Ephemeral Diffie-Hellman Over COSE (EDHOC)

draft-ietf-lake-edhoc-03

LAKE, IETF, December 2020

Issue updates since IETF 109

- **Resumption (#25)**

- **Agreement/negotiation of parameters (#11, #23)**
  - New appendix in -03 that describes the parameters that need to be agreed upon between Initiator and Responder has been added. Currently some overlap between the new appendix and the section “Communication/Negotiation of Protocol Features”.

- **More ways to Identify certificates ('kid’, ‘c5u’, c5t’) (#32, #33)**
- **Verification of intended peer (#8)**
  - COSE WG have on ongoing discussion on how to identify a certificate with ‘kid’.
  - Request to add ‘c5u’, c5t’ to the EDHOC test vectors. The CBOR certificate draft will add the subject private key to enable this.
  - New text in -03 on why SIGMA require a “subject name” and the kind of misbinding attacks this mitigates. This is a bit related to ongoing discussion on the security of ‘x5u’ in COSE.

- **Distinguish error message (#30)**
  - Text in -03 on how to distinguish error message bases on text item.
  - The issue has spawned several new discussions. Are there a need to distinguish message_1 from message_3 except the connection ID and 5-tuple? Is the error message something the implementor define or should EDHOC define the error messages?

Issue updates since IETF 109

- **Shall we replace HKDF with a more general extract-and-expand to allow KMAC? (#19):**
  - Changed from HKDF-Extract and HKDF-Expand to general Extract-and-Expand. For SHA-2, HKDF is used, for SHAKE, KMAC is used.
  - (Note that this does change anything when SHA-256 is used. We are also currently not planning on adding SHAKE cipher suites, but SHAKE (like all other COSE algorithms) can be used with private cipher suites.

- **Shall we specify EDHOC in terms of KEM? (#17):**
  - We tried to implement EDHOC with the HPKE interfaces.
  - The signature mode maps quite well to the unauthenticated HPKE interfaces, with the difference that HPKE forces Extract-and-Expand and EDHOC uses Extract and Expand as separate functions and use the intermediate key to derive several keys with Expand.
  - The Static Diffie-Hellman modes does not map well to HPKE as HPKE do GenerateKeyPair() inside Encap(), while EDHOC relies on doing GenerateKeyPair() outside of “Encap()” as an essential optimization.
  - The conclusion is therefore that this change should not be done as it would change the key derivation quite much as well as significantly increase message size.
  - We should however consider to future proof EDHOC so it can be updated with PQC KEMs. Unclear if “Static DH” authentication with PQC KEMs provide any benefits compared to PQC signatures.

Register cipher suites with high security (#35)

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

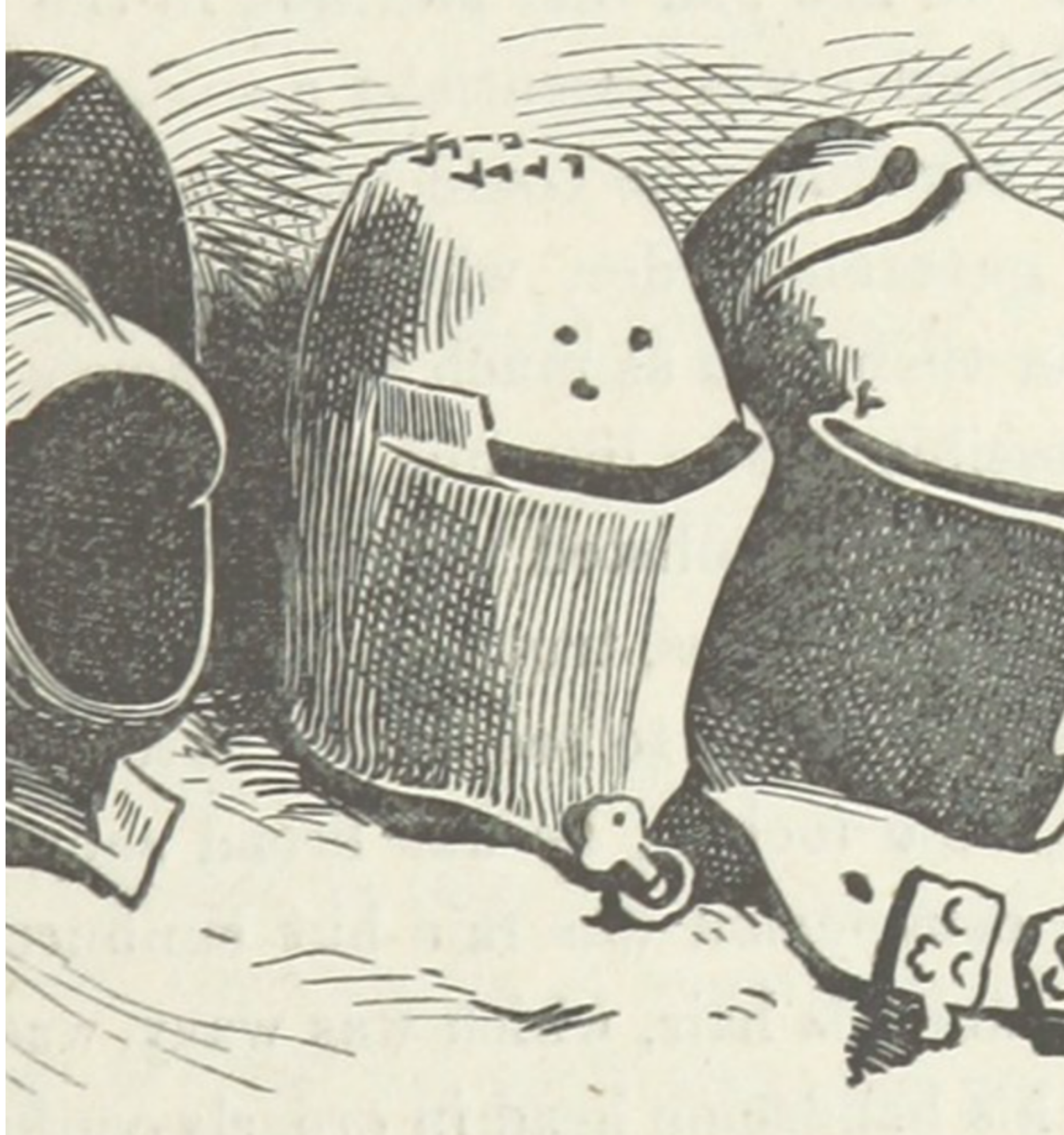
— Several independent requests to include cipher suites compatible with CNSA. One request to add a general non-constrained cipher suite.

— We have added the following two registered cipher suites to -03. The first one is compatible with CNSA and the second uses the algorithms popular on the web.

  — A256GCM, SHA-384, P-384, ES384, P-384, A256GCM, SHA-384

  — A128GCM, SHA-256, X25519, ES256, P-256, A128GCM, SHA-256

— We have also added additional text describing that EDHOC can be used with all COSE algorithms/curves by using the private cipher suites.



Open Issues

- Rekeying OSCORE AEAD (#20)
- ID encryption in message_2 (#34)

- **Delivery receipt for message_3 / key confirmation (#10, #18)**
- **TEE Assumptions (#5)**
- **Forward and backward secrecy (#24)**
- **SHA-512, signature algorithms, and MTI cipher suite (#2, #21, #22)**

Rekeying OSCORE AEAD (#20)

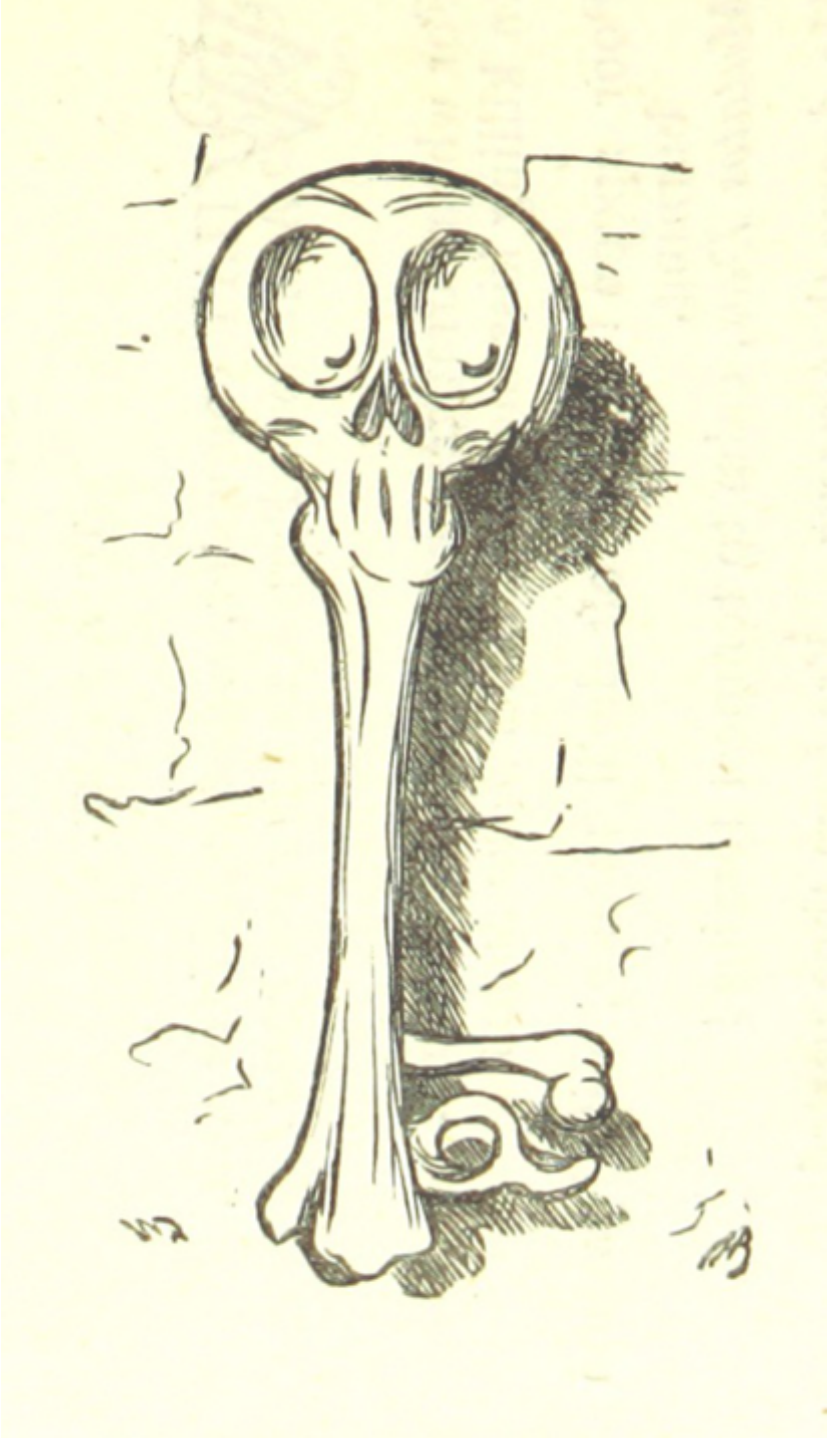
- **Shall we solve rekeying of AEAD within EDHOC, or let the data protection protocol, e.g. OSCORE, handle it more efficiently? (#20)**

- 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?**



Rekeying OSCORE AEAD (#20)

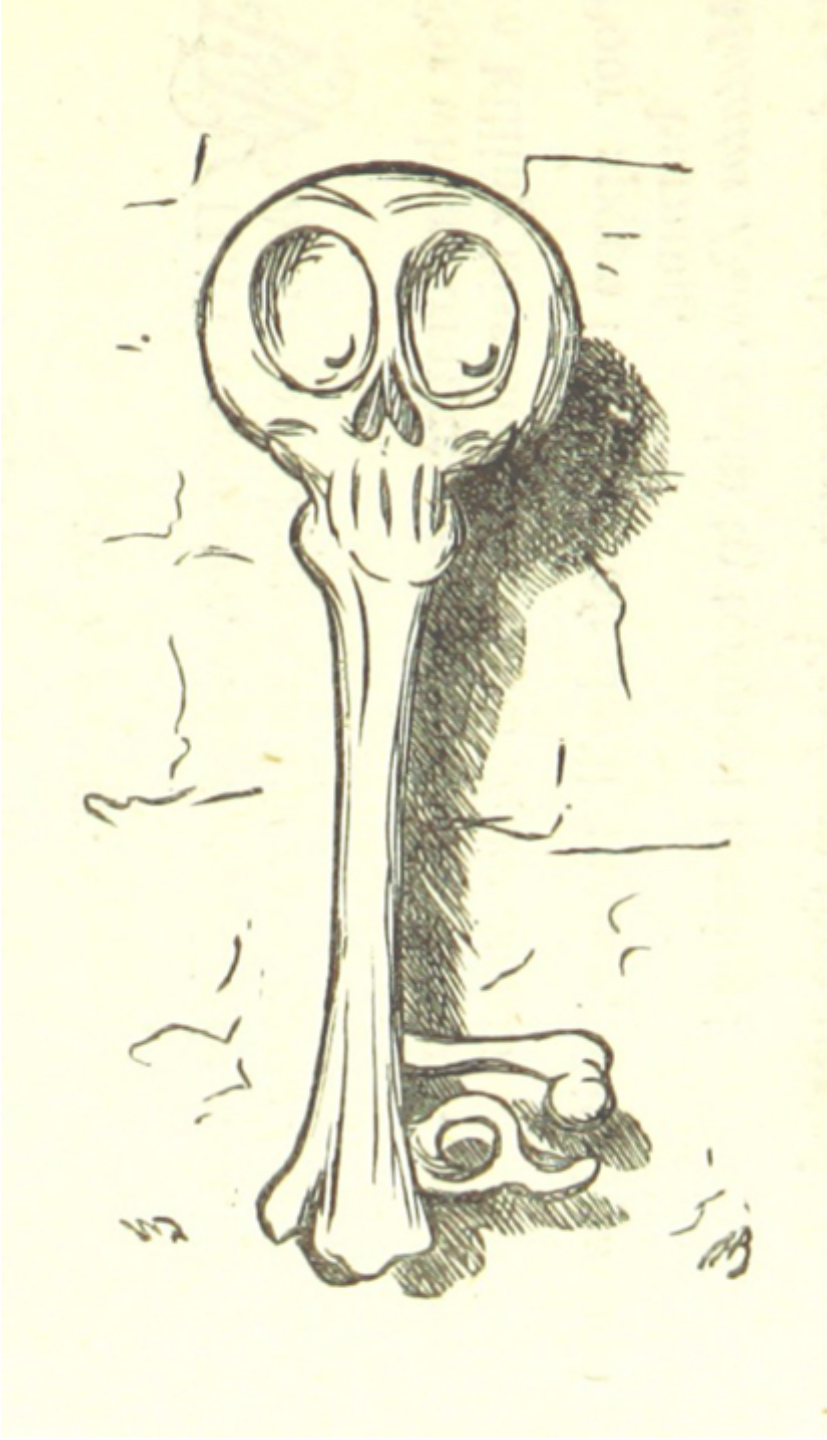
- **Shall we solve rekeying of AEAD within EDHOC, or let the data protection protocol, e.g. OSCORE, handle it more efficiently? (#20)**

- CORE WG discussed rekeying and forward secrecy at IETF 109. The discussion has continued after IETF 109. Preliminary conclusions are that
  - New CORE draft will specify OSCORE AEAD counters
  - OSCORE RFC 8613 appendix B.2 (or an update) is needed for lightweight rekeying in OSCORE. Appendix B.2 exchange authenticated nonces and switch keys.
  - Good if lightweight forward secrecy (i.e., a hash-chain) can be done in EDHOC to avoid changes in OSCORE. EDHOC keys might be stored in TEE.

- Lightweight FS can be achieved by "hashing" the key PRK_4x3m with the EDHOC-Exporter. The nonce make sure that the "hash-chain" does not have short cycles. The following function has been made in -03

```
EHDOC-Exporter-FS( nonce ):
       PRK_4x3m = Extract( [ "TH_4", nonce ], PRK_4x3m )
```

- FFS where the nonce comes from (EDHOC, OSCORE) and if it is a counter, a random number, or counter+random number.
- The use of OSCORE appendix B.2 together with the EHDOC-Exporter-FS function align on a high level with the original mechanism proposed by Karthik
  https://mailarchive.ietf.org/arch/msg/lake/vkJunXEQZ33HP9YpNByQesEW7l8/



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.

ID encryption in message_2 (#34)

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

— The agreement from IETF 109 was to specify new modes of AES and ChaCha20 for message_2 similar to TLS 1.3.
— After trying to implement this we think it is a bad idea as it makes things quite complicated for developers and complicated the specification. In EDHOC it is not enough with AES-ECB and the ChaCha20 block cipher as in TLS 1.3. EDHOC would need AES-CTR and the ChaCha20 stream cipher.
— This makes the specification a bit complicated and it makes it a bit complicated for developers with a COSE implementation as they must dig out AES and ChaCha20 and implement stream cipher modes.
— We would like to bring this up for discussion again. We think both 1 and 2) would be better choices with low complexity for developers.

1. **Generate keystream with HDKF-Expand.**
2. **Encrypt like message_3 and remove the tag (if there is a well-defined tag).**

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.

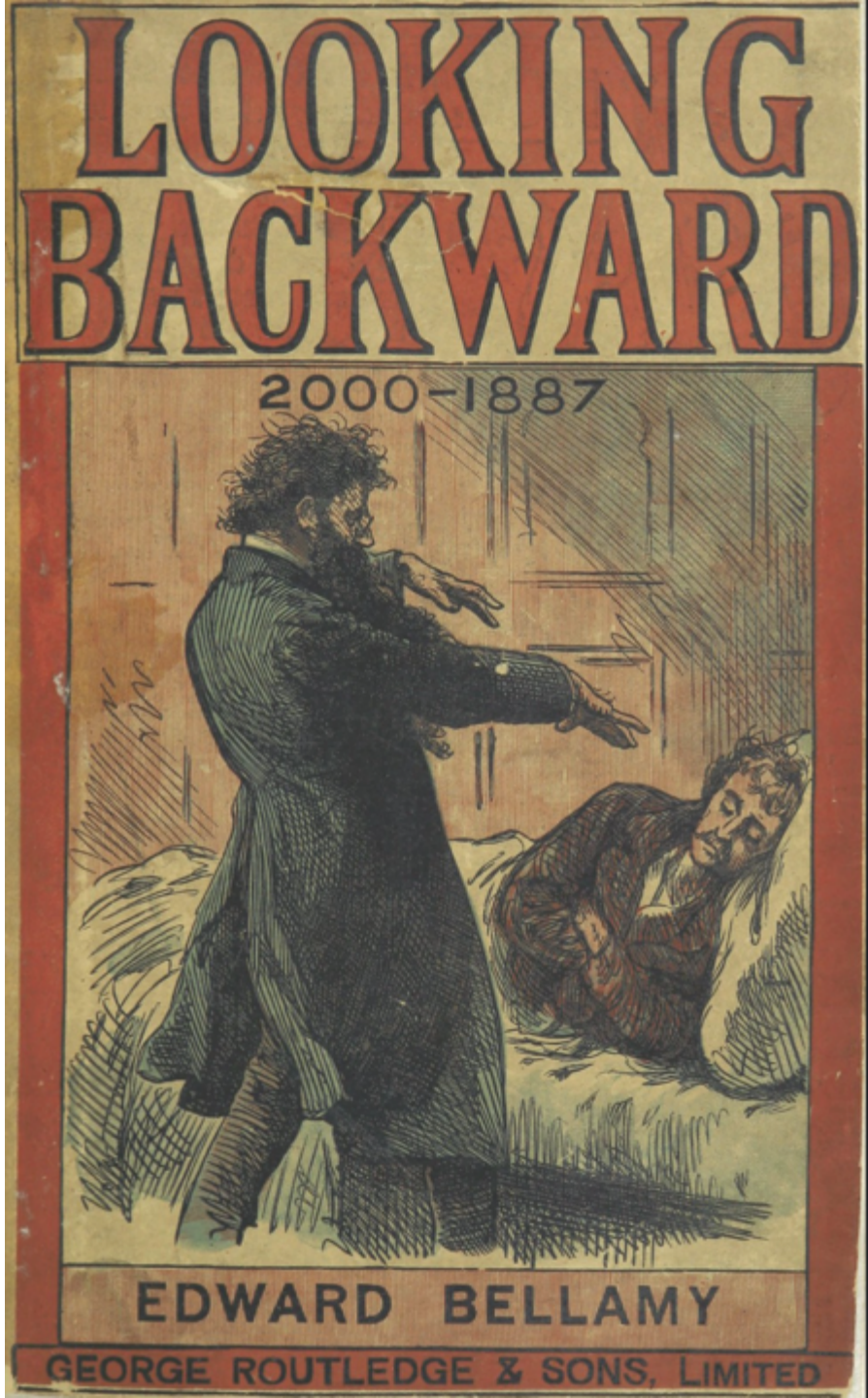
Forward and backward secrecy (#24)

- **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)

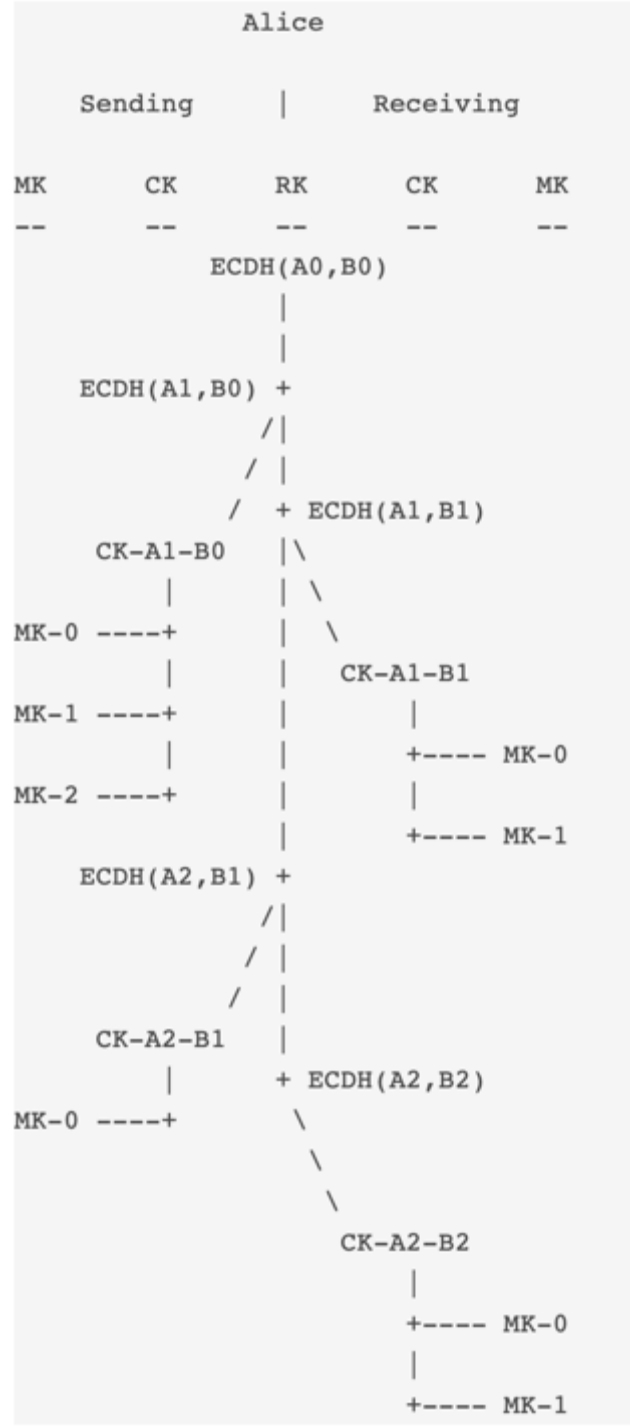
- Some solutions only protect against attackers that are passive after the compromise. Other solutions protect also against active attackers. Might also be differences depending on for which messages the attacker eavesdropped on/was active.

- 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?**



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