Key Update for OSCORE (KUDOS)

draft-ietf-core-oscore-key-update-03

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF 115, CoRE WG, November 7th, 2022

Recap

- (1) Key Update for OSCORE (KUDOS)
  - Renew the Master Secret and Master Salt; derive new Sender/Recipient keys
  - No change to the ID Context; can achieve Perfect Forward Secrecy
  - Loosely inspired by Appendix B.2 of OSCORE

- (2) AEAD Key Usage Limits in OSCORE (**)
  - Excessive use of the same key can enable breaking security properties of the AEAD algorithm*
  - Defining appropriate limits for OSCORE, for a variety of algorithms
  - Defining counters for key usage; message processing details; steps when limits are reached

- (3) Update of OSCORE Sender/Recipient IDs (**)
  - Exchanging desired new Recipient ID through a new CoAP Option

** Candidates for splitting out (see later slides)

*See also *draft-irtf-cfrg-aead-limits*

Rekeying procedure

- Key Update for OSCORE (KUDOS)
  - Message exchange to share nonces N1 and N2
  - Nonces are placed in new field in OSCORE CoAP option
  - *UpdateCtx()* function for deriving new OSCORE Security Context using the nonces and ‘x’ bytes
  - Extended OSCORE Option

```
 0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
|1|0|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+

 <- 1 byte -> <----- s bytes ------> <- 1 byte -> <--- m + 1 bytes --->
+------------+----------------------+---------------------------------+
| s (if any) | kid context (if any) | x (if any) | nonce (if any)     |
+------------+----------------------+------------+--------------------+
                                   /              \____
                                  /                    |
                                 /   0 1 2 3 4 5 6 7   |
+------------------+            |   +-+-+-+-+-+-+-+-+  |
| kid (if any) ... |            |   |0|0|b|p|   m   |  |
+------------------+            |   +-+-+-+-+-+-+-+-+  |
```

Nonce length

'x' byte contains additional signaling flags

Client-initiated rekeying

```
                         Client               Server
                      (initiator)          (responder)
                            |                   |
Generate N1                 |                   |
                            |                   |
CTX_1 =                     |                   |
  updateCtx(X1, N1,         |                   |
            CTX_OLD)        |                   |
                            |                   |
                            |    Request #1     |
Protect with CTX_1          |------------------>|
                            | OSCORE Option:    | CTX_1 =
                            |   ...             |  updateCtx(X1, N1,
                            |   d flag: 1       |            CTX_OLD)
                            |   X1              |
                            |   Nonce: N1       | Verify with CTX_1
                            |   ...             |
                            |                   | Generate N2
                            |                   |
                            |                   | CTX_NEW =
                            |                   |  updateCtx(Comb(X1,X2),
                            |                   |            Comb(N1,N2),
                            |                   |            CTX_OLD)
                            |                   |
                            |    Response #1    |
                            |<------------------| Protect with CTX_NEW
CTX_NEW =                   | OSCORE Option:    |
 updateCtx(Comb(X1,X2),     |   ...             |
           Comb(N1,N2),     |   Partial IV: 0   |
              CTX_OLD)      |   ...             |
                            |                   |
Verify with CTX_NEW         |  d flag: 1        |
                            |  X2               |
Discard CTX_OLD             |  Nonce: N2        |
                            |  ...              |

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

OSCORE flag bits

- Updates to bit registrations, based on discussion and agreement in [1]
  - As before bit 15, 'd', indicates a KUDOS message (presence of nonce and x)
  - Defined bit 0 for signaling a second flag byte (instead of bit 1)
    - No concrete other plan for bit 0 otherwise
    - This is an additional point about KUDOS updating RFC 8613
  - Changed the status of bit 1 from "Reserved" to "Unassigned"
  - Plan to soon request registration of bits 8, 16, 24, 32, 40, 48 as further extension bits

```
 0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
|1|0|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+

 <- 1 byte -> <----- s bytes ------> <- 1 byte -> <--- m + 1 bytes --->
+------------+----------------------+---------------------------------+
| s (if any) | kid context (if any) | x (if any) | nonce (if any)     |
+------------+----------------------+------------+--------------------+
                                   /              \____
                                  /                    |
                                 /   0 1 2 3 4 5 6 7   |
+------------------+            |   +-+-+-+-+-+-+-+-+  |
| kid (if any) ... |            |   |0|0|b|p|   m   |  |
+------------------+            |   +-+-+-+-+-+-+-+-+  |
```

Objections?

[1] https://mailarchive.ietf.org/arch/msg/core/x_Ix5a4PV-XcrvmLECtsC_CmoYs/

Method for context update

- **Previously** *updateCtx()* had two paths for key update
  - One based on EDHOC-KeyUpdate() (Method 1)
  - One based on Extract and Expand (Method 2)
  - Discussed at the CoRE interim meeting on 2022-09-28 [2]
    - Why not keep only Method 2?
    - No additional benefits from EDHOC-KeyUpdate

```
updateCtx(X, N, CTX_IN) {

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

  X_cbor = bstr .cbor X // CBOR bstr wrapping of X
  N_cbor = bstr .cbor N // CBOR bstr wrapping of N

  X_N = X_cbor | N_cbor

  oscore_key_length = < Size of CTX_IN.MasterSecret in bytes >

  Label = "key update"

  MSECRET_NEW = KUDOS-Expand-Label(CTX_IN.MasterSecret, Label,
                                   X_N, oscore_key_length)
              = KUDOS-Expand(CTX_IN.MasterSecret, ExpandLabel,
                             oscore_key_length)

  MSALT_NEW = N;

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

  Return CTX_OUT;

}
```

New version of updateCtx

- **Now** *updateCtx()* relies only on Expand
  - Only one code path: simplified implementations
  - Building internal value X_N for key derivation is easier
  - No need for fallback or signaling in case EDHOC-KeyUpdate can't be used
  - No need to support EDHOC or to think of EDHOC for the original key establishment

[2] https://datatracker.ietf.org/meeting/interim-2022-core-13/session/core

Not locked to HKDF anymore

- The updateCtx() function has been generalized
  - Previously, it used specifically HKDF-Expand()
  - Now it uses KUDOS-Expand()
    - Interface to the key derivation function used by OSCORE
  - This ensures flexibility and is future-proof

- If OSCORE uses an HKDF Algorithm …
  - KUDOS-Expand is mapped to HKDF-Expand
  - This would be the typical functionality of OSCORE today

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

- A potential, future update to RFC 8613 that admits a different KDF for OSCORE ...
  - … must define the mapping between that key derivation function and KUDOS-Expand()

Signal KUDOS support in EDHOC

EAD items are optional data that can be exchanged during an EDHOC execution

- Defined an EDHOC EAD item for signaling KUDOS support
  - The sender peer indicates if it supports KUDOS and in which modes
    - Peers learn of each other's KUDOS support during EDHOC execution

- Registered *ead_label* and defined values: ASK, NONE, FULL, PART
  - FULL or PART in EDHOC message_2 also asks the other peer to indicate whether it supports KUDOS in EDHOC message_3

```
EDHOC                                                    EDHOC
Initiator                                            Responder
|                                                            |
|               EAD_1: (TBD_LABEL, ASK)                      |
+----------------------------------------------------------->|
|                      message_1                             |
|                                                            |
|               EAD_2: (TBD_LABEL, FULL)                     |
|<-----------------------------------------------------------+
|                      message_2                             |
|                                                            |
|               EAD_3: (TBD_LABEL, FULL)                     |
+----------------------------------------------------------->|
|                      message_3                             |
|                                                            |
```

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.

Thoughts? Objections?

Further updates from IETF 114

- **Forbid sending non-KUDOS messages during a KUDOS execution**

- **In the client-initiated version of KUDOS**
  - The server's Partial IV is included in its KUDOS response message
  - This prevents reusing the same pair (AEAD nonce, key)
  - Later open point on how to better make this a general rule for OSCORE

- **Clarify what a CAPABLE and non-CAPABLE device must support**
  - Not CAPABLE device MUST support no-FS mode
  - CAPABLE device MUST support FS mode and SHOULD support no-FS mode

- **Restructured section about reasons for rekeying**

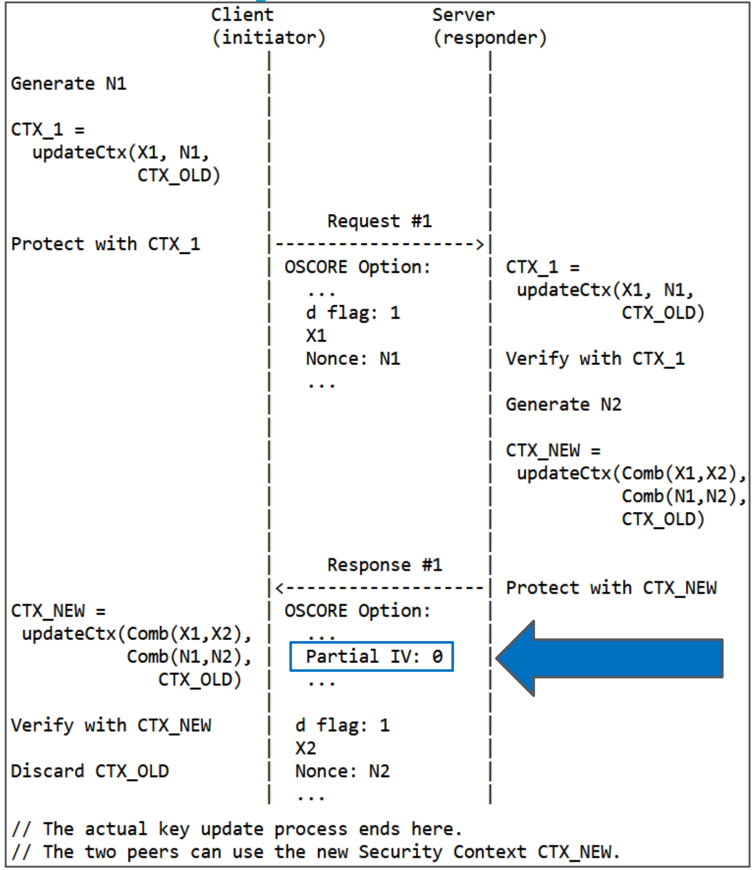
- **Improved retention policies of CTX_OLD**

Open point: Partial IV in responses

- **The Server MUST include a PIV in Response #1**
- This prevents a reuse of the same pair (AEAD nonce, key) from the server, as otherwise shown in this table:

Peer	Message	Nonce	Sender key from	Pair reuse
Client	Request #1	A	CTX_1	No
Server	Response #1	A	CTX_NEW	No
Client	Request #2	A	CTX_NEW	No
Server	Response #2	A	CTX_NEW	YES

- **This is now an ad-hoc fix for client-initiated KUDOS**
  - The server-initiated version does not have this problem.
  - For simplicity, it can be a general update for OSCORE
  - *If an OSCORE response is protected with a different Security Context than the corresponding request was unprotected with, the server MUST include its Sequence Number as Partial IV.*
  - *An exception is Appendix B.2 of RFC 8613, which does not have this problem by construction.*



```
                    Client                Server
                 (initiator)           (responder)
                      |                     |
Generate N1           |                     |
                      |                     |
CTX_1 =               |                     |
  updateCtx(X1, N1,   |                     |
            CTX_OLD)  |                     |
                      |                     |
                      |     Request #1      |
Protect with CTX_1    |-------------------->|
                      | OSCORE Option:      | CTX_1 =
                      |   ...               |  updateCtx(X1, N1,
                      |   d flag: 1         |            CTX_OLD)
                      |   X1                |
                      |   Nonce: N1         | Verify with CTX_1
                      |   ...               |
                      |                     | Generate N2
                      |                     |
                      |                     | CTX_NEW =
                      |                     |  updateCtx(Comb(X1,X2),
                      |                     |            Comb(N1,N2),
                      |                     |            CTX_OLD)
                      |                     |
                      |     Response #1     |
                      |<--------------------| Protect with CTX_NEW
CTX_NEW =             | OSCORE Option:      |
 updateCtx(Comb(X1,X2),|   ...              |
           Comb(N1,N2),|  Partial IV: 0     |
              CTX_OLD) |   ...              |
                      |                     |
Verify with CTX_NEW   |  d flag: 1          |
                      |  X2                 |
Discard CTX_OLD       |  Nonce: N2          |
                      |  ...                |

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

Objections?

Future structure of the document

- **Content on AEAD limits – Section 2 and Appendix A**
  - Split out into a separate, WG document?
  - From the 2022-09-28 CoRE interim meeting [2]: strong preference to split out.
  - Shall we confirm to do it?

- **Method for updating the OSCORE Sender/Recipient IDs – Section 5**
  - This can be run stand-alone or embedded in a KUDOS execution
  - Split out into a separate, WG document?
  - From the 2022-09-28 CoRE interim meeting [2]: mild preference or no opinion to split out.
  - We still need work on that section (mainly discuss examples and preserving observation)
  - Proposal: keep for now and bring it up again when the section is completed?

- **If both splits happen, this documents would be focused on KUDOS**

[2] https://datatracker.ietf.org/meeting/interim-2022-core-13/session/core

Main next steps

- **Addressed open points from the previous slides**
  - Document restructuring/split
  - General rule for Partial IV in responses across a key update

- **Text discussing soft limits vs. hard limits**
  - Based on feedback from Rafa Marin-Lopez

- **OSCORE ID update examples**
  - Textual description of provided examples
  - Preservation of ongoing Observation

- **Comments and reviews are welcome!**

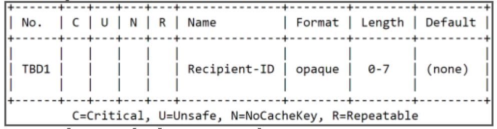
Thank you!

Comments/questions?

https://github.com/core-wg/oscore-key-update

Update of Sender/Recipient IDs

- Method for updating peers' OSCORE Sender/Recipient IDs
  - Based on earlier discussions on the mailing list [1][2] and on [3]
  - This procedure can be embedded in a KUDOS execution or run standalone
  - This procedure can be initiated by a client or by a server
  - Content moved from old appendix to document body and improved (Section 5)



No.	C	U	N	R	Name	Format	Length	Default
TBD1					Recipient-ID	opaque	0-7	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

- Properties
  - The sender indicates its new wished Recipient ID in the new Recipient-ID Option (class E)
  - Both peers have to opt-in and agree in order for the IDs to be updated
  - Changing IDs practically triggers derivation of new OSCORE Security Context
  - Must not be done immediately following a reboot (e.g., KUDOS must be run first)
  - Offered Recipient ID must be not used yet under (Master Secret, Master Salt, ID Context)
  - Received Recipient ID must not be used yet as own Sender ID under the same triple

- Examples are provided in Sections 5.1.1 and 5.1.2

[1] https://mailarchive.ietf.org/arch/msg/core/GXsKO4wKdt3RTZnQZxOzRdIG9QI/
[2] https://mailarchive.ietf.org/arch/msg/core/ClwcSF0BUVxDas8BpgT0WY1yQrY/
[3] https://github.com/core-wg/oscore/issues/263#issue-946989659