Key Update for OSCORE (KUDOS)

draft-ietf-core-oscore-key-update-08

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF CoRE WG meeting – IETF 120 – July 24th, 2024

Recap

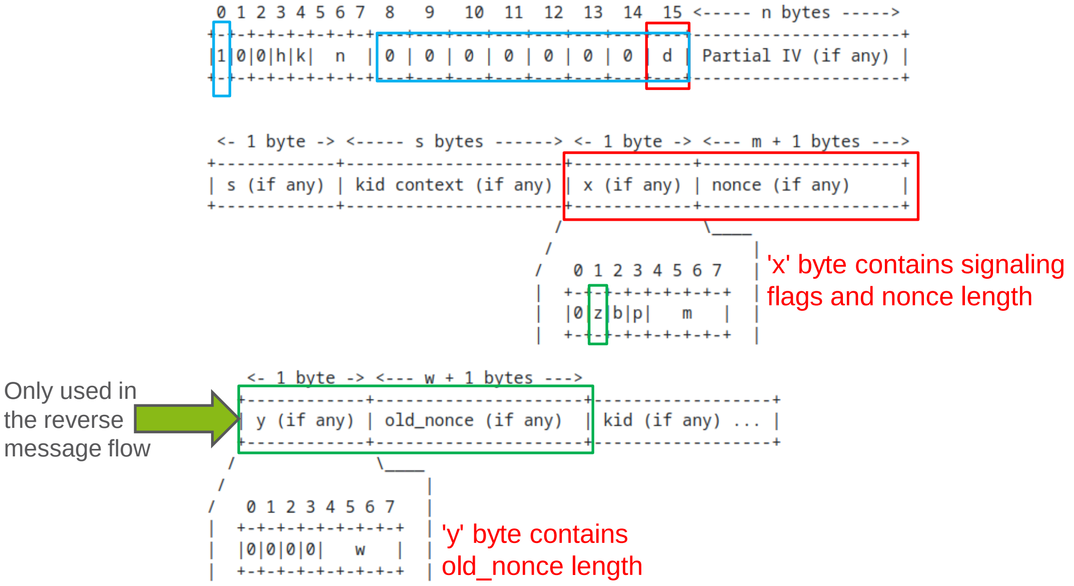
- (1) Key Update for OSCORE (KUDOS)
  - Renew the Master Secret and Master Salt; derive new Sender/Recipient keys
  - No change to the ID Context; can achieve Perfect Forward Secrecy
  - Agnostic of the key establishment method originally used
  - Loosely inspired by Appendix B.2 of OSCORE

- (2) AEAD Key Usage Limits in OSCORE
  - Was split out as a separate draft as of March 2023: *draft-ietf-core-oscore-key-limits*

- (3) Procedure for updating OSCORE Sender/Recipient IDs
  - Was split out as a separate draft as of March 2024: *draft-ietf-core-oscore-id-update*

Rekeying Procedure

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



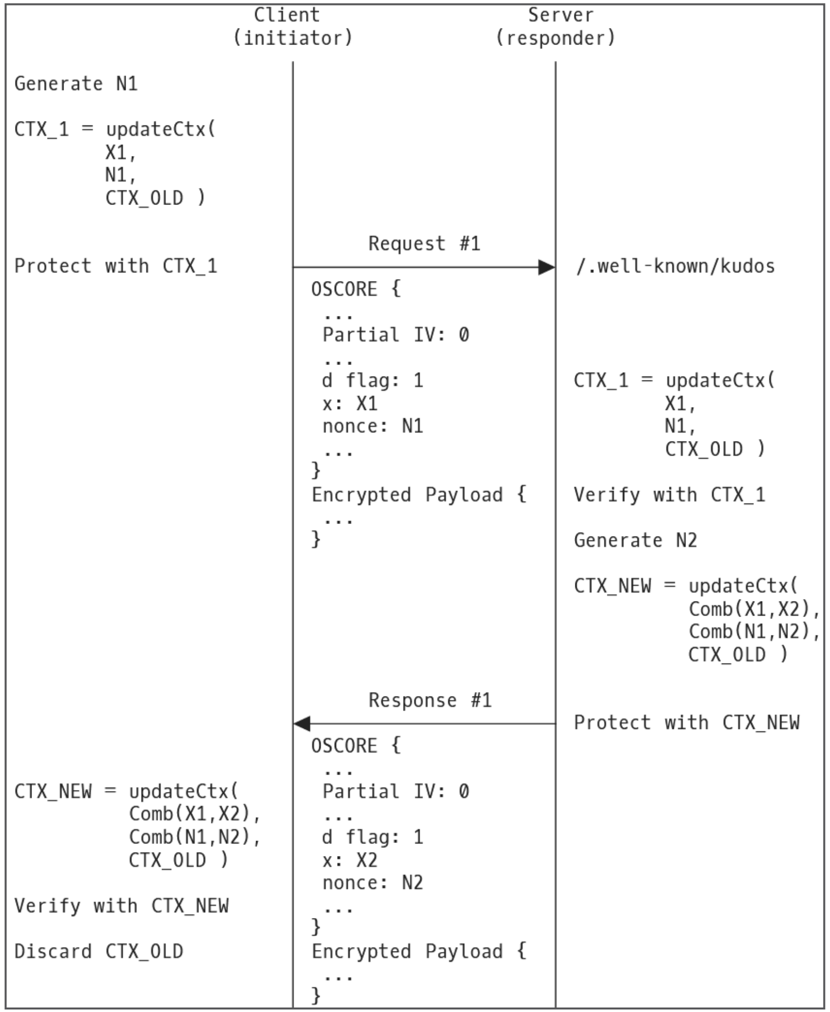
```
 0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
|1|0|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+

 <- 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   |
                                 |  +-+-+-+-+-+-+-+-+  |
                                 |  |0|z|b|p|   m   |  |
                                 |  +-+-+-+-+-+-+-+-+  |

    <- 1 byte -> <--- w + 1 bytes --->
   +------------+---------------------+------------------+
   | y (if any) | old_nonce (if any)  | kid (if any) ... |
   +------------+---------------------+------------------+
   /              \____
  /                    |
 /   0 1 2 3 4 5 6 7   |
 |  +-+-+-+-+-+-+-+-+  |
 |  |0|0|0|0|   w   |  |
 |  +-+-+-+-+-+-+-+-+  |
```

'x' byte contains signaling flags and nonce length
Only used in the reverse message flow
'y' byte contains old_nonce length

KUDOS forward message flow



```
                          Client                 Server
                       (initiator)            (responder)
                            |                      |
Generate N1                 |                      |
                            |                      |
CTX_1 = updateCtx(          |                      |
        X1,                 |                      |
        N1,                 |                      |
        CTX_OLD )           |                      |
                            |                      |
                            |      Request #1      |
Protect with CTX_1          |--------------------->| /.well-known/kudos
                            | OSCORE {             |
                            |  ...                 |
                            |  Partial IV: 0       |
                            |  ...                 |
                            |  d flag: 1           | CTX_1 = updateCtx(
                            |  x: X1               |         X1,
                            |  nonce: N1           |         N1,
                            |  ...                 |         CTX_OLD )
                            | }                    |
                            | Encrypted Payload {  | Verify with CTX_1
                            |  ...                 |
                            | }                    | Generate N2
                            |                      |
                            |                      | CTX_NEW = updateCtx(
                            |                      |           Comb(X1,X2),
                            |                      |           Comb(N1,N2),
                            |                      |           CTX_OLD )
                            |                      |
                            |      Response #1     |
                            |<---------------------| Protect with CTX_NEW
                            | OSCORE {             |
                            |  ...                 |
CTX_NEW = updateCtx(        |  Partial IV: 0       |
          Comb(X1,X2),      |  ...                 |
          Comb(N1,N2),      |  d flag: 1           |
          CTX_OLD )         |  x: X2               |
                            |  nonce: N2           |
Verify with CTX_NEW         |  ...                 |
                            | }                    |
Discard CTX_OLD             | Encrypted Payload {  |
                            |  ...                 |
                            | }                    |
```

Updates to v-08 (1/3)

- **Note on using the CoAP No-Response Option**
  - KUDOS Response #1 might not be the direct response to KUDOS Request #1, but rather a response to a different request
  - If the client knows for sure that this will be the case, the client may use the CoAP No-Response option in its KUDOS Request #1, and KUDOS will still complete

- **Avoid problems for two simultaneously started key updates**
  - It may happen that two peers initiate KUDOS simultaneously, that is:
    - Both peers first act as initiator in a KUDOS execution, sending the first KUDOS message
    - Then, both peers act as responder in the other KUDOS execution
  - To avoid problems, the two KUDOS executions must not both finish
  - **Solution described in Section 4.3.3:**
    - If P1 is Initiator in a KUDOS execution E1 with P2, and ...
    - P1 receives a first KUDOS message from P2 for starting a KUDOS execution E2, ...
    - then P1 MUST abort the execution E2 and MUST reply to P2 with a CoAP reset message

Updates to v-08 (2/3)

- **New OSCORE Security Contexts have a Notification Number that is uninitialized**
  - Notification Number is used for replay detection of Observe Notifications (see RFC 8613)
  - Value: largest Partial IV of the received notifications for an associated Observe registration
  - Clarified that newly derived OSCORE Security Contexts has this value set as uninitialized

- **Editorial improvements and fixes**
  - General clarifications
  - Restructuring and splitting of long section *Key Update with Forward Secrecy*
    - Now split into 4.3.1- 4.3.5

- **IANA considerations for CoAP Option Numbers Registry**
  - Requested to update the "OSCORE" entry in the "CoAP Option Numbers" registry, with a reference to this document
  - As this document is specifying an updated, extended format of the CoAP OSCORE Option

Updates to v-08 (3/3)

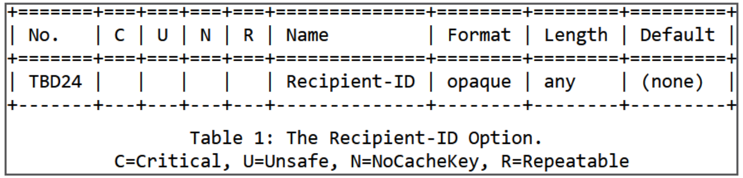
- **Expanded security considerations**
  - Added reference to relevant paper *Security of Symmetric Ratchets and Key Chains - Implications for Protocols like TLS 1.3, Signal, and PQ3* [1]
  - New paragraph describing relevant information about KUDOS from that paper
    - Rekeying with a symmetric key exchange is not intended to substitute an 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)

- **Discuss possible deadlock situation on servers**
  - It might be the case that a peer is only a CoAP server (i.e., cannot send requests)
  - If such a server reaches key usage limits for its OSCORE Recipient Key:
    - It cannot safely decrypt further incoming messages
    - It cannot execute KUDOS as initiator, as it cannot decrypt a non-KUDOS protected request
  - That server can only run KUDOS if the client starts KUDOS using the forward message flow

[1] https://eprint.iacr.org/2024/220

Update of Sender/Recipient IDs

- **Recap: Method for updating peers' OSCORE Sender/Recipient IDs**
  - This procedure can be embedded in a KUDOS execution or run standalone
  - This procedure can be initiated by a client or by a server



No.	C	U	N	R	Name	Format	Length	Default
TBD24					Recipient-ID	opaque	any	(none)

Table 1: The Recipient-ID Option.
C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

- **Properties**
  - The message 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 <u>not</u> be done immediately following a reboot if run standalone (e.g., KUDOS must be run first)
  - Offered Recipient ID must <u>not</u> be used yet under the same (Master Secret, Master Salt, ID Context)
  - Received Recipient ID must <u>not</u> be used yet as own Sender ID under the same triple

- **Examples are provided in Sections 2.1.1 and 2.1.2**

Document Updates

Submitted version -01 before cutoff for IETF 120

- **Specify that the Recipient-ID Option might be empty**
  - The requested new Recipient ID may be the empty byte string
  - If so, the option value SHALL be empty (Option Length = 0)

- **Failure cases when running the ID update procedure integrated with KUDOS**
  - The KUDOS procedure succeeds, while the OSCORE ID update procedure fails
    - Use the newly derived OSCORE Security Context CTX_NEW
    - Continue using the old Sender and Recipient IDs
  - The OSCORE ID update procedure succeeds, while the KUDOS procedure fails
    - Continue using the old OSCORE Security Context CTX_OLD
    - Use the new Sender and Recipient IDs

- **Enforce maximum length for the Recipient IDs**
  - Handling failure case: the length of the received Recipient-ID Option exceeds the maximum length of OSCORE Sender/Recipient IDs for the OSCORE AEAD algorithm
  - The length of the Recipient-ID Option must not exceed the length of the AEAD nonce minus 6

Summary and next steps

- **Related point on OSCORE key usage limits document**
  - Submitted new version -03 in July 2024
  - Monitoring updates to *cfrg-aead-limits* and waiting for possible feedback

- **Process the recent KUDOS review by Christian Amsüss - Thanks!**
  - See mail: https://mailarchive.ietf.org/arch/msg/core/QGS8QfeySlrTKYRvFnEH7laxBDk/

- **KUDOS implementation**
  - Finished implementation in Java supporting the forward message flow [2]
  - Mature implementation in C for Contiki-NG supporting server-side forward message flow

- **Comments and reviews are welcome!**

[2] https://github.com/rikard-sics/californium/blob/oscore_cmd/cf-oscore/

Thank you!

Comments/questions?

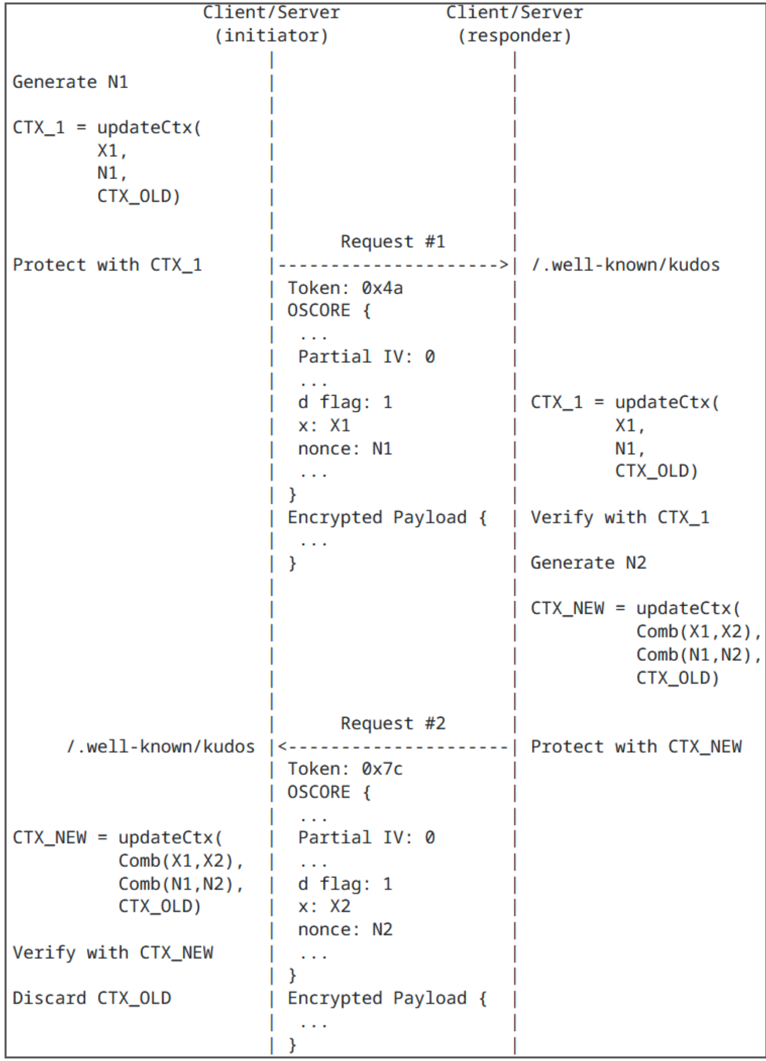
https://github.com/core-wg/oscore-key-update

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

Backup

Flexible Message Pair

- **An execution of KUDOS does not need to be a request/response message pair**
  - Instead, more flexible messages flows can be allowed (e.g., two CoAP requests)
- **E.g., a scenario using the Resource Directory where both KUDOS messages are requests**
- **Other alternatives are also possible**
  - Second KUDOS message is a response to a different request than the first KUDOS message
  - Could be the case where there are ongoing observations between the peers



```
                      Client/Server          Client/Server
                       (initiator)            (responder)
                            |                      |
Generate N1                 |                      |
                            |                      |
CTX_1 = updateCtx(          |                      |
        X1,                 |                      |
        N1,                 |                      |
        CTX_OLD)            |                      |
                            |                      |
                            |      Request #1      |
Protect with CTX_1          |--------------------->| /.well-known/kudos
                            | Token: 0x4a          |
                            | OSCORE {             |
                            |  ...                 |
                            |  Partial IV: 0       |
                            |  ...                 |
                            |  d flag: 1           | CTX_1 = updateCtx(
                            |  x: X1               |         X1,
                            |  nonce: N1           |         N1,
                            |  ...                 |         CTX_OLD)
                            | }                    |
                            | Encrypted Payload {  | Verify with CTX_1
                            |  ...                 |
                            | }                    | Generate N2
                            |                      |
                            |                      | CTX_NEW = updateCtx(
                            |                      |           Comb(X1,X2),
                            |                      |           Comb(N1,N2),
                            |                      |           CTX_OLD)
                            |                      |
                            |      Request #2      |
     /.well-known/kudos     |<---------------------| Protect with CTX_NEW
                            | Token: 0x7c          |
                            | OSCORE {             |
                            |  ...                 |
CTX_NEW = updateCtx(        |  Partial IV: 0       |
          Comb(X1,X2),      |  ...                 |
          Comb(N1,N2),      |  d flag: 1           |
          CTX_OLD)          |  x: X2               |
                            |  nonce: N2           |
Verify with CTX_NEW         |  ...                 |
                            | }                    |
Discard CTX_OLD             | Encrypted Payload {  |
                            |  ...                 |
                            | }                    |
```

KUDOS Messages as Regular Application Messages

- **Allow the client to initiate KUDOS with a 'normal' application message**
  - The client wants to send an application request to the server. Thus, this message also serves as a KUDOS message.
  - Practically KUDOS request messages can target any resource at the server:
    - In the forward message flow, the client sends the application message that it currently wants to send as a KUDOS message
    - The server cannot be sure the request is fresh, thus if it requires freshness it MUST respond with a protected 4.01 response.
    - Then the client re-sends a non-KUDOS request protected with CTX_NEW, typically with the same content as the first request.
- **The *./well-known/kudos* resource can still be used**
  - For instance, if the client does not want to send any application request currently
  - In this case, the CoAP request composed before OSCORE protection should not include an application payload

```
                     Client                Server
                   (initiator)          (responder)
                        |                      |
Generate N1             |                      |
                        |                      |
CTX_1 = updateCtx(      |                      |
        X1,             |                      |
        N1,             |                      |
        CTX_OLD)        |                      |
                        |                      |
                        |      Request #1      |
Protect with CTX_1      |--------------------->| /temp
                        | OSCORE {             |
                        |  ...                 |
                        |  Partial IV: 0       |
                        |  ...                 |
                        |  d flag: 1           | CTX_1 = updateCtx(
                        |  x: X1               |         X1,
                        |  nonce: N1           |         N1,
                        |  ...                 |         CTX_OLD)
                        | }                    |
                        | Encrypted Payload {  | Verify with CTX_1
                        |  0.03 (PUT)          |
                        |  ...                 |
                        |  Application Payload |
                        | }                    | Generate N2
                        |                      |
                        |                      | CTX_NEW = updateCtx(
                        |                      |             Comb(X1,X2),
                        |                      |             Comb(N1,N2),
                        |                      |             CTX_OLD)
                        |                      |
                        |      Response #1     |
                        |<---------------------| Protect with CTX_NEW
                        | OSCORE {             |
                        |  ...                 |
CTX_NEW = updateCtx(    |  Partial IV: 0       |
            Comb(X1,X2),|  ...                 |
            Comb(N1,N2),|  d flag: 1           |
            CTX_OLD)    |  x: X2               |
                        |  nonce: N2           |
Verify with CTX_NEW     |  ...                 |
                        | }                    |
Discard CTX_OLD         | Encrypted Payload {  |
                        |  4.01 (Unauthorized) |
                        |  ...                 |
                        |  Application Payload |
                        | }                    |
```

Key Limits Overview

- **Working group document**
  - Content split out from *Key Update for OSCORE (KUDOS)* (draft-ietf-core-oscore-key-update)
  - Discussed during previous core interim on 2022-09-28 [1]
  - Also discussed and confirmed during IETF 115 [2]

- **Content of the draft: 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

[1] https://datatracker.ietf.org/meeting/interim-2022-core-13/session/core
[2] https://datatracker.ietf.org/meeting/115/session/core

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