AEAD key usage limits in OSCORE

draft-hoeoglund-core-oscore-key-limits

Rikard Höglund, RISE
Marco Tiloca, RISE

CoRE WG interim meeting, April 28th, 2021
Problem Recap (1/2)

› OSCORE uses AEAD algorithms to provide security
  – Confidentiality and Integrity

› Forgery attack against AEAD algorithms
  – Adversary may break the security properties of the AEAD algorithm
  – Reference draft-irtf-cfrg-aead-limits-01

› AEAD limits and their impact on OSCORE
  – Defining appropriate limits for OSCORE
  – How the forgery attack and the limits affect OSCORE
  – Necessary steps to take during message processing (e.g. counting)
  – What actions to take if the limits are exceeded (e.g. rekeying)
Problem Recap (2/2)

› What you need to count
  – ‘q’: the number of messages protected with specific key, i.e. the number of times the key has been used to encrypt data
  – ‘v’: the number of forgery attempts that have been made against a specific key, i.e. the amount of failed decryptions for a key

› Relevant parameters for OSCORE, added to the OSCORE Security Context
  – Counting number of times a Sender Key has been used for encryption (‘count_q’)
  – Counting number of times a Recipient Key has been used for failed decryption (‘count_v’)
  – Both of these have associated limits ‘limit_q’ and ‘limit_v’

› If the limits are exceeded the context must be rekeyed
  – The draft also offers an overview of methods for rekeying OSCORE
Updates since IETF 110 (1/2)

› Table with ‘q’ and ‘v’ limits for further algorithms
  – These are based on the formulas in the CFRG document

<table>
<thead>
<tr>
<th>Algorithm name</th>
<th>Limit for 'q'</th>
<th>Limit for 'v'</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEAD_AES_128_CCM_8</td>
<td>8388608 (2^23)</td>
<td>112 (2^6.8)</td>
</tr>
<tr>
<td>AEAD_AES_128_CCM</td>
<td>8388608 (2^23)</td>
<td>15337958 (2^23.9)</td>
</tr>
<tr>
<td>AEAD_AES_128_GCM</td>
<td>23703419 (2^24)</td>
<td>1.1518e+18 (2^60)</td>
</tr>
<tr>
<td>AEAD_AES_256_GCM</td>
<td>23703419 (2^24)</td>
<td>1.1518e+18 (2^60)</td>
</tr>
<tr>
<td>AEAD_CHACHA20_POLY1305</td>
<td>68719476736 (2^36)</td>
<td>-</td>
</tr>
</tbody>
</table>

› Extended section about methods for OSCORE rekeying
  – Also added bootstrapping towards a LWM2M Bootstrap Server as an alternative
  – That can provide a client with an updated Security Context (if the material on the Bootstrap Server was updated)
  – Both the LWM2M Client and the LWM2M Server can initiate bootstrapping
Updates since IETF 110 (2/2)

› State that messages detected as replays do not affect ‘count_v’
  – As these are replays they should not be counted as failed decryptions/forgery attempts

› 'exp' timestamp for OSCORE Security Context expiration
  – Added this parameter to the Security Context
  – Integer value similar to a Unix timestamp
  – When this specific time is reached a peer MUST stop using this Security Context to process any incoming or outgoing messages

› General editorial improvements
Open Points (1/2)

› Default lifetime of a Security Context
  – ‘exp' has to be set when installing a Security Context (now + lifetime)
  – A default lifetime should be defined (if not provided otherwise)
  – Lifetimes and ‘exp’ on the peers do not have to match

› Periodic saving of ‘count_q’ and ‘count_v’ by constrained devices
  – Allow safely continuing to use a Security Context after reboot
  – Will reduce number of writes to nonvolatile memory
  – Similar to solution outlined in OSCORE Appendix B.1 for storing SSN
  – Considerations on storing rates vs rekeying rates
    › If 'count_v' is saved with a too large rate, it will jump forward a lot on reboot
    – Documenting this procedure – Just as B.1 but applied to these counters?
Open Points (2/2)

› Further explore optimizations to track ‘count’q’
  – (SSN+X), with X the outgoing messages without Partial IV
  – Rely only on SSN, sacrificing accuracy and accepting more frequent rekeyings

› Can the limits be defined in a more general location like the COSE alg registry?
  – If the limits are general per algorithm they could be placed there

› How do we adapt the limits to be OSCORE specific
  – Possibly considering different probabilities p_q and p_v
  – What authoritative and appropriate reference to use to produce those?
  – Synchronizing with the work John Mattsson is doing on this
Thank you!

Comments/questions?

Backup Slides
Optimization for ‘count_q’ (1/2)

› Pro: No need to keep an explicit ‘count_q’
› Con: Pessimistic overestimation; rekeying earlier than needed

› At any point in time, an endpoint has made at most \( ENC = (SSN + SSN^*) \) encryptions, where:
  – SSN is its own Sender Sequence Number.
  – SSN* is the other endpoint's Sender Sequence Number. That is, SSN* is an overestimation of the responses without Partial IV that this endpoint has sent
Optimization for ‘count_q’ (2/2)

› Before performing an encryption, an endpoint stops and invalidates the Security Context if \((SSN + X) > \text{‘limit_q’}\), where \(X\) is determined as follows:

› If this endpoint is producing an outgoing response, \(X\) is the Partial IV in the request it is responding to

› If this endpoint is producing an outgoing request, \(X\) is the highest Partial IV value marked as received in its Replay Window, or \((\text{REPLAY\_WINDOW\_SIZE} - 1)\) if it has received no messages yet from the other endpoint
   – That is, \(X\) is the highest Partial IV seen from the other point, i.e. its highest seen Sender Sequence Number