Content Recap

› OSCORE (RFC8613) uses AEAD algorithms to provide security
  – Need to follow limits in number of encryptions and failed decryptions, before rekeying
  – Excessive use of the same key can enable breaking security properties of the AEAD algorithm*

› (1) Defined Key Update for 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

› (2) AEAD Key Usage Limits in OSCORE
  – Defining appropriate limits for OSCORE, for a variety of algorithms
  – Defining counters for key usage; message processing details; steps when limits are reached

*See also draft-irtf-cfrg-aead-limits
Key Update Recap

› Method for rekeying OSCORE
  - Key Update for OSCORE (KUDOS)
  - Client and server exchange nonces N1 and N2
  - $\text{UpdateCtx()}$ function for deriving new OSCORE Security Context using the nonces
  - Extended OSCORE Option
    › IANA: can bits "1" and "15" be "1 (suggested)" and "15 (suggested)"? --> We do need and prefer exactly "1" and "15"
    › 'id detail' renamed to 'nonce'

![Diagram showing Key Update process]

• 'x' byte enriched with additional signaling flags
Key Update without FS (1/2)

› Alternative KUDOS mode without Forward Secrecy
  – Text moved from old Appendix to document body and improved (Section 4.4)
  – Stateless key update; needed for devices that cannot store to persistent memory

› Signaling through a new 'p' bit in the 'x' byte of the OSCORE Option
  – 'p' set to 0 ==> sender's wish to run KUDOS in FS mode (original mode)
  – 'p' set to 1 ==> sender's wish to run KUDOS in no-FS mode
  – If \( p = 0 \) in both KUDOS messages ==> use the FS mode
  – If \( p = 1 \) in both KUDOS messages ==> use the no-FS mode

› When using the FS-mode
  – The latest Security Context CTX_OLD is used as is, and FS is preserved
  – Devices capable of writing to persistent memory should initiate the procedure with 'p' set to 0
When using the no-FS mode
- FS is sacrificed due to at least one peer unable to write to persistent memory
- Before starting KUDOS, the CTX_OLD is modified to ensure that:
  - Master Secret = Bootstrap Master Secret, and Master Salt = Bootstrap Master Salt.
  - Every execution of KUDOS between these peers will consider this same Secret/Salt pair

Agreed downgrading to no-FS mode
- 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, with 'p' set to 1
  - Client responder: send a protected request, with 'p' set to 1
  - In either case, abort KUDOS
- Then, the initiator may retry with 'p' set to 1

Section 4.4.1 has an extensive discussion on handling keying material and reboot
Content moved from old appendix to document body and extended (Section 4.5)

Problem recap:
1. The client starts an observation Obs1 by sending a request Req1 with req_piv X
2. The two peers run KUDOS, and reset their Sender Sequence Number (SSN) to 0.
3. 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.
4. A notification sent by the server for Obs1 and a response to Req2 would both cryptographically match against Req1 and Req2 by OSCORE external_aad.

Solution: "Long-jumping" of OSCORE Sender Sequence Numbers (SSNs)
   – After completing KUDOS, a peer determines PIV* as the highest req_piv among all the ongoing observations where it is client.
   – The peer updates its SSN to be (PIV* + 1)
Preserving Observations (2/2)

› Signaling through a new 'b' bit in the 'x' byte of the OSCORE Option
  – 'p' set to 0 ==> sender's wish to **cancel** all common observations beyond key update
  – 'p' set to 1 ==> sender's wish to **keep** all common observations beyond key update

› Simple "all-or-nothing" approach
  – If p = 1 in both KUDOS messages, peers keep their observations, otherwise they are cancelled

› A client ever wishing to preserve its observations:
  – MUST NOT silently forget them
  – Has to use cancellation requests (Observe:1)
    › Observations are purged only if receiving a confirmation from the server

› Even though key update is not of interest at the present moment ...
  – A peer might run KUDOS to quickly cancel the ongoing observations with the other peer!

Comments? Questions?
Update of Sender/Recipient IDs

› Method for updating peers' OSCORE Sender/Recipient IDs
  – Based on earlier discussions on the mailing list [1][2] and on [3]
  – This procedure can be embedded in a KUDOS execution or run standalone
  – This procedure can be initiated by a client or by a server
  – Content moved from old appendix to document body and improved (Section 5)

› Properties
  – The 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 (e.g., KUDOS must be run first)
  – Offered Recipient ID must be not used yet under (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 5.1.1 and 5.1.2

[1] https://mailarchive.ietf.org/arch/msg/core/GXsKO4wKdl3RTZnQZxOzRdIg9QI/
[2] https://mailarchive.ietf.org/arch/msg/core/ClwcSF0BUVxDas8BpgT0WY1yQrY/
Further Updates (1/2)

- Defined signaling bits present in the 'x' byte
  - Four least significant bits encode the 'nonce' length in bytes minus 1 (length indication for 'nonce')
  - Fifth least significant bit is the "No Forward Secrecy" 'p' bit (controls using FS or no-FS mode)
  - Sixth least significant bit is the "Preserve Observations" 'b' bit (controls preserving observations or not)
  - The two most significant bits are reserved for now

- Redesigned the updateCtx() function
  - updateCtx(N, CTX_IN) --> updateCtx(X, N, CTX_IN)
    - 'x' bytes also as input --> Covered by key derivation --> Integrity protected
  - Still two available methods
    - METHOD 1: use EDHOC-KeyUpdate, if EDHOC was used to derive the first Ctx
    - METHOD 2: a simple plain use of HKDF-Expand(), if EDHOC was not used
  - When using METHOD 1
    - Aligned with the new EDHOC-KeyUpdate(), with input a CBOR byte string
    - Defined rules about when replacing the old EDHOC keys PRK_out and PRK_exporter
Further Updates (2/2)

- X1 and X2: raw value of 'x' in the OSCORE Option of 1st/2nd KUDOS message
- N1 and N2: raw value of 'nonce' in the OSCORE Option of 1st/2nd KUDOS message

Before `updateCtx()`, blends the Xs and Ns into X and N
  - Message 1: X = X1 and N = N1
  - Message 2: X = bstr .cbor X1 | bstr .cbor X2 , N = bstr .cbor N1 | bstr .cbor N2

Invoke `updateCtx(X, N, ...)`, which blends X and N into a single CBOR byte string X_N
  - X_cbor = bstr .cbor X
  - N_cbor = bstr .cbor N
  - X_N = bstr .cbor (X_cbor | N_cbor)
  - X_N is used as input to EDHOC-KeyUpdate() or to HKDF-Expand()
Open points & Next steps

› Continue addressing the issues on the Github repo [1]

› Proposal: reorganize/split updateCtx() into
  – A preamble to compute X_N and then invoke ...
  – … METHOD 1, based on EDHOC-KeyUpdate() or ...
  – … METHOD 2, based on HKDF-based

› Proposal: agreed fallback to METHOD 2
  – E.g., an EDHOC session is not valid anymore
  – New signaling bit in the 'x' byte to use when running KUDOS; same as when agreeing on no-FS

› Implementation built on existing implementation of OSCORE in Java based on Californium

› Comments and reviews are welcome!

[1] https://github.com/core-wg/oscore-key-update/issues
Thank you!

Comments/questions?

https://github.com/core-wg/oscore-key-update
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
  - "nonce" 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
+----------+----------+----------+----------+----------+----------+----------+----------+
|0|1|0|h|k| n | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d |
+----------+----------+----------+----------+----------+----------+----------+----------+
```

```
<- 1 byte -> <- s bytes -> <- 1 byte -> <- m + 1 bytes ->
+----------+----------+----------+----------+----------+----------+----------+----------+
| s (if any) | kid context (if any) | x (if any) | nonce (if any) |
+----------+----------+----------+----------+----------+----------+----------+----------+
```

'b' = Preserve Observations
'p' = No Forward Secrecy
'm' = Length of nonce -1
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 $l = 2^{10}$

<table>
<thead>
<tr>
<th>Algorithm name</th>
<th>IA probability</th>
<th>CA probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEAD_AES_128_CCM</td>
<td>$2^{-64}$</td>
<td>$2^{-66}$</td>
</tr>
<tr>
<td>AEAD_AES_128_GCM</td>
<td>$2^{-97}$</td>
<td>$2^{-89}$</td>
</tr>
<tr>
<td>AEAD_AES_256_GCM</td>
<td>$2^{-97}$</td>
<td>$2^{-89}$</td>
</tr>
<tr>
<td>AEAD_CHACHA20_POLY1305</td>
<td>$2^{-73}$</td>
<td>-</td>
</tr>
</tbody>
</table>

› 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?

<table>
<thead>
<tr>
<th>'q', 'v' and 'l'</th>
<th>IA probability</th>
<th>CA probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>q=2^20, v=2^20, l=2^8</td>
<td>2^-44</td>
<td>2^-70</td>
</tr>
<tr>
<td>q=2^15, v=2^20, l=2^8</td>
<td>2^-44</td>
<td>2^-80</td>
</tr>
<tr>
<td>q=2^10, v=2^20, l=2^8</td>
<td>2^-44</td>
<td>2^-90</td>
</tr>
<tr>
<td>q=2^20, v=2^15, l=2^8</td>
<td>2^-49</td>
<td>2^-70</td>
</tr>
<tr>
<td>q=2^15, v=2^15, l=2^8</td>
<td>2^-49</td>
<td>2^-80</td>
</tr>
<tr>
<td>q=2^10, v=2^15, l=2^8</td>
<td>2^-49</td>
<td>2^-90</td>
</tr>
<tr>
<td>q=2^20, v=2^14, l=2^8</td>
<td>2^-50</td>
<td>2^-70</td>
</tr>
<tr>
<td>q=2^15, v=2^14, l=2^8</td>
<td>2^-50</td>
<td>2^-80</td>
</tr>
<tr>
<td>q=2^10, v=2^14, l=2^8</td>
<td>2^-50</td>
<td>2^-90</td>
</tr>
<tr>
<td>q=2^20, v=2^10, l=2^8</td>
<td>2^-54</td>
<td>2^-70</td>
</tr>
<tr>
<td>q=2^15, v=2^10, l=2^8</td>
<td>2^-54</td>
<td>2^-80</td>
</tr>
<tr>
<td>q=2^10, v=2^10, l=2^8</td>
<td>2^-54</td>
<td>2^-90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>'q', 'v' and 'l'</th>
<th>IA probability</th>
<th>CA probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>q=2^20, v=2^20, l=2^6</td>
<td>2^-44</td>
<td>2^-74</td>
</tr>
<tr>
<td>q=2^15, v=2^20, l=2^6</td>
<td>2^-44</td>
<td>2^-84</td>
</tr>
<tr>
<td>q=2^10, v=2^20, l=2^6</td>
<td>2^-44</td>
<td>2^-94</td>
</tr>
<tr>
<td>q=2^20, v=2^15, l=2^6</td>
<td>2^-49</td>
<td>2^-74</td>
</tr>
<tr>
<td>q=2^15, v=2^15, l=2^6</td>
<td>2^-49</td>
<td>2^-84</td>
</tr>
<tr>
<td>q=2^10, v=2^15, l=2^6</td>
<td>2^-49</td>
<td>2^-94</td>
</tr>
<tr>
<td>q=2^20, v=2^14, l=2^6</td>
<td>2^-50</td>
<td>2^-74</td>
</tr>
<tr>
<td>q=2^15, v=2^14, l=2^6</td>
<td>2^-50</td>
<td>2^-84</td>
</tr>
<tr>
<td>q=2^10, v=2^14, l=2^6</td>
<td>2^-50</td>
<td>2^-94</td>
</tr>
<tr>
<td>q=2^20, v=2^10, l=2^6</td>
<td>2^-54</td>
<td>2^-74</td>
</tr>
<tr>
<td>q=2^15, v=2^10, l=2^6</td>
<td>2^-54</td>
<td>2^-84</td>
</tr>
<tr>
<td>q=2^10, v=2^10, l=2^6</td>
<td>2^-54</td>
<td>2^-94</td>
</tr>
</tbody>
</table>
Key update overview

- Defined a new method for rekeying OSCORE
  - Key Update for OSCORE (KUDOS)
  - Client and server exchange nonces N1 and N2
  - `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
  - Mode with FS (stateful) and without FS (stateless)
  - Possibility to preserve ongoing observations
  - Possibility to update Recipient/Sender IDs
"Long-Jumping"