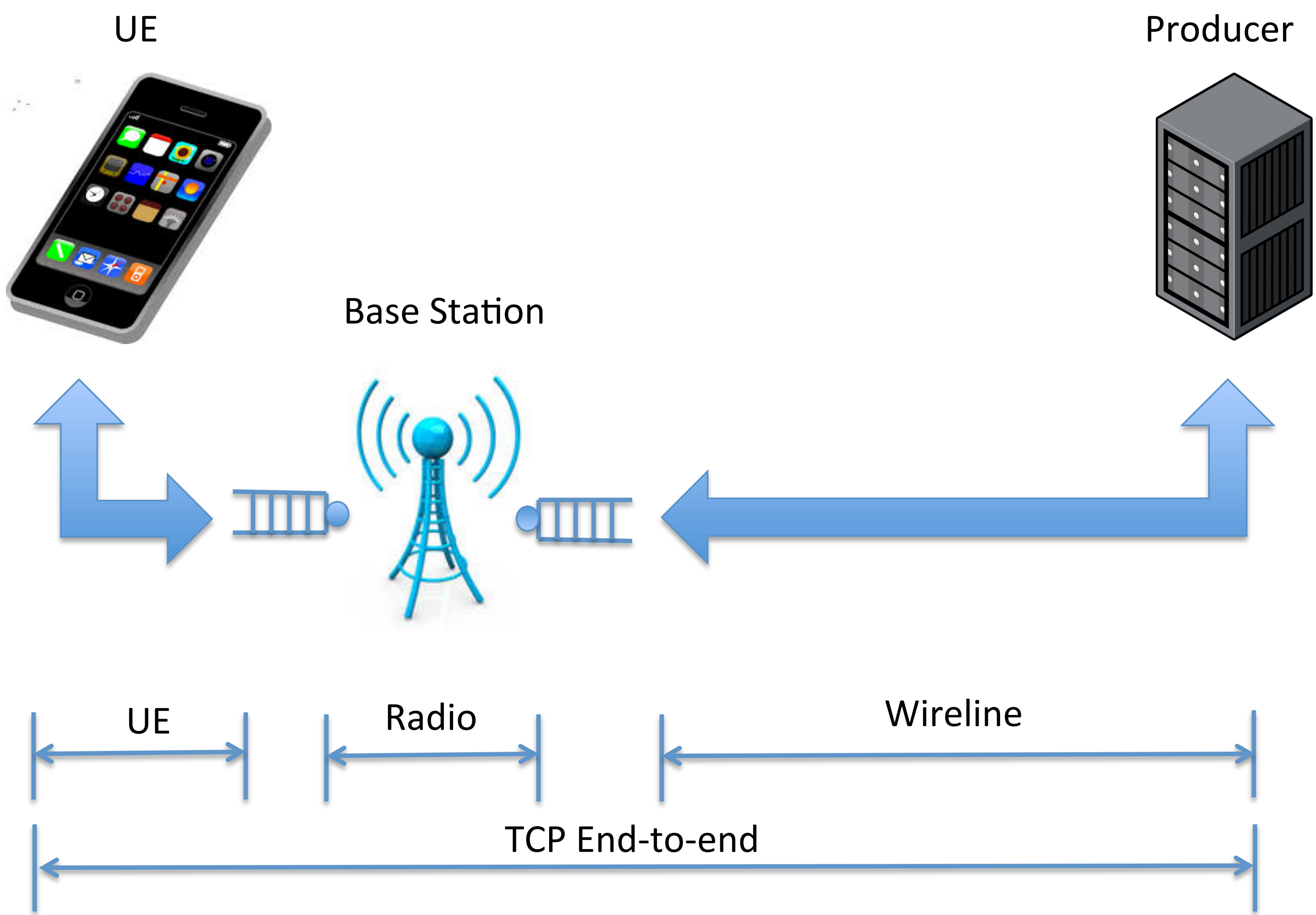
Secure Transport Offload with Encrypted PEPs

Christopher A. Wood
UCI and PARC

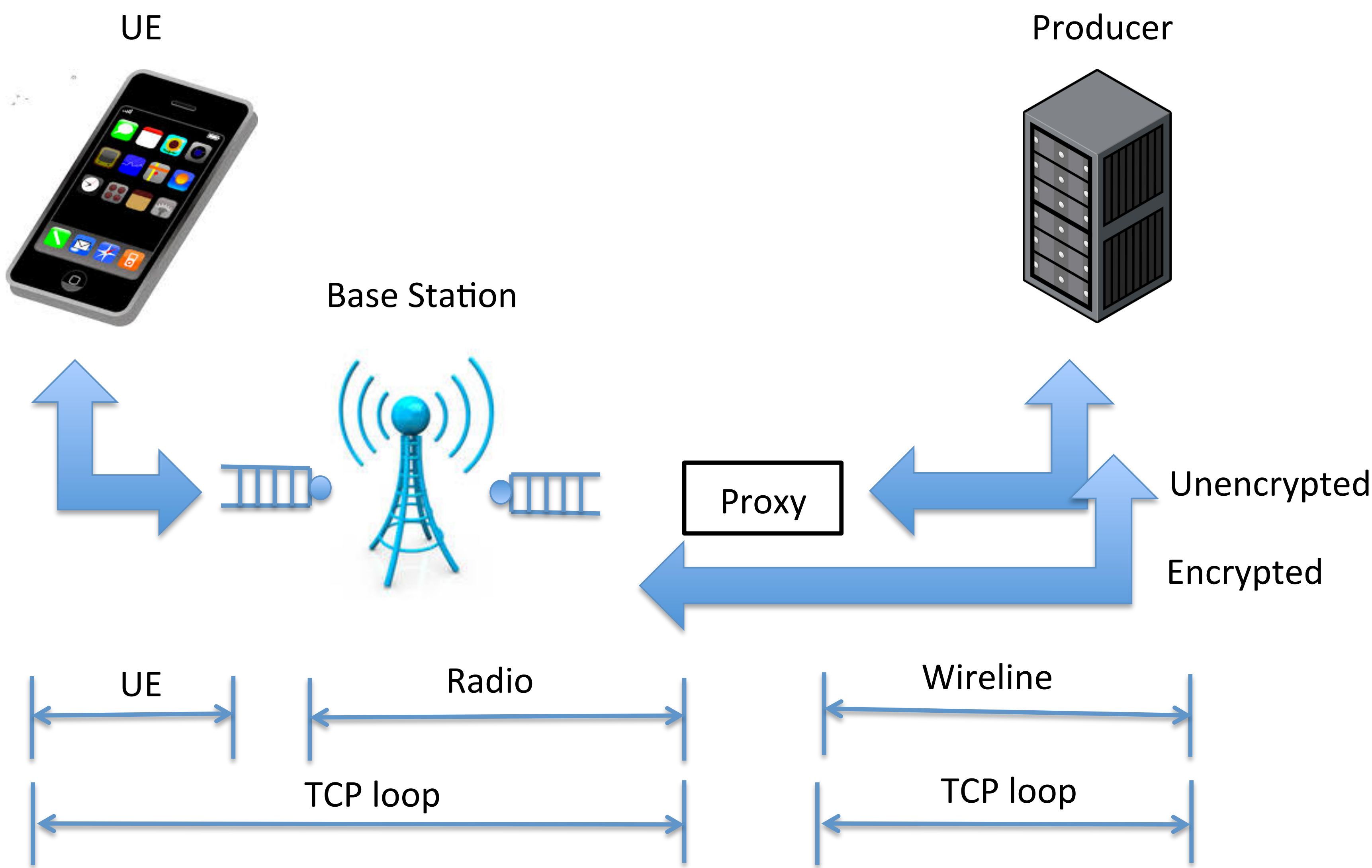
ICNRG Interim Meeting – IETF 96 – Berlin

July 17, 2016

LTE Setup



LTE Setup with PEPs

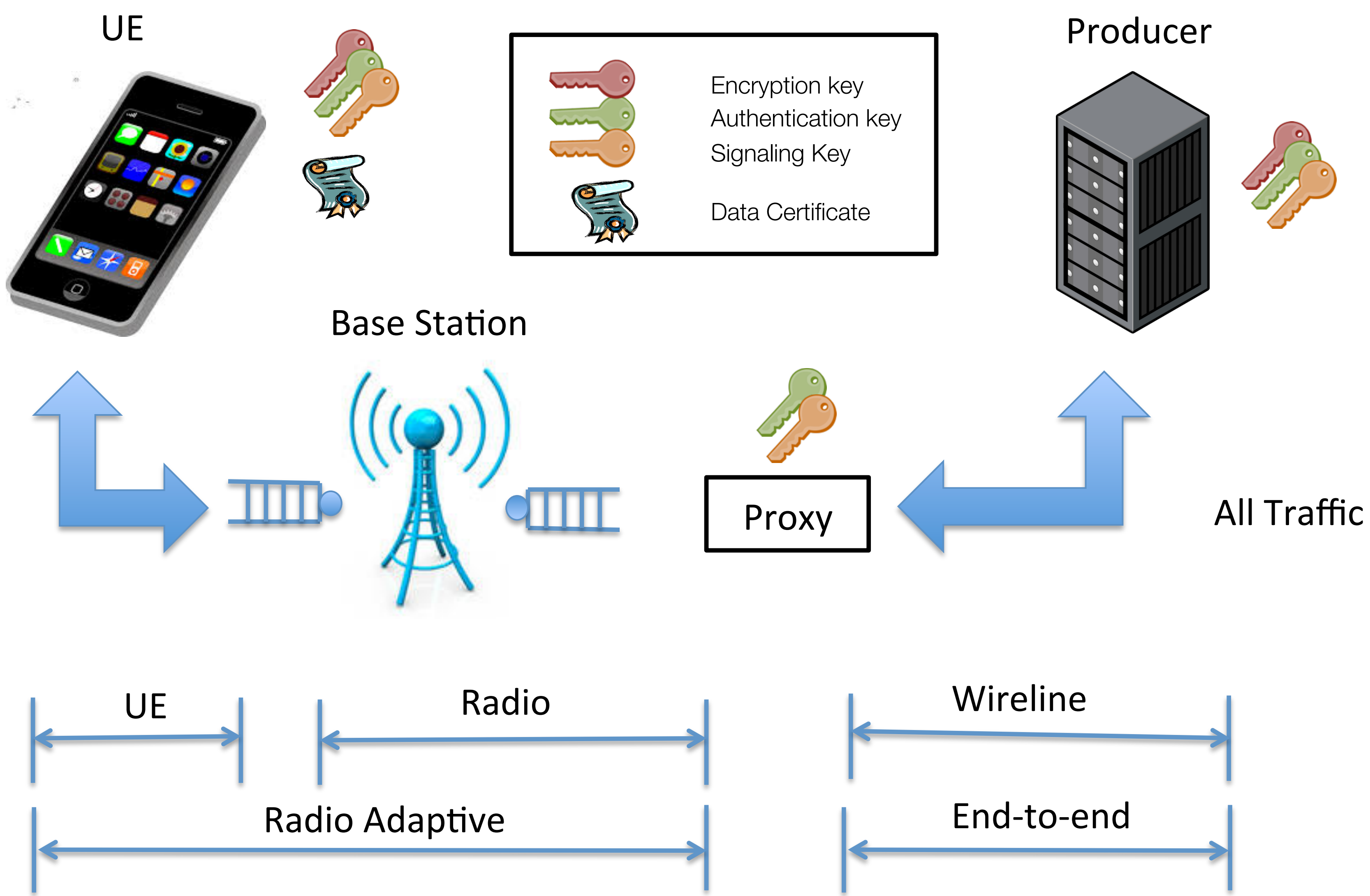


Problems

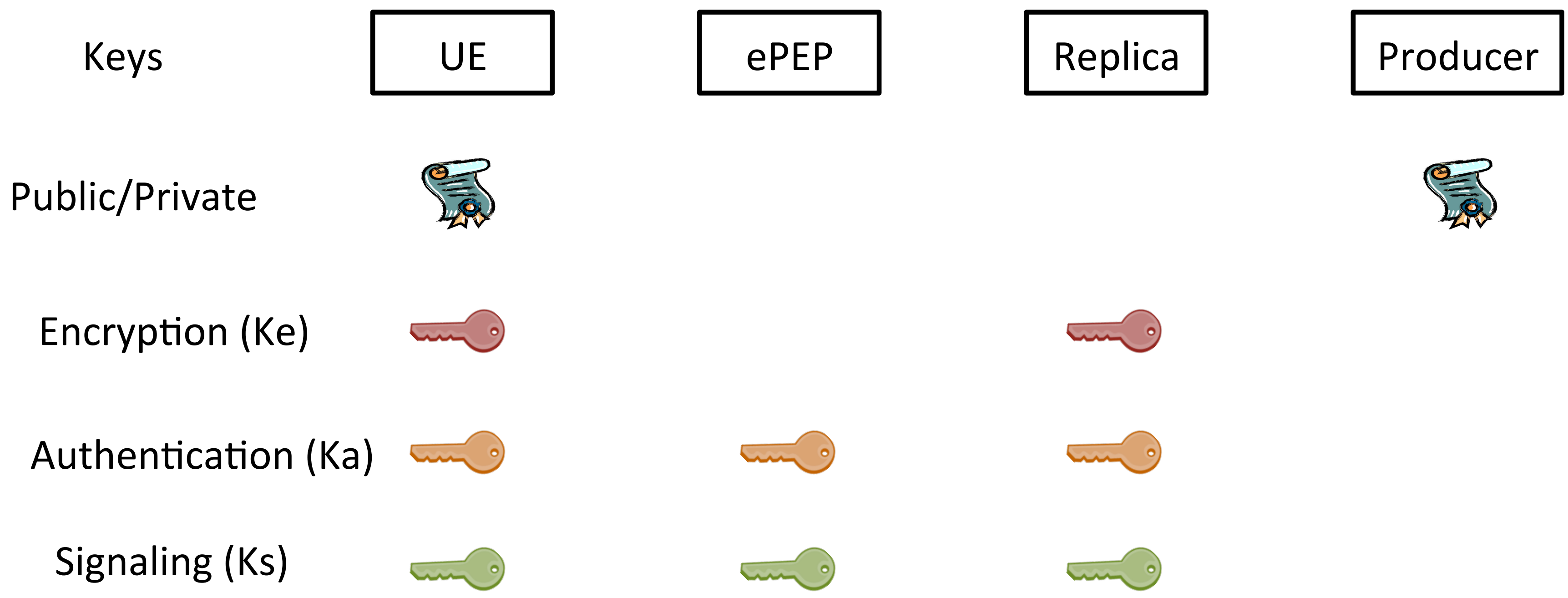
- Traditional PEPs require the ability to peek into TCP packets to operate [1]
- End-to-end encryption prevents these PEPs

[1] Caini, Carlo, Rosario Firrincieli, and Daniele Lacamera. "PEPsal: a Performance Enhancing Proxy for TCP satellite connections." IEEE Aerospace and Electronic Systems Magazine 22.8 (2007): 7-16.

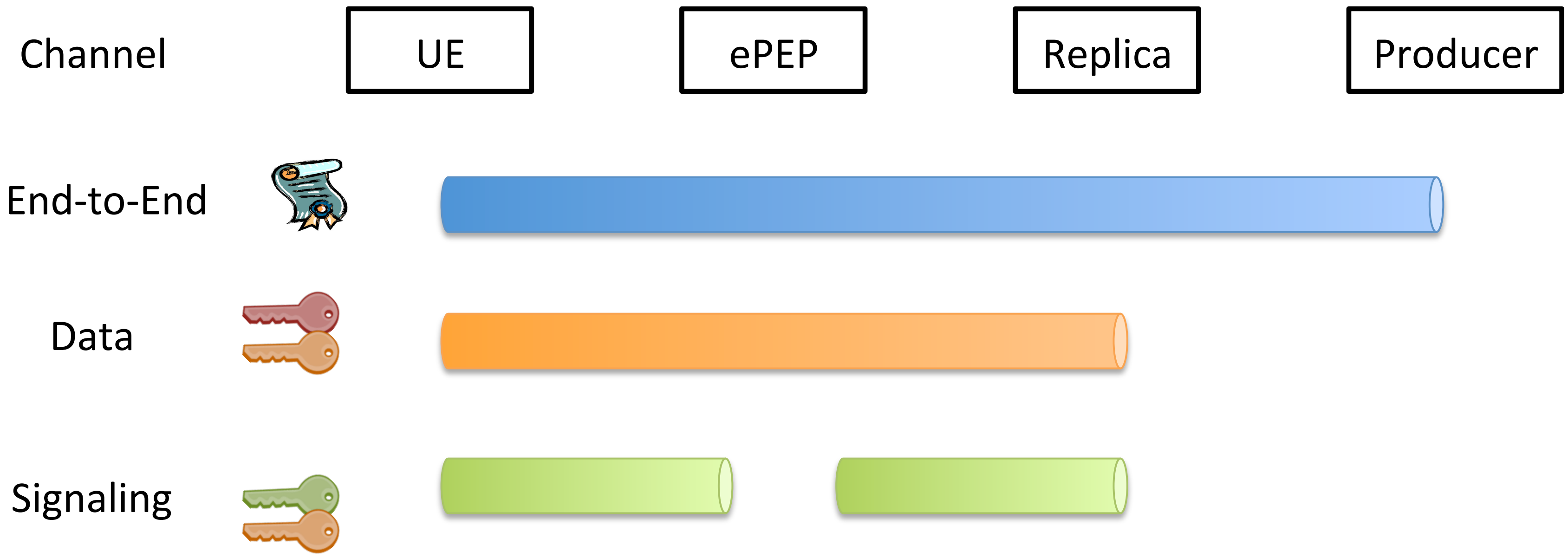
CCNx ePEP Setup



Key Ownership



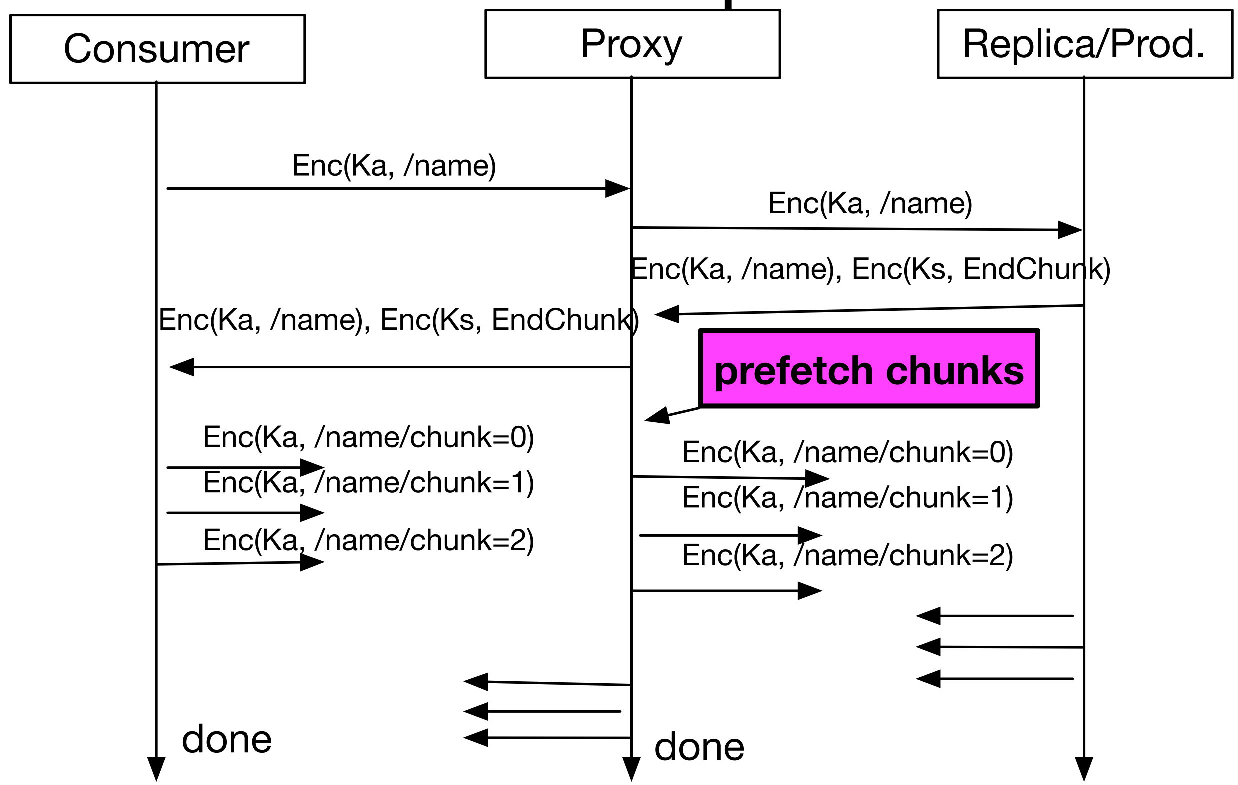
Key Scope



ePEP Transfer Protocols

- ePEPs operate over sessions
- Data “transactions” exist within sessions
- Each transaction can be driven by one of two protocols
  - Chunk-based transfer protocol
    - A single transaction is for a specific number of chunks
    - ePEP prefetches all chunks on behalf of UE
  - Manifest-based transfer protocol
    - A single transaction is for a manifest tree
    - ePEP resolves the manifest tree and replies to UE interests

Chunk-Based Transport Protocol



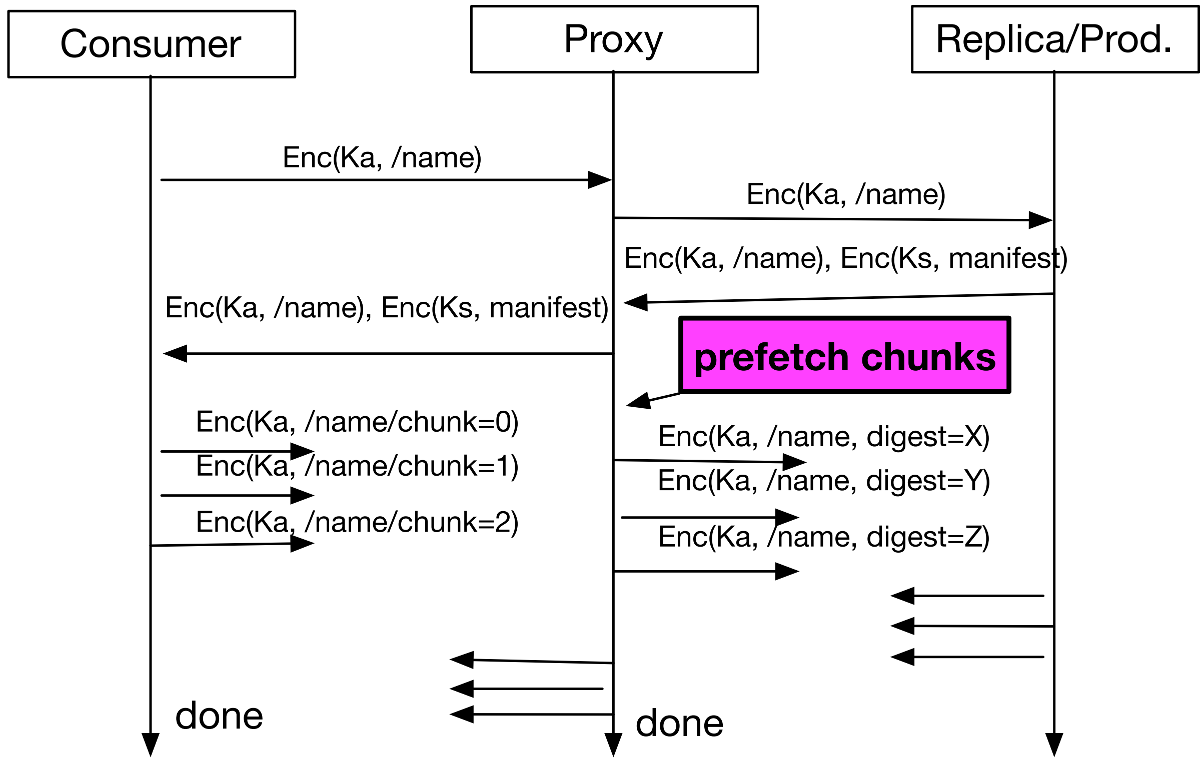
Proxy behavior:

- Prefetch chunks as a byte stream and return them to the consumer

Consumer behavior:

- Ask for each chunk
- Verify each chunk signature before consumption

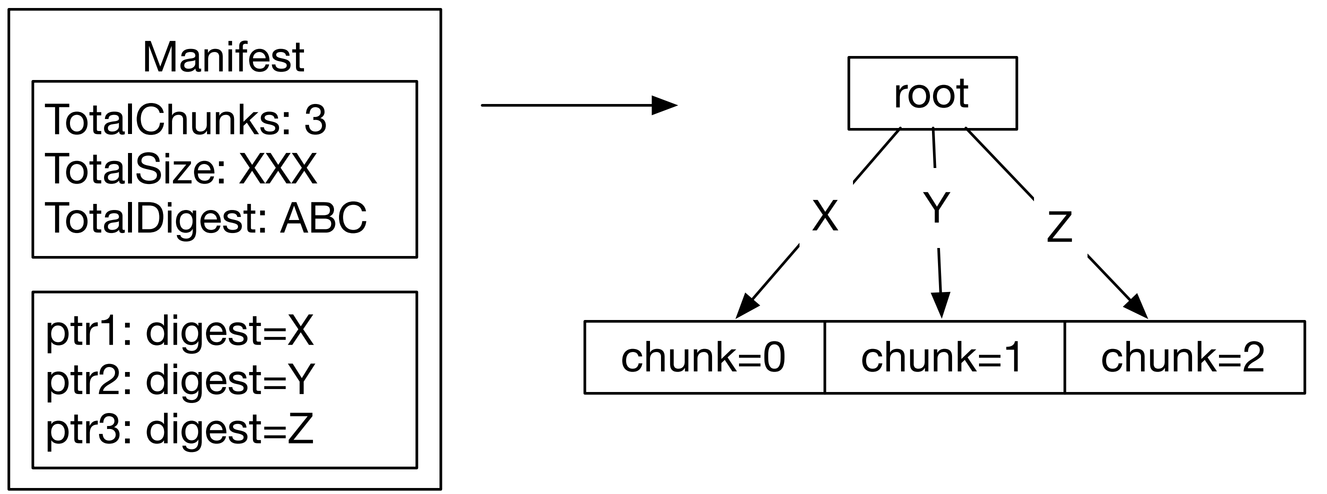
Manifest-Based Transport Protocol



Proxy behavior:
- Recursively prefetch the transport manifest
- Verify each manifest node with Ka
- Only respond to consumer interests for chunks (**leafs**) when they are located
  (this ensures that only application data is sent over the air)

Consumer behavior:
- Fetch each chunk and verify with Ka
- Verify signature on root manifest
- Verify total data hash upon completion

Chunk-to-Leaf Mapping



Key Distribution

1. UE and Producer create session U-P
2. UE and Producer derive Ke, Ks, and Ka from the traffic secret
3. UE and Replica create a session U-R
4. UE transfers Ks, Ka to Replica over U-R

Key Derivation

- For each transaction with ID i
  - Derive K_e^i, K_a^i, K_s^i from K_e, K_a, K_s using a KDF (RFC 5869)

$$K_j = KDF(K_e \text{ XOR } i)$$

Control Protocol

- Update transaction keys:
  - Triggers:
    - UE issues control interest to Replica with UPDATE_KEYS message
  - Behavior:
    - All keys are evolved using the KDF
- Open transaction:
  - Triggers:
    - UE issues new interest with corresponding random transaction ID
  - Behavior:
    - Replica and Producer derive transaction keys and respond with ACK
- Close transaction (and drop keys)
  - Triggers:
    - UE issues control interest to Replica with CLOSE_TX message and transaction ID
  - Behavior:
    - Drop transaction keys

Control Protocol (continued)

- Drop in-memory manifests:
  - Triggers:
    - UE interests include cumulative chunk number and bitmap of received chunks
  - Behavior:
    - Replica drops in-memory manifests and all sub-trees
- Drop session:
  - Triggers:
    - UE issues control interest to Replica and Producer with CLOSE_SESSION message
  - Behavior:
    - Replica and producer drop session keys
- Cancel a transaction:
  - Triggers:
    - UE issues control interest to Replica and Producer with CANCEL_TX message and transaction ID
  - Behavior:
    - Replica drops all transaction keys

Future Directions

- Modify UE and Replica channel to minimize the number of interests sent

Current Status

- Session generation (via CCNxKE) done
- Transport protocol implementation in progress
- Transaction management implementation in progress