Key Update for OSCORE (KUDOS)

draft-ietf-core-oscore-key-update-06

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF CoRE WG interim – January 17th, 2024

Recap

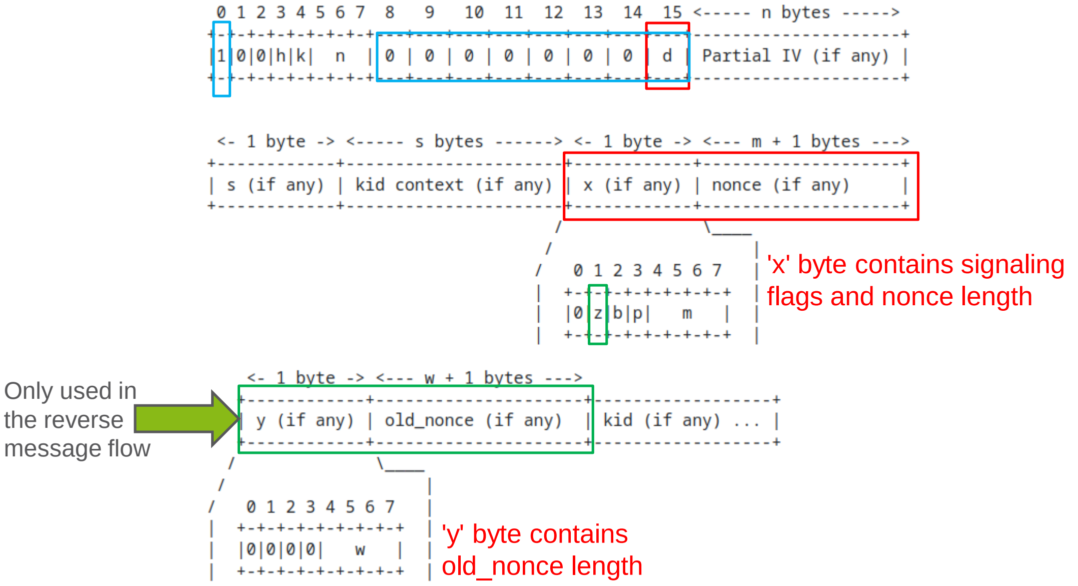
- (1) Key Update for OSCORE (KUDOS)
  - Renew the Master Secret and Master Salt; derive new Sender/Recipient keys
  - No change to the ID Context; can achieve Perfect Forward Secrecy
  - Agnostic of the key establishment method originally used
  - Loosely inspired by Appendix B.2 of OSCORE

- (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
  - This will be split out as a separate draft, as agreed during IETF 118

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



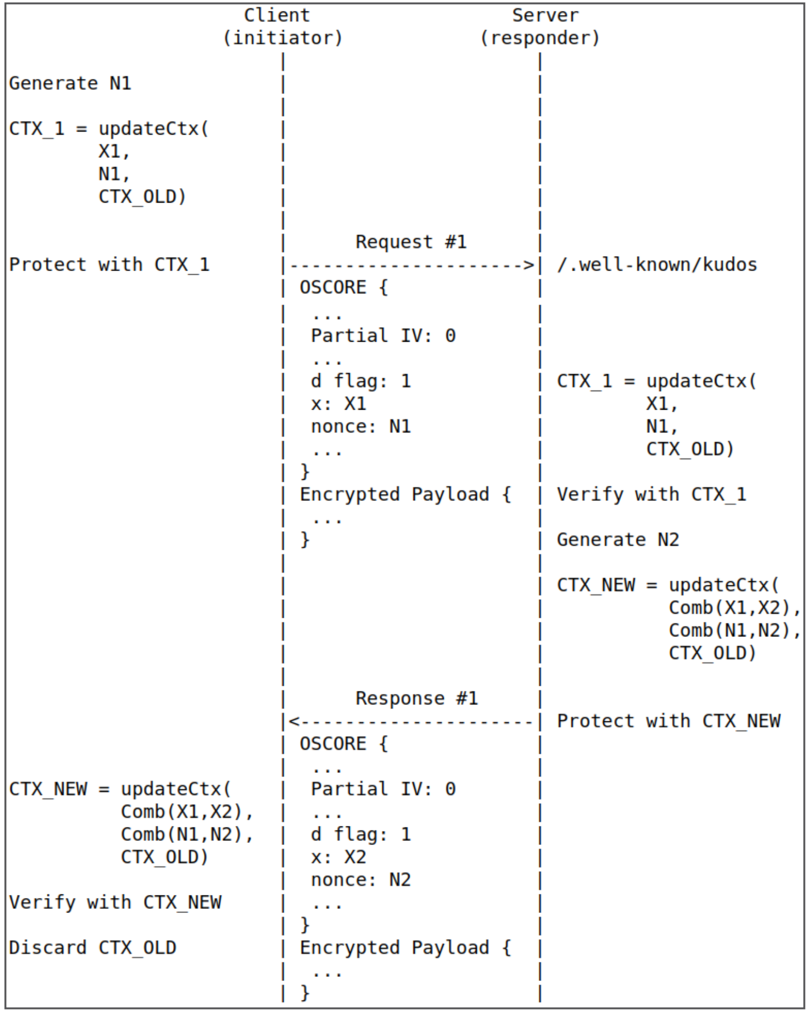
```
 0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
|1|0|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+

 <- 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 {  |
                            |  ...                 |
                            | }                    |
```

Longer nonces

› **Idea proposed by Christian Amsüss at [1] and [2]**

› **In *draft-amsuess-core-cachable-oscore***
  – A Request Hash is computed over a plain CoAP request
  – The request is turned into an OSCORE-protected Deterministic Request
  – The Request Hash is transported in the Request-Hash option of the Deterministic Request
  – Any alternative to the Request-Hash option?

› **Proposal**
  – In KUDOS, enable longer nonces than 16 bytes
  – In the same 'nonce' field of the OSCORE option;
    OR
  – In the CoAP payload, pre-pended to the OSCORE ciphertext
    › Take advantage of fragmentation with Block-wise

[1] https://github.com/core-wg/oscore-key-update/issues/95
[2] https://gitlab.com/chrysn/core-cachable-oscore/-/issues/38

Proposal and Considerations

- **Proposal**
  - In KUDOS, enable longer nonces than 16 bytes
  - In the same 'nonce' field of the OSCORE option;

    OR
  - In the CoAP payload, pre-pended to the OSCORE ciphertext
    - Take advantage of fragmentation with Block-wise

- **3 points need to be considered**
  - Encoding of the larger nonce size
  - Positioning of the nonce
    - CoAP payload vs. 'nonce' field of the OSCORE option
  - How this fits with KUDOS and Cacheable OSCORE

Encoding of the nonce size

- **Current encoding**
  - The nonce size SIZE is between 1 and 16 bytes
  - Encoded in the 'x' field of the OSCORE option, the 4-bit subfield 'm' has value (SIZE - 1)
  - After the 'x' field, the 'nonce' field is present

```
<-   1 byte   -> <---   8 bytes   --->
+-+-+-+-+-+-+-+-+--------------------+
|0|z|b|p|0|1|1|1|       nonce        |
+-+-+-+-+-+-+-+-+--------------------+
          m = 7
```

- **Proposed new encoding**
  - The nonce size is between 1 and 65805 bytes
  - Similar to the encoding of CoAP option number delta in CoAP options
  - Within the 'x' field of the OSCORE option, the 4-bit subfield 'm' specifies either:
    - A value between 0 and 12 included, as (SIZE - 1); or
    - The value 13, indicating that a 1-byte 'ext_m' field follows and has content an 8-bit unsigned integer with value (SIZE - 13 - 1); or
    - The value 14, indicating that a 2-byte 'ext_m' field follows and has content a 16-bit unsigned integer with value (SIZE - 269 - 1).
  - Within the 'x' field of the OSCORE option, the value 15 for the 4-bit subfield 'm' is reserved and cannot be used.

```
<-   1 byte   -> <-   1 byte    -> <---  32 bytes   --->
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+--------------------+
|0|z|b|p|1|1|0|1|0|0|0|1|0|0|1|0|       nonce        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+--------------------+
         m = 13     ext_m = 18
```

- **Conclusion: Doable and would work to enable very large nonces**

Positioning of the nonce

- **Where to place the nonce/hash value V?**
  - In the OSCORE option, as in current KUDOS; or
  - In the CoAP payload, prepended to the OSCORE ciphertext

- **KUDOS – V is the KUDOS nonce**
  - V must be integrity protected is in both requests and responses
  - Integrity protection is achieved by taking V as input to the key derivation
  - Conclusion: placing V in the OSCORE option and the CoAP payload would both work fine

- **Cacheable OSCORE – V is the Request Hash value**
  - V in the Deterministic Request
    - Involved in the key derivation --> Integrity protected --> Its position is irrelevant
  - V in the Response to a Deterministic Request
    - In the OSCORE option --> Integrity protected, since the external_aad includes the OSCORE option
    - In the CoAP payload --> It cannot be integrity protected
  - Conclusion: placing V in the OSCORE option works fine; the CoAP payload does not work

The OSCORE option would work correctly. Is it also convenient?

Does it pay off?

- **Need for longer nonces in KUDOS?**
  - KUDOS already supports 16 byte nonces; longer nonces should not be needed for KUDOS itself
  - Longer nonces also require a very good source of entropy on devices

- **Cacheable OSCORE – Complexity of parsing messages**
  - The Request Hash value is in the 'nonce' field of the OSCORE option
  - In both messages, the OSCORE option is extended as in KUDOS messages, but this is not a key update!
  - In order to understand that this is not about key update:
    - The server has to rely on the request 'kid' indicating the Sender ID of the Deterministic Client
    - The client has more simply to rely on the response being a reply to a Deterministic Request

- **Cacheable OSCORE – Communication overhead**
  - In responses, the Request-Hash option is treated as Class I and is usually elided (normative SHOULD)
  - The option OSCORE cannot similarly be elided from the response to a Deterministic Request
  - This results in larger responses, compared to when using the (elided) Request-Hash option

Keep the current design in both documents?

Other points related to KUDOS

- **OSCORE limits document**
  - Submitted new version –02 (rebumb)
  - Waiting for updates to *cfrg-aead-limits* and possible feedback from John Preuß Mattsson
- **Procedure for OSCORE ID update**
  - Will be split into new document as decided during IETF 118

- **More flexible KUDOS message flow**
  - Decoupling from the currently rigid request/response flow
- **KUDOS messages as regular application messages**
  - E.g., an application request can also be a KUDOS message
- **Usage of non-random nonces in KUDOS**
  - Possible, but limited to CAPABLE devices, for which random nonces remain preferred

- **Implementations**
  - Starting up implementation work in Java and C

Thank you!

Comments/questions?

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

Backup

More flexible message flow

- **An execution of KUDOS is based on a request/response message pair**
  - Forward message flow: *KUDOS request* then *KUDOS response*
  - Reverse message flow: *KUDOS response* then *KUDOS request*

- **Shall we allow for more variations of message flows?**
  - Consider a scenario using the Resource Directory where both KUDOS messages are requests

1. EP -> RD: POST /.well-known/rd - **KUDOS message 1 (request)**
2. RD -> EP: GET /.well-known/core - **KUDOS message 2 (request)**
3. EP -> RD: response with payload of /.well-known/core
4. RD -> EP: response to the POST

Credits to Christian Amsüss for this scenario

- **Other scenarios 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

KUDOS messages as regular application messages

- **Currently KUDOS messages are 'special' types of messages**
  - For instance, in the forward message flow the client sends the first KUDOS message to */.well-known/kudos,* without any payload

- **Can we instead allow the client to initiate KUDOS with a 'normal' application message?**
  - That is, the client wishes to send an actual application request to the server. Then this message can also serve as a KUDOS message.
  - Implications would be:
    - KUDOS request messages can target any resource at the server
    - In the forward message flow, the client would send the application message that it currently wants to send as a KUDOS message
    - The caveat is that the server cannot be sure the request is fresh, thus it may want to respond with 4.01. This can in turn signal to the client to re-run KUDOS.

Usage of non-random nonces

- **Currently we always refer to the nonces as random values**
  - However, in some scenarios using counters instead can make sense
  - CAPABLE devices which can persist the context over a reboot may keep a nonce counter. This also ensures that the nonces are not re-used.

- **Shall we explicitly allow usage of counters as nonces?**
  - Currently we do not explicitly forbid, nor allow it
  - Taking RFC9203 (OSCORE ACE profile) as example: *The use of a 64-bit long random number as the nonce's value is RECOMMENDED in this profile.*

- **Proposed solution**
  - Non-CAPABLE devices: MUST use random values
  - CAPABLE devices: SHOULD use random values, but may use counters if it enforces measures to ensure their freshness and accepts the privacy implications

Splitting out OSCORE IDs update

- **Method for updating the OSCORE Sender/Recipient IDs – Section 5 & Appendix A**
  - The ID update procedure can be run stand-alone or embedded in a KUDOS execution
  - Fundamentally it is a separate procedure and not related to KUDOS

- **Shall we split this out into a separate, WG document?**
  - We raised this in the past but without a strong consensus to proceed
  - Combining Section 5 and Appendix A, this content currently spans around 16 pages

- **One benefit of doing this is to focus this document specifically on KUDOS**