Key Update for OSCORE (KUDOS)

draft-ietf-core-oscore-key-update

Rikard Höglund, RISE
Marco Tiloca, RISE

CoRE WG interim meeting, December 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

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

Key update overview

- Defined a new method for rekeying OSCORE
  - Key Update for OSCORE (KUDOS)
  - Client and server exchange nonces R1 and R2
  - *UpdateCtx()* function for deriving new OSCORE Security Context using the nonces

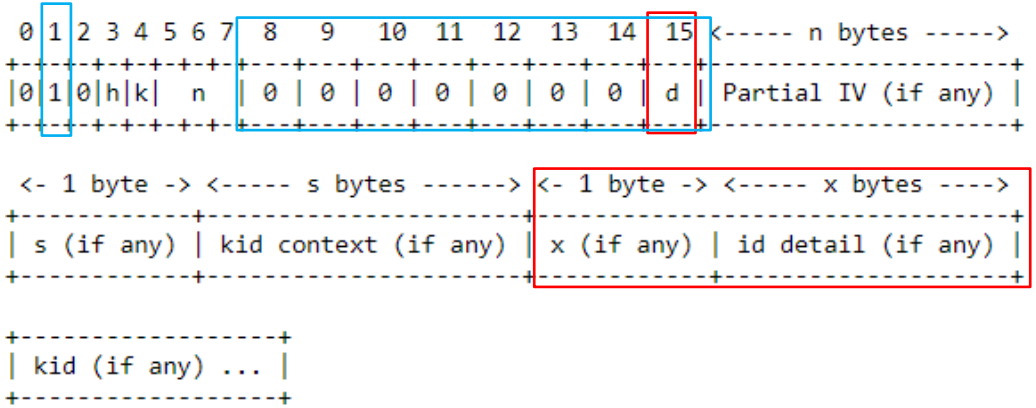
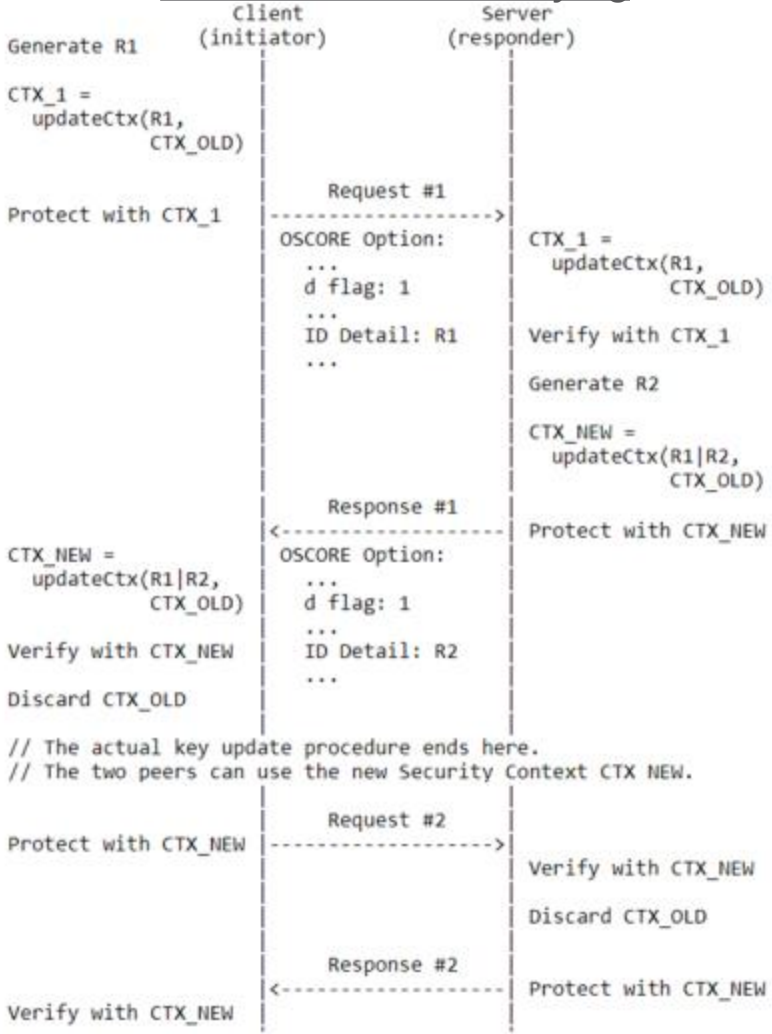


Figure 3: The OSCORE option value, including 'id detail'

Client-initiated rekeying



Open points for today

- Preservation of observations after Key Update
  - Different alternatives for preserving observations
- Key Update without Perfect Forward Secrecy (PFS)
  - New mode of KUDOS without PFS
- Renewal of Sender/Recipient IDs

Keeping observations (1/2)

- The client starts an observation Obs1 by sending a request Req1 with req_piv=X
- The two peers run KUDOS, and resets their Sender Sequence Number (SSN) to 0.
- Later on, while Obs1 is still ongoing, the client sends a new request Req2 also with req_piv=X. This is not necessarily an observation request.
- Problem: a notification sent by the server for Obs1 or a response to Req2 would both cryptographically match against Req1 and Req2.

- Current approach: terminate observations after completing KUDOS.
- Two possible alternative approaches: "long-jumping" or "skipping" SSNs.
- Either chosen approach has to apply to all observations and all key updates.

Keeping observations (2/3)

- "Long-jumping" approach
  - When wishing to send a first request after key update, the client determines the PIV* as the highest req_piv among all the ongoing observations.
  - The client updates its SSN to be (PIV* + 1)

- Pros and Cons
  - Pro: no need to take any particular action for every outgoing request.
  - Con: several SSN values become unusable and yield an early use of large-size values on the wire (in the OSCORE option of all following requests), thus increasing communication overhead.

Keeping observations (3/3)

- "Skipping" approach
  - The client holds a list containing the req_piv among the observation requests of all observations
  - Before sending any request, the client checks if its current SSN is in the list L. If this is the case, the client increments its SSN and checks the value against the list L again.
  - As soon as a value is found as not being in the list, this is used as Partial IV in the OSCORE option of the request.

- Pros and Cons
  - Pro: no "long jump" and waste of SSN values; instead, only the already taken ones are selectively skipped.
  - Pro: no additional communication overhead due to early use of large-size Partial IVs.
  - Con: more processing for each outgoing request, since the client has to check the list L.

- Which one should we take? Terminating, long-jumping or skipping? Any further?

Key Update without PFS (1/4)

- Admit alternative KUDOS mode without PFS ?
  - Raised on the CoRE mailing list in [1]
  - Benefit: allows to be stateless across rebooting, like OSCORE Appendix B.2
    - Good for constrained devices that cannot store information to persistent memory

- Extension to KUDOS, enabling the selection of a no-PFS mode
  - Define new bit 'p' in the second flag byte of the OSCORE option
  - 'p' set to 0 ==> run KUDOS in PFS mode;
  - 'p' set to 1 ==> run KUDOS in non-PFS mode

- New concepts
  - Bootstrap Master Secret & Bootstrap Master Salt
    - If provisioned they are stored on disk, and they are never changed by the device
  - Latest Master Secret and Latest Master Salt
    - From the latest derived OSCORE Security; should be stored on disk by a device capable to do so

[1] https://mailarchive.ietf.org/arch/msg/core/EL0yHxQrP2DQwHxo6ojnQedvFbY/

Key Update without PFS (2/4)

- On reboot, check if you have a (Latest Master Secret, Latest Master Salt) on disk

- If yes:
  - Use its content to derive an OSCORE context CTX_OLD
  - Run KUDOS as initiator

- If no, check if you have a (Bootstrap Master Secret, Bootstrap Master Salt) on disk
  - If yes:
    - Use its content to derive an OSCORE context CTX_OLD
    - Run KUDOS as initiator
  - If no, run something else to establish a first Security Context (e.g., EDHOC)

Key Update without PFS (3/4)

- If the 'p' bit is set to 1, KUDOS is run in no-PFS mode
  - PFS is sacrificed due to at least one peer unable to save to persistent memory
  - Before starting KUDOS, CTX_OLD is possibly modified to always ensure that:
    - Master Secret = Bootstrap Master Secret, and Master Salt = Bootstrap Master Salt.
    - ==> Every execution of KUDOS between these peers will always consider this same Secret/Salt pair
    - ==> Hence no Perfect Forward Secrecy is ensured anymore
    - But all the other properties of KUDOS remain! (i.e., compared to OSCORE Appendix B.2)

- This mode of KUDOS requires a peer to have Bootstrap Master Secret/Salt ...
  - … Which practically implies the other peer has them too

Key Update without PFS (4/4)

- If the 'p' bit is set to 0, KUDOS is run in PFS mode
  - This is equivalent to the original KUDOS procedure defined in the draft
  - The Security Context CTX_OLD is used as is, and PFS is preserved
  - Devices capable of writing to persistent memory should initiate the procedure with p set to 0

- Agreed downgrading
  - If the initiator sets 'p' to 0, the responder might not follow-up (if unable to write to disk)
    - Server responder: return a protected 5.03 error response to Request #1, with 'p' set to 1
    - Client responder: send a protected Request #2, with 'p' set to 1
    - In either case, abort KUDOS
  - Then, the initiator may retry with 'p' set to 1
    - To support the best possible common thing

- Overall good approach and way forward?

Renew Sender/Recipient IDs (1/3)

- Possibility to update peers Sender and Recipient IDs
  - Based on earlier discussions on mailing list [2][3]

- Properties
  - Each endpoint can specify its own new Recipient ID (similar to EDHOC)
  - Accepting to update the Sender/Recipient IDs is optional; explicit confirmation is needed
  - Can be embedded in a KUDOS execution or run standalone
  - Possible for both client and server to initiate this procedure
  - 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)
    - Admit exception? - Use of OSCORE Appendix B.1 to avoid nonce reuse

[2] https://mailarchive.ietf.org/arch/msg/core/GXsKO4wKdt3RTZnQZxOzRdIG9QI/
[3] https://mailarchive.ietf.org/arch/msg/core/ClwcSF0BUVxDas8BpgT0WY1yQrY/

Renew Sender/Recipient IDs (2/3)

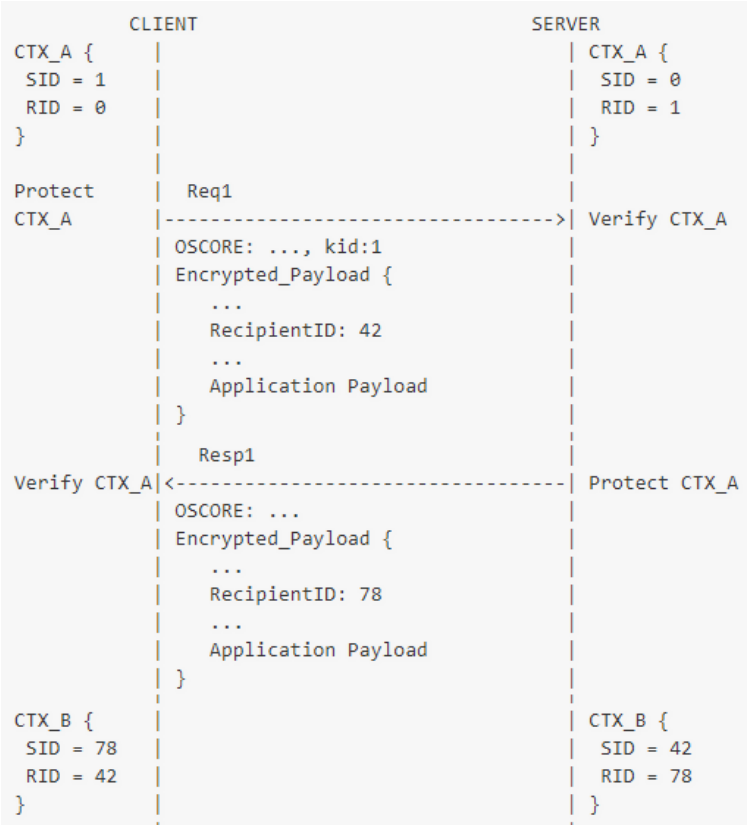
- Defined new CoAP Option to carry the desired Recipient ID
  - Proposed option number 24 (00011000)
  - The content of this option is the new Recipient ID of the message sender
  - (Of course the peer should pick and offer a free recipient ID for the used ID Context)
  - Class E option for OSCORE processing

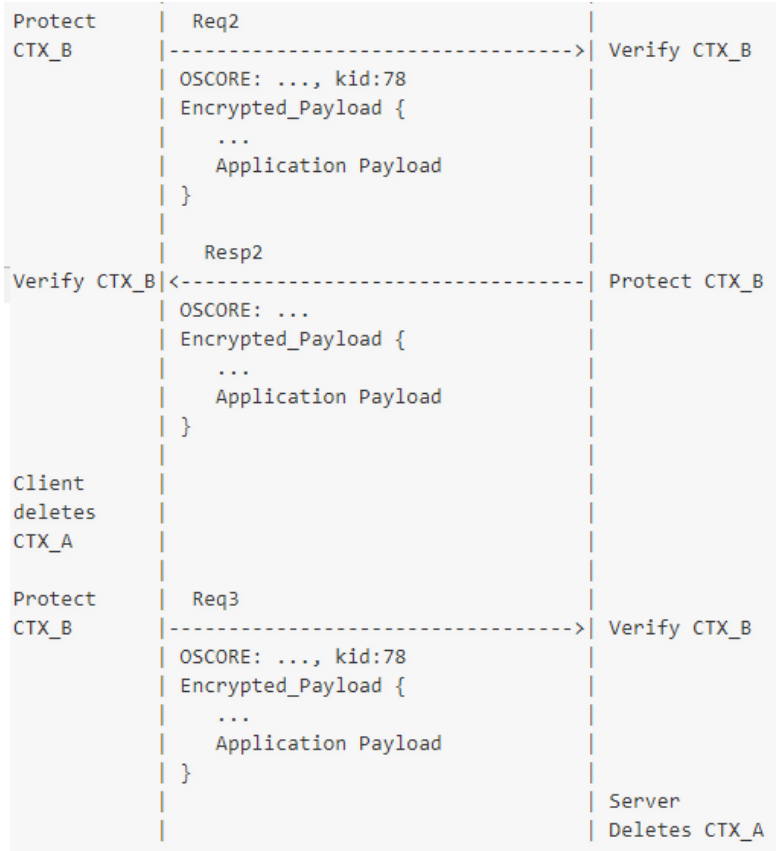
No.	C	U	N	R	Name	Format	Length	Default
TBD1					RecipientID	opaque	0-7	(none)

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

- Fine for the option to be elective? This is for having it optional to accept
- Any better way than a CoAP option?

Renew Sender/Recipient IDs (3/3)





Next steps

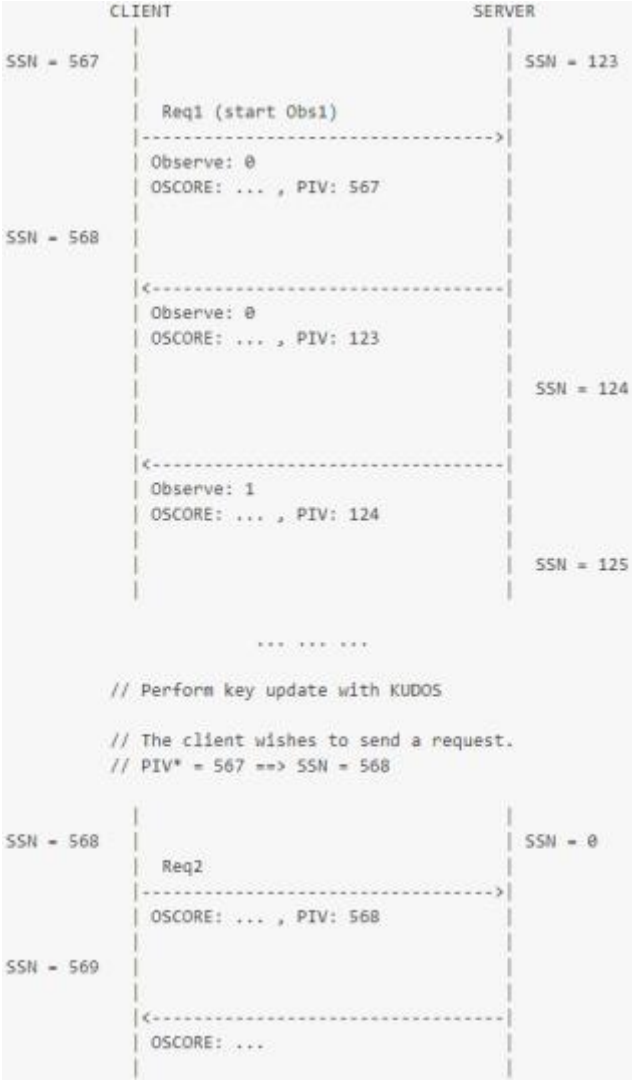
- Address open points and issues
  - Further refinement of key limits
  - Refinements to the KUDOS procedure and its extensions
    - Reuse applicable considerations from OSCORE Appendix B.2
  - Which KUDOS messages can contain actionable payload?
- Implementation

Thank you!

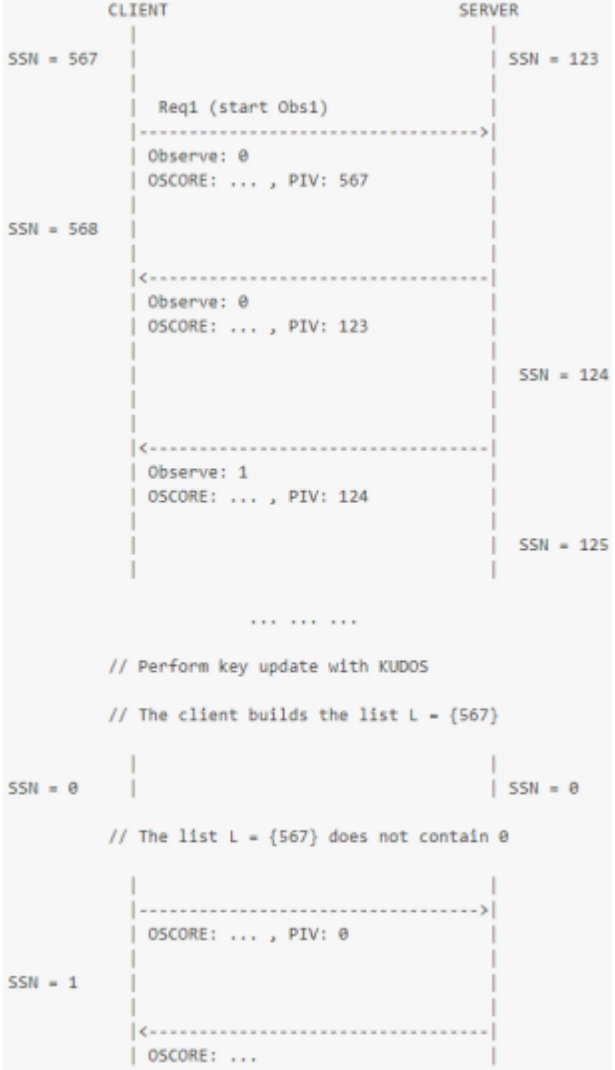
Comments/questions?

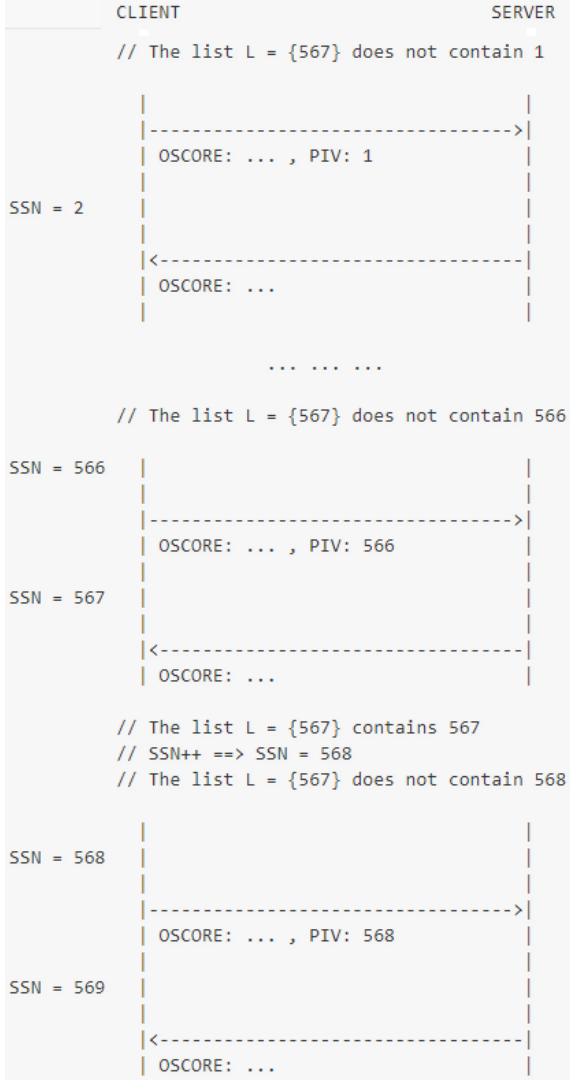
https://github.com/core-wg/oscore-key-update

“Long-Jumping”



“Skipping”


CLIENT
SERVER
SSN = 567
SSN = 123
Req1 (start Obs1)
Observe: 0
OSCORE: ... , PIV: 567
SSN = 568
Observe: 0
OSCORE: ... , PIV: 123
SSN = 124
Observe: 1
OSCORE: ... , PIV: 124
SSN = 125
... ...
// Perform key update with KUDOS
// The client builds the list L = {567}
SSN = 0
SSN = 0
// The list L = {567} does not contain 0
OSCORE: ... , PIV: 0
SSN = 1
OSCORE: ...


CLIENT
SERVER
// The list L = {567} does not contain 1
OSCORE: ... , PIV: 1
SSN = 2
OSCORE: ...
... ...
// The list L = {567} does not contain 566
SSN = 566
OSCORE: ... , PIV: 566
SSN = 567
OSCORE: ...
// The list L = {567} contains 567
// SSN++ ==> SSN = 568
// The list L = {567} does not contain 568
SSN = 568
OSCORE: ... , PIV: 568
SSN = 569
OSCORE: ...

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

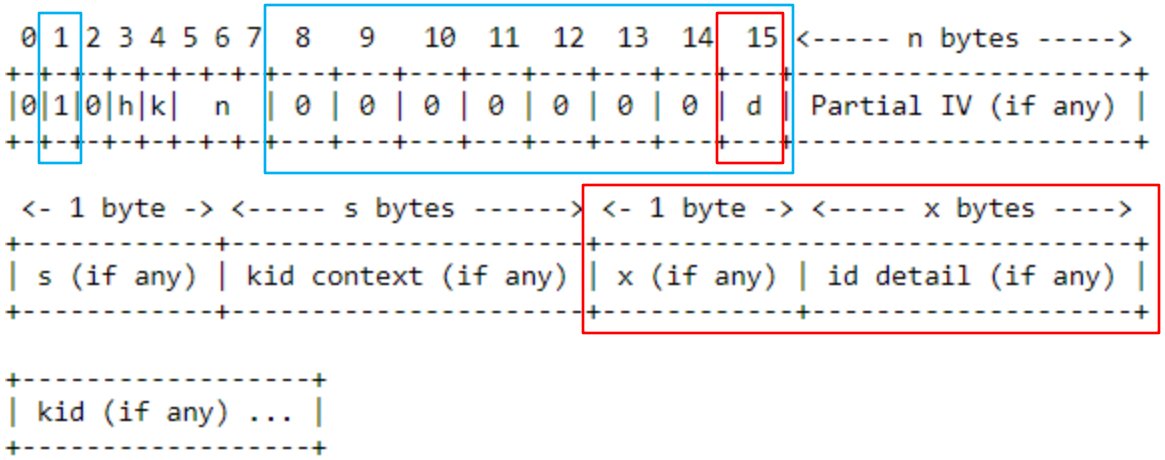


Figure 3: The OSCORE option value, including 'id detail'

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 overview

- Defined a new method for rekeying OSCORE
  - Key Update for OSCORE (KUDOS)
  - Client and server exchange nonces R1 and R2
  - *UpdateCtx()* function for deriving new OSCORE Security Context using the nonces

- 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

NEW › Mode with PFS (stateful) and without PFS (stateless)

NEW › Possibility to update Recipient/Sender IDs

Client-initiated rekeying