Key Establishment in the Space Environment

Implementing MLS inside BPSec

Benjamin Dowling benjamin.dowling@kcl.ac.uk
Britta Hale britta.hale@nps.edu
Xisen Tian xisen.tian1@nps.edu
Bhagya Wimalasiri b.m.wimalasiri@sheffield.ac.uk

March 17, 2025

Space & Interplanetary Communication

The terrestrial internet as we know it operates under certain assumptions.

Terrestrial Communication	Interplanetary Communication
Continuous availability of links and devices	Discontinuous availability
Low latency and fast RTT, e.g., RTT between two points is approximately 100 to 300 ms	Latency
Low packet loss rate	Packet loss, attenuation, jamming potential
Bi-directional data links	Data links can be simplex
Ubiquitous client-server model	Handshakes are unreliable/costly

Session establishment handshakes can be unreliable
packet loss incurs further delays in key establishment

Bundle Protocol

- Bundle Protocol is (roughly) comparable to IP
- BPSec integrates confidentiality and integrity guarantees (viewed as roughly similar to IPSec, but application layer)

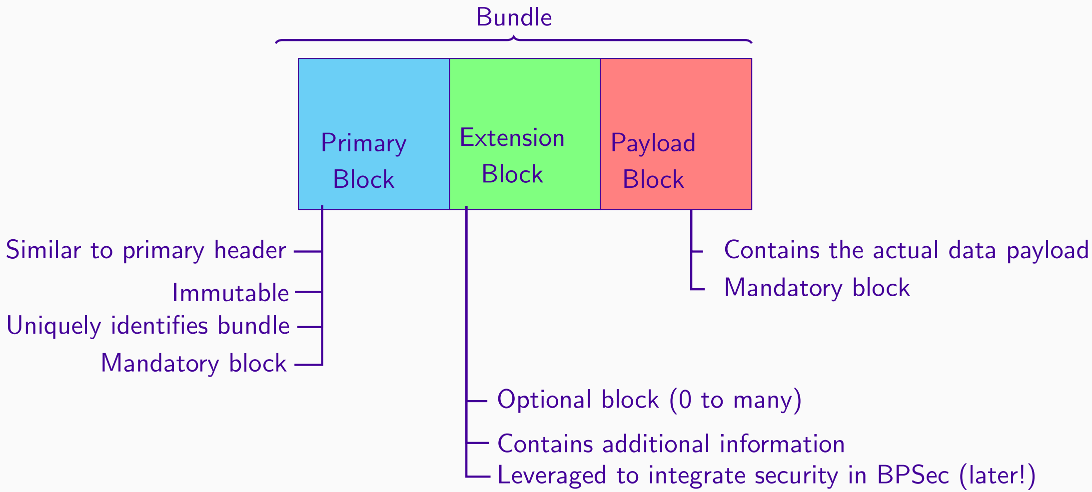
Application	HTTP/TLS
Transport	TCP/UDP
Network	IP/IPSec
Physical/Datalink	Ethernet

Internet Protocols

Application	BPSec
Transport	BP
Network	LTP
Physical/Datalink	Radio

Deepspace Protocols

Generic Bundle Message Format



BPSec: Bundle Integrity/Confidentiality Block

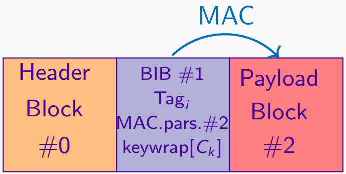
For a bundle block $\overrightarrow{\mathtt{M}}_i$, a BIB/BCB can be constructed in two ways:

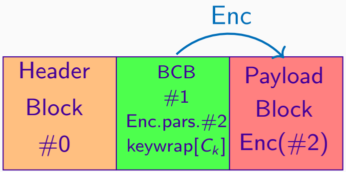
Key wrapped keys BIB:

1. $\mathsf{Tag}_i \leftarrow \mathsf{MAC.Auth}(k_{\mathsf{BIB}}, \overrightarrow{\mathtt{M}}_i)$
2. $C_k \leftarrow \mathsf{KeyWrap.Enc}(k_{\mathsf{KW}}, k_{\mathsf{BIB}})$
3. $k_{\mathsf{BIB}} \leftarrow \mathsf{KeyWrap.Dec}(k_{\mathsf{KW}}, C_k)$
4. $\{0,1\} \leftarrow \mathsf{MAC.Vfy}(k_{\mathsf{BIB}}, \mathsf{Tag}_i)$

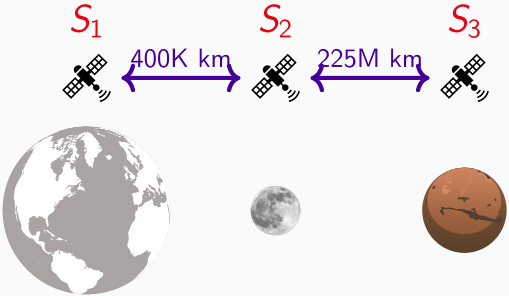
Key wrapped keys BCB:

1. $C_i \leftarrow \mathsf{Enc}(k_{\mathsf{BCB}}, \overrightarrow{\mathtt{M}}_i)$
2. $C_k \leftarrow \mathsf{KeyWrap.Enc}(k_{\mathsf{KW}}, k_{\mathsf{BCB}})$
3. $k_{\mathsf{BCB}} \leftarrow \mathsf{KeyWrap.Dec}(k_{\mathsf{KW}}, C_k)$
4. $\overrightarrow{\mathtt{M}}_i \leftarrow \mathsf{Dec}(k_{\mathsf{BCB}}, C_i)$



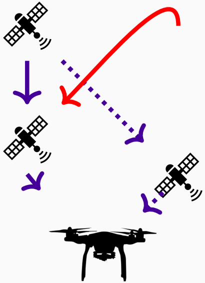


Bundle Protocol Security: Assumptions and Goals



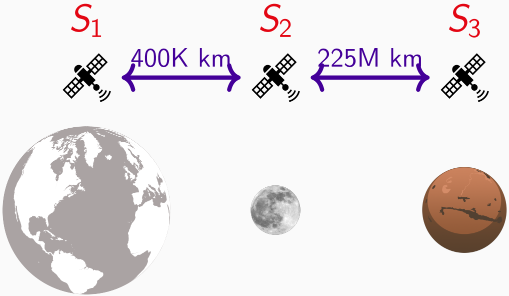
- **Goal**: Non-interactive application layer secure channel
  - Extreme distances and intermittent availability discourage handshakes/key negotiations
  - Assumes pre-established keys/symmetric keys and identities sharing keys known
- `https://eprint.iacr.org/2025/023.pdf`

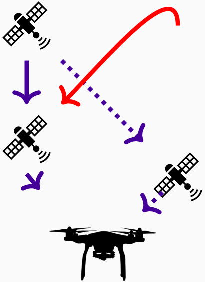
Bundle Protocol Security: Assumptions and Goals



- **Goal**: Non-interactive application layer secure channel
  - Want authorized intermediate nodes to be able to **process** messages on behalf of destination
    - authorized $\Longrightarrow$ share secret keys
    - thus, can consider groups of nodes
  - `https://eprint.iacr.org/2025/023.pdf`

Bundle Protocol Security: Assumptions and Goals





- Problem: Symmetric keys potentially not updated for long periods of time
    - Goal: Need to consider and handle usage limits
    - Goal: Multiple groups (with distinct keys) processing messages without re-encryption?
- Solution (?): **Key wrapping**: Two birds, one stone?

Key Wrapping Limitations

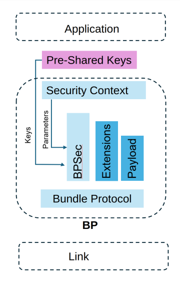
Key wrapping merely prolongs, does not solve the issue:

```
5.5.  Limits on Key Usage

   There are cryptographic limits on the amount of plaintext which can
   be safely encrypted under a given set of keys.  [AEAD-LIMITS]
   provides an analysis of these limits under the assumption that the
   underlying primitive (AES or ChaCha20) has no weaknesses.
   Implementations SHOULD do a key update as described in Section 4.6.3
   prior to reaching these limits.

   For AES-GCM, up to 2^24.5 full-size records (about 24 million) may be
   encrypted on a given connection while keeping a safety margin of
   approximately 2^-57 for Authenticated Encryption (AE) security.  For
   ChaCha20/Poly1305, the record sequence number would wrap before the
   safety limit is reached.

Rescorla                     Standards Track                   [Page 84]
RFC 8446                           TLS                       August 2018
```



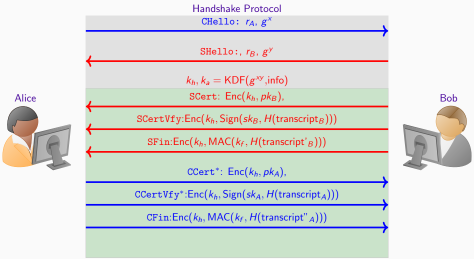
Also, what happens when keys are compromised, due to bad usage, bad handling?

How can we refresh these keys?

Proposed Solution: Key Exchange

Key exchange protocols enable two (or more) parties to generate **fresh, independent, shared, session keys**

Most famous example of a key exchange protocol is TLS:

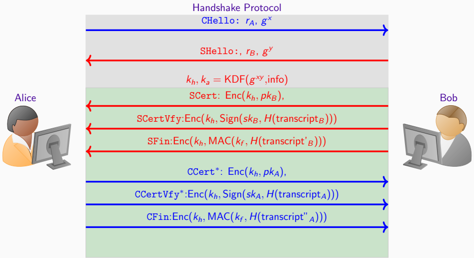


- **Forward Secrecy**: Compromise of long-term keys don't impact previous communciation security.
- **Provides Key Update**: Alice and Bob can ratchet forward their secrets $\mathtt{secret}_{n+1} = \mathsf{HKDF\text{-}Expand}(\mathtt{secret}_n, \mathtt{traffic\ update}, \emptyset, \mathsf{H\_len})$.

But, TLS **isn't** a good choice for the asynchronous setting.

Proposed Solution: Key Exchange Handshakes

TLS's handshake requires synchronous communication:



- **No Post-Compromise Security**: Future Confidentiality lost once keys compromised.
- **High Risk in Low Latency**: 0-RTT mode has known issues (see RFC 8446)
- **Assumes Synchronous Communication**: Alice and Bob both must be interactive to perform handshake.

But, once we establish keys, we could use Key Update to establish fresh keys, right?

TLS Key Update

TLS's key update mechanism does not refresh randomness

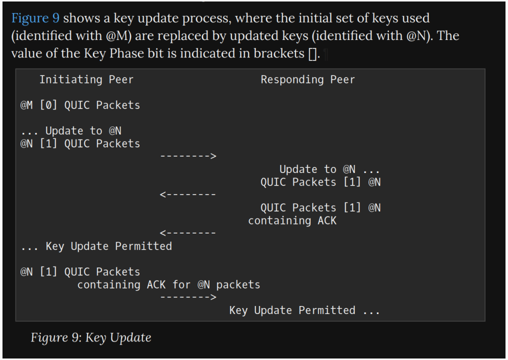


Figure 9 shows a key update process, where the initial set of keys used (identified with @M) are replaced by updated keys (identified with @N). The value of the Key Phase bit is indicated in brackets [].

```
   Initiating Peer                    Responding Peer

@M [0] QUIC Packets

... Update to @N
@N [1] QUIC Packets
                      -------->
                                         Update to @N ...
                                      QUIC Packets [1] @N
                      <--------
                                      QUIC Packets [1] @N
                                    containing ACK
                      <--------
... Key Update Permitted

@N [1] QUIC Packets
         containing ACK for @N packets
                      -------->
                                 Key Update Permitted ...
```

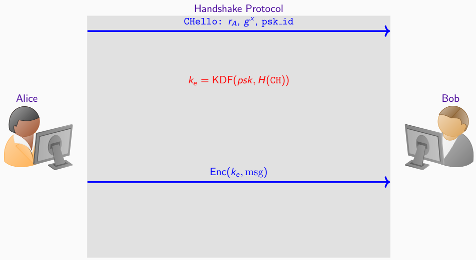
Figure 9: Key Update

- **Provides Key Update**: Alice and Bob can ratchet forward their secrets $\texttt{secret}_{n+1} = \text{HKDF-Expand}(\texttt{secret}_n, \texttt{traffic update}, \emptyset, \text{H\_len})$.
- **Forward Secrecy**: Compromise of previous secrets don't impact previous communication security.
- **No Post-Compromise Security**: Future Confidentiality lost once keys compromised.

But TLS has 0RTT, right?

TLS 0RTT

One potential solution proposal that has been floated to support **asynchronous communication** is 0RTT.



However:

- *TLS does not provide inherent replay protection for 0-RTT data.*
  - very much leaves things to application/implementor...
- **Loses Forward Secrecy**: Once the PSK is leaked, all messages using PSK are known.

Which led us to the question...

...Is key exchange the right protocol here?

Support for asynchronicity, recovery from packet loss, achieving PFS and PCS implies an demand signal for **Continuous Key Agreement (CKA)**

- Key exchange protocols are not CKAs – KEX are **session-based** protocols.
- The analysis of TLS focuses on the **per-session guarantees.** Their composition is also analysed, but these guarantees are broadly session-oriented and TLS is not analysed for security in the "spread out" handshake sense of a CKA. Compromise models are different for CKAs and KEX!

...Why not use an existing standardized CKA and use it to establish keys for BPSec? **RFC9420, Messaging Layer Security**.

(Bonus: this only adds key establishment messages the rest of BPSec can be used as-is)**

Why MLS?

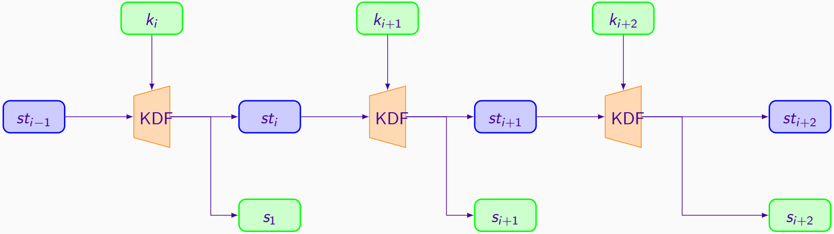
Goals – Setting

- **secure continuous key agreement protocol**
- **Asynchronous**, key updates can be unidirectional
- **Sessions are long-lasting**, can evolve updates over even years
- Low bandwidth & complexity
- (Optional) Can handle groups of devices and dynamic group membership

Goals – Security

- E2E Authenticity and Confidentiality
- Adversarial Model
    - MiTM: Network, Delivery Service compromised
    - Corruption: Can heal from state compromise
    - Adaptive and Active Adversary
- **Forward and Post-Compromise Security**

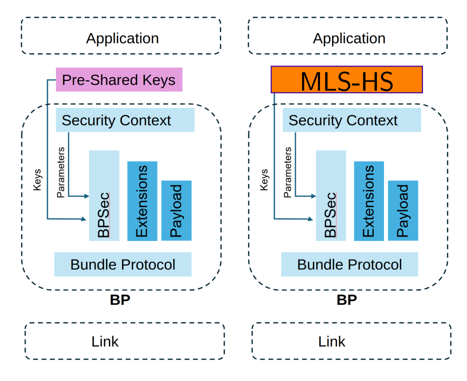
MLS: Epochs



- Keys k found at the root of a Ratchet Tree are called "commit secrets" and are used to generate keys for encrypting messages.
- Members can "Update" the tree, generate a new commit secret k_{i+1}, which advances the group to the next epoch $i + 1$
- Commit secrets k_i are combined with secrets generated from the previous epoch to achieve post-compromise security

(The key schedule has more to it than this... this is an overly simplistic view but gives a sense of CKAs)

Proposal: BPSec-MLS



- Addition of key agreement messages to BPSec
- Achieve PCS and Asynchronous Updates
- be resilient against delays from session setup

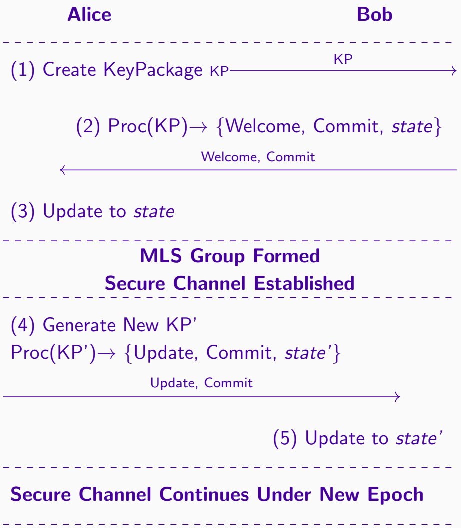
PoC and Benchmarking

PoC & Benchmarking Specs

- Testbed: 11th Gen Intel(R) Core(TM) i7-11370H @3.30GHz with 16GB onboard RAM
- PoC developed using quic-go (for TLS) and cisco/go-mls (for MLS) libraries

Security Function	TLS	MLS
Handshake/Join Group	1.7 ms	2 ms
PCS Key Update	N/A	1.7 ms

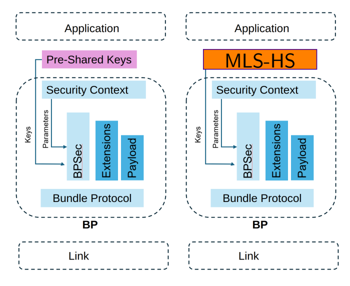
Table 1: *Performance Comparison between TLS and MLS Cryptographic Operations*



Summary

Core Idea:

- Add MLS key agreement within BPSec.
- Otherwise BPSec remains the same.
- **Is the WG interested in this approach?**



Next Steps:

- If WG interest in this approach, we propose BPSec extension

Alice **Bob**

(1) Create KeyPackage KP $\xrightarrow{\text{KP}}$

(2) Proc(KP)$\rightarrow$ {Welcome, Commit, *state*}

$\xleftarrow{\text{Welcome, Commit}}$

(3) Update to *state*

MLS Group Formed

Secure Channel Established

(4) Generate New KP'

Proc(KP')$\rightarrow$ {Update, Commit, *state*'}

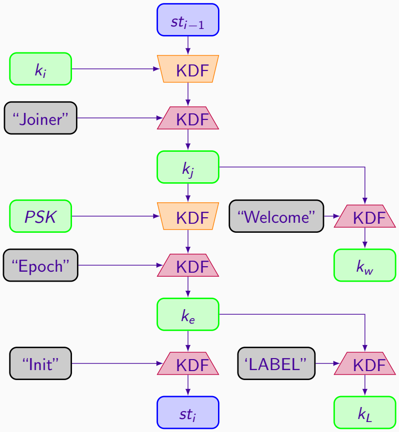
$\xrightarrow{\text{Update, Commit}}$

(5) Update to *state*'

Secure Channel Continues Under New Epoch

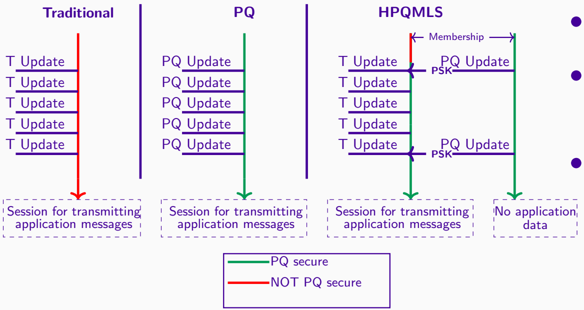
MLS Key Schedule

However, MLS isn't quite this simple - we combine the updated i-th commit secret k_i with the previous "secret state" st_{i-1} to generate keying material for various purposes.



k_L	Label
sender_data_secret	"sender data"
encryption_secret	"encryption"
exporter_secret	"exporter"
authentication_secret	"authentication"
external_secret	"external"
confirmation_key	"confirm"
membership_key	"membership"
resumption_secret	"resumption"

One Step Further? Post Quantum MLS



- PQ can be costly
- Hybrid strikes a balance between security & efficiency
- Potential amortized design
  - Use two sessions – T and PQ
  - combine via exported PSKs.
  - **Flexibility**: PQ updates are tunable