Identifier Update for OSCORE

draft-ietf-core-oscore-id-update-05

Rikard Höglund, RISE
Marco Tiloca, RISE

IETF CoRE WG meeting – IETF 124 – November 7th, 2025

Recap

- **Method for updating peers' OSCORE Sender/Recipient IDs**
  - This procedure can be initiated by a client or by a server
  - Peers start with an original OSCORE Security Context, CTX_A
    - ...and use the new IDs for deriving a new OSCORE Security Context CTX_B
- **Properties**
  - The message sender indicates its new wished Recipient ID, the other peer acks it
  - Both peers have to opt-in and agree in order for the IDs to be updated
  - Must not be done immediately following a reboot if run standalone (e.g., KUDOS must be run first)
  - Offered Recipient ID must not be used yet under the same (Master Secret, Master Salt, ID Context)
  - Received Recipient ID must not be used yet as own Sender ID under the same triple
  - **Overall goal**: Mitigate privacy issues due to message correlation and tracking of OSCORE peers

Design in line with KUDOS (1/2)

No.	C	U	N	R	Name	Format	Length	Default
TBD24					Recipient-ID	opaque	any	(none)

Table 1: The Recipient-ID Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

No.	C	U	N	R	Name	Format	Length	Default
TBD32					Recipient-ID-Ack	opaque	any	(none)

Table 2: The Recipient-ID-Ack Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

- **The procedure starts when either peer**
  - Sends a message including the Recipient-ID Option, or
  - Receives such a message from the other peer
- **During the procedure**
- **Sending a first message**
  - The first messages sent after the procedure has started must include the Recipient-ID Option, if this peer hasn't offered its Recipient ID already
- **Acknowledgment**
  - If a peer has received a valid message from the other peer including the Recipient-ID Option, it must include the Recipient-ID-Ack Option in subsequent messages (with value the Recipient ID received from the other peer)
- **Sending Subsequent Messages**
  - A peer must send a message with the Recipient ID Option regularly, specifically when the local timer REPEAT_TIMER expires

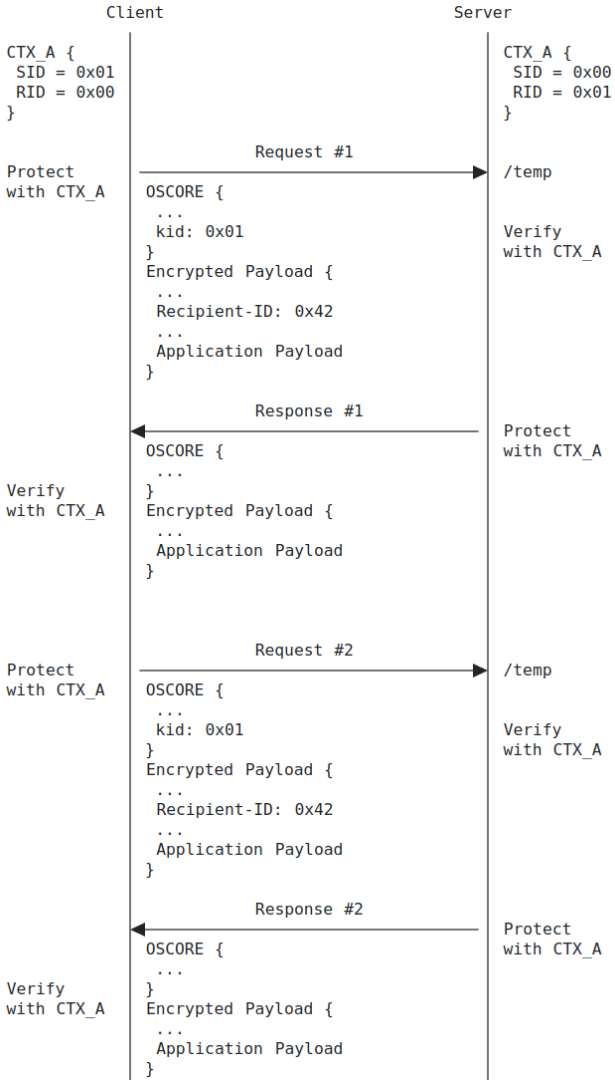
Design in line with KUDOS (2/2)

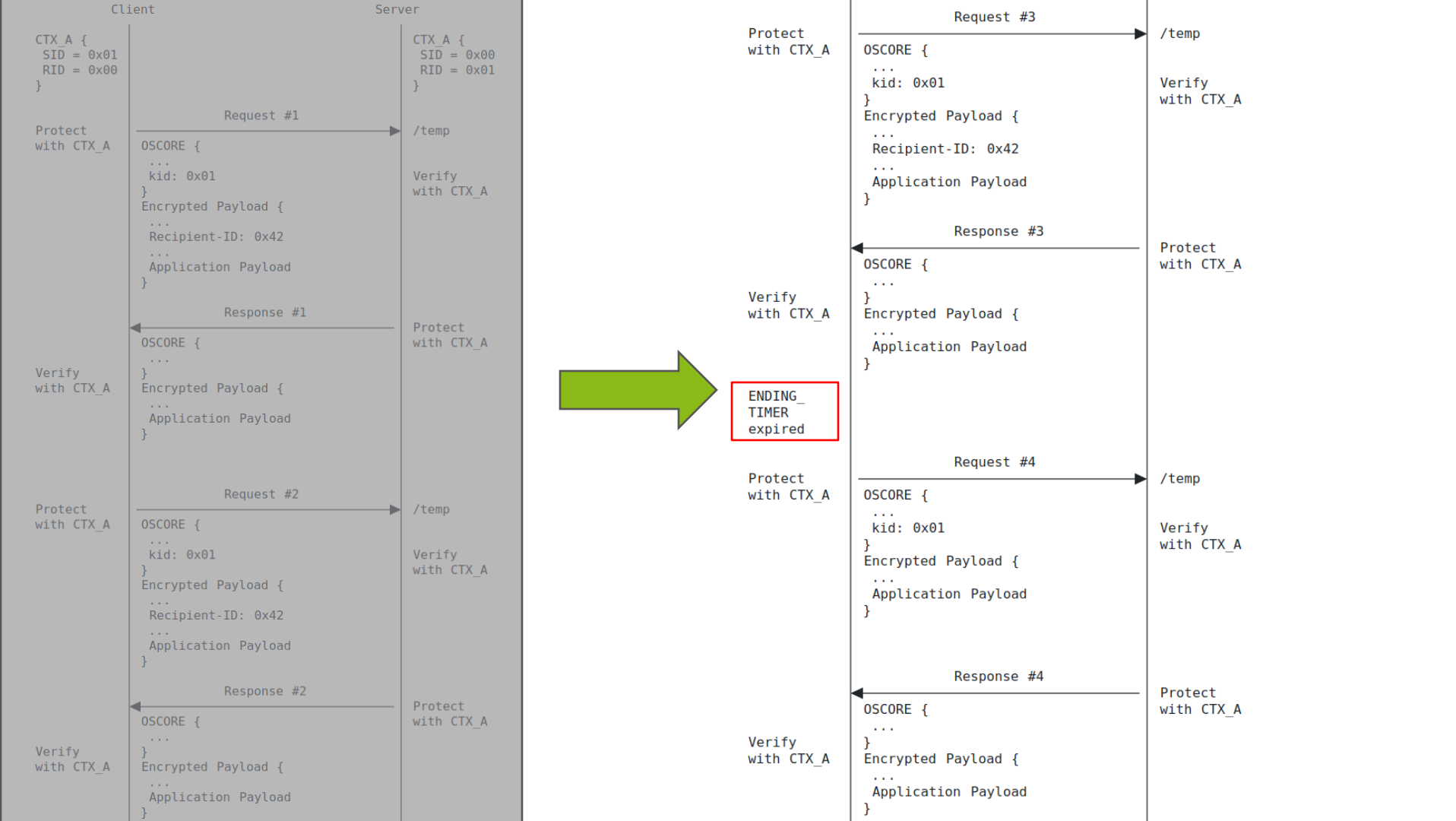
- **Procedure Completion**
  - **Success**
    - If a peer has received and successfully verified at least three message from the other peer containing the Recipient-ID-Ack Option
    - Now safe to delete CTX_A (does not mean that CTX_A has to be deleted at this point)
    - CTX_B is considered valid and can be used
      - The peers can start using it after a network migration
  - **Failure**
    - An ENDING_TIMER, is maintained and started when the procedure starts
    - If the ENDING_TIMER expires, the procedure times out without confirmation, and fails
    - The offered Recipient ID must be discarded and added to the list of IDs to prevent reuse

Changes for v-05

- **Editorial improvements**

- **Add additional message flow examples, including failure case**
  - OSCORE ID Update Procedure Initiated with a Response Message
    - Successful execution of the procedure, initiated by a CoAP response
  - Failure of the OSCORE ID Update Procedure Initiated with a Request Message
    - The client repeatedly tries sending requests to the client including the Recipient-ID option, but does not receive acknowledgments in the form of responses containing the Response-ID-Ack option from the server
    - ...thus the client eventually reaches the expiration of its ENDING_TIMER, aborts the OSCORE ID update procedure, and proceeds to continue communication with normal OSCORE messages


Client
Server
CTX_A {
SID = 0x01
RID = 0x00
}
CTX_A {
SID = 0x00
RID = 0x01
}
Request #1
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Verify
with CTX_A
Response #1
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Application Payload
}
Request #2
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Verify
with CTX_A
Response #2
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Application Payload
}


Client
Server
CTX_A {
SID = 0x01
RID = 0x00
}
CTX_A {
SID = 0x00
RID = 0x01
}
Request #1
/temp
Protect
with CTX_A
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Verify
with CTX_A
Response #1
Protect
with CTX_A
OSCORE {
...
}
Encrypted Payload {
...
Application Payload
}
Verify
with CTX_A
Request #2
/temp
Protect
with CTX_A
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Verify
with CTX_A
Response #2
Protect
with CTX_A
OSCORE {
...
}
Encrypted Payload {
...
Application Payload
}
Verify
with CTX_A
ENDING_
TIMER
expired
Request #3
/temp
Protect
with CTX_A
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Verify
with CTX_A
Response #3
Protect
with CTX_A
OSCORE {
...
}
Encrypted Payload {
...
Application Payload
}
Verify
with CTX_A
Request #4
/temp
Protect
with CTX_A
OSCORE {
...
kid: 0x01
}
Encrypted Payload {
...
Application Payload
}
Verify
with CTX_A
Response #4
Protect
with CTX_A
OSCORE {
...
}
Encrypted Payload {
...
Application Payload
}
Verify
with CTX_A

Summary and next steps

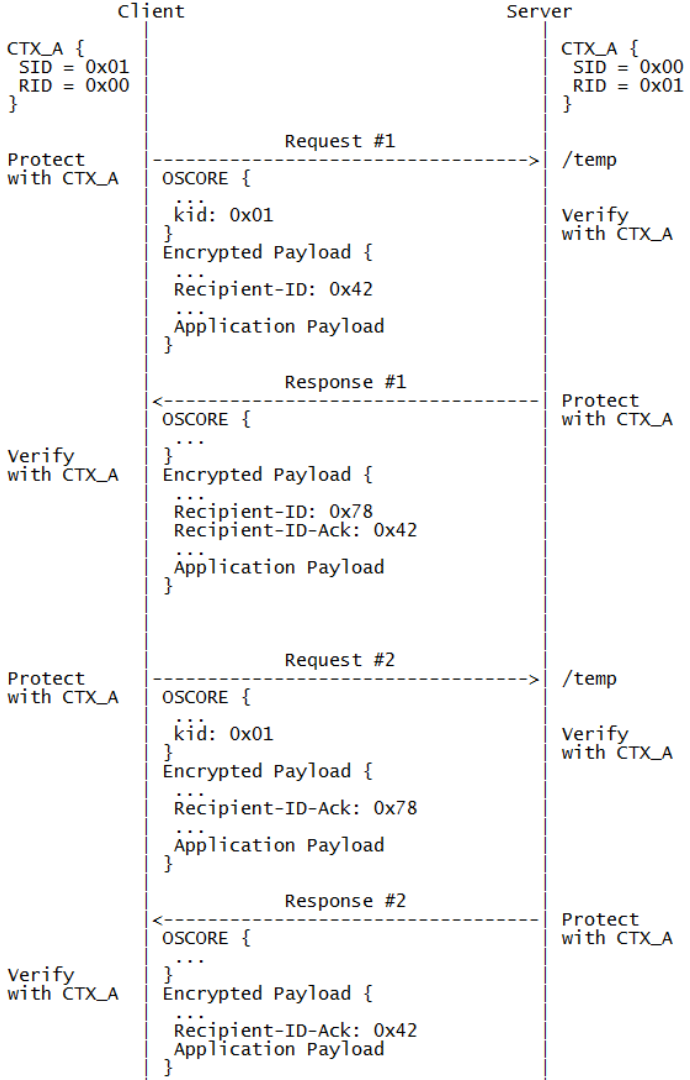
- **Consider message counting as alternative to the timers**
  - i.e., alternative to ENDING_TIMER and REPEAT_TIMER
- **Examples of running ID update integrated with KUDOS, considering the change to the KUDOS design**
  - Add message flow examples
  - Textual description of how this should work
- **Implement the OSCORE ID Update procedure**
  - Starting from our Java OSCORE implementation
- **Comments and reviews are welcome!**

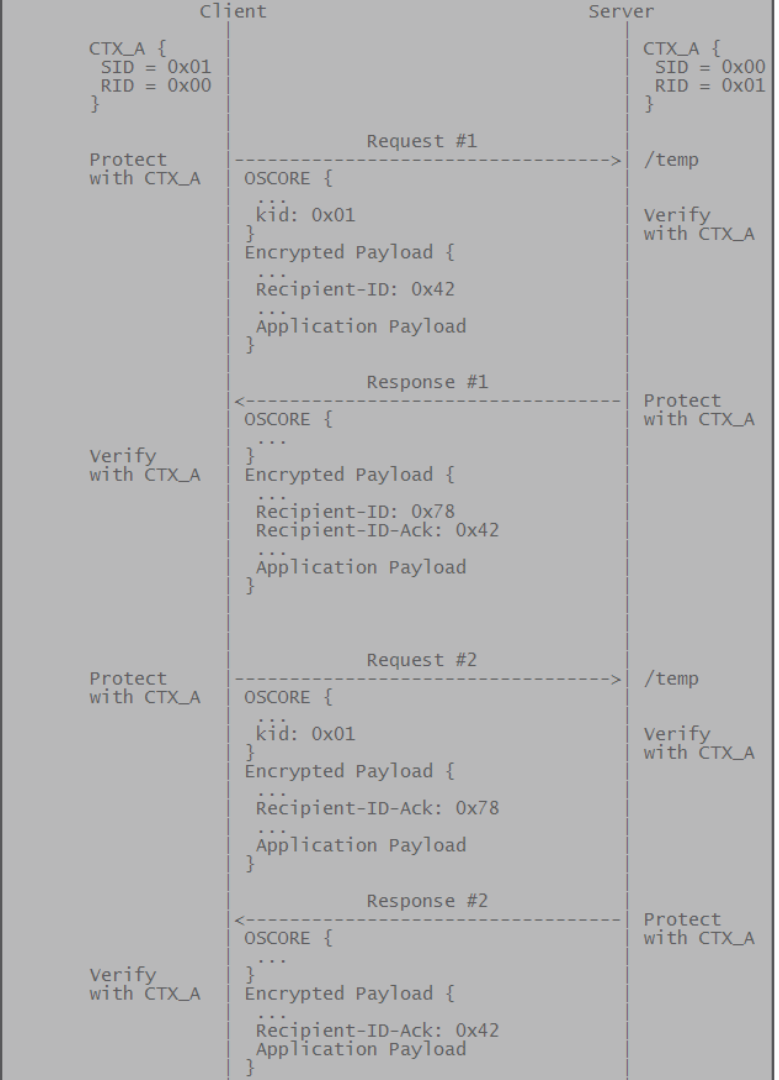
Thank you!

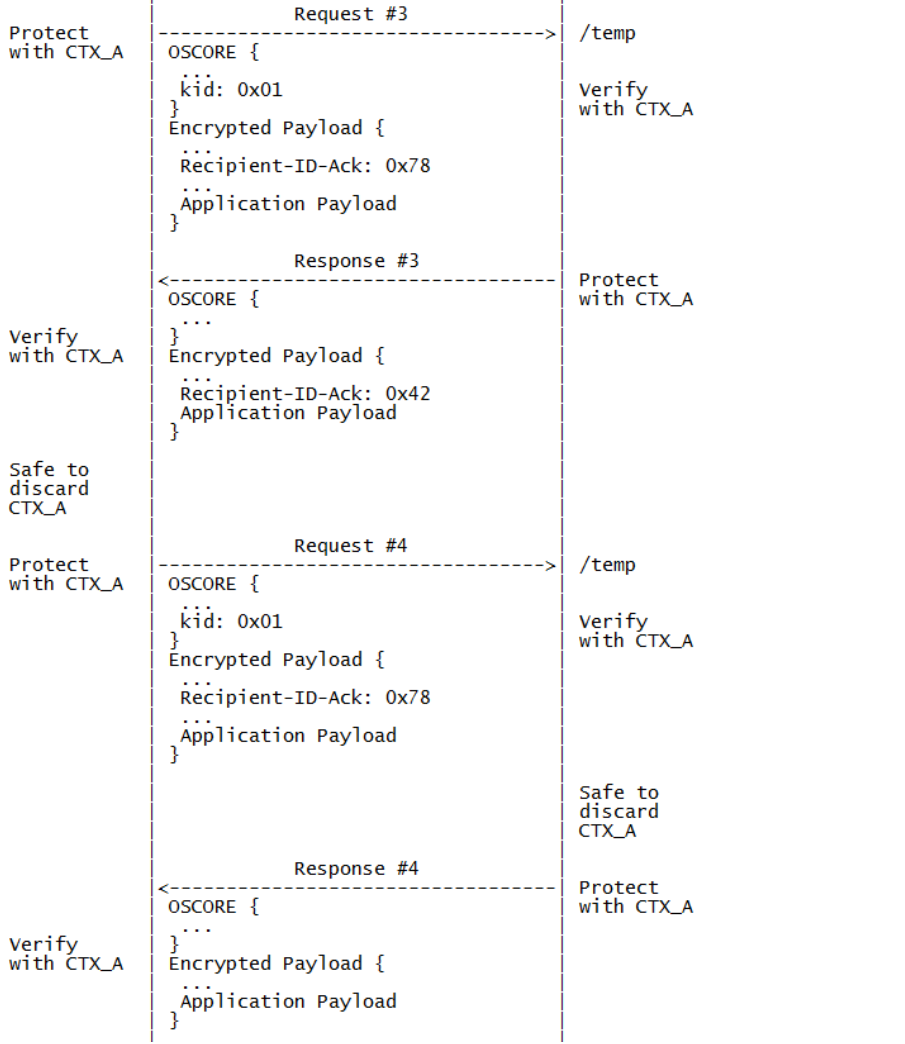
Comments/questions?

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

Backup


Client
Server
CTX_A {
SID = 0x01
RID = 0x00
}
CTX_A {
SID = 0x00
RID = 0x01
}
Request #1
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Response #1
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID: 0x78
Recipient-ID-Ack: 0x42
...
Application Payload
}
Request #2
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x78
...
Application Payload
}
Response #2
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x42
Application Payload
}


Client
Server
CTX_A {
SID = 0x01
RID = 0x00
}
CTX_A {
SID = 0x00
RID = 0x01
}
Request #1
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID: 0x42
...
Application Payload
}
Response #1
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID: 0x78
Recipient-ID-Ack: 0x42
...
Application Payload
}
Request #2
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x78
...
Application Payload
}
Response #2
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x42
Application Payload
}


Request #3
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x78
...
Application Payload
}
Response #3
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x42
Application Payload
}
Safe to
discard
CTX_A
Request #4
Protect
with CTX_A
/temp
OSCORE {
...
kid: 0x01
}
Verify
with CTX_A
Encrypted Payload {
...
Recipient-ID-Ack: 0x78
...
Application Payload
}
Safe to
discard
CTX_A
Response #4
Protect
with CTX_A
OSCORE {
...
}
Verify
with CTX_A
Encrypted Payload {
...
Application Payload
}