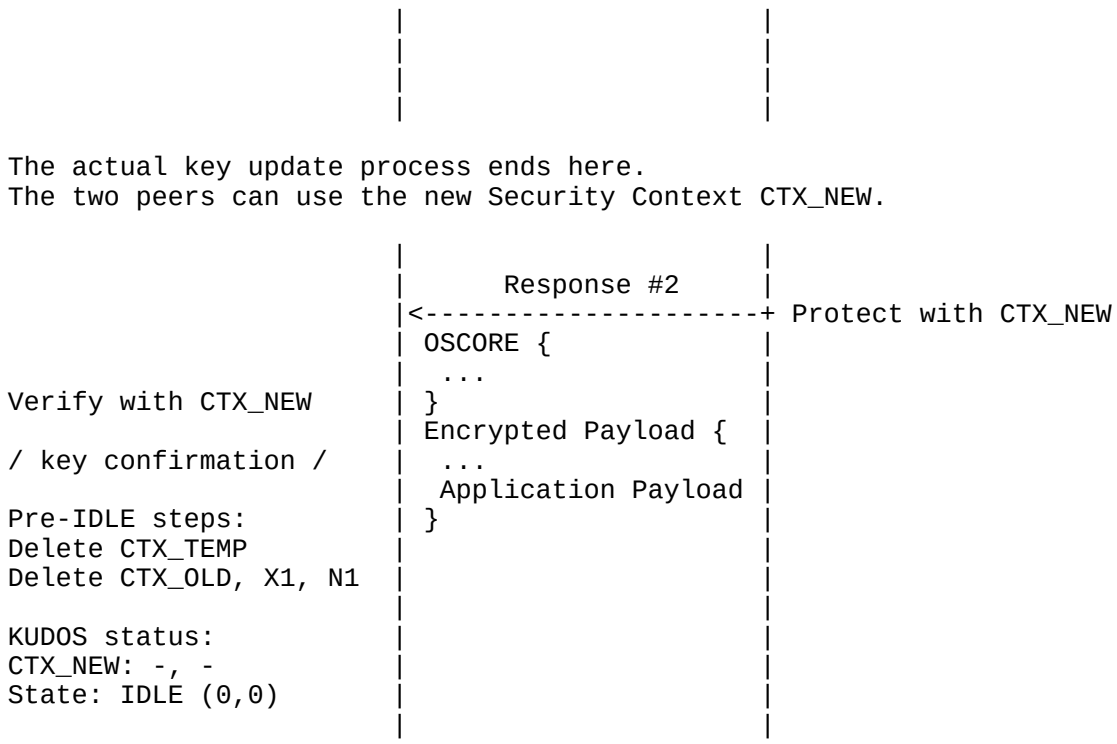


A.2. Successful KUDOS Execution Initiated with a Response Message

The following shows a succesful execution of KUDOS where KUDOS is started by the server sending a divergent KUDOS message as a CoAP response.




Response #1
Protect with CTX_TEMP
OSCORE {
...
Partial IV: 0
...
d flag: 1
x: X1 = b'00000111'
nonce: N1
...
}
Encrypted Payload {
...
}
KUDOS status:
CTX_OLD: X1, N1
State: BUSY (1,0)
CTX_TEMP = updateCtx(
0x,
X1 | N1,
CTX_OLD)
Verify with CTX_TEMP
KUDOS status:
- CTX_OLD: -,-
- State: BUSY (0,1)
Generate N2, X2
CTX_NEW = updateCtx(
X2 | N2,
X1 | N1,
CTX_OLD)
Request #2
/.well-known/kudos
Protect with CTX_NEW
OSCORE {
...
d flag: 1
x: X2 = b'01000111'
nonce: N2
...
}
Encrypted Payload {
...
Application Payload
}
KUDOS status:
- CTX_OLD: X2, N2
- State: PENDING (1,1)
CTX_NEW = updateCtx(
X1 | N1,
X2 | N2,
CTX_OLD)
Verify with CTX_NEW
/ key confirmation /
Pre-IDLE steps:
Delete CTX_TEMP
Delete CTX_OLD, X1, N1
KUDOS status:
- CTX_NEW: -,-
- State: IDLE (0,0)



The actual key update process ends here.
The two peers can use the new Security Context CTX_NEW.

Appendix B. Document Updates

This section is to be removed before publishing as an RFC.

B.1. Version -10 to -11

* Extended security considerations.
* Clarifications and editorial improvements.
* Updates to IANA considerations according to IANA early reviews.
* Extended section about updated protection of CoAP responses.
* Discussion about combining usage of KUDOS with profiles of ACE.
* Optimization for traversing the state machine upon reception of a divergent message.

B.2. Version -09 to -10

* Major re-design building on a state machine driving the KUDOS execution.

B.3. Version -08 to -09

* Merge text about avoiding in-transit requests during a key update into a single subsection.

* Improved error handling.

* Editorial improvements and clarifications.

* State that the EDHOC EAD item must be used as non-critical.

* Extended description and updates values for KUDOS communication overhead.

* Introduce special case when non-CAPABLE devices may operate in FS Mode.

* Add parameter for signaling KUDOS support when using the ACE OSCORE profile.

* Enable using the reverse message flow for peers that are only CoAP servers.

* Further clarifications about achieving key confirmation and deletion of old contexts.

* Restructure distribution of content about FS and no-FS mode.

* Warn of consequences of running KUDOS with insufficient margin.

* Stressed usefulness of core.kudos for safe KUDOS requests without side effects.

B.4. Version -07 to -08

* Add note about usage of the CoAP No-Response Option.

* Avoid problems for two simultaneously started key updates.

* Set Notification Number to be uninitialized for new OSCORE Security Contexts.

* Handle corner case for responder that reached its key usage limits.

* Re-organizing main section about Forward Secrecy mode into subsections.

* IANA considerations for CoAP Option Numbers Registry to refer to this draft for the OSCORE option.

* Use AASVG in diagrams.

* Use actual tables instead of figures.

* Clarifications and editorial improvements.

* Extended security considerations with reference to relevant paper.

B.5. Version -06 to -07

* Removed material about the ID update procedure, which has been split out into a separate draft.

* Allow non-random nonces for CAPABLE devices.

* Editorial improvements.

* Permit flexible message flow with KUDOS messages as any request/response.

* Enable sending KUDOS messages as regular application messages.

B.6. Version -05 to -06

* Mandate support for both the forward and reverse message flow.

* Mention the EDHOC and OSCORE profile of ACE as method for rekeying.

* Clarify definition of KUDOS (request/response) message.

* Further extend the OSCORE option to transport N1 in the second KUDOS message as a request.

* Mandate support for the no-FS mode on CAPABLE devices.

* Explain when KUDOS fails during selection of mode.

* Explicitly forbid using old keying material after reboot.

* Editorial improvements.

B.7. Version -04 to -05

* Note on client retransmissions if KUDOS execution fails in reverse message flow.

* Specify what information needs to be written to non-volatile memory to handle reboots.

* Extended recommendations and considerations on minimum size of nonces N1 & N2.

* Arbitrary maximum size of the Recipient-ID Option.

* Detailed lifecycle of the OSCORE IDs update procedure.

* Described examples of OSCORE IDs update procedure.

* Examples of OSCORE IDs update procedure integrated in KUDOS.

* Considerations about using SCHC for CoAP with OSCORE.

* Clarifications and editorial improvements.

B.8. Version -03 to -04

* Removed content about key usage limits.

* Use of "forward message flow" and "reverse message flow".

* Update to RFC 8613 extended to include protection of responses.

* Include EDHOC_KeyUpdate() in the methods for rekeying.

* Describe reasons for using the OSCORE ID update procedure.

* Clarifications on deletion of CTX_OLD and CTX_NEW.

* Added new section on preventing deadlocks.

* Clarified that peers can decide to run KUDOS at any point.

* Defined preservation of observations beyond OSCORE ID updates.

* Revised discussion section, including also communication overhead.

* Defined a well-known KUDOS resource and a KUDOS resource type.

* Editorial improvements.

B.9. Version -02 to -03

* Use of the OSCORE flag bit 0 to signal more flag bits.
* In UpdateCtx(), open for future key derivation different than HKDF.
* Simplified updateCtx() to use only Expand(); used to be METHOD 2.
* Included the Partial IV if the second KUDOS message is a response.
* Added signaling of support for KUDOS in EDHOC.
* Clarifications on terminology and reasons for rekeying.
* Updated IANA considerations.
* Editorial improvements.

B.10. Version -01 to -02

* Extended terminology.
* Moved procedure for preserving observations across key updates to main body.
* Moved procedure to update OSCORE Sender/Recipient IDs to main body.
* Moved key update without forward secrecy section to main body.
* Define signaling bits present in the ’x’ byte.
* Modifications and alignment of updateCtx() with EDHOC.
* Rules for deletion of old EDHOC keys PRK_out and PRK_exporter.
* Describe CBOR wrapping of involved nonces with examples.
* Renamed ’id detail’ to ’nonce’.
* Editorial improvements.

B.11. Version -00 to -01

* Recommendation on limits for CCM_8. Details in Appendix.
* Improved message processing, also covering corner cases.
* Example of method to estimate and not store 'count_q'.
* Added procedure to update OSCORE Sender/Recipient IDs.
* Added method for preserving observations across key updates.
* Added key update without forward secrecy.

Acknowledgments

The authors sincerely thank Christian Amsss, Carsten Bormann, Simon Bouget, Rafa Marin-Lopez, John Preu Mattsson, and Gran Selander for their feedback and comments.

The work on this document has been partly supported by the Sweden's Innovation Agency VINNOVA and the Celtic-Next projects CRITISEC and CYPRESS; and by the H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259).

Authors' Addresses

Rikard Hglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: rikard.hoglund@ri.se

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se