Key Update for OSCORE (KUDOS)

draft-ietf-core-oscore-key-update-10

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF CoRE WG meeting – IETF 122 – March 18th, 2025

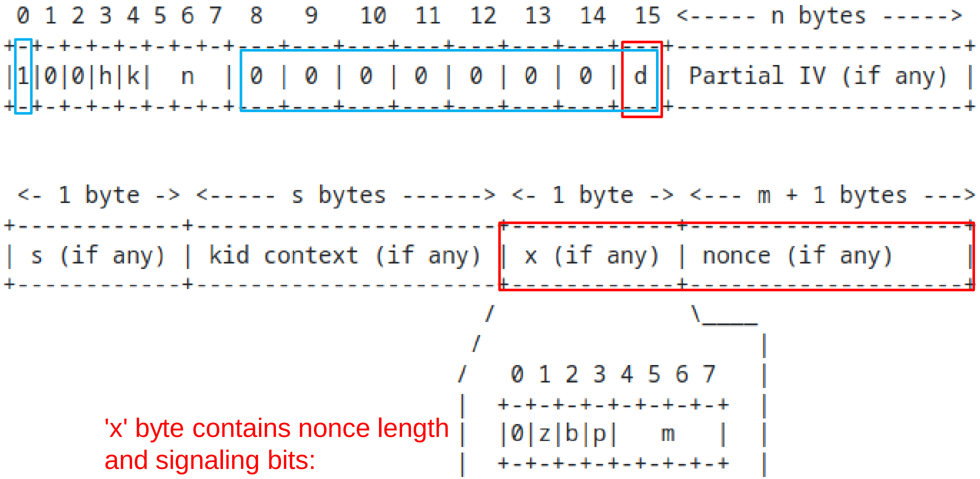
Recap

- 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 Forward Secrecy
  - Agnostic of the key establishment method originally used
  - Loosely inspired by Appendix B.2 of OSCORE
  - The peers update their current context CTX_OLD, deriving a new context CTX_NEW
  - Now re-designed using a more flexible and simpler approach – Thanks Christian!

Rekeying Procedure

- **Key Update for OSCORE (KUDOS)**
  - Message exchange to share two nonces N1 and N2
    - Decoupled from request/response and client/server concepts
  - Nonces are placed in new fields in OSCORE CoAP option
  - *UpdateCtx()* function for deriving new OSCORE Security Context using the two nonces, two ‘x’ bytes and CTX_OLD
  - Two modes
    - FS mode providing forward secrecy
    - No-FS mode for very constrained devices
  - No change of OSCORE identifiers
  - Expected to complete in 1 round trip

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   |  |
                                 |  +-+-+-+-+-+-+-+-+  |
```

'x' byte contains nonce length and signaling bits:
- 'b': Preserving observations
- 'p': FS / no-FS mode
- 'z': KUDOS message type

Main changes for v-10

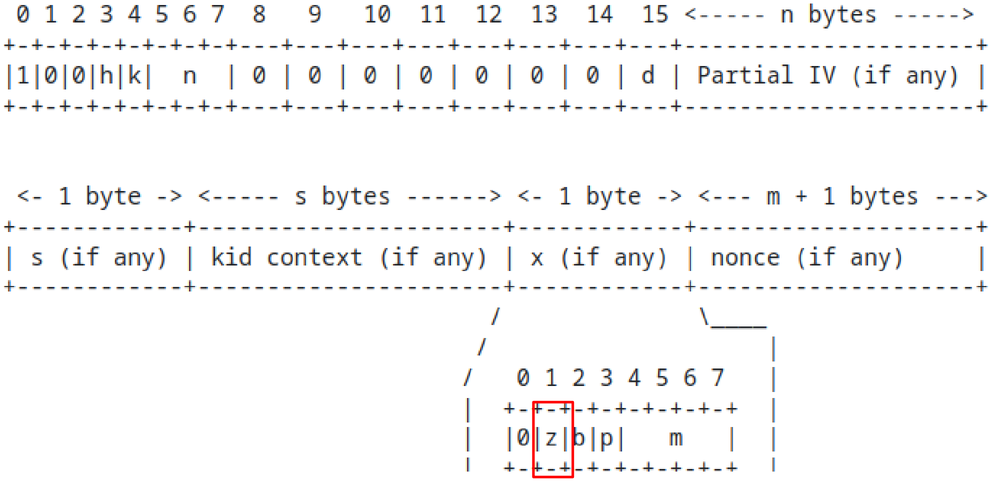
- **Major re-design based on an updated state machine**
- **More flexible design**
  - Removed concept of forward/reverse message flow
  - Removed "first" and "second" KUDOS message
  - Removed rigid roles of "Initiator" and "Responder"
  - Removed ordering of nonces by time of arrival/transmission
  - No need to send the other peer's nonce on the wire
- **Less aspects and potential issues to think/worry about**
  - More adaptive to different types of message exchanges
  - Simpler to ensure the absence of deadlocks
- **Formally defined CTX_BOOTSTRAP**
  - Relevant for non-CAPABLE devices, unable to write in persistent memory
  - The context built from the Bootstrap Master Secret/Salt and used in the no-FS mode
    - Always used as "baseline" Security Context to derive the new one with updateCtx()

KUDOS Message Types

- **Two types of KUDOS messages, distinguished by the 7th least significant bit 'z' in the 'x' byte**
  - Indicates if only one or both nonces have been exchanged

- **z = 0: “divergent message”**
- This message is protected with the temporary Security Context CTX_TEMP (was CTX_1)
- The sender peer is offering its own nonce in the message and waiting to receive the other peer’s nonce.

- **z = 1: “convergent message”**
- This message is protected with the final Security Context CTX_NEW.
- The sender peer is offering its own nonce in the message, has received the other peer’s nonce, and is going to wait for key confirmation



```
 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   |  |
                                 |  +-+-+-+-+-+-+-+-+  |
```

Note: The z bit used to have another meaning

Main changes for v-10

- **A pair (X, Nonce) offered by a peer is bound to CTX_OLD**
  - ...with the intent to use the same pair until a successful KUDOS execution completes
  - This pair is generated before invoking updateCtx(), in case a pair is not already associated with the CTX_OLD
  - The pair is deleted upon deletion of CTX_OLD

- **Lexicographical sorting of the input in updateCtx()**
  - ...to make the solution agnostic of the nonces sending/arrival time
  - Same output for updateCtx(X1 | N1, X2 | N2, CTX_OLD) and updateCtx(X2 | N2, X1 | N1, CTX_OLD).
  - The peers get the same output regardless of which nonce they consider to be "first" and "second"

KUDOS States

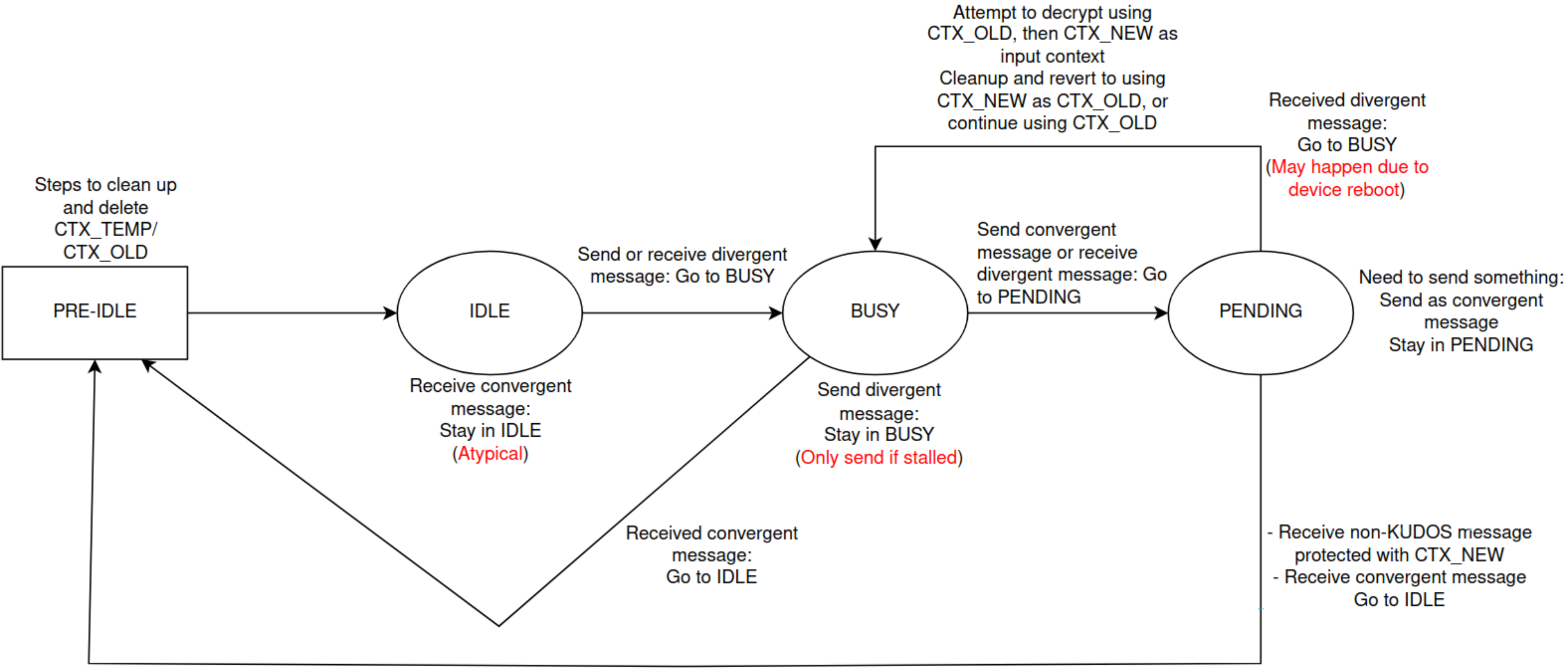
- **Three possible states: IDLE, BUSY, and PENDING**
  - Normally, the peer is in the IDLE state, i.e., in "equilibrium“

- **A peer starts a KUDOS execution upon entering the BUSY state**

- **A peer successfully 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

- **A peer can locally represent its current state using 2 bits**
  - (00) IDLE - The peer is not running KUDOS
  - (01) BUSY - The peer has not offered a nonce, but has received the nonce from the other peer
  - (10) BUSY - The peer 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

KUDOS State Machine



PRE-IDLE

Steps to clean up and delete CTX_TEMP/ CTX_OLD

IDLE

Receive convergent message: Stay in IDLE (Atypical)

Send or receive divergent message: Go to BUSY

BUSY

Attempt to decrypt using CTX_OLD, then CTX_NEW as input context
Cleanup and revert to using CTX_NEW as CTX_OLD, or continue using CTX_OLD

Send divergent message: Stay in BUSY (Only send if stalled)

Received convergent message: Go to IDLE

Send convergent message or receive divergent message: Go to PENDING

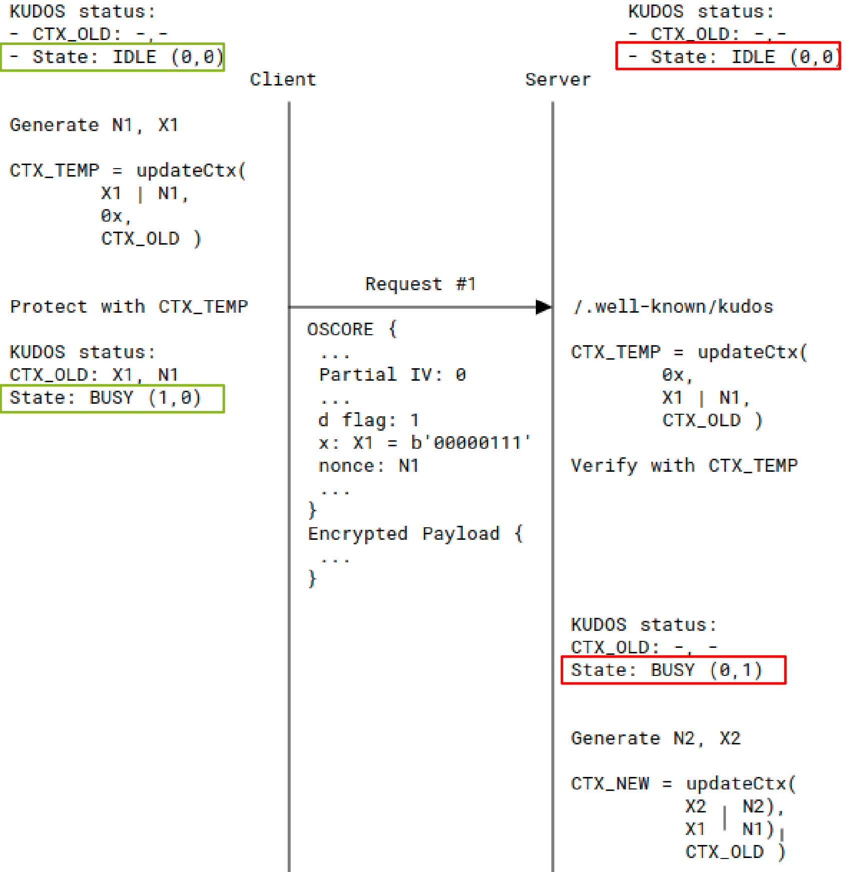
PENDING

Received divergent message: Go to BUSY (May happen due to device reboot)

Need to send something: Send as convergent message Stay in PENDING

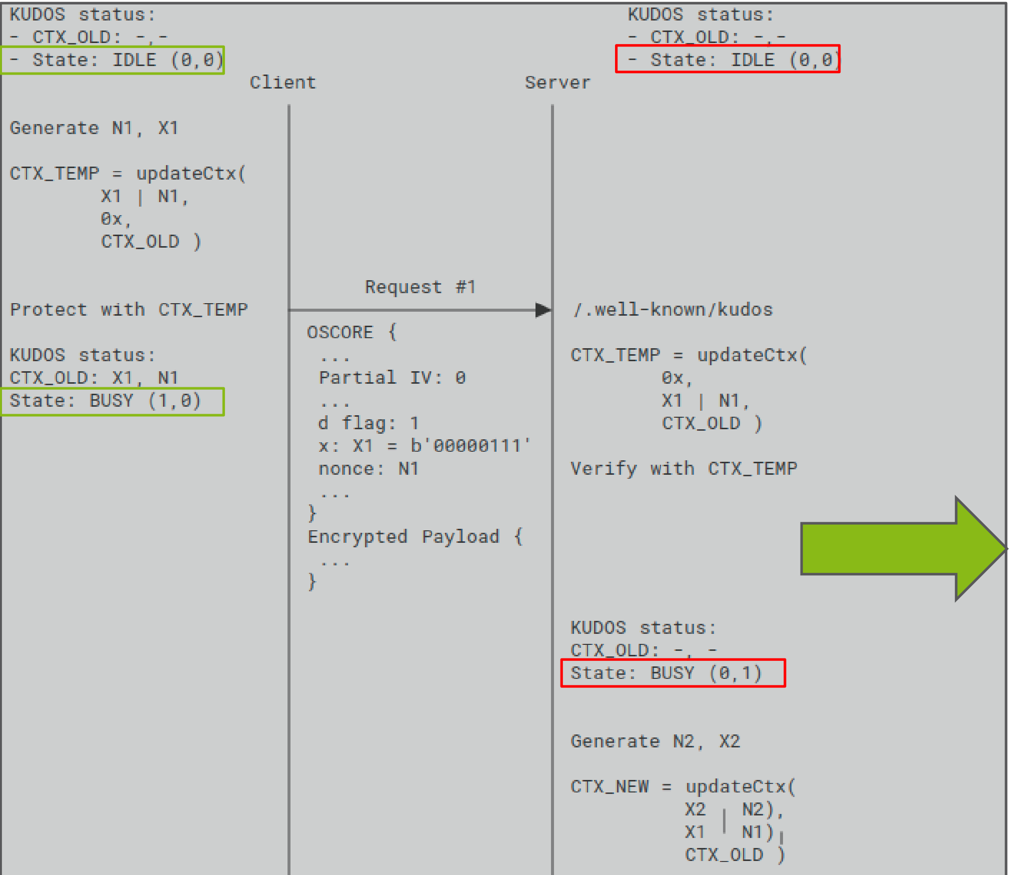
- Receive non-KUDOS message protected with CTX_NEW
- Receive convergent message
Go to IDLE

Example Execution

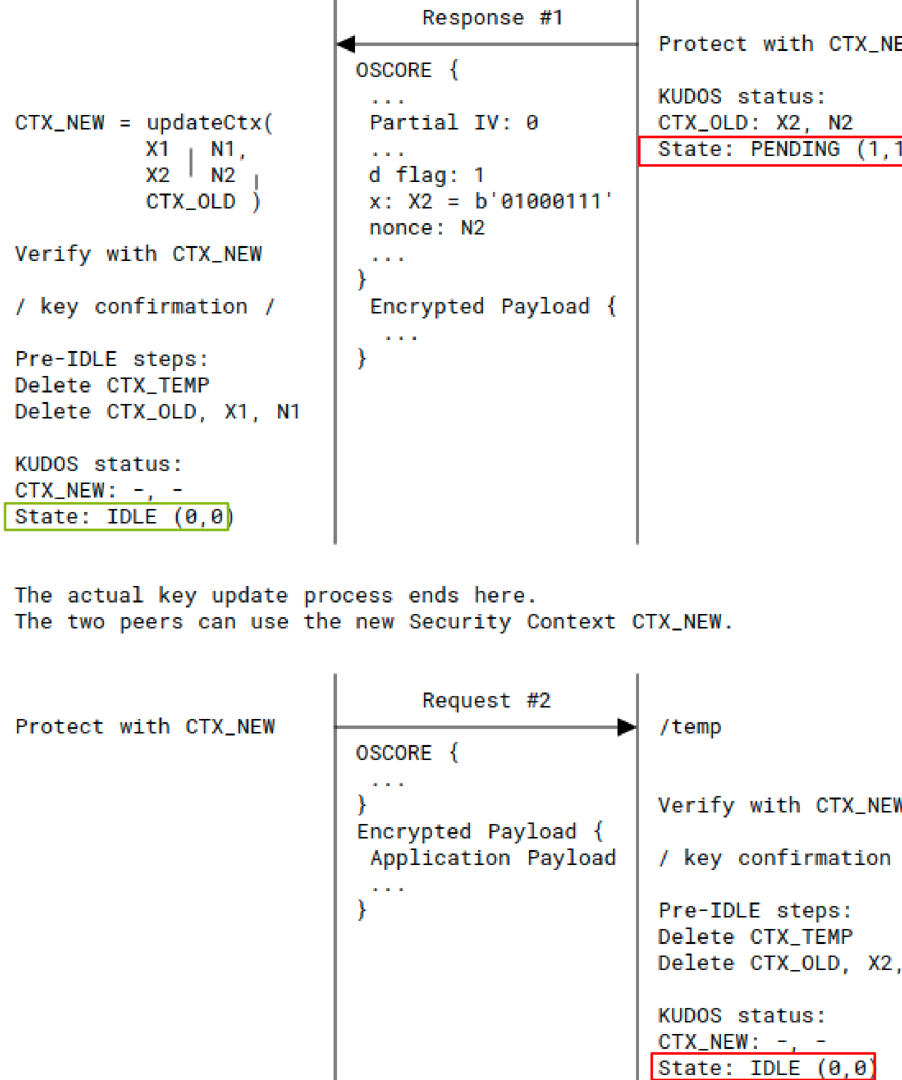


```
KUDOS status:                                          KUDOS status:
- CTX_OLD: -,-                                         - CTX_OLD: -,-
- State: IDLE (0,0)                                    - State: IDLE (0,0)
                      Client                   Server

Generate N1, X1

CTX_TEMP = updateCtx(
        X1 | N1,
        0x,
        CTX_OLD )


                            Request #1
Protect with CTX_TEMP    ----------------------->  /.well-known/kudos
                          OSCORE {
KUDOS status:              ...                      CTX_TEMP = updateCtx(
CTX_OLD: X1, N1            Partial IV: 0                    0x,
State: BUSY (1,0)          ...                              X1 | N1,
                           d flag: 1                        CTX_OLD )
                           x: X1 = b'00000111'
                           nonce: N1                Verify with CTX_TEMP
                           ...
                          }
                          Encrypted Payload {
                           ...
                          }


                                                    KUDOS status:
                                                    CTX_OLD: -, -
                                                    State: BUSY (0,1)


                                                    Generate N2, X2

                                                    CTX_NEW = updateCtx(
                                                              X2 | N2),
                                                              X1 | N1)|
                                                              CTX_OLD )
```

Example Execution



```
KUDOS status:                                    KUDOS status:
- CTX_OLD: -,-                                   - CTX_OLD: -,-
- State: IDLE (0,0)                              - State: IDLE (0,0
                     Client                 Server
                        |                      |
Generate N1, X1         |                      |
                        |                      |
CTX_TEMP = updateCtx(   |                      |
        X1 | N1,        |                      |
        0x,             |                      |
        CTX_OLD )       |                      |
                        |                      |
                        |      Request #1      |
Protect with CTX_TEMP   |--------------------->| /.well-known/kudos
                        | OSCORE {             |
KUDOS status:           |  ...                 | CTX_TEMP = updateCtx(
CTX_OLD: X1, N1         |  Partial IV: 0       |         0x,
State: BUSY (1,0)       |  ...                 |         X1 | N1,
                        |  d flag: 1           |         CTX_OLD )
                        |  x: X1 = b'00000111' |
                        |  nonce: N1           | Verify with CTX_TEMP
                        |  ...                 |
                        | }                    |
                        | Encrypted Payload {  |
                        |  ...                 |
                        | }                    |
                        |                      |
                        |                      | KUDOS status:
                        |                      | CTX_OLD: -, -
                        |                      | State: BUSY (0,1)
                        |                      |
                        |                      | Generate N2, X2
                        |                      |
                        |                      | CTX_NEW = updateCtx(
                        |                      |          X2 | N2),
                        |                      |          X1 | N1)|
                        |                      |          CTX_OLD )
```



```
                              Response #1
                         |<------------------------| Protect with CTX_NE
                         | OSCORE {                |
                         |  ...                    | KUDOS status:
CTX_NEW = updateCtx(     |  Partial IV: 0          | CTX_OLD: X2, N2
          X1 | N1,       |  ...                    | State: PENDING (1,1
          X2 | N2 |      |  d flag: 1              |
          CTX_OLD )      |  x: X2 = b'01000111'    |
                         |  nonce: N2              |
Verify with CTX_NEW      |  ...                    |
                         | }                       |
/ key confirmation /     | Encrypted Payload {     |
                         |  ...                    |
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.

                               Request #2
Protect with CTX_NEW     |------------------------>| /temp
                         | OSCORE {                |
                         |  ...                    |
                         | }                       | Verify with CTX_NEW
                         | Encrypted Payload {     |
                         |  Application Payload    | / key confirmation
                         |  ...                    |
                         | }                       | Pre-IDLE steps:
                         |                         | Delete CTX_TEMP
                         |                         | Delete CTX_OLD, X2,
                         |                         |
                         |                         | KUDOS status:
                         |                         | CTX_NEW: -, -
                         |                         | State: IDLE (0,0)
```

Key Usage Limits & ID Update

- **ID Update (draft-ietf-core-oscore-id-update)**
  - **Recap**: Method for updating peers' OSCORE Sender/Recipient IDs. Can be initiated by a client or by a server
  - Submitted new version in January with main change to split overly long sections
  - Next steps
    - Re-consider its design, to be in the same spirit of the new KUDOS design
    - Add more examples including failure cases

- **Key Usage Limits (draft-ietf-core-oscore-key-limits)**
  - **Recap**: OSCORE-specific safe limits for Sender/Recipient Key usage
    - Safe number of encryptions using the Sender Key
    - Safe number of failed decryptions using the Recipient Key
  - Next steps: Align to the document from CFRG as it develops [1]
    - There is ongoing discussion in the CFRG mailing list [2]

[1] https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-limits/

[2] https://mailarchive.ietf.org/arch/msg/cfrg/CQ6MaMX1t96qxzxJK8cpTPVA46g/

Summary and next steps

- **Consider simplifications and improvements to the state machine**
  - E.g., avoiding complex and resource-intensive paths when determined not needed

- **KUDOS implementations**
  - Update the implementation in Java [1] to be aligned with the latest design
  - Update the implementation in C for Contiki-NG to be aligned with the latest design

- **Process a few remaining minor issues captured in the Github repo**

- **Comments and reviews are welcome!**

[1] 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

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

Backup

Key Usage 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]

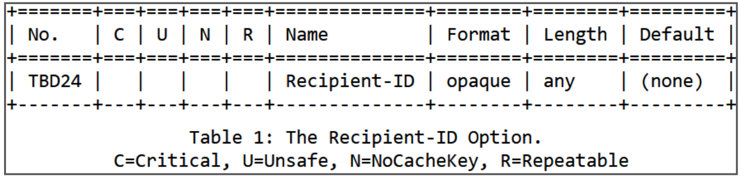
- **Content of the draft: AEAD Key Usage Limits in OSCORE**
  - Excessive use of the same key can enable breaking security properties of the AEAD algorithm*
  - Defining appropriate limits for OSCORE, for a variety of algorithms
  - Defining counters for key usage; message processing details; steps when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

Update of Sender/Recipient IDs

- **Recap: Method for updating peers' OSCORE Sender/Recipient IDs**
  - 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 not be done immediately following a reboot if run standalone (e.g., KUDOS must be run first)
  - Offered Recipient ID must not be used yet under the same (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 2.1.1 and 2.1.2**