Ephemeral Diffie-Hellman Over COSE (EDHOC)

draft-ietf-lake-edhoc-02

LAKE, IETF 109, November 2020

Changes

Changes from -00 to -01

- Removed PSK Method.
- Removed examples with certificate by value.
- Editorial changes.

Changes from -01 to -02

- Added section 1.2. “Use of EDHOC” to better describe the use cases for EDHOC.
- Clarified how to encode the bstr_identifiers (C_I, C_R, kid_I, kid_R). New bstr_identifiers subsection added.
- Added more text describing what an identity or ”small set of identities” can be in practice.
- Added text on key confirmation. Recommended that I and R have explicit key confirmation of the key PRK_4x3m before they store the keying material (PRK_4x3m and TH_4) long-term.
- Added test vectors for EDHOC authenticated with static Diffie-Hellman keys.
- Editorial changes.

Open Issues

- Agreement/negotiation of parameters (#11, #23)
- Rekeying OSCORE AEAD (#20)
- Future-proofing EDHOC (#17, #19)
- More ways to Identify certificates ('kid', 'c5u', c5t') (#32, #33)
- Verification of intended peer (#8)
- Resumption (#25)
- Distinguish error message (#30)
- ID encryption in message_2 (#34)
- Delivery receipt for message_3 / key confirmation (#10, #18)
- TEE Assumptions (#5)
- Forward and backward secrecy (#24)
- Register cipher suites with high security (#35)
- SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)

Agreement/negotiation of parameters (#11, #23)

- **Agreement/negotiation of parameters/options (#23)**
- **Agreement of method (#11)**

- Parameters agreed beforehand:
  - Method, Transport and Correlation, Use and format of AD_1, AD_2, AD_3
- Parameters negotiated/verified:
  - Cipher suite
- Chosen by each party (no current requirement and agreement beforehand):
  - Raw public key OR certificate: CRED_R and CRED_I
  - Type of credential identifier: ID_CRED_R and ID_CRED_I

- EDHOC has parameters/options that are set in different ways. Agreeing beforehand reduces flexibility. Negotiation might increase message sizes and roundtrips. Letting each party chose parameters in the protocol run allows an on-path attacker to block parameters that can be seen from the message size, and may affect availability as the parties may not support the same COSE options.

- **Are we happy with the assumptions on which parameters to negotiate and which are agreed beforehand?**

Rekeying OSCORE AEAD (#20)

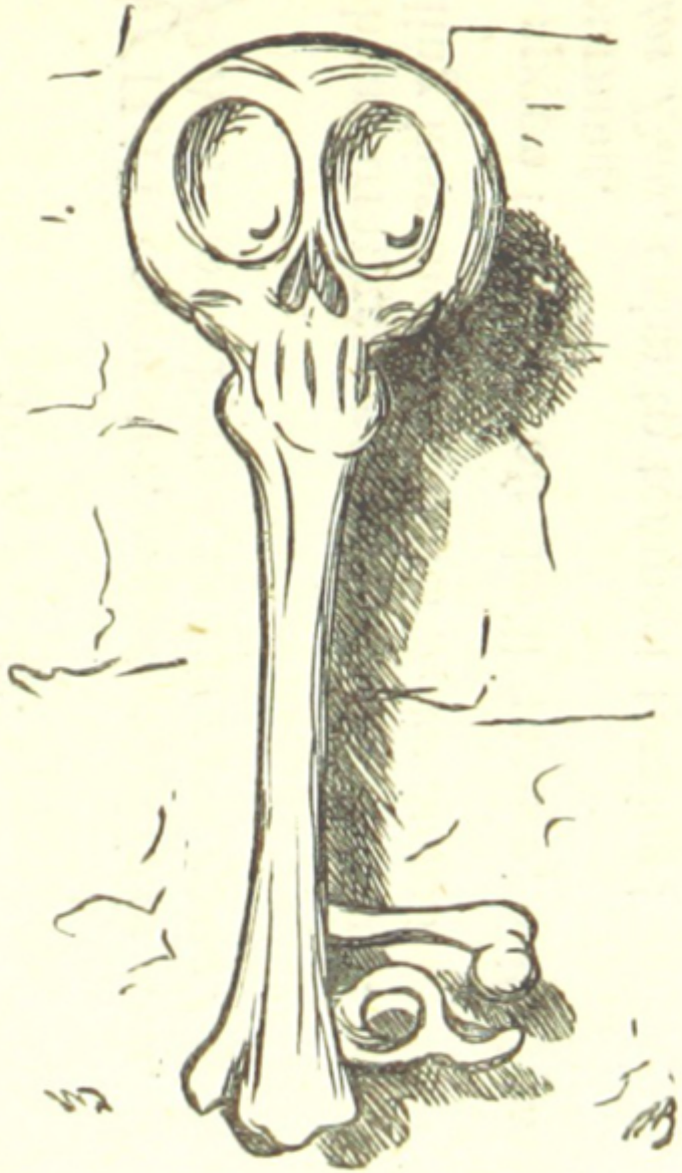
- **Shall we solve rekeying of AEAD within EDHOC, or let the data protection protocol, e.g. OSCORE, handle it more efficiently? (#20)**

- CFRG are working on a document specifying equations to calculate AEAD limits for the number of encryption operations *q* and the number of forgery attempts *v*. TLS, DTLS, and QUIC has adopted strict limits based on the same equations. https://tools.ietf.org/html/draft-irtf-cfrg-aead-limits

- Limits are based on a target probability for forgery of a single packet or distinguishability from a random string. Packet length (plaintext + additional data) also affects the limits. Having strict limits is not a problem if re-keying is easy.

- EDHOC use each AEAD key only once, but it might be a problem for OSCORE. Re-keying can be done in EDHOC or OSCORE. DTLS 1.3 sets the limits for CCM to q = 2^{23} and v = $2^{23.5}$ and states that CCM_8 MUST NOT used in DTLS without additional safeguards against forgery.

- **Should the IETF IoT community discuss reasonable limits for q and v for CCM_8?**
- **Overall, does this sound like something better to do in (OS)CORE?**



Future-proofing EDHOC (#19)

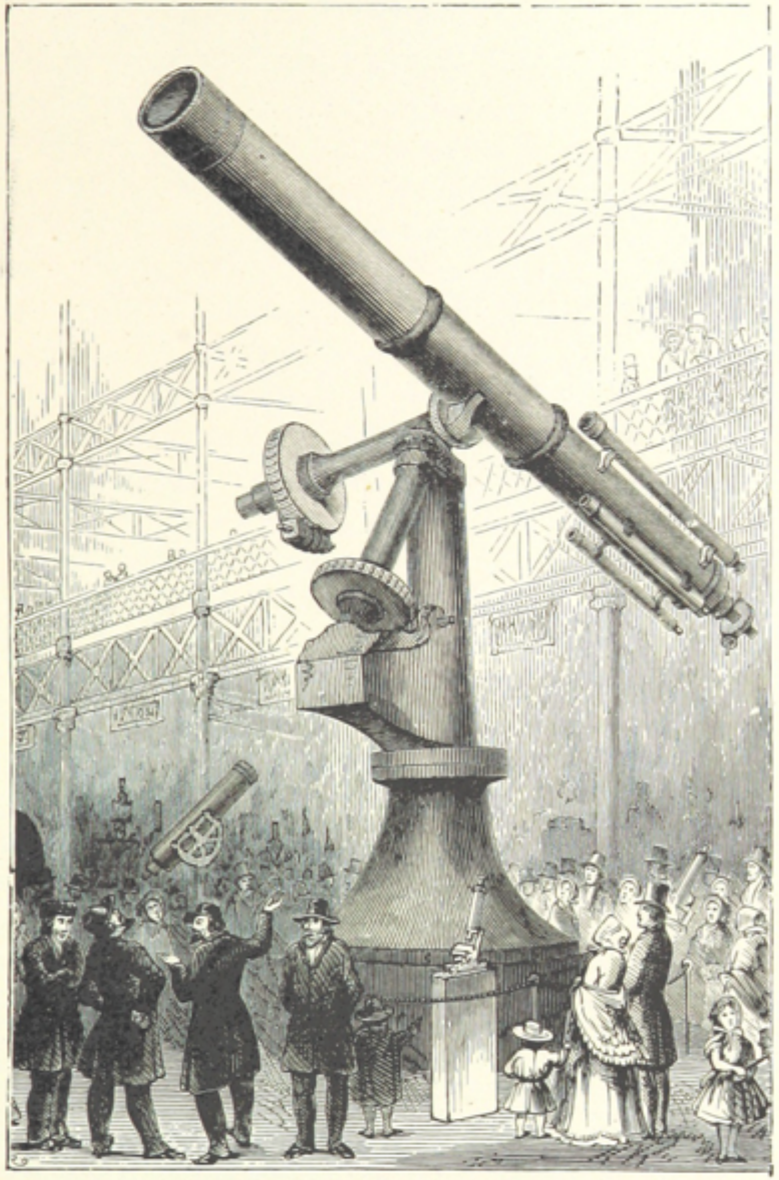
— **Shall we replace HKDF with a more general extract-and-expand to allow KMAC? (#19)**

— HMAC is needed to mitigate the length extension weakness of SHA-2. SHAKE does not have this weakness and NIST has therefore standardized the simple and efficient KMAC mode.

— Suggestion to specify EDHOC in terms and “Extract” and “Expand” instead of “HKDF-Extract” and ”HKDF-Expand”. Use HMAC when the hash function is SHA2. Use KMAC when the hash function is SHAKE.

PRK = Extract(salt, IKM) = KMAC128(salt, IKM, 256, "KDF extraction")
OKM = Expand(PRK, L, info) = KMAC128(PRK, info, L, "KDF expansion")

— **Specify use of KMAC for extract-and-expand?**
— **Ask CFRG for advice?**



TELESCOPE IN EXHIBITION, 1851.

Future-proofing EDHOC (#17)

- **Shall we specify EDHOC in terms of KEM? (#17)**

- All new PQC algorithms from NIST is expected to be specified as Key Encapsulation Methods (KEM). Might be a good idea to specify EDHOC in terms of KEM so that is is prepared to be used with future KEMs. This would be purely a specification change, no change to messages or implementation.
- EDHOC can be specified using the CFRG HPKE functions (using the exact definitions would change implementations):

(skX, pkX) = **GenerateKeyPair**()	(skX, pkX) = **GenerateKeyPair**()
(shared_secret, enc) = **Encap**(pk),	(shared_secret, enc) = **AuthEncap**(pkR, skS)
shared_secret = **Decap**(enc, sk),	shared_secret = **AuthDecap**(skR, pkS)

- ECDHE can quite easily be specified as a KEM. CFRG HPKE does exactly this.

  (sk, pk) = (x, g^x)   (shared_secret, enc) = (g^{xy}, g^y)

- While future PQC KEMs are likely adhere to the Encap, Decap interfaces, it seems uncertain that they will adhere to the CFRG HPKE AuthEncap, AuthDecap interfaces. Or? PQC algorithms might require additional precautions when keys are used more than once. In AuthEncap() both pkR, skS would be used more than once.

- **Wait until it is clear how future PQC algorithm interfaces looks like?**
- **Ask CFRG for advice?**

More ways to Identify certificates ('kid’, ‘c5u’, c5t’) (#32, #33)

— **Identifying a certificate with 'kid’ (#32)**

— **Reference draft-mattsson-cose-cbor-cert-compress (#33)**

— Currently the specification assumes that certificates are identified with ’x5t’ or ‘x5u’ and RPK are identified with a ‘kid’.

— COSE WG is planning to work on ”cbor certificates” and are expected to register ‘c5t’, ‘c5u’, …

— An ‘x5t’ identifier such as ”{ 34: [-15, h'FC79990F2431A3F5’] }” is at least 14 bytes when encoded while a ‘kid’ can be as small as 1 byte (in EDHOC). This might limit the use of certificates compared to RPK in very constrained environments.

— If a raw public key in EDHOC can be identified with a ‘kid’, then a certificate could as well. Specifying this would enable certificates by reference to be used with as small messages as RPK. Making them fit in 51 bytes LoRaWAN and 5-hop 6TiSCH.

— **Specify that a certificate can be identified with a ’kid’?**

— **Examples with ‘c5t’, ‘c5u’ when/if adopted by COSE WG?**

Verification of intended peer (#8)

- **Should the “subject name” in raw public keys be recommended or even mandatory? (#8)**

- When raw public keys are used in EDHOC, EDHOC allows the parties to agree on an optional “subject name” that are included in the MAC/Signature (i.e. not sent on the wire). This protects against attacks where an attacker has succeeded in registering a public key with a different ”subject name”.

- We would like more input from people deploying constrained IoT devices and what their operational requirements on identity are. Do they see the public key itself as the identity? Is a subject name (like EUI-64) available in many or all cases?

- Suggestions that the word ”identity” is a bit too high level and that the draft should only talk about ”public key” and “subject name”.

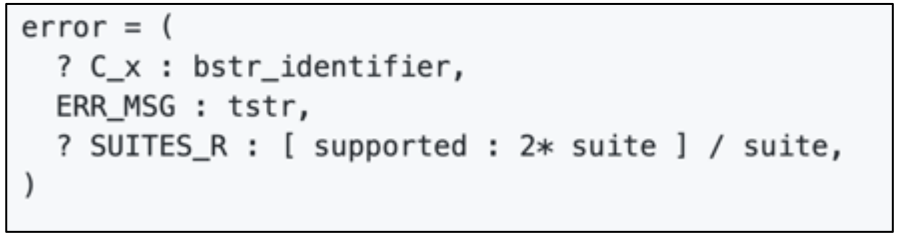
- **Recommend/mandate use of subject name?**
- **Identity/Identifier/subject name terminology?**

Resumption (#25)

- **Is there any need for a resumption mechanism in EDHOC? (#25)**
- Resumption provides a way to minimize state when it is unclear when/if there will be future connections. Also provides re-keying, ”forward secrecy”, and guarantees that future messages are fresh.
- IoT connections are typically long lived. Re-keying and freshness (ECHO option) can be achieved more efficiency in OSCORE. OSCORE re-keying will be discussed in CORE on Friday. ”Forward secrecy” is discussed in another issue. Seems to be agreement that resumption itself is not needed as a feature.
- **Specific resumption method is not needed?**

Distinguish error message (#30)

- **How to distinguish a regular message from an error message (#30)**



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

— error	? int / bstr, **tstr**
— message_1	int, [] / int
— message_2	? int / bstr, bstr
— message_3	? int / bstr, bstr

— The error message has a tstr as the 1st or 2nd CBOR item. Is this good enough for distinguishing?

— **If 1st or 2nd CBOR object is tstr (major type 3) then error message? Add guidance in draft?**

ID encryption in message_2 (#34)

- **How to do encryption without integrity in message_2 (#34)**

— As the Responder sends its identity to an unauthenticated part, there is no need to have IND-CCA encryption against active attackers. IND-CPA encryption is enough in this case. (everything is integrity protected by the inner MAC).

1. The current specification generates a long encryption key and perform XOR cipher.
2. Remove the tag from AEAD ciphertext. Only works when AEAD has a well-defined tag.
3. Associate a IND-CPA encryption alg with each AEAD. Requires table. (AES-CCM, AES-GCM -> AES-CTR, ChaCha20-Poly1305 -> ChaCha20)

- **1), 2), or 3)?**

THE CAPELLO E CAPOTE OF THE AZORES.
FROM A PHOTOGRAPH.

Page 138.

Delivery receipt for message_3 / key confirmation (#10, #18)

- **Optional message_4 for key confirmation (#18)**
- **Injective agreement issue (was: G_IY in session key material) (#10)**

— The Initiator would typically want a delivery receipt for message_3 / explicit key confirmation of PRK_4x3m, otherwise the Initiator does not know if the Responder has received and accepted message_3.

— To get explicit key confirmation the Initiator needs to receive a MAC from the Responder. The MAC can be an OSCORE Response, OSCORE Request, or any other MAC.

— Sending message_3 in OSCORE as specified in draft-palombini-core-oscore-edhoc and requiring a response solves the problem for use cases where the EDHOC Initiator is OSCORE client. When the EDHOC Initiator is OSCORE server, a first OSCORE request is needed to provide key confirmation.

— For use cases where relying on OSCORE or any other messages with a MAC is not suitable, it has been suggested to add an optional fourth EDHOC message_4 with a MAC. Concern that message_4 increases overhead and complexity. An optional fourth message could for example be specified in an appendix. Initiator could indicate whether the Responder must send a message_4 (and fail if this is not received)

— **Introduce optional fourth EDHOC message_4 or only rely on OSCORE?**

— **Add recommendation that client send OSCORE request in parallel with or soon after message_3?**

TEE Assumptions (#5)

- **Which information and cryptographic operations can be expected to be generated/stored/performed inside a TEE? (#5)**

- Long-term public authentication keys?
- Ephemeral public keys?
- PRK_4x3m and EDHOC-Exporter?
- EDHOC protocol?
- OSCORE protocol?

- This would be useful to write recommendation about. Current draft recommends “as much as possible…”

- It is also an input to protocol design: ”Compromise of key X does not lead to compromise of key Y” does not matter practically if X and Y are stored together.

- **Assume/recommend that all EDOC keys (authentication keys, ephemeral keys, PRK_4x3m) are kept in TEE?**

THE ANGEL'S STORY.

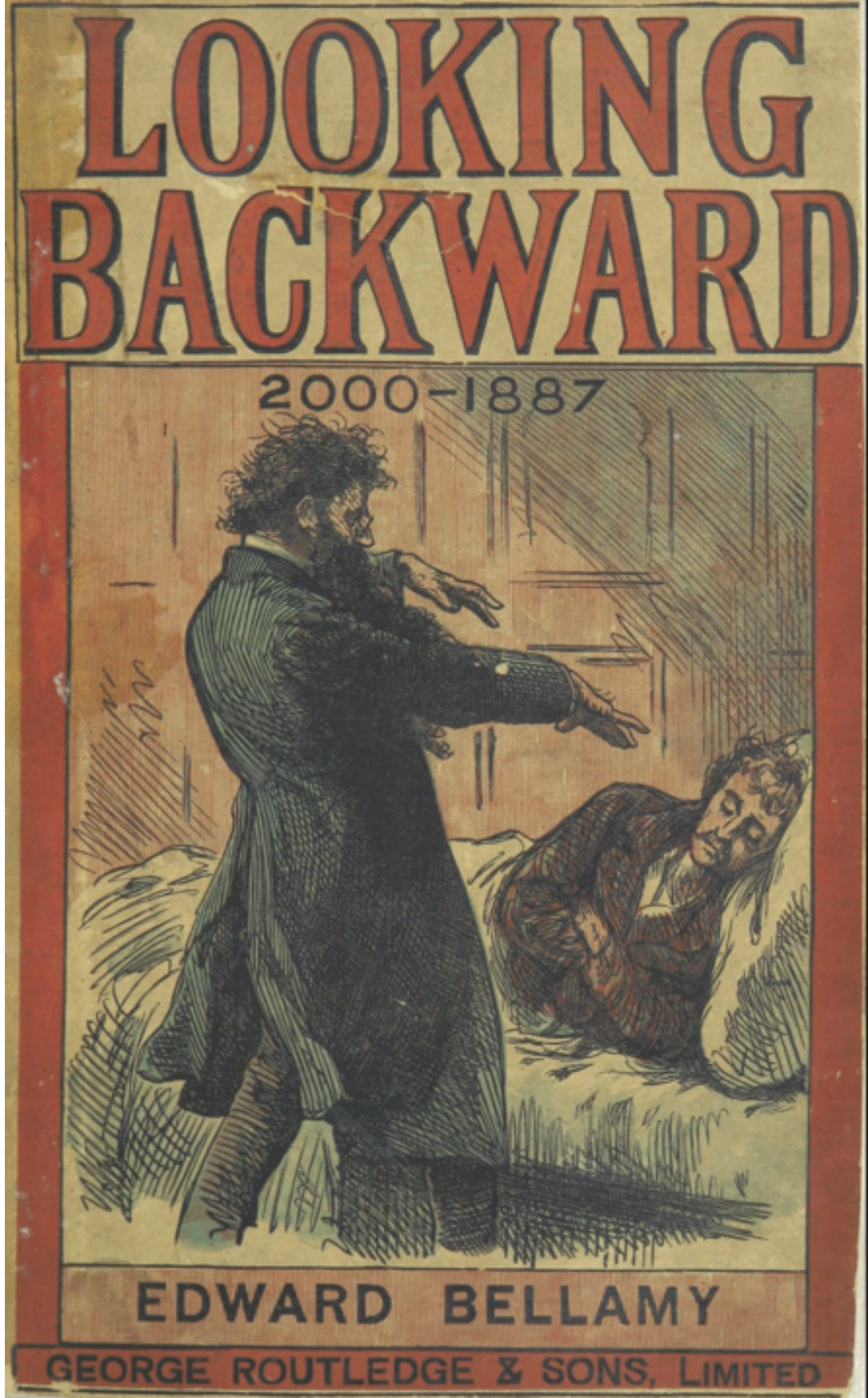
Forward and backward secrecy (#24)

- **Forward and backward secrecy (#24)**
- **Related to AEAD rekeying (#20) and resumption (#25) and TEE (#5)**

- Is there a need for a new lightweight protocol component that provides both forward and backward secrecy? Or is it sufficient to rerun EDHOC periodically and maybe PSK-based FS in OSCORE with a chain of hashed session keys or by exchanging nonces or as part of a rekeying solution?

- Goal is that an attacker compromising the session state used to protect message with sequence number *s* shall not be able to decrypt/forge messages with sequence number *s* - *r* and *s* + *r*, where *s* is a security parameter (limits for time can be achieved with a solution for sequence numbers).

- **Rerunning EDHOC every for every time the OSCORE sequence number is congruent with 0 mod r achieves both forward and backward secrecy.**



Forward and backward secrecy (#24)

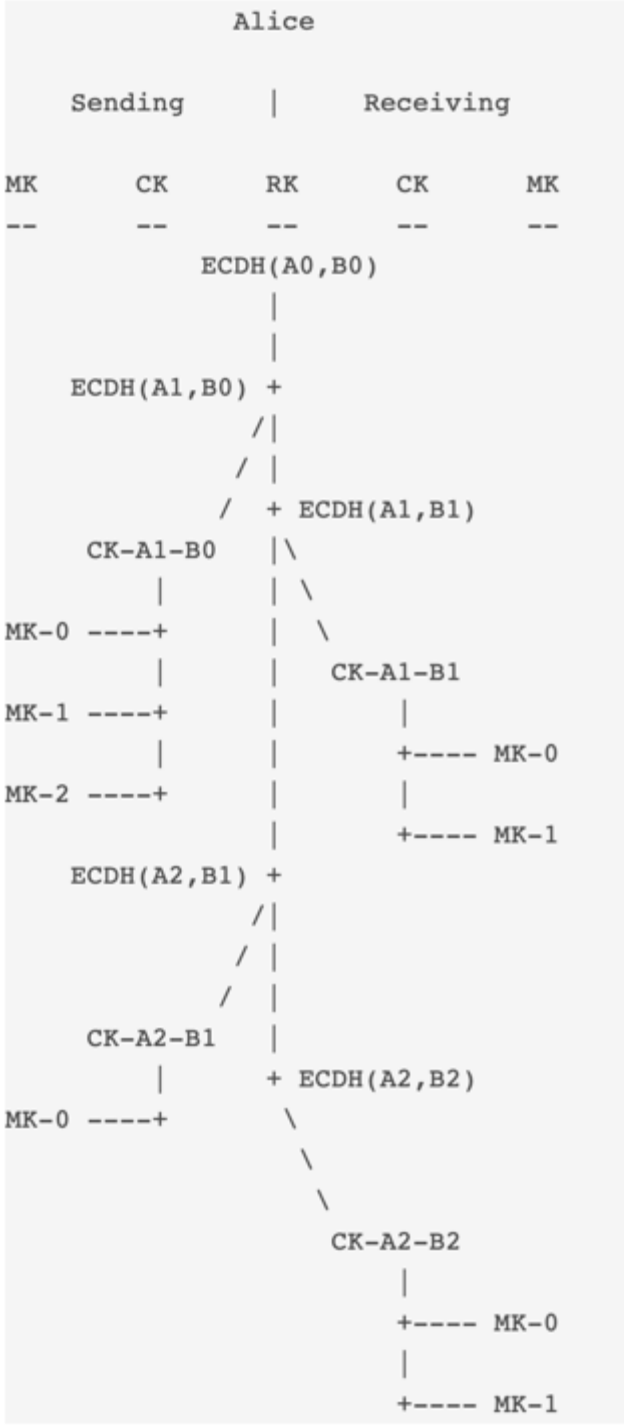
- Some solutions only protect against attackers that are passive after the compromise. Other solutions protect also against active attackers. Might also be differences depending on for which messages the attacker eavesdropped on/was active.

- One potential solution would be to add cryptographic ratcheting similar to the Signal protocol. The shared secret could be updated every r^{th} message. See figure from Signal on the right →

- Several comments that we should avoid adding complexity and code size unless necessary. The size of added code to firmware updates, should be compared to the message sizes or rerunning EDHOC.

- Adding any form of key update to EDHOC and OSCORE adds severe problems with synchronization. A solution should self-synchronizing in the way that the receiver knows from the received OSCORE message which key to use.

- **Sufficient to optionally rerun EDHOC periodically and discuss/specify lightweight ways to get FS as part of rekeying discussion in (OS)CORE?**
- **Recommendation in EDHOC/OSCORE to have policies for rerunning EDHOC based on time/number of OSCORE messages?**



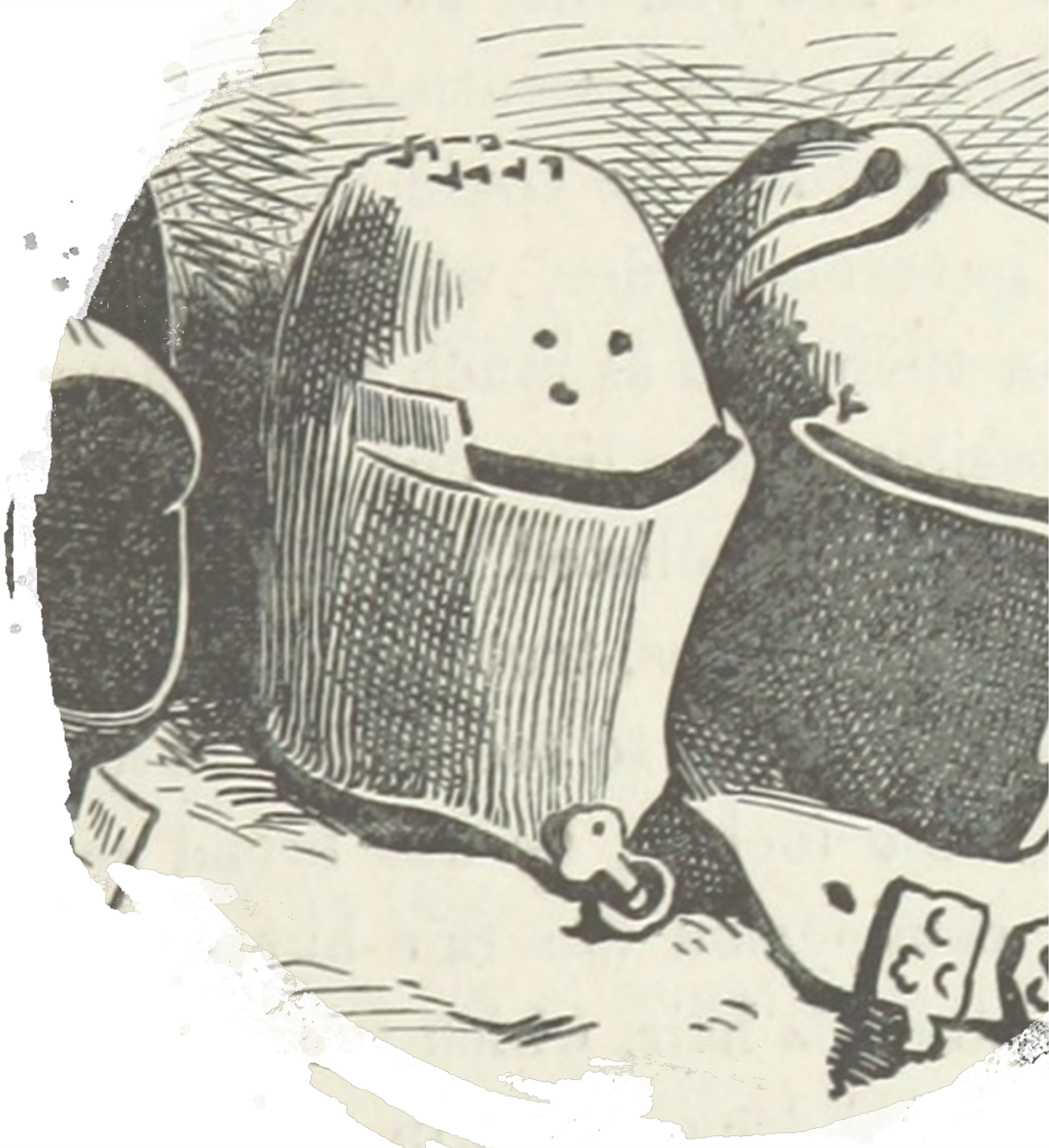
```
              Alice

    Sending     |     Receiving

MK      CK      RK      CK      MK
--      --      --      --      --
            ECDH(A0,B0)
                |
                |
    ECDH(A1,B0) +
               /|
              / |
             /  + ECDH(A1,B1)
     CK-A1-B0   |\
         |      | \
MK-0 ----+      |  \
         |      |   CK-A1-B1
MK-1 ----+      |       |
         |      |       +---- MK-0
MK-2 ----+      |       |
                |       +---- MK-1
    ECDH(A2,B1) +
               /|
              / |
             /  |
     CK-A2-B1   |
         |      + ECDH(A2,B2)
MK-0 ----+       \
                  \
                   \
                    CK-A2-B2
                        |
                        +---- MK-0
                        |
                        +---- MK-1
```

Register cipher suites with high security (#35)

- **Register cipher suites with high security (#35)**

- There has been interest to use EDHOC with high level security, i.e. AES-256 and 384 bit ECC.

- **Register CNSA compliant cipher suite for government and financial use?**



SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)

- **Cipher suites requiring multiple SHA (#2)**
- **Use of SHA-512 in constrained IoT (#21)**
- **Mandatory to implement cipher suite (#22)**

- Many comments regarding **device support**, **performance**, and **security** pointing in different directions..

- **Device support:** ECDSA, SHA-256, P-256 is the current default choice and has very wide support. SHA-512 is not supported on many IoT devices and the question is if it ever will. Currently SHA-256 have wide support and is often HW accelerated. Adding also SHA-512 requires more code storage. If SHA-256 gets replaces it would likely be with SHAKE128 or some future XOF emerging from the NIST lightweight standardization protect (e.g. Gimli) that can do both AEAD and hashing.

SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)

- **Performance:** Having high performance ECC algorithms are important to reduce latency. While some earlier benchmarks indicated huge performance benefits with Curve25519 and Ed25519 compared to P-256 ECDH and ECDSA, a large part of the difference seem to have been due to unoptimized P-256 implementations. On some platforms Ed25519 seems to be significantly faster, while they seem to have equal performance on other platforms.
- http://essay.utwente.nl/75354/1/DNSSEC%20curves.pdf
- https://bearssl.org/speed.html

- **Security:** We have received several comments that people would like some of the security improvements in Ed25519 compared to ECDSA with P-256. The Minerva attack last year used ECDSA side-channels to do practical recovery of the long-term private key. Several academic papers have also shown that deterministic ECC signatures like Deterministic ECDSA or Ed25519 are vulnerable to side-channel attacks
- https://minerva.crocs.fi.muni.cz/
- https://tools.ietf.org/html/draft-mattsson-cfrg-det-sigs-with-noise

SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)

- **Mandatory-to-implement (MTI) cipher suite:**
- No ideal MTI ECC algorithms. Concern with support for Ed25519 in legacy low end microcontrollers. Concern with performance and security of ECDSA and P-256. Ed25519 with SHA-256 would be an improvement for many. ECDSA, SHA-256, P-256 might be the only thing that can currently be mandated.
- **COSE (RFC 8152):** "Applications need to determine the set of security algorithms that are to be used. When selecting the algorithms to be used as the mandatory-to-implement set …"
- **Recommend that implementations provide algorithms based on both P-256 and Curve25519 if they can, at least one of them?**
- **Do like COSE and let application determine MTI?**

SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)

- **What is the future IoT signature algorithm?**
  - ECDSA/Schnorr/EdDSA/qDSA
  - Weierstrass/Edwards/Montgomery
  - SHA-256/SHAKE128/Gimli
  - Deterministic/Random/Deterministic + Random

- ECDSA with SHA-256 + P-256 is the current default choice that has very wide support. Several companies has commented that they want to move away from ECDSA and P-256 for security and performance reasons.
- Ed25519 mandates use of SHA-512 and mandates deterministic nonce generation making it quite unsuitable for IoT. ECDSA25519 solves some of these problems but is still ECDSA.
  https://tools.ietf.org/html/draft-ietf-lwig-curve-representations-13
  https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf

- **Encourage IETF work on Schorr/EdDSA/qDSA suitable for future IoT?**

WANTED

CONTINUE ISSUE DISCUSSION ON GITHUB

MORE REVIEWS

PARTICIPATE IN INTEROP