Logging System for Long-lifetime Data Validation

Yingdi Yu
UCLA

Lifetime of data vs. signing key

- Lifetime of a data packet
  - depends on data usage
  - may exist for a long time
  - even forever

Lifetime of data vs. signing key

- Lifetime of a data packet
  - depends on data usage
  - may exist for a long time
  - even forever
- Lifetime of a signing key
  - must be limited

How to maintain long-lived data

How to maintain long-lived data

- Re-sign data with a new key
  - maintenance is complicated
    - key rollover
    - publishing re-signed data

How to maintain long-lived data

- Re-sign data with a new key
  - maintenance is complicated
    - key rollover
    - publishing re-signed data
- Can we sign data once and leave it alone?
  - post-fact validation
    - validate data with an expired key?

Post-Fact Validation

Post-Fact Validation

- Key was valid at the moment of signing
  - though it is invalid now

Post-Fact Validation

- Key was valid at the moment of signing
  - though it is invalid now
- Check if the signature was generated during the valid period of the key

Post-Fact Validation

- Key was valid at the moment of signing
  - though it is invalid now
- Check if the signature was generated during the valid period of the key
- Can we have a time machine to go back?
  - a logging system may help!

What to log?

- Assume we have a honest logger

What to log?

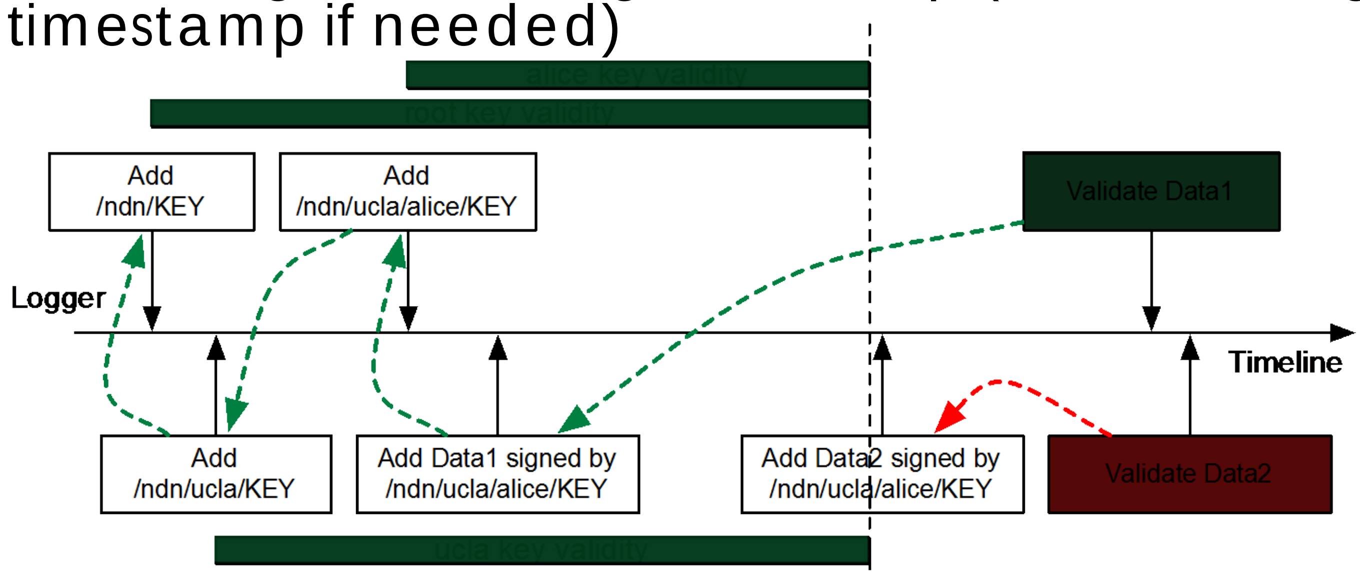
- Assume we have a honest logger
- Given a long-lived data
  - data name: retrieve data when necessary
  - data digest: integrity checking
  - signing timestamp

What to log?

- Assume we have a honest logger
- Given a long-lived data
  - data name: retrieve data when necessary
  - data digest: integrity checking
  - signing timestamp
- But also signing key
  - name, digest, inserting timestamp (and revoking timestamp if needed)

What to log?

- Assume we have a honest logger
- Given a long-lived data
  - data name: retrieve data when necessary
  - data digest: integrity checking
  - signing timestamp
- But also signing key
  - name, digest, inserting timestamp (and revoking timestamp if needed)



Secure logger

Secure logger

- A trusted third party?
  - not every one will trust the same third party
  - no entity lasts forever

Secure logger

- A trusted third party?
  - not every one will trust the same third party
  - no entity lasts forever
- Publicly auditable logger
  - anyone can audit the logger
    - data signers, data consumers, certificate issuers, independent third parties, …

Secure logger

- A trusted third party?
  - not every one will trust the same third party
  - no entity lasts forever
- Publicly auditable logger
  - anyone can audit the logger
    - data signers, data consumers, certificate issuers, independent third parties, …
  - force logger to behave honestly

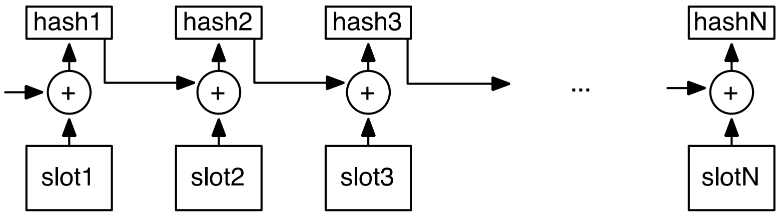
Secure logger

- A trusted third party?
  - not every one will trust the same third party
  - no entity lasts forever
- Publicly auditable logger
  - anyone can audit the logger
    - data signers, data consumers, certificate issuers, independent third parties, …
  - force logger to behave honestly
  - tamper-evident log

Tamper-Evident Log

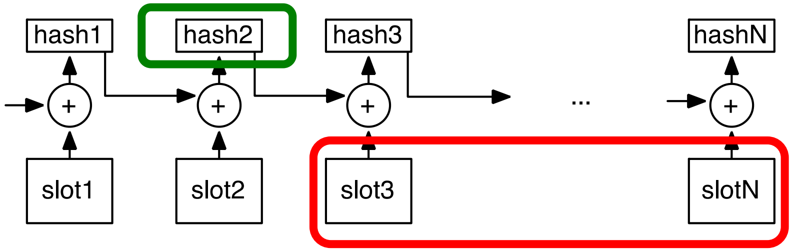
Tamper-Evident Log

- Hash chain
  - Bitcoin
  - simple, space effient
  - slow to check



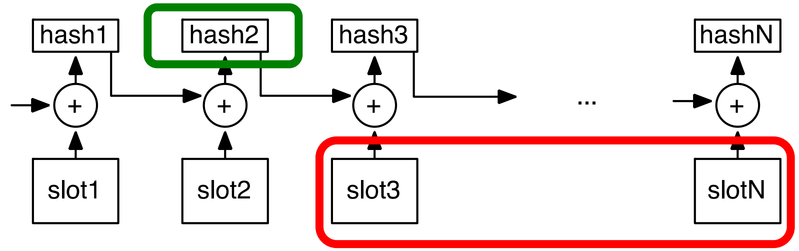
Tamper-Evident Log

- Hash chain
  - Bitcoin
  - simple, space effient
  - slow to check

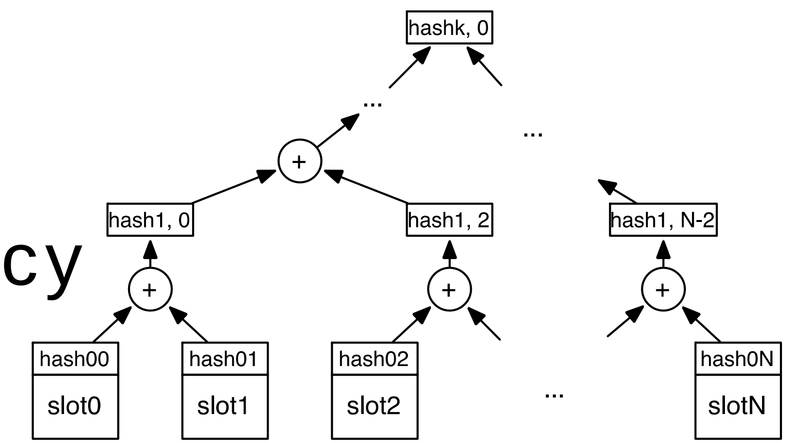


Tamper-Evident Log

- Hash chain
  - Bitcoin
  - simple, space effient
  - slow to check

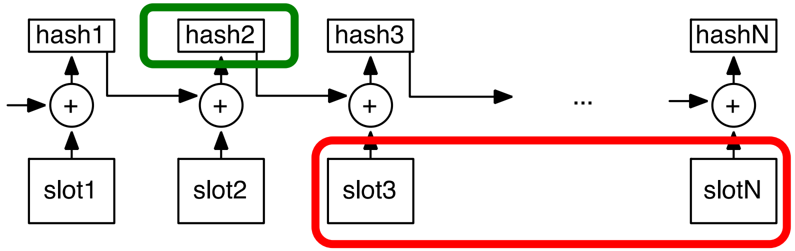


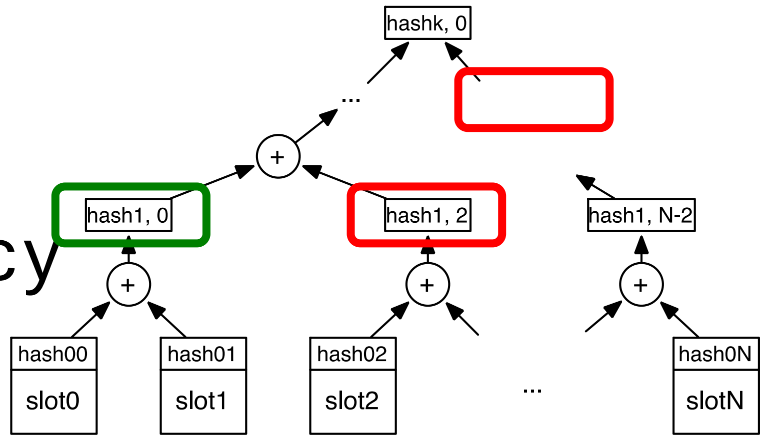
- MerkleTree
  - Certificate Transparency
  - efficient checking



Tamper-Evident Log

- Hash chain
  - Bitcoin
  - simple, space effient
  - slow to check
- MerkleTree
  - Certificate Transparency
  - efficient checking

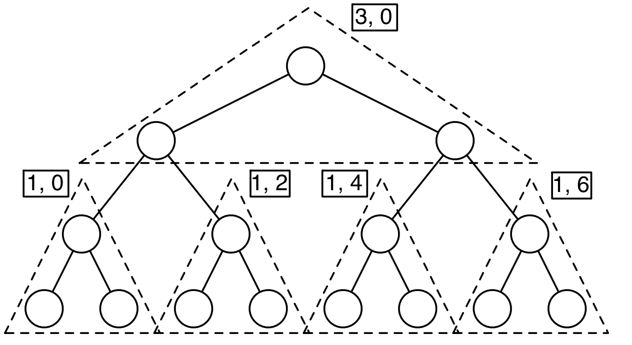




MerkleTree in NDN

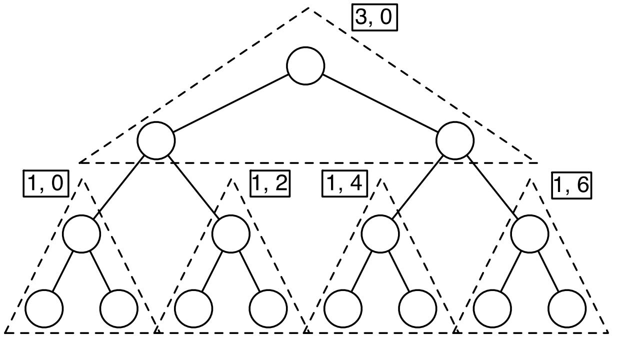
MerkleTree in NDN

- A MerkleTree consists of sub-trees



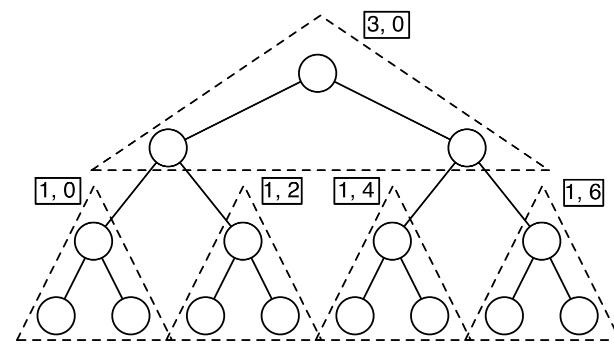
MerkleTree in NDN

- A MerkleTree consists of sub-trees
- Each sub-tree
  - fixed by its root
    - **easy to verify**



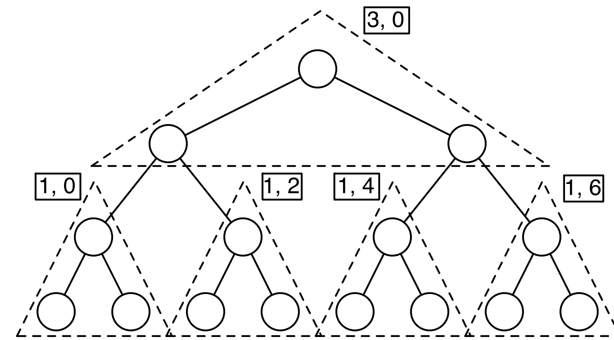
MerkleTree in NDN

- A MerkleTree consists of sub-trees
- Each sub-tree
  - fixed by its root
    - **easy to verify**
  - fixed by its index (level, seqNo)
    - **easy to retrieve**



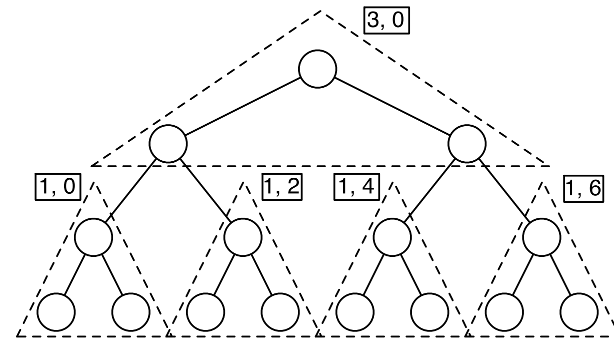
MerkleTree in NDN

- A MerkleTree consists of sub-trees
- Each sub-tree
  - fixed by its root
    - **easy to verify**
  - fixed by its index (level, seqNo)
    - **easy to retrieve**
  - once complete, become frozen
    - **can be cached**



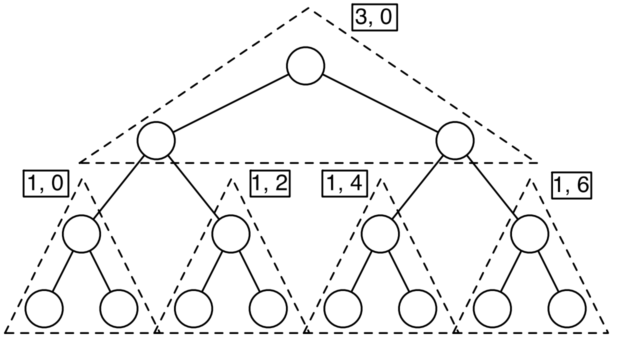
MerkleTree in NDN

- A MerkleTree consists of sub-trees
- Each sub-tree
  - fixed by its root
    - **easy to verify**
  - fixed by its index (level, seqNo)
    - **easy to retrieve**
  - once complete, become frozen
    - **can be cached**
- Encode each sub tree in a data packet
  - name: /<loggerPrefix>/[subTreeIndex]/[digest]
  - content: node digests in BFS order



MerkleTree in NDN

- A MerkleTree consists of sub-trees
- Each sub-tree
  - fixed by its root
    - **easy to verify**
  - fixed by its index (level, seqNo)
    - **easy to retrieve**
  - once complete, become frozen
    - **can be cached**

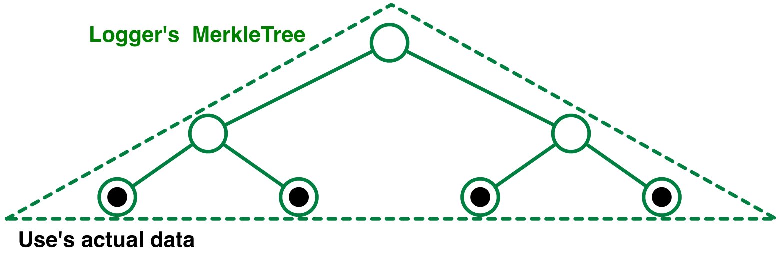


- Encode each sub tree in a data packet
  - name: /<loggerPrefix>/[subTreeIndex]/[digest]
  - content: node digests in BFS order
- Leaf node
  - name: /<loggerPrefix>/leaf/[seqNo]
  - detailed info (signed data, timestamp…)

Storage of log & data

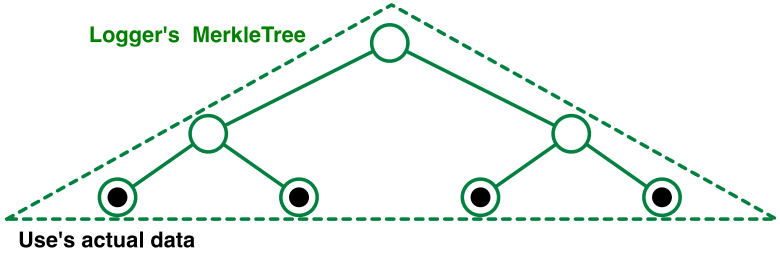
Storage of log & data

- Store log & data separately
  - Loggers maintain log
  - Users maintain actual data
    - no need to retrieve log for unavailable data



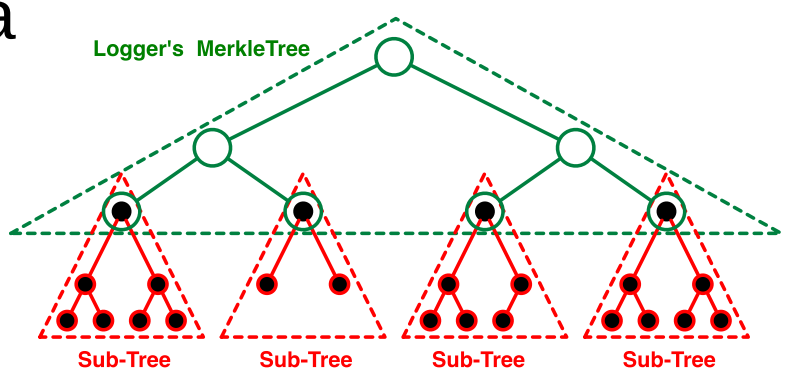
Storage of log & data

- Store log & data separately
  - Loggers maintain log
  - Users maintain actual data
    - no need to retrieve log for unavailable data
- User cannot change actual data
  - digest is fixed in log



Storage of log & data

- Store log & data separately
  - Loggers maintain log
  - Users maintain actual data
    - no need to retrieve log for unavailable data
- User cannot change actual data
  - digest is fixed in log
- Users may even keep a sub-tree
  - contain a user's own data
  - could be incomplete
  - root digest is fixed in log



Multiple Loggers

Multiple Loggers

- Loggers may serve different purposes
  - different namespaces, different trust models
    - e.g., each organization may have its own logger to log their own data

Multiple Loggers

- Loggers may serve different purposes
  - different namespaces, different trust models
    - e.g., each organization may have its own logger to log their own data
- Loggers synchronize with each other

Multiple Loggers

- Loggers may serve different purposes
  - different namespaces, different trust models
    - e.g., each organization may have its own logger to log their own data
- Loggers synchronize with each other
  - improve redundancy

Multiple Loggers

- Loggers may serve different purposes
  - different namespaces, different trust models
    - e.g., each organization may have its own logger to log their own data
- Loggers synchronize with each other
  - improve redundancy
  - automatically audit each other

Multiple Loggers

- Loggers may serve different purposes
  - different namespaces, different trust models
    - e.g., each organization may have its own logger to log their own data
- Loggers synchronize with each other
  - improve redundancy
  - automatically audit each other
  - using/extending ChronoSync
    - each logger has its own prefix & seqNo

Hash Agility

- Temper-evident log is based on hash function
- A hash function may be broken eventually

Hash Agility

- Temper-evident log is based on hash function
- A hash function may be broken eventually
- Two copies using different hash functions
  - one is relatively stronger than the other
    - e.g., Sha256(B), Sha3-384(B)

Hash Agility

- Temper-evident log is based on hash function
- A hash function may be broken eventually
- Two copies using different hash functions
  - one is relatively stronger than the other
    - e.g., Sha256(B), Sha3-384(B)
  - assume: not broken on the same day
    - weaker broken, stronger still valid
    - enough time to reconstruct another copy with a stronger hash at that time
    - hopefully, it rarely happens

Conclusion

- logging system enables
  - post-fact validation
  - usage of short-lived keys
- Secure logging system through public auditing
- Increase redundancy of certificate provisioning

Thank you!

yingdi@cs.ucla.edu