Key Update for OSCORE (KUDOS)

draft-hoeglund-core-oscore-key-limits-02

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF 112, CoRE WG, November 8th, 2021

Recap

- OSCORE (RFC8613) uses AEAD algorithms to provide security
  - Need to follow limits in key usage and number of failed decryptions, before rekeying
  - Excessive use of the same key can enable breaking security properties of the AEAD algorithm
  - Reference **draft-irtf-cfrg-aead-limits-03**

- (1) Study of AEAD limits and their impact on OSCORE
  - Defining appropriate limits for OSCORE, for a variety of algorithms
  - Defining counters for key usage; message processing details; steps when limits are reached
  - Taking into account John Mattsson's input at the April CoRE interim [1]

- (2) Defined a new method for rekeying OSCORE (KUDOS)
  - Loosely inspired by Appendix B.2 of OSCORE
  - Goal: renew the Master Secret and Master Salt; derive new Sender/Recipient keys from those
  - Achieves Perfect Forward Secrecy

[1] https://datatracker.ietf.org/meeting/110/materials/slides-110-saag-analysis-of-usage-limits-of-aead-algorithms-00.pdf

Key limits (1/3)

Confidentiality Advantage (CA): Probability of breaking confidentiality properties

Integrity Advantage (IA): Probability of breaking integrity properties

- Recap on AEAD limits
  - Discussed in **draft-irtf-cfrg-aead-limits-03**
  - Limits key use for encryption (q) and invalid decryptions (v)
  - This draft defines fixed values for ‘q’, ‘v’, and ‘l’ and from those calculate CA & IA probabilities
    - IA & CA probabilities must be acceptably low

- Now explicit size limit of protected data to be sent in a new OSCORE message
  - The probabilities are influenced by ‘l’, i.e., maximum message size in cipher blocks
  - Implementations should not exceed 'l', and it has to be easy to avoid doing so
  - New text: *the total size of the COSE plaintext, authentication Tag, and possible cipher padding for a message may not exceed the block size for the selected algorithm multiplied with 'l'‘*

- New table (Figure 3) showing values of ‘l’ not just in cipher blocks but actual bytes

Key limits (2/3)

- Increased value of ‘l’ (message size in blocks) for algos except AES_128_CCM_8
  - Increasing ‘l’ from 2^8 to 2^10 should maintain secure CA and IA probabilities
  - draft-irtf-cfrg-aead-limits mentions aiming for CA & IA lower than to 2^-50
    - They have added a table in that document with calculated ‘q’ and ‘v’ values

q = 2^20, v = 2^20, and l = 2^10

Algorithm name	IA probability	CA probability
AEAD_AES_128_CCM	2^-64	2^-66
AEAD_AES_128_GCM	2^-97	2^-89
AEAD_AES_256_GCM	2^-97	2^-89
AEAD_CHACHA20_POLY1305	2^-73	-

- Intent is to increase 'q', 'v' and/or 'l' further. Should we?
  - Since we are well below 2^-50 for CA & IA currently

Key limits (3/3)

- Updated table of ‘q’, ‘v’ and ‘l’ for AES_128_CCM_8
  - Added new value for ‘v’, still leaving CA and IA less than 2^-50
  - Is it ideal to aim for CA & IA close to 2^-50 as defined in the CRFG document?

'q', 'v' and 'l'	IA probability	CA probability
q=2^20, v=2^20, l=2^8	2^-44	2^-70
q=2^15, v=2^20, l=2^8	2^-44	2^-80
q=2^10, v=2^20, l=2^8	2^-44	2^-90
q=2^20, v=2^15, l=2^8	2^-49	2^-70
q=2^15, v=2^15, l=2^8	2^-49	2^-80
q=2^10, v=2^15, l=2^8	2^-49	2^-90
q=2^20, v=2^14, l=2^8	2^-50	2^-70
q=2^15, v=2^14, l=2^8	2^-50	2^-80
q=2^10, v=2^14, l=2^8	2^-50	2^-90
q=2^20, v=2^10, l=2^8	2^-54	2^-70
q=2^15, v=2^10, l=2^8	2^-54	2^-80
q=2^10, v=2^10, l=2^8	2^-54	2^-90

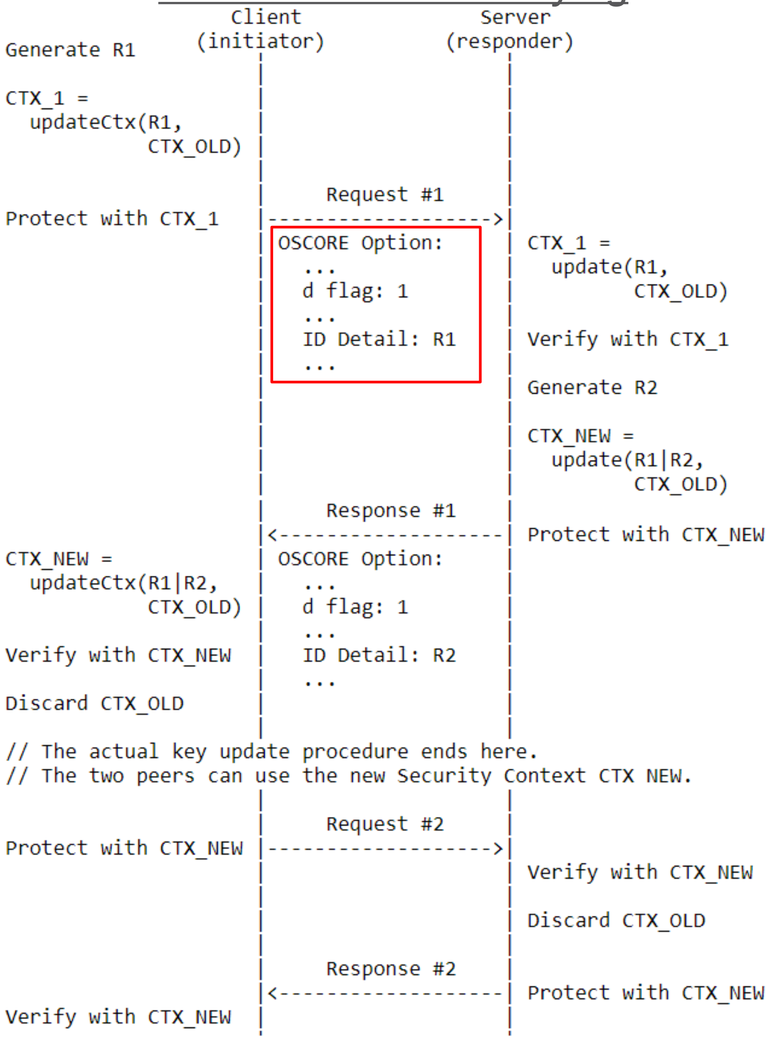
'q', 'v' and 'l'	IA probability	CA probability
q=2^20, v=2^20, l=2^6	2^-44	2^-74
q=2^15, v=2^20, l=2^6	2^-44	2^-84
q=2^10, v=2^20, l=2^6	2^-44	2^-94
q=2^20, v=2^15, l=2^6	2^-49	2^-74
q=2^15, v=2^15, l=2^6	2^-49	2^-84
q=2^10, v=2^15, l=2^6	2^-49	2^-94
q=2^20, v=2^14, l=2^6	2^-50	2^-74
q=2^15, v=2^14, l=2^6	2^-50	2^-84
q=2^10, v=2^14, l=2^6	2^-50	2^-94
q=2^20, v=2^10, l=2^6	2^-54	2^-74
q=2^15, v=2^10, l=2^6	2^-54	2^-84
q=2^10, v=2^10, l=2^6	2^-54	2^-94

Key update (1/4)

- Defined a new method for rekeying OSCORE
  - Key Update for OSCORE (KUDOS) - Named procedure
  - Client and server exchange two nonces R1 and R2
  - *UpdateCtx()* function for deriving new OSCORE Security Context using the nonces
  - Current Sec Ctx (to renew) ==> Intermediate Sec Ctx ==> **New Sec Ctx**

- Properties
  - Can be initiated by either the client or server
  - Completes in one round-trip (after that, the new Security Context can be used)
  - Only one intermediate Security Context is derived
  - The ID Context does not change
  - Robust and secure against peer rebooting
  - Compatible with prior key establishment using the EDHOC protocol

Client-initiated rekeying



Key update (2/4)

- No more R1 in the Response #1 for the **client-initiated** rekeying
  - Just like in OSCORE Appendix B.2
  - Simply not needed: Response #1 correlates to Request #1 through the CoAP Token

Before

```
                    |      Request #1      |
Protect with CTX_1  |--------------------->| CTX_1 =
                    | OSCORE Option:       |   update(R1,
                    |   ...                |          CTX_OLD)
                    |   d flag: 1          |
                    |   ...                |
                    |   ID Detail: R1      | Verify with CTX_1
                    |   ...                |
                    |                      | Generate R2
                    |                      |
                    |                      | CTX_NEW =
                    |                      |   update(R1|R2,
                    |                      |          CTX_OLD)
                    |      Response #1     |
                    |<---------------------| Protect with CTX_NEW
CTX_NEW =           | OSCORE Option:       |
  updateCtx(R1|R2,  |   ...                |
          CTX_OLD)  |   d flag: 1          |
                    |   ...                |
Verify with CTX_NEW |  ID Detail:R1|R2     |
                    |   ...                |
Discard CTX_OLD     |                      |
```

After

```
                    |      Request #1      |
Protect with CTX_1  |--------------------->| CTX_1 =
                    | OSCORE Option:       |   update(R1,
                    |   ...                |          CTX_OLD)
                    |   d flag: 1          |
                    |   ...                |
                    |   ID Detail: R1      | Verify with CTX_1
                    |   ...                |
                    |                      | Generate R2
                    |                      |
                    |                      | CTX_NEW =
                    |                      |   update(R1|R2,
                    |                      |          CTX_OLD)
                    |      Response #1     |
                    |<---------------------| Protect with CTX_NEW
CTX_NEW =           | OSCORE Option:       |
  updateCtx(R1|R2,  |   ...                |
          CTX_OLD)  |   d flag: 1          |
                    |   ...                |
Verify with CTX_NEW |   ID Detail: R2      |
                    |   ...                |
Discard CTX_OLD     |                      |
```

Key update (3/4)

- Recommendations on minimum length of R1 and R2 values
  - R1 and R1 | R2 are used as nonces
  - Motivation is based on similar considerations for Appendix B.2 in RFC8613
  - We now recommend minimum 8 bytes, is this sufficient?
  - Further text needs to be added as in Appendix B.2. e.g. mentioning the birthday paradox

- Currently MUST terminate ongoing observations after rekeying (derived CTX_NEW)
  - Possible to keep them ongoing for a price, i.e. admitting an earlier use of large Partial IVs
  - Possible solution: after a rekeying, the client considers PIV* as the highest req_piv among all the ongoing observations. Then, when the client starts the first new observation, the SSN jumps to PIV*+1, thus every observation request has a PIV greater than PIV*.
  - Drawback: Big jumps in PIV, i.e., faster consumption and larger communication overhead
  - (More complicated solutions like reserving some PIVs in a bit-map is also possible)
  - Is it worth keeping observations ongoing across a rekeying? Plan is to not keep observations

Key update (4/4)

- Added and discussed 6TiSCH as use case
  - 6TiSCH uses OSCORE Appendix B.2 to handle failure events
  - If the 6TiSCH JRC severely fails, it can use Appendix B.2 with the pledges (RECOMMENDED)
  - The new key update procedure is a good replacement, especially for 6TiSCH
  - Among its intrinsic advantages compared to Appendix B.2, it preserves the ID Context across rekeying
    - 6TiSCH uses ID Context as pledge identifier, meaning that:
    - → A key update would not change pledge identifier, which remains unchanged in the long run
    - → The JRC does not need anymore to do a remapping between new ID Context and pledge identifier
    - → ID Contexts and pledge identifiers can be used as intended at setup/deploy time

- The update to RFC8613 includes also “deprecating and replacing” its Appendix B.2
  - Ok with this?

More general updates

- Improved Table of Content structure
  - Key Limits
  - Current rekeying methods
  - New rekeying methods
    - Building blocks
    - Client-initiated procedure
    - Server initiated procedure
    - Policies
    - Discussion

- Editorial improvements
  - Terminology harmonization
  - Alignment to most recent EDHOC interfaces
  - Use of RFC8126 terminology in IANA considerations
  - Updated title to *Key Update for OSCORE (KUDOS)* - Feedback on title?

Next steps

- Address open points, including:
  - Material to save to disk to support rebooting
  - Reuse applicable considerations from OSCORE Appendix B.2
  - Update security considerations
  - Further refinement of key limits

- The document foundation and the key update protocol are stable

- Plan to implement

- WG adoption?

Thank you!

Comments/questions?

https://gitlab.com/rikard-sics/draft-hoeglund-oscore-rekeying-limits/

OSCORE Option update

- OSCORE Option: defined the use of flag bit 1 to signal presence of flag bits 8-15
- Defined flag bit 15 -- 'd' -- to indicate:
  - This is a OSCORE key update message
  - "id detail" is specified (length + value); used to transport a nonce for the key update

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

 <- 1 byte -> <----- s bytes ------> <- 1 byte -> <----- x bytes ---->
+------------+----------------------+------------+--------------------+
| s (if any) | kid context (if any) | x (if any) | id detail (if any) |
+------------+----------------------+------------+--------------------+

+------------------+
| kid (if any) ... |
+------------------+
```

Figure 3: The OSCORE option value, including 'id detail'