Outline

— Changes from -04 $\rightarrow$ -05
— Error messages
— Selected open issues
Main Changes -04 → -05

- Comments from implementers:
  - Optional “subject name” in CRED_x for COSE_Key
  - Reference cose-x509 for transport of certificate
  - Default values for length of OSCORE secret/salt

- Rename and modify key update function
  - EDHOC-KeyUpdate( nonce ):
    PRK_4x3m = Extract( nonce, PRK_4x3m )

- Clarifications of error messages

- Updated security considerations
  - Target security level, key confirmation, crypto algs

- Updated test vectors (including changes in -04)
  - Updated text on applicability statement
Error Messages
Error Messages

• Classes of Error Messages (#74)

— Current error message format

```python
error = (
    ? C_x : bstr_identifier,
    DIAG_MSG : tstr,
    ? SUITES_R : [ supported : 2* suite ] / suite,
)
```

— DIAG_MSG
  — mandatory (may be empty)
  — human-readable diagnostic message in English
— Intended for software engineers during debugging
— SHOULD be provided to the calling application
— SHOULD be logged

— What diagnostic messages needs to be standardized, if any?

Discussion at IETF 109
Example grouping TLS 1.3 error alerts

1. `handshake_failure`, `insufficient_security` → Parameter negotiation error
2. `decode_error`, `illegal_parameter`, `unexpected_message` → Message (field) error
3. `bad_certificate`, `unsupported_certificate`, `certificate_revoked`, `certificate_expired`, `certificate_unknown`, `unknown_ca`, `bad_certificate_status_response` → Credential/context error
4. `decrypt_error` → Decrypt error
5. `missing_extension`, `unsupported_extension` → Auxiliary Data error
6. `access_denied` → Access denied
7. `internal_error` → Internal error

`bad_record_mac`, `protocol_version`, `record_overflow`, `inappropriate_fallback`, `unrecognized_name`, `unknown_psk_identity`, `no_application_protocol`, `certificate_required` → N/A
Candidate EDHOC error classes

1. Selected cipher suite not supported

2. Message (field) error (syntax error or incorrect protocol field of expected message)

3. Credential/context error (error related to establish certificate/raw public key context for processing message, including certificate expired, revoked, unsupported, corrupt, unknown CA, or other issue in processing the credential)

4. Decrypt error (a cryptographic operation failed, including being unable to correctly verify a signature or a MAC in the protocol message)

5. Auxiliary data error (unsupported or missing auxiliary data)

6. Access denied (credential valid but peer not allowed access)

7. Internal error (error not related to the protocol or the peer)
Examples: Mapping processing to error codes

5.2.3. Responder Processing of Message 1
The Responder SHALL process message_1 as follows:

• Decode message_1 \(\rightarrow 2\)
• Verify that the selected cipher suite is supported and that no prior cipher suite in SUITES_1 is supported. \(\rightarrow 1\)
• Pass AD_1 to the security application. \(\rightarrow 5\)

5.3.3. Initiator Processing of Message 2
The Initiator SHALL process message_2 as follows:

• Decode message_2 \(\rightarrow 2\)
• Retrieve the protocol state using the connection identifier C_I and/or other external information such as the CoAP Token and the 5-tuple. \(\rightarrow 3\)
• Decrypt CIPHERTEXT_2 \(\rightarrow 4\)
• Verify that the identity of the Responder is an allowed identity for this connection \(\rightarrow 6\)
• Verify Signature_or_MAC_2 using the algorithm in the selected cipher suite. \(\rightarrow 4\)
• Pass AD_2 to the security application. \(\rightarrow 5\)
5.4.3. Initiator Processing of Message 3

- Decode message_3 → 1
- Retrieve the protocol state using the connection identifier C_R and/or other external information such as the CoAP Token and the 5-tuple. → 3
- Decrypt and verify the outer COSE_Encrypt0 → 4
- Verify that the identity of the Initiator is an allowed identity for this connection → 6
- Verify Signature_or_MAC_3 using the algorithm in the selected cipher suite. → 4
- Pass AD_3 to the security application. → 5

Examples: Mapping processing to error codes
Comments from IOTOPS

— Error code encoded as unsigned integer value
  — symbolic label like in (D)TLS, linked to an integer value, to make code and specs better readable.
— Reserve one of the status codes for success, so that the status reporting can also use the standard value in case of no error.
— To avoid complexity, allow implementations to use a high-level code with not too much detail.
  — For those implementations that can afford to add more complex/detailed diagnostics, use sub-codes.
  — second-level etc. error codes could be standardized also later on, on a need basis.
— Too much detail in the error reporting reveals information that an attacker can use to do a next, more targeted attack.
— Error message for “failure to begin” and “intermediate progress”. E.g. “join proxy not found”.
— Create a registry to which things can be added applying and extending existing RFC.
— As few as possible different ones: what’s the difference in actionable next steps.
  — If there is no difference then diagnostics could be the same.
— Best diagnostic requires operational experience
— “0x7411 Only my admin can help, ticket #345687”

https://mailarchive.ietf.org/arch/msg/iotops/KcWcK19kACOYnimRqFNJDBu_MAk/
Candidate EDHOC error classes (annotated)

1. **Selected cipher suite not supported** → only this error “actionable” by EDHOC engine

2. **Message field error** (syntax error or incorrect protocol field of expected message) → this error may not be possible to send e.g. if connection id cannot be retrieved

3. **Credential/context error** (error related to establish certificate/raw public key context for processing message, including certificate expired, revoked, unsupported, corrupt, unknown CA, or other issue in processing the credential) → sufficiently specific?

4. **Decrypt error** (a cryptographic operation failed, including being unable to correctly verify a signature or a MAC in the protocol message)

5. **Auxiliary data error** (unsupported or missing auxiliary data)

6. **Access denied** (credential valid but peer not allowed access)

7. **Internal error** (error not related to the protocol or the peer) → "try again" is potentially actionable, but could alternatively be handled by transport
Discussion

— What to specify? What to mandate?
— Actionable, for whom?
  — EDHOC engine or administrator?
— Complexity of protocol vs. granularity of diagnostics
— Diagnostic information should not simplify an attack
— Use of transport error message and diagnostics
— Group “non-actionable” into one class, e.g. “Unspecified error”
— Allow application defined error message content

— Proposal:
  — Error message content dependent on code
  — Make an IANA register
  — Mandate only two error codes:
    — “Unspecified error”: error = (0, tstr)
    — “Selected ciphersuite not supported”: error = (1, SUITES_R)
  — Define also “Application dependent”: error = (TBD, any)

OLD
error = (?
  C_x : bstr_identifier,
  DIAG_MSG : tstr,
  ? SUITES_R : [ supported : 2*
    suite ] / suite,
)

NEW
error = (?
  C_x : bstr_identifier,
  ERR_CODE : int,
  ERR_INFO : any,
)

Initial content of IANA register
0 / tstr / Unspecified
1 / SUITES_R / Selected ciphersuite not supported
Selected Open Issues
Applicability statement

- Applicability statement can depend on conditions during EDHOC execution (#80)
- Optional message_4 for key confirmation (#18)

- The applicability statement (Appendix C) lists parameters to be agreed between Initiator and Responder for a particular use of EDHOC
  - How to detect an EDHOC message, method, transport correlation, which optional message fields are present, type of CRED_x, type of ID_CRED_x, use of message_4, etc.

- Further clarifications are needed:
  - One endpoint may support different applicability statements
  - The EDHOC engine must be able to associate a particular protocol execution to an applicability statement
  - Some parameters may be determined from transport, others may depend on protocol message field. For example:
    - METHOD_CORR may be associated to CoAP resource, or transport parameter
    - Auxiliary Data may determine type of CRED_x
Message format and processing

- Add guidelines for distinguishing received messages (#39)
- Change message_1 format (#61)

- What is needed to distinguish messages depends on the expected processing
- High level processing needs to be clarified, e.g.
  - Applicability statement → which optional message fields are present → connection id/transport correlation → security context → expected message fields

Target setting for disambiguation:
- No need for a priori restriction of C_R/C_I
  - due to disambiguation between messages
  - even if endpoint is Initiator and Responder in multiple connections
- No need to parse entire CBOR sequence to retrieve context.
- Should work for message_1/2/3/4 and error message.
Disambiguation of messages

— Current version allows for disambiguation but can be simplified and made more robust
— Proposal to add initial byte to message_1:

message_1 ? null, int, ...
message_2 ? int / bstr, bstr, ...
message_3 ? int / bstr, bstr, ...
message_4 ? int / bstr, bstr, ...

error ? int / bstr, int (tstr in -05), ...

collection
identifier

— Comments/objections?
AD in message_4

— v-05:
  
  message_4 = (  
    data_4,  
    MAC_4 : bstr,  
  )  
  
  data_4 = (  
    ? C_I : bstr_identifier,  
  )  

  Compute COSE_Encrypt0 ... :  
  • protected = h''  
  • external_aad = << TH_4 >>  
  • plaintext = h''

— Key for the MAC derived from the Exporter, using label “EDHOC_message_4_Key”.

— Request to include auxiliary data in message_4
  — Support for implementations expecting response to AD_3 in requests (#18):

  message_4 = (  
    data_4,  
    CIPHERTEXT_4 : bstr,  
  )  
  
  data_4 = (  
    ? C_I : bstr_identifier,  
  )  

  Compute COSE_Encrypt0 ... :  
  • protected = h''  
  • external_aad = << TH_4 >>  
  • plaintext = AD_4

— Key for the ciphertext derived from the Exporter, using label “EDHOC_message_4_Key”.

— Comments/objections?
Message duplication and idempotency

- Loss of Message_4 (#65)
- State idempotency (#85)

- Message_4 enables key confirmation
  - “In deployments where no protected application message is sent from the Responder to the Initiator, the Responder MUST send message_4.”
- What if message_4 never reaches the Initiator (e.g. lost CoAP non-confirmable)? If Initiator resends message_3 how does the Responder act?
  - Message de-deduplication is a complication for constrained implementations.
  - Not supported by all CoAP implementations
- Significant simplification if EDHOC behaves as an idempotent resource
  - Same message in response to earlier message
  - No new security processing of earlier messages
  - Response message may be cached (in case of message_4, recreated)

- Any objection to specifying EDHOC to behave as idempotent resource?
Misc. message processing

- When key-confirmation is needed latest (#63)
- Max retransmissions of EDHOC messages (#64)

- What shall the Initiator do if not receiving key confirmation?
- For how long shall it retransmit? When shall it clear the security context?

- In general: EDHOC needs application input
- Key confirmation coincides with reachability:
  - Initiator verifies protected message ⇔ message reached endpoint and right key was used by the Responder.
- Current text:
  
  “While the Initiator can securely send protected application data, the Initiator SHOULD NOT permanently store the keying material PRK_4x3m and TH_4 until the Initiator is assured that the Responder has actually computed the key PRK_4x3m (explicit key confirmation).”

- Additional clarifications?
Test vectors as JSON

- Test vectors as JSON (#78)
  - Requested by several implementers
  - Already used by one research institute
  - Who wants to be involved in making a template?
Test vectors additions

- Test vector additions (#47)
  - obsolete PSK test vectors
  - add TH4 output
  - add exporter, exporter outputs, keyupdate, to all test vectors
  - add KEYSTREAM_2
  - print out public keys.
  - add real certificates to test vectors - 0:CBOR native and DER (and possibly later 1:ASN.1 translated)
  - Add ciphersuites 2 and 3 to test vectors, ECDSA keys should print out full y coordinate
  - Add error message
  - Add message_4
  - Add message_1 C_1 dummy null byte

Anyone that can provide a first run with certs or suite 2 or 3?
Connection identifier encoding

- **Coding density for bstr_identifier (#79)**
  - bstr_identifier makes compact encoding of short CBOR bstr as CBOR int
    - Mapping to positive/negative 1-byte CBOR integers
    - Mapped back to bstr since connection identifier is used as OSCORE Sender ID (which is bstr)
    - Reduces EDHOC overhead but slight complication in specification
    - Primarily used with connection identifiers, also with ID_CRED_x in case of kid

  - Two ideas:
    1. Use the same trick for efficient encoding of longer CBOR bstr
       - Not necessarily more complicated, since there is already a mapping step
    2. Decouple the EDHOC types from representation of OSCORE Sender IDs
       - Allow bstr_identifier to be either int or bstr
       - Make the mapping of OSCORE together with rest of OSCORE context establishment

  - **Shall we do 1 and/or 2 (or none)?**
Revisit of MTI cipher suite

• MTI cipher suite (#22)

  — Outcome of previous discussion:
    — Less constrained devices SHOULD implement suite 0 and 2.
    — Constrained endpoints SHOULD implement suite 0 or 2.

  — Comment: Why did we not mandate suite 2 (ECDSA)?
  — Previous considerations: “Change MTI cipher suite to one based on ECDSA, has negative consequences on performance and security, which will remain for long time.”
  — Performance:
    a) processing time (ECDSA hardware acceleration pre-dominantly for constrained devices)
    b) code footprint (no availability of SHA-512 acceleration)
    c) message overhead (same)
(if time permits) misc. open issues

- Opportunistic use (#88)
- COSE header map for public key (#82)
- Deterministic CBOR encoding (#71)
- Identifying a certificate with 'kid' (#32)