AEAD Key Usage Limits in OSCORE
Key Update for OSCORE
draft-hoeoglund-core-oscore-key-limits-01

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF 111, CoRE WG, July 28th, 2021
OSCORE (RFC8613) uses AEAD algorithms to provide security
  – Confidentiality and Integrity

Need to follow limits in key usage and failed decryptions, before rekeying
  – Otherwise, it is possible to break the security properties of the AEAD algorithm
  – Reference draft-irtf-cfrg-aead-limits-03

(1) AEAD limits and their impact on OSCORE
  – Defining appropriate limits for OSCORE
  – Originally starting from the same assumptions in TLS
  – Revisited based on John Mattsson's input at the April CoRE interim
Draft Overview (2/2)

› (2) Updates to OSCORE
  – Counters in the Security Context: key encryption use (q) and invalid decryptions (v)
  – Necessary steps to take during message processing (counting)
  – Update the keys when the limits are exceeded (rekeying)

› (3) Defined a new method for rekeying OSCORE (new)
  – Loosely inspired by Appendix B.2 of OSCORE
  – Goal: renew the Master Secret and Master Salt; derive new keys from those
  – Achieves Perfect Forward Secrecy
Key Limits (1/2)

- Selected fixed values for ‘q’, ‘v’, and ‘l’
  - \( q = 2^{20}, v = 2^{20} \) and \( l = 2^8 \) // ‘l’ is the max message length in cipher blocks
  - Based on earlier discussions and John Mattsson’s presentation

- Table with ‘IA’ and ‘CA’ probabilities based on those values
  - These are based on the formulas in the CFRG document

<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^{-68} )</td>
<td>( 2^{-70} )</td>
</tr>
<tr>
<td>AEAD_AES_128_GCM</td>
<td>( 2^{-99} )</td>
<td>( 2^{-89} )</td>
</tr>
<tr>
<td>AEAD_AES_256_GCM</td>
<td>( 2^{-99} )</td>
<td>( 2^{-89} )</td>
</tr>
<tr>
<td>AEAD_CHACHA20_POLY1305</td>
<td>( 2^{-75} )</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 1: Probabilities for algorithms based on chosen q, v and l values.

Integrity Advantage (IA):
Probability of breaking integrity properties

Confidentiality Advantage (CA):
Probability of breaking confidentiality properties
Specific look at AEAD_AES_128_CCM_8
- Due to short Tag length the limits can be most problematic here

Table with ‘IA’ and ‘CA’ probabilities for various values of ‘q’, ‘v’ and ‘l’

From the considered values the best triple is \((q = 2^{20}, v = 2^{10}, l = 2^{8})\)
- The question is if an IA of \(2^{54}\) is good enough?
OSCORE Key Update method (1/3)

- Defined a new method for rekeying OSCORE
  - Client and server exchange two nonces R1 and R2
  - \textit{UpdateCtx()} function for deriving new OSCORE Security Context using the nonces
  - Current Sec Ctx (to renew) ==> Intermediate Sec Ctx ==> \textbf{New Sec Ctx}

- Properties
  - Only one intermediate Security Context is derived
  - The ID Context does not change
  - Can be initiated by either the client or server
  - It is robust and secure against a peer rebooting
  - The procedure completes in one round-trip (after that, the new context can be used)
  - Compatible with possible prior key establishment through the EDHOC protocol
OSCORE Key Update method (2/3)

- Key update messages are OSCORE-protected and self-evident
- 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

![Diagram of OSCORE Key Update method](image)
**OSCORE Key Update method (3/3)**

### Client (initiator)

**Generate R1**

<table>
<thead>
<tr>
<th>CTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**CTX 1** = updateCtx(R1, CTX OLD)

**Protect with CTX 1**

Request #1

**OSCORE Option:**

- d flag: 1
- ID Detail: R1

**Generate R2**

<table>
<thead>
<tr>
<th>CTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
</tr>
</tbody>
</table>

**CTX NEW** = updateCtx(R1|R2, CTX OLD)

**Discard CTX OLD**

### Server (responder)

**Update CTX**

```plaintext
updateCtx( N, CTX_IN ) {
    CTX_OUT  // The new Security Context
    MSECRET_NEW  // The new Master Secret
    MSALT_NEW  // The new Master Salt

    if <the original Security Context was established through EDHOC> {
        EDHOC-KeyUpdate( N )
        // This results in updating the key PRK_4x3m of the EDHOC session,
        // i.e., PRK_4x3m = Extract( N, PRK_4x3m )

        MSECRET_NEW = EDHOC-Exporter( "OSCORE Master Secret", key_length )
        - EDHOC-KDF(PRK_4x3m, TH_4, "OSCORE Master Secret", key_length )

        MSALT_NEW = EDHOC-Exporter( "OSCORE Master Salt", salt_length )
        - EDHOC-KDF( PRK_4x3m, TH_4, "OSCORE Master Salt", salt_length )
    }
    else {
        Master Secret Length = < Size of CTX_IN.MasterSecret in bytes >
        MSECRET_NEW = HKDF-Expand-Label(CTX_IN.MasterSecret, Label, N, Master Secret Length)
        - HKDF-Expand(CTX_IN.MasterSecret, HkdfLabel, Master Secret Length)

        MSALT_NEW = N;
    }
    < Derive CTX_OUT using MSECRET_NEW and MSALT_NEW, together with other parameters from CTX_IN >
    Return CTX_OUT;
}
```
Summary and Next Steps

› Twofold update to OSCORE
  – Tracking and reacting to defined key limits, to preserve security of the AEAD cipher
  – New efficient key update procedure (rekeying) with Perfect Forward Secrecy
    † Initially planned as a separate draft
    † Preference at the April CoRE interim to have it in this document

› Take and adopt feedback on the new key limits and especially for CCM_8

› Main next steps are tracked as Gitlab issues in [1]

› Need for reviews, on both key limits and key update

Thank you!

Comments/questions?