Profiling EDHOC for CoAP and OSCORE

draft-ietf-core-oscore-edhoc-03

Francesca Palombini, Ericsson
Marco Tiloca, RISE
Rikard Höglund, RISE
Stefan Hristozov, Fraunhofer AISEC
Göran Selander, Ericsson

CoRE WG interim meeting, April 27th, 2022
Recap

EDHOC: lightweight authenticated key exchange
- Developed in the LAKE Working Group
- Main use: establish an OSCORE Security Context
- Normally, two round-trips before using OSCORE

Scope of this document
- Focus on EDHOC for OSCORE, transported over CoAP
- Optimized key establishment workflow (main item)
  - Single request with EDHOC option, combining final EDHOC message_3 and first OSCORE-protected application request
- Conversion of OSCORE IDs to EDHOC IDs
- OSCORE-specific processing of EDHOC messages
- Extension/consistency of EDHOC application templates
- Web linking for discovery of EDHOC resources and their application templates (through target attributes)
Update since IETF 112

› EDHOC+OSCORE request – Client processing
  – Not more than 1 “outstanding interaction” (see Section 4.7 of RFC 7252) such that
    › They are EDHOC+OSCORE requests for the same server
    › They are related to the same EDHOC session identified by C_R
  – → A client “impatient” to obtain a response does not flood the server

› EDHOC+OSCORE request – Server processing
  – Once finished processing EDHOC message_3 …
  – … rebuild the OSCORE-protected application request and …
  – … remove the EDHOC option (now explicitly stated)
    › Not needed from then on
    › Analogous to removing the OSCORE option after decryption
    › Ensures correct processing when both inner and outer blockwise are used

Comments/objections?
Update since IETF 112

› Selection of EDHOC connection identifiers, on client and server
   – More precise guidelines, as selection of OSCORE Recipient IDs
   – Consistent with uniqueness requirements from RFC 8613
     › SHOULD be an available Recipient ID overall
     › MUST be available among the Security Contexts with zero-length ID-Context

› Editorial fixes/improvements
   – “Perfect forward secrecy” → “Forward secrecy”
   – Improved all example figures
   – Highlighted that C_R is NOT in the payload of the EDHOC+OSCORE request
     › The server recomputes it from the ‘kid’ of the OSCORE option
Update since IETF 112

› When can the EDHOC+OSCORE request get too big?
  – Use of large ID_CRED_I in EDHOC, e.g., as a certificate chain
  – Use of a large EAD_3 for External Authorization Data

› Use of Blockwise for the EDHOC+OSCORE request – Client side
  – OSCORE protection of each inner block as usual
  – If the protected block is not the first one (i.e., Block1.NUM ≠ 0)
    › The client MUST NOT add the EDHOC option, but sends the protected request as is
    › → Only the first inner block conveys EDHOC data
  – If the protected block is the first one (i.e., Block1.NUM = 0) and …
    › … (EDHOC message_3 | OSCORE ciphertext) > MAX_UNFRAGMENTED_SIZE … then
    › … abort and possibly switch to the original vanilla EDHOC workflow
    › No further inner blockwise can happen once the EDHOC+OSCORE request is assembled
Update since IETF 112

› Use of Blockwise for the EDHOC+OSCORE request – Server side
  – If the EDHOC+OSCORE request has Block options, then outer blockwise is used
  – First, the server collects all the outer blocks of the (first inner block of the) request
  – Then, the server can process the EDHOC data and complete EDHOC as usual

› The new text on blockwise brought back an old question
  – In case blockwise is used for the EDHOC+OSCORE request …
  – … when does the optimized workflow stop being convenient to use?
Optimized workflow and blockwise

Definitions

- A: size of application payload
- B: size of EDHOC message
- LIMIT: maximum amount of transmittable bytes before using blockwise, e.g.:
  - UDP maximum datagram size, i.e., 64 KiB
  - IPv6 MTU, i.e., 1280 bytes
- OVERHEAD: overall overhead from different layers (including OSCORE processing)
- LIMIT* = (LIMIT – OVERHEAD): practical limit for the application to consider

Sending the EDHOC+OSCORE request is going to work fine if

- In case inner blockwise is not used,  
  \[(A \leq \text{LIMIT}^*) \land \land (B \leq \text{LIMIT}^*) \land ((A + B) \leq \text{LIMIT}^*)\]  
  OR
- In case inner blockwise is used,  
  \[(B \leq \text{LIMIT}^*) \land \land ((\text{BLOCK}\_\text{SIZE} + B) \leq \text{LIMIT}^*)\]

  Only the application payload can be split into blocks
Optimized workflow and blockwise

› **Practical guidelines for using the EDHOC+OSCORE request**
  – If \( B > \text{LIMIT}^* \), the EDHOC+OSCORE request cannot be used
  – If \((A > \text{LIMIT}^*) \ || \ ((A + B) > \text{LIMIT}^*)\), it is necessary to use inner blockwise
    › BLOCK_SIZE has to be chosen such that \((\text{BLOCK} \_\text{SIZE} + B) \leq \text{LIMIT}^*\)
    › Inner blockwise might be used even if not strictly due to exceeding \text{LIMIT}^*

› **If inner blockwise is used**
  – The round-trips to complete EDHOC and exchange OSCORE-protected data are
    › Optimized workflow w/ blockwise \( \Rightarrow RT' = 1 + \text{ceil}(A / \text{BLOCK} \_\text{SIZE}) \)
    › Original workflow w/ blockwise \( \Rightarrow RT'' = 1 + \text{ceil}(A / \text{BLOCK} \_\text{SIZE}) + \text{ceil}(B / \text{BLOCK} \_\text{SIZE}) \)
  – \( RT' < RT'' \Rightarrow \) The optimized workflow is always more convenient

› **Is it always overall worth it?**
Optimized workflow and blockwise

- **Practical guidelines for using the EDHOC+OSCORE request**
  - If \( B > \text{LIMIT}^* \), the EDHOC+OSCORE request cannot be used
  - If \( (A > \text{LIMIT}^*) \) \( \text{or} \) \( ((A + B) > \text{LIMIT}^*) \), it is necessary to use inner blockwise
    - BLOCK_SIZE has to be chosen such that \( ((\text{BLOCK}\_\text{SIZE} + B) \leq \text{LIMIT}^*) \)
    - Inner blockwise might be used even if not strictly due to exceeding \( \text{LIMIT}^* \)

- **Corner case: \((A \leq \text{LIMIT}^*) \) \&\& \( ((A + B) > \text{LIMIT}^*) \)**
  - Inner blockwise is necessary for the optimized workflow but not for the original workflow!
  - The round-trips to complete EDHOC and exchange OSCORE-protected data are
    - Optimized workflow with blockwise \( \rightarrow RT' = 1 + \text{ceil}(A / \text{BLOCK}\_\text{SIZE}) \)
    - Original workflow without blockwise \( \rightarrow RT'' = 3 \)
  - \( RT' \leq RT'' \rightarrow \) The optimized workflow can be **not worse** in terms of RTT
    - It depends on the used BLOCK_SIZE, ideally resulting in only 2 blocks, hence in 2 RTTs
    - It still requires using the EDHOC+OSCORE request and inner blockwise …
Optimized workflow and blockwise

› **Main takeaway**
  – When inner blockwise is used, the optimized workflow yields less RTTs

› **Corner case:** \((A \leq \text{LIMIT}^*) \&\& ((A + B) > \text{LIMIT}^*)\)
  – The optimized workflow requires inner blockwise but …
  – … the original workflow does not require inner blockwise
  – The optimized workflow can still be not worse, but it is overall less convenient
    › No advantage in terms of round-trips anyway, thus …
    › No reason for client and server to perform extra processing steps

› **Proposal: in the corner case above, the client**
  – SHOULD NOT use the optimized workflow
  – SHOULD revert to the original workflow
Next steps

› Text on using the optimized workflow or not when using blockwise
  – The analytical model of the previous slides is a starting point

› Revise and simplify text related to OSCORE/EDHOC identifiers
  – Due to expected changes for EDHOC identifiers (to be intrinsically byte strings only)

› More next steps
  – Use of “URI compression” option from Christian once it is available
    › https://datatracker.ietf.org/meeting/interim-2021-core-05/materials/slides-interim-2021-core-05-sessa-core-option-for-well-known-resources-00.pdf
  – Security considerations

› We have running code built for Eclipse Californium (Java)
  – Aligned to EDHOC v-12; updates expected based on next EDHOC revision
    › https://github.com/rikard-sics/californium/tree/edhoc

› Comments are reviews are welcome!
Thank you!
Comments/questions?

https://github.com/core-wg/oscore-edhoc/
EDHOC + OSCORE request