Secure Channels
Summer Term 2018

Problem Set 7

Prof. Stefan Lucks, Eik List

Bauhaus-Universität Weimar

12 July 2018
Update: 13 July 2018

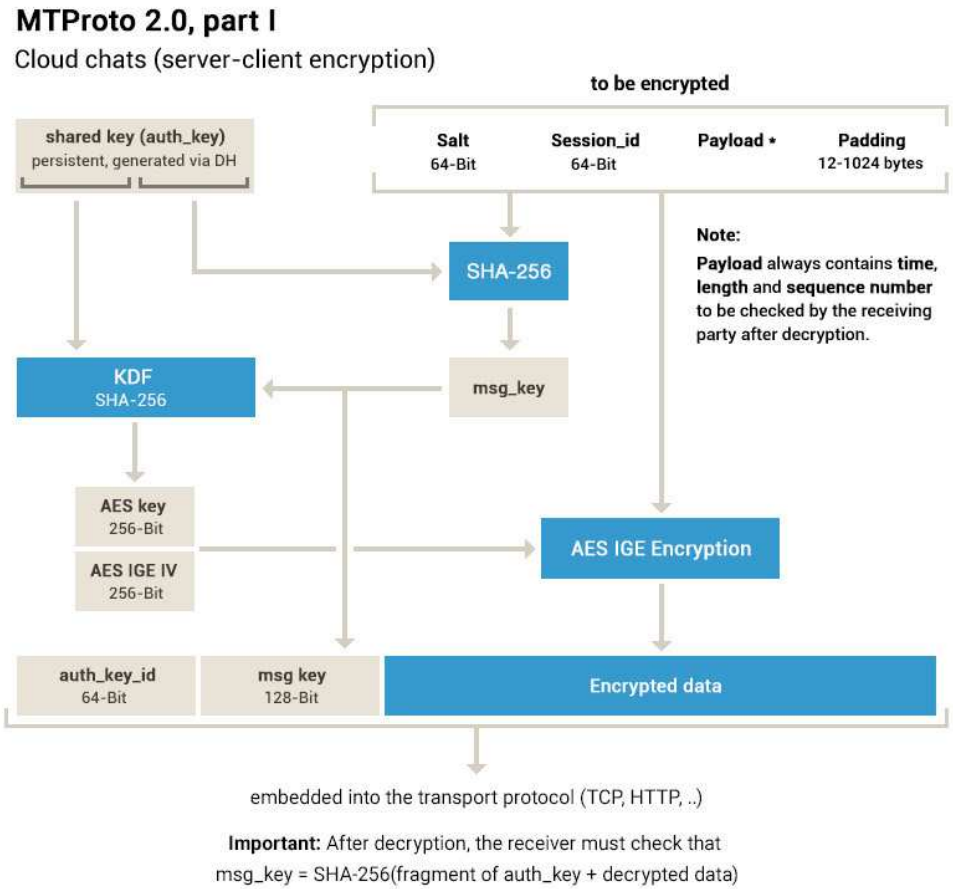
Agenda

- Auditing Cryptographic Protocols
- Auditing Cryptographic Implementations
- Own Protocols

Section 1

Auditing Cryptographic Protocols

Auditing Crypto: MTProto 2

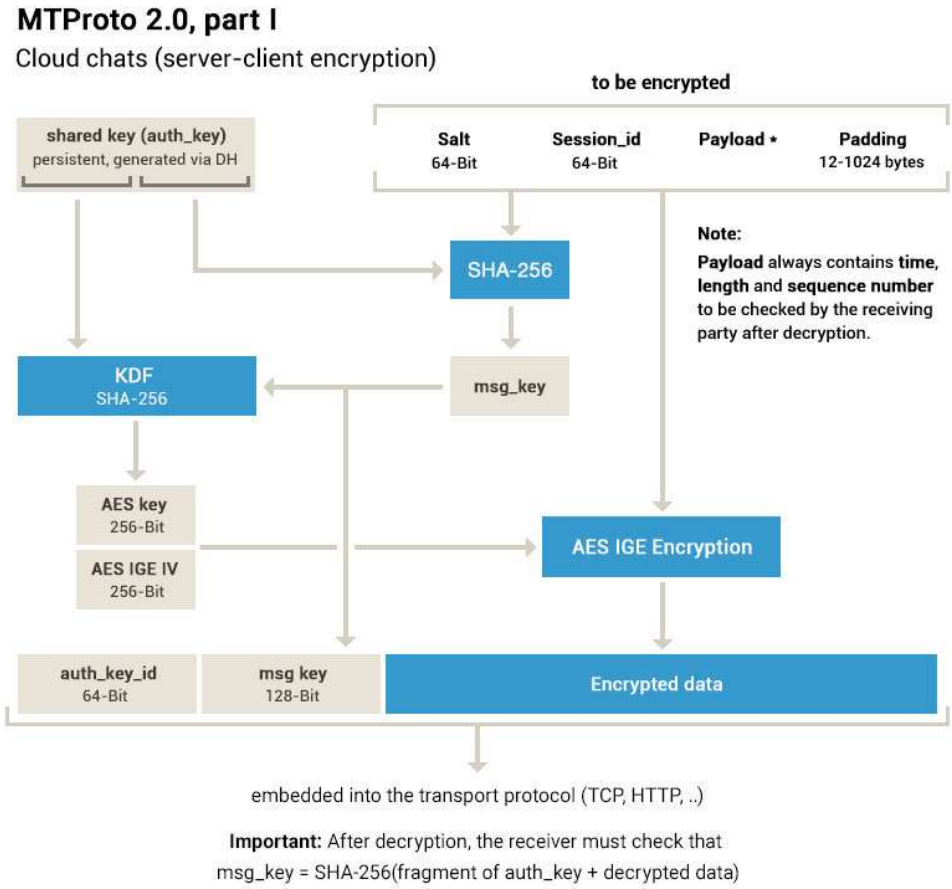


`https://core.telegram.org/mtproto`

Auditing Crypto: MTProto 2

Issues:

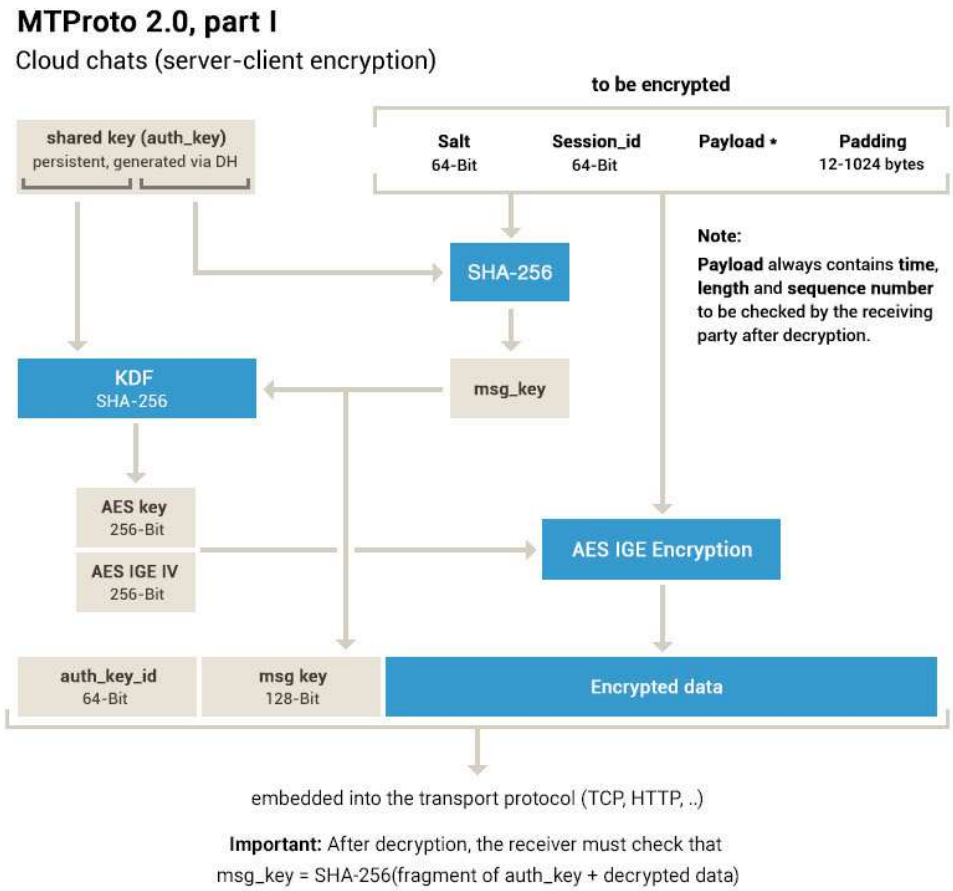
- Usage of single-keyed SHA-256 as MAC
- Weak KDF
- KDF is not collision-resistant: 4 key bytes of Auth-key unused
- Key ID is exchangeable
- ...



Auditing Crypto: MTProto 2

Usage of Weak Cryptographic Schemes

- Single-keyed SHA-256 is not an SUF-CMA-secure MAC
- $F(K \,\|\, M)$ is vulnerable to extension attacks



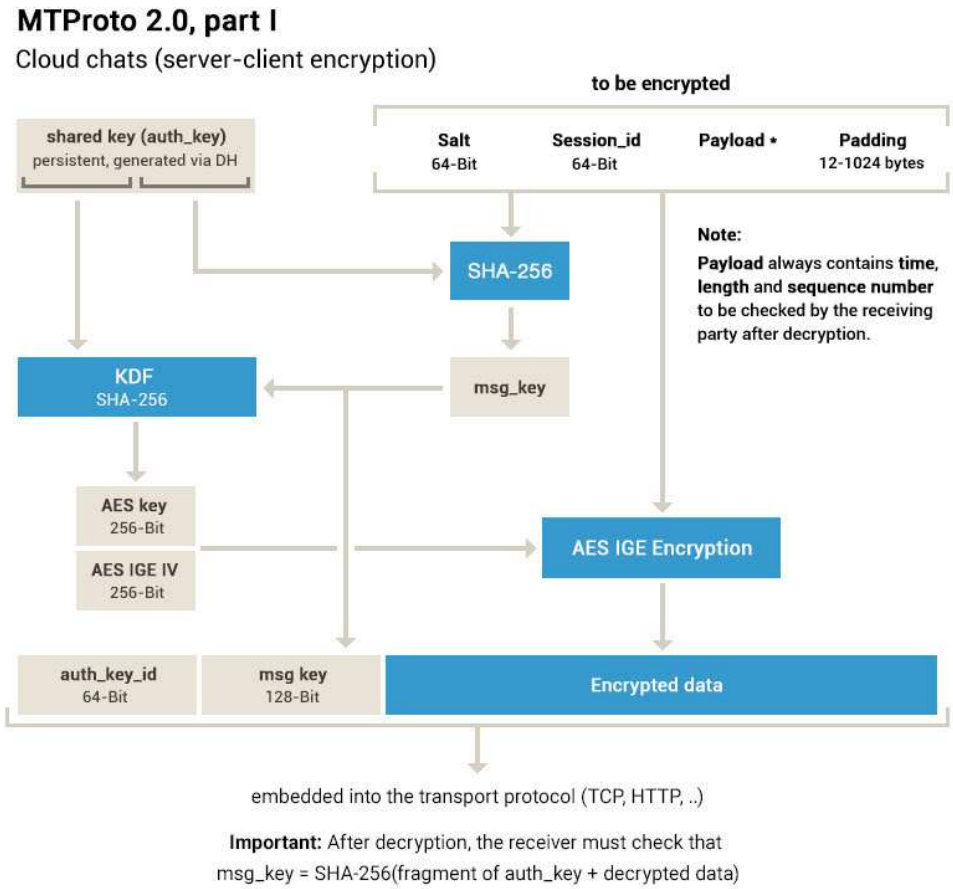

```
1 msg_key_large = sha256(self.auth_key[88: 88 + 32] + data + padding).digest()
```

https://github.com/pyrogram/pyrogram/blob/develop/pyrogram/session/session.py

Auditing Crypto: MTProto 2

Usage of Weak Cryptographic Schemes

- 2x SHA-256 is not a secure KDF
- Does not thwart offline adversaries

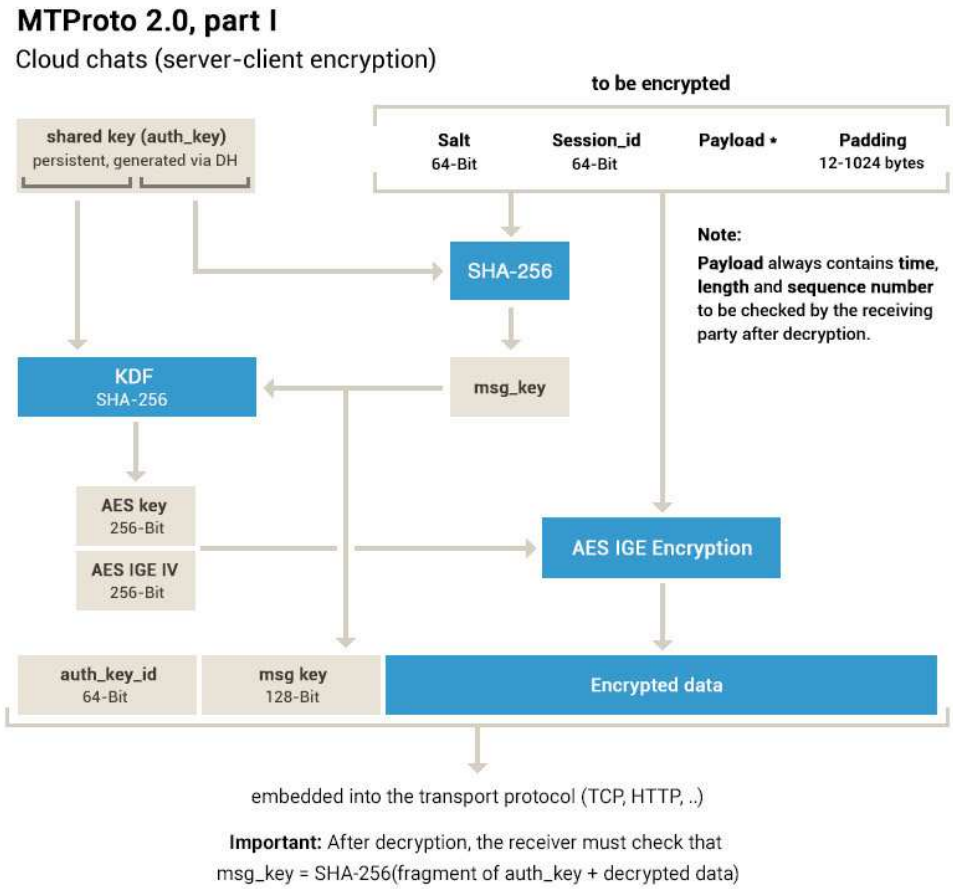



```
1 sha256_a = sha256(msg_key + auth_key[x: x + 36]).digest()
2 sha256_b = sha256(auth_key[x + 40:x + 76] + msg_key).digest()
3
4 aes_key = sha256_a[:8] + sha256_b[8:24] + sha256_a[24:32]
5 aes_iv = sha256_b[:8] + sha256_a[8:24] + sha256_b[24:32]
```

Auditing Crypto: MTProto 2

Usage of Weak Cryptographic Schemes

- KDF is not collision-resistant
- `auth_key[36: 40]` appear unused

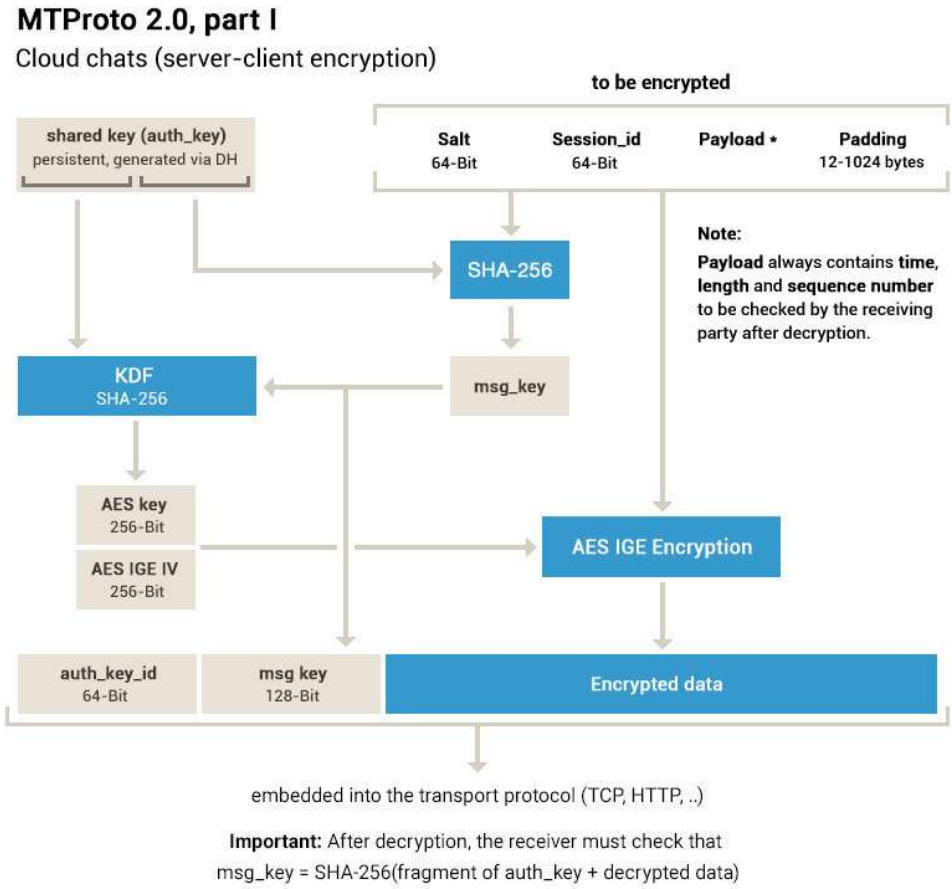



```
1 sha256_a = sha256(msg_key + auth_key[x: x + 36]).digest()
2 sha256_b = sha256(auth_key[x + 40:x + 76] + msg_key).digest()
3
4 aes_key = sha256_a[:8] + sha256_b[8:24] + sha256_a[24:32]
5 aes_iv = sha256_b[:8] + sha256_a[8:24] + sha256_b[24:32]
```

Auditing Crypto: MTProto 2

Replay attacks

- Auth key ID could be manipulatabled
- Old packet may be replayed



Section 2

Auditing Implementations

Auditing Implementations: Pyrogram

- Use of naive RSA encryption
- Use of weak random generation with `random`
- Use of naive padding

Pyrogram

Issue 1

Usage of Weak Cryptographic Schemes

- Use of naive RSA encryption

```
@classmethod
def encrypt(cls, data: bytes, fingerprint: int) -> bytes:
    return int.to_bytes(
        pow(
            int.from_bytes(data, "big"),
            cls.server_public_keys[fingerprint].e,
            cls.server_public_keys[fingerprint].m
        ),
        256,
        "big"
    )
```

https://github.com/pyrogram/pyrogram/blob/develop/pyrogram/crypto/rsa.py

Pyrogram

Issue 2

Usage of Weak Cryptographic Schemes

- Use of naive padding

```
1 padding = urandom(- (len(data) + len(sha)) % 255)
2 data_with_hash = sha + data + padding
3 encrypted_data = RSA.encrypt(data_with_hash, public_key_fingerprint)
```

https://github.com/pyrogram/pyrogram/blob/develop/pyrogram/session/auth.py

Pyrogram

Issue 3

Potential Side Channels

```
1  @classmethod
2  def ctr(cls, data: bytes, key: bytes, iv: bytearray, state: bytearray) -> bytes:
3      cipher = pyaes.AES(key)
4      ...
5
6      try:
7          iv[k] += 1
8          break
9      except ValueError:
10         iv[k] = 0
```

https://github.com/pyrogram/pyrogram/blob/develop/pyrogram/crypto/aes.py

Section 3

A Custom Protocol

A Custom Protocol for Instant Messaging

- AE scheme
- Primitives
- Decide for low-level protocol(s) (TCP/UDP/WebRTC etc.)

Stateful AE

- Level 1: Integrity and Privacy
- Level 2: + Packet Reordering
- Level 3: + Packet Replay
- Level 4: + Packet Loss

Setting

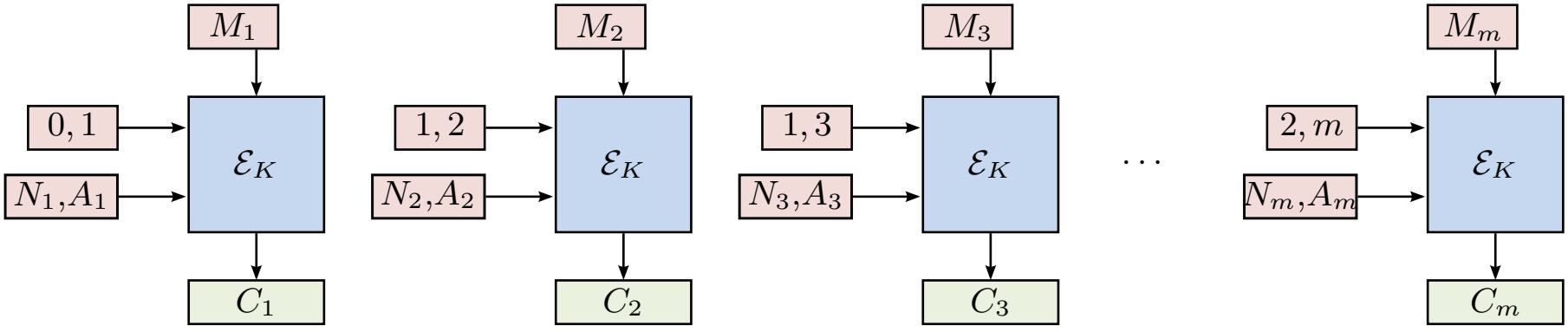
Environment:

- Probably: off-the-shelf 64-bit smartphones
- Messages: small messages
- Small associated data of up to 1kB
- Key(s) already exchanged

Security Goals:

Goal	Relevance	Level
Authentication of communication partners	essential	64-bit
Privacy of communication	essential	64-bit
Authenticity of communication	essential	64-bit
End-to-end encryption	essential	64-bit
Reorder, replay, and drop protection	essential	64-bit
Length hiding	essential	
Forward secrecy	essential	
Robustness against nonce misuse	optional	
Robustness against release of unverified plaintexts	optional	

Protocol Suggestion 1

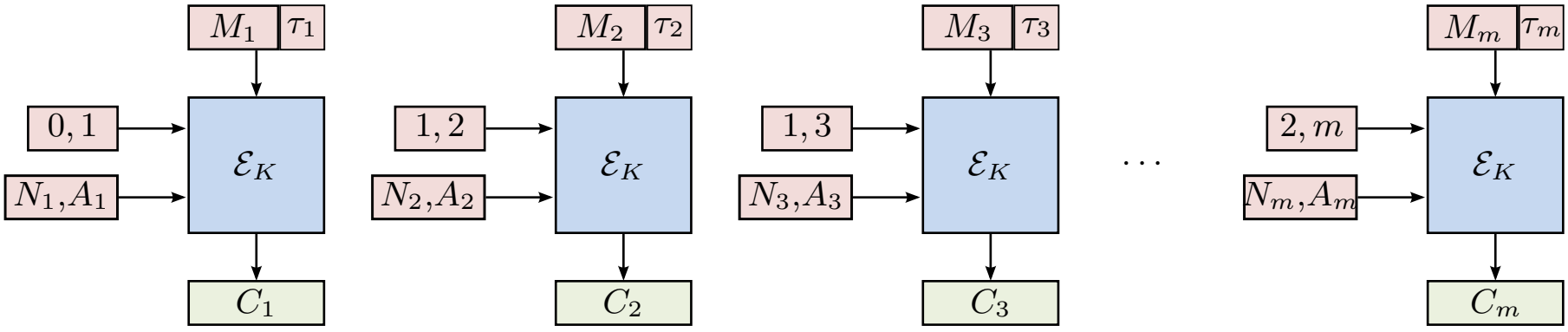


- sAE Scheme: STREAM-like
  - i = start/middle/end
  - j = message index in a sequence
- AE Scheme: SIV-like
  - misuse-resistant, messages are small and on-line not necessary
- Low-level protocols: TCP
  - Ensures order and proper packet transport

1: **function** $\mathcal{E}_K(i, j, N, A, M)$
2: $A' \leftarrow \textsc{EncodeA}(i, j, A)$
3: $C \leftarrow \mathcal{E}_K^{N,A'}(M)$
4: **return** C

1: **function** $\textsc{EncodeA}(i, j, A)$
2: **return** $\langle i \rangle \,\|\, \langle j \rangle \,\|\, A$

Protocol Suggestion 2



- sAE Scheme: STREAM-like
  - τ_i = per-message stretch (length-hiding)
- AE Scheme: HCTR with AES-256 and PHash-AES-256
  - Tweakable STPRP with Encode-then-Encipher $\Longrightarrow$ Robust AE
  - Messages are small and on-line not necessary
  - Standardized single primitive
- Low-level protocols: TCP
  - Ensures order and proper packet transport

function $\mathcal{E}_K(i, j, \tau, N, A, M)$
  $V \leftarrow \textsc{EncodeA}(i, j, \tau, N, A)$
  $M' \leftarrow \textsc{EncodeM}(M, \tau)$
  $C \leftarrow \mathcal{E}_K^V(M')$
  return C

function $\textsc{EncodeA}(i, j, \tau, N, A)$
  return $\langle i \rangle \,\|\, \langle j \rangle \,\|\, \langle \tau \rangle \,\|\, N \,\|\, A$

function $\textsc{EncodeM}(M, \tau)$
  return $M \,\|\, \langle \tau \rangle \,\|\, 0^{\tau}$

Questions?