ETH zürich

Applied Crypto Group
D-INFK

Security Analysis of Proton Key Transparency

Thore Göbel

Master Thesis Final

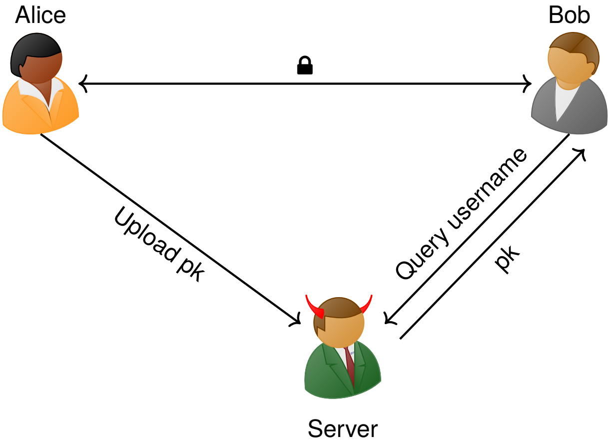
Outline

1. Motivation
2. ProtonKT Architecture
3. Security Analysis

Outline

1. Motivation
2. ProtonKT Architecture
3. Security Analysis

Internet Messaging

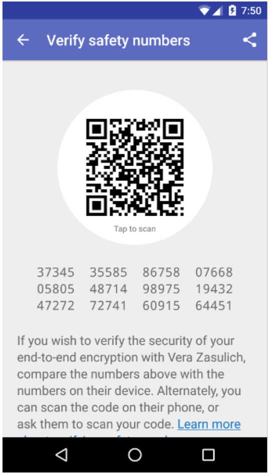


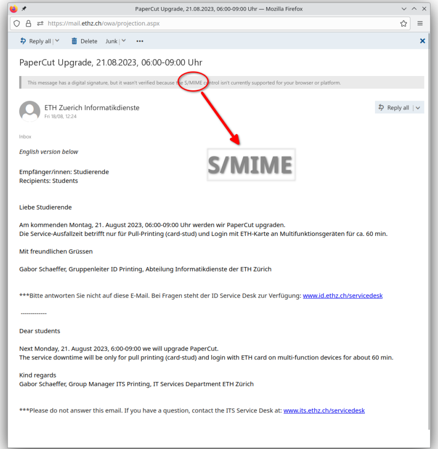
A Key Directory is just a Database

Username	Public Key
alice	~~pkA~~ pkEve
bob	pkB

```
UPDATE table_keys SET pk="pkEve" WHERE username="alice";
```

Existing Solutions: Out-of-Band || Certificates





Key Transparency Goals

Goal 1: make key verification automatic

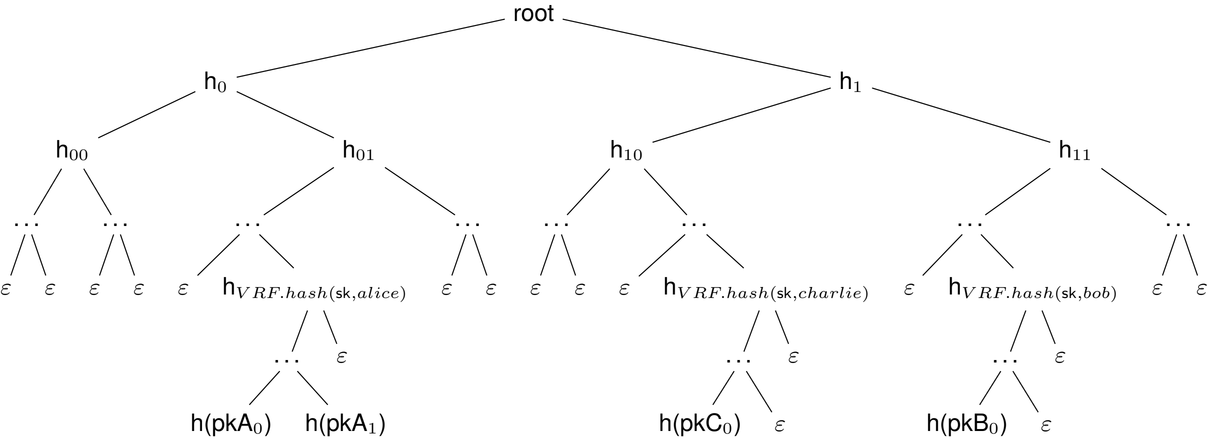
Goal 2: make server behaviour auditable (“transparent”)

Key Transparency in the Real World

- Keybase (docs), Zoom (Whitepaper)
- WhatsApp (blog , Stanford Security Seminar talk)
- Apple iMessage (blog)
- IETF Working Group
- Proton (this talk)

Outline

Build a Merkle Hash Tree from the Key Directory



$$leafindex = VRF.verify(\mathsf{pk}, \mathsf{label}, \pi_{vrf}) \mid\mid \mathsf{rev}$$

Verifiable Random Function (VRF)

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow VRF.kgen()$$

$$\beta \leftarrow VRF.hash(\mathsf{sk}, \alpha)$$

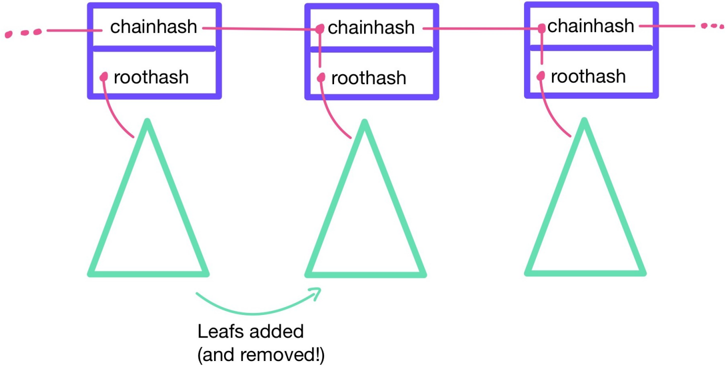
$$\pi \leftarrow VRF.prove(\mathsf{sk}, \alpha)$$

$$\beta / \perp \leftarrow VRF.verify(\mathsf{pk}, \alpha, \pi)$$

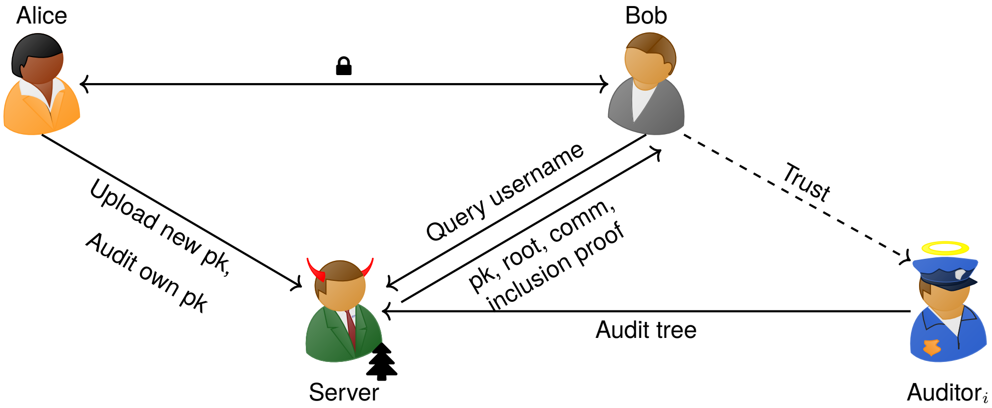
Properties: Pseudorandomness, Collision Resistance, Uniqueness

Trees Across Epochs

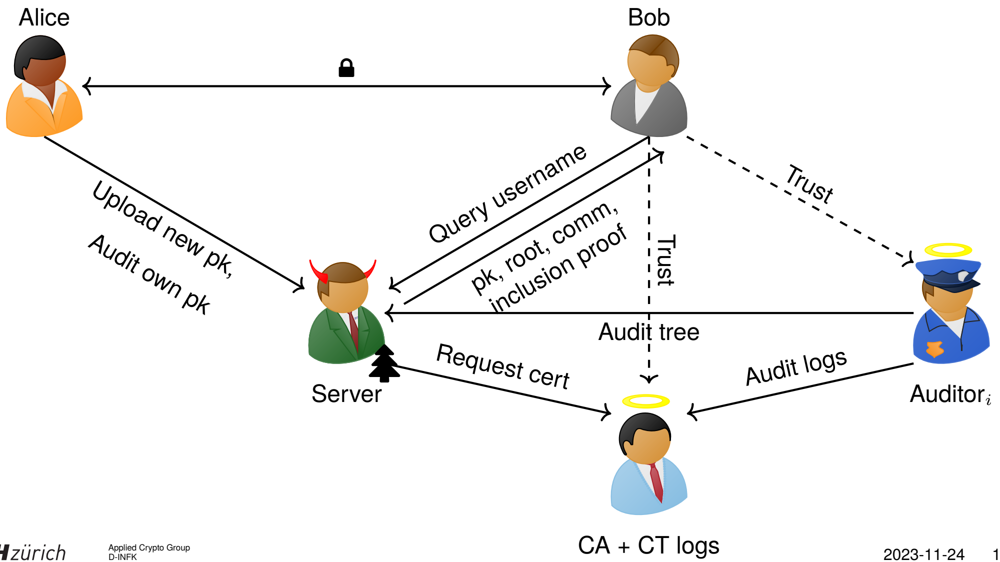
$$chainhash_i = h(chainhash_{i-1} \,||\, roothash_i)$$



System Overview and Roles

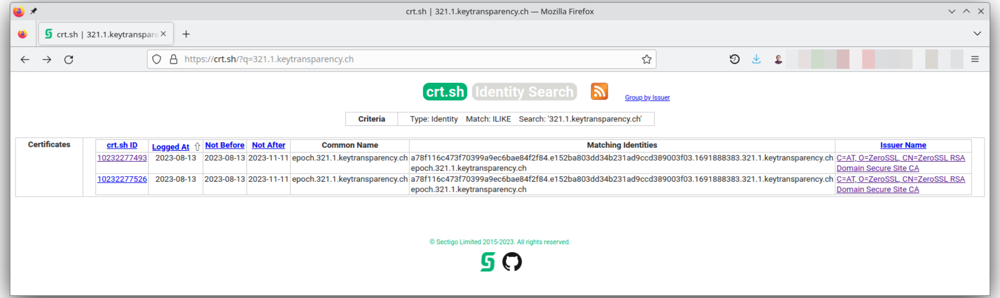


System Overview and Roles



Committing to the Tree Root

`chainhash[0:32].chainhash[32:64].timestamp.epochid.1.keytransparency.ch.`



Tree Leaves

- $leafindex = VRF.verify(\mathsf{pk}, \mathsf{label}, \pi_{vrf}) \mid\mid \mathsf{rev}$
- $\mathsf{val}_{abs} = \emptyset$
- $\mathsf{val}_{incl} = \{keylist,\ minEpochId\}$
- $\mathsf{val}_{obs} = \{ObsolenceToken,\ minEpochId\}$

Tree Leaves

- $leafindex = VRF.verify(\mathsf{pk}, \mathsf{label}, \pi_{vrf}) \mid\mid \mathsf{rev}$
- $\mathsf{val}_{abs} = \emptyset$
- $\mathsf{val}_{incl} = \{keylist, minEpochId\}$
- $\mathsf{val}_{obs} = \{ObsolenceToken, minEpochId\}$

- $leafhash_{abs} = \varepsilon$
- $leafhash_{incl} = h\Big(h(keylist) \mid\mid minEpochId\Big)$
- $leafhash_{obs} = h\Big(h(ObsolenceToken) \mid\mid minEpochId\Big)$

Deletions

Deletion of leaf rev allowed $\iff$ leaf $\mathsf{rev}+1$ inserted ≥ 90 days ago

$$\{keylist_1,\, minEpochId_1\}, \{ObsolenceToken_2,\, minEpochId_2\}, \{keylist_3,\, minEpochId_3\}, \ldots$$

ProtonKT Subprotocols (simplified)

- ProtonKT.RequestInsertion(label, $keylist$)
- ProtonKT.Publish($\{$label$_i$, $keylist_i\}_i$)

ProtonKT Subprotocols (simplified)

- ProtonKT.RequestInsertion(label, $keylist$)
- ProtonKT.Publish($\{$label$_i$, $keylist_i\}_i$)

- ProtonKT.QueryEpoch(t)
- ProtonKT.QueryValue($roothash_t$, label)

ProtonKT Subprotocols (simplified)

- ProtonKT.RequestInsertion(label, $keylist$)
- ProtonKT.Publish($\{$label$_i, keylist_i\}_i$)

- ProtonKT.QueryEpoch(t)
- ProtonKT.QueryValue($roothash_t$, label)

- ProtonKT.SelfAudit($roothash_t$, label, $keylist$)
- ProtonKT.PromiseAudit($roothash_t, promises$)
- ProtonKT.ExtAudit()

Outline

Security Properties

Consistency: for a given $(\mathsf{label}, \mathsf{rev})$, we agree on (τ, val).

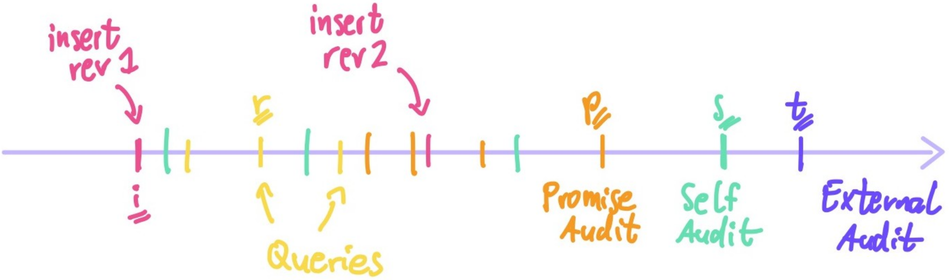
Consistency between queries and Self Audits $\implies$ correctness of keys

Security Property: Query-to-SelfAudit Consistency

We say that ProtonKT provides *Query-to-SelfAudit Consistency*, if

- whenever there was a successful External Audit of epoch t
- and client A runs a successful Self Audit SA for its label at epoch $s \leq t$ and SA passes with $latestRev \geq \mathsf{rev}$,
- and prior to epoch t A has run a successful Self Audit at least once every $\mathsf{DeletionParam}$ (e.g. every 90 days),
- and a query Q for label in epoch $r \leq t$ returned outcome $O = (\tau, \mathsf{rev}, \mathsf{val})$,
- and – if Q returned O as a promise P – there was a successful Promise Audit that sees P at an epoch p with $r < p \leq t$,
- then client A agrees that $(\tau, \mathsf{rev}, \mathsf{val})$ is the expected outcome for rev.

Security Property: Query-to-SelfAudit Consistency



Adversary Model

The adversary can:

- Control the network (active network adversary, Dolev-Yao). Reorder, replay, drop, insert, modify messages.
- Corrupt the KT server. Insert, modify, delete leaves in the Merkle tree.

The adversary cannot:

- Break SHA-256 collision resistance, break ECVRF uniqueness.
- Prevent External Auditors from seeing all CT log entries.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 1: server cannot equivocate on root hash.

- Assume two executions Q, U of $\mathsf{ProtonKT.QueryEpoch}(t)$ accepted $roothash_t^Q \neq roothash_t^U$. Also assume an External Audit passed.
- Then there must exist $chainhash_t^Q = h(chainhash_{t-1}^Q || roothash_t^Q)$ and $cert^Q$, and same for U.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 1: server cannot equivocate on root hash.

- Assume two executions Q, U of $\mathsf{ProtonKT.QueryEpoch}(t)$ accepted $roothash_t^Q \neq roothash_t^U$. Also assume an External Audit passed.
- Then there must exist $chainhash_t^Q = h(chainhash_{t-1}^Q || roothash_t^Q)$ and $cert^Q$, and same for U.
- *Case 1* $(chainhash_t^Q \neq chainhash_t^U)$*:* Then $cert^Q \neq cert^U$. But these certs must be in CT logs. Then the External Audit finds the equivocation. If it doesn't, then a CT log was malicious or the auditor does not have a global view of CT.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 1: server cannot equivocate on root hash.

- Assume two executions Q, U of $\mathsf{ProtonKT.QueryEpoch}(t)$ accepted $roothash_t^Q \neq roothash_t^U$. Also assume an External Audit passed.
- Then there must exist $chainhash_t^Q = h(chainhash_{t-1}^Q || roothash_t^Q)$ and $cert^Q$, and same for U.
- *Case 1 ($chainhash_t^Q \neq chainhash_t^U$):* Then $cert^Q \neq cert^U$. But these certs must be in CT logs. Then the External Audit finds the equivocation. If it doesn't, then a CT log was malicious or the auditor does not have a global view of CT.
- *Case 2 ($chainhash_t^Q = chainhash_t^U$):* Then we have $(pch||rh) \neq (pch'||rh')$ such that $h(pch||rh) = h(pch'||rh')$. Contradiction to SHA-256 collision resistance.

CA/CT can Discredit the Server

Problem:

`chainhash[0:32].chainhash[32:64].timestamp.epochid.1.keytransparency.ch.`

`chainhash'[0:32].chainhash'[32:64].timestamp.epochid.1.keytransparency.ch.`

CA + CT logs

CA/CT can Discredit the Server

Problem:

`chainhash[0:32].chainhash[32:64].timestamp.epochid.1.keytransparency.ch.`

`chainhash'[0:32].chainhash'[32:64].timestamp.epochid.1.keytransparency.ch.`

CA + CT logs

Possible solution:

`sig[0:32].sig[32:64].sig[64:96].sig[96:128].epochid.1.keytransparency.ch.`

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it)

- Assume a query and a Self Audit disagree on the outcome for rev:

  $$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

  Also assume an External Audit passed.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it)

- Assume a query and a Self Audit disagree on the outcome for rev:

  $$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

  Also assume an External Audit passed.
- *Case 1 ($idx^Q \neq idx^A$):*
  Let leaf index $idx^Q = VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^Q)||\mathsf{rev}$, and
  $idx^A = VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^A)||\mathsf{rev}$.
  But $VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^Q) \neq VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^A)$ contradicts uniqueness of ECVRF.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it)

- Assume a query and a Self Audit disagree on the outcome for rev:

  $$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

  Also assume an External Audit passed.
- *Case 1 ($idx^Q \neq idx^A$):*
  Let leaf index $idx^Q = VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^Q)||\mathsf{rev}$, and
  $idx^A = VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^A)||\mathsf{rev}$.
  But $VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^Q) \neq VRF.verify(\mathsf{sk}, \mathsf{label}, \pi^A)$ contradicts uniqueness of ECVRF.
- *Case 2 (Q and SA at same epoch):* By non-equivocation, Q and SA agree on $roothash_t$.
  (Case for different epochs: omitted.)

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it, continued)

$$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

So far: same tree root hash, same leaf index.

- *Case 3 ($leafhash^Q_{idx} \neq leafhash^A_{idx}$):* Hash collision on the path to the root.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it, continued)

$$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

So far: same tree root hash, same leaf index.

- *Case 3 ($leafhash^Q_{idx} \neq leafhash^A_{idx}$):* Hash collision on the path to the root.
- *Case 4 ($leafhash^Q_{idx} = leafhash^A_{idx}$):*
  - *Case 4.1 ($\mathsf{val}^Q \neq \mathsf{val}^A$):* omitted.
  - *Case 4.2 ($\mathsf{val}^Q = \mathsf{val}^A$):*

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it, continued)

$$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

So far: same tree root hash, same leaf index.

- *Case 3 ($leafhash^Q_{idx} \neq leafhash^A_{idx}$):* Hash collision on the path to the root.
- *Case 4 ($leafhash^Q_{idx} = leafhash^A_{idx}$):*
  - *Case 4.1 ($\mathsf{val}^Q \neq \mathsf{val}^A$):* omitted.
  - *Case 4.2 ($\mathsf{val}^Q = \mathsf{val}^A$):*
    - *Case 4.1.0 ($\tau^Q = abs, \tau^A = abs$):* $O^Q = O^A$, done.
    - *Case 4.1.1 ($\tau^Q = abs, \tau^A \neq abs$):* Absence has $\mathsf{val}_{abs} = \varepsilon$ but inclusion/absence have values.
    - *Case 4.1.2 ($\tau^Q \neq abs, \tau^A = abs$):* same.

Manual Analysis of Query-to-SelfAudit Consistency

Analysis goal 2: Query-to-SelfAudit (part of it, continued)

$$O^Q = (\tau^Q, \mathsf{rev}, \mathsf{val}^Q) \neq (\tau^A, \mathsf{rev}, \mathsf{val}^A) = O^A$$

So far: same tree root hash, same leaf index.

- *Case 3 ($leafhash^Q_{idx} \neq leafhash^A_{idx}$):* Hash collision on the path to the root.
- *Case 4 ($leafhash^Q_{idx} = leafhash^A_{idx}$):*
  - *Case 4.1 ($\mathsf{val}^Q \neq \mathsf{val}^A$):* omitted.
  - *Case 4.2 ($\mathsf{val}^Q = \mathsf{val}^A$):*
    - *Case 4.1.0 ($\tau^Q = abs, \tau^A = abs$):* $O^Q = O^A$, done.
    - *Case 4.1.1 ($\tau^Q = abs, \tau^A \neq abs$):* Absence has $\mathsf{val}_{abs} = \varepsilon$ but inclusion/absence have values.
    - *Case 4.1.2 ($\tau^Q \neq abs, \tau^A = abs$):* same.
    - *Case 4.1.3 ($\tau^Q = incl, \tau^A = obs$):* Then $\mathsf{val}^Q \stackrel{\text{Def}}{=} \{keylist, minEpochId\} = \{ObsolenceToken, minEpochId\} \stackrel{\text{Def}}{=} \mathsf{val}^A$.
      I.e. Q interprets the first field as a keylist and A as an ObsolenceToken.
      This is a contradiction to the fact that the algorithms check that the keylist is JSON-encoded and that ObsolenceToken is a non-empty hex value.
    - *Case 4.1.4 ($\tau^Q = obs, \tau^A = incl$):* same.

Better Leaf Hashes

- $leafhash_{incl} = h\Big(h(keylist) \mid\mid minEpochId\Big)$
- $leafhash_{obs} = h\Big(h(ObsolenceToken) \mid\mid minEpochId\Big)$

Better Leaf Hashes

- $leafhash_{incl} = h\big(h(keylist) \mid\mid minEpochId\big)$
- $leafhash_{obs} = h\big(h(ObsolenceToken) \mid\mid minEpochId\big)$

- $leafhash^{proposed}_{incl} = h\big(h(``1'' \mid\mid keylist) \mid\mid minEpochId\big)$
- $leafhash^{proposed}_{obs} = h\big(h(``2'' \mid\mid ObsolenceToken) \mid\mid minEpochId\big)$

Server can Delay Promise Audit

```
if (promise.expectedMinEpochID > currentEpoch.EpochID) {
return LocalStorageAuditStatus.RetryLater;
}

/* ... only further down the Maximum Merge Delay is checked ... */
if (isTimestampTooOld(promise.creationTimestamp)) {
throwKTError('promise was ignored beyond MMD');
}
```

Formal Analysis of Query-to-SelfAudit Consistency

- Used new Tamarin festures (subterm, natural numbers).
- Tried to prove Query-to-SelfAudit Consistency but ran into a limitation of how Tamarin handles induction.
- Still a useful exercise to understand the protocol better, to find gaps in your understanding.

Conclusion

- ProtonKT Specification
- Adversary Model, Security Property
- Manual Analysis
- Formal Analysis with dead-end
- Recommendations: sign chainhash in CT to prevent discrediting, make type explicit in leaf hash, server could delay Promise Audit forever

Questions?

Formal Analysis of Query-to-SelfAudit Consistency

```
rule CT_Insert:
[ In(<epoch_id, chainhash>) ]
--[ CtInsertChainhash(epoch_id, chainhash) ]->
[ !CT(epoch_id, chainhash) ]
```

Tamarin Prover

Model 1: Tree as Persistent Facts.

```
!TreeLeaf($label, val, %rev, %min_epoch_id)
```

Tamarin Prover

Model 1: Tree as Persistent Facts.

```
!TreeLeaf($label, val, %rev, %min_epoch_id)
```

Model 2: Trees as Terms.

```
roothash = h( <'head', h(ut_0), h(ut_1), ..., h(ut_n), 'tail'> )
ut = < $label, <%n, val_n>, ..., <%3, val_3>, <%2, 'empty'>, 'rest' >
```

$\Longrightarrow$ Only 3 levels, not binary, but more tree-ish.

Tamarin Prover

Model 1: Tree as Persistent Facts.

```
!TreeLeaf($label, val, %rev, %min_epoch_id)
```

Model 2: Trees as Terms.

```
roothash = h( <'head', h(ut_0), h(ut_1), ..., h(ut_n), 'tail'> )
ut = < $label, <%n, val_n>, ..., <%3, val_3>, <%2, 'empty'>, 'rest' >
```

$\Longrightarrow$ Only 3 levels, not binary, but more tree-ish.

We didn't find a proof, we ran into a limitation of Tamarin's induction mechanism.

Proving Query-to-SelfAudit Consistency with Tamarin

Problem: Query and Self Audit can happen in different epochs $r \neq s$, w.l.o.g. $r < s$.
Recall: $chainhash_i = h(chainhash_{i-1} \,||\, roothash_i)$
Thus: $chainhash_r \sqsubset chainhash_s$
To reason about the leaves, we need to reason about how the tree(s) evolved, thus we need to reason about the chainhashes.

Proving Query-to-SelfAudit Consistency with Tamarin

Problem: Query and Self Audit can happen in different epochs $r \neq s$, w.l.o.g. $r < s$.
Recall: $chainhash_i = h(chainhash_{i-1} \,||\, roothash_i)$
Thus: $chainhash_r \sqsubset chainhash_s$
To reason about the leaves, we need to reason about how the tree(s) evolved, thus we need to reason about the chainhashes.

Lemma:

```
"All ch1 ch2 #i #j. Ch(ch1)@i & Ch(ch2)@j ==> ch1 << ch2"
```

Proving Query-to-SelfAudit Consistency with Tamarin

Problem: Query and Self Audit can happen in different epochs $r \neq s$, w.l.o.g. $r < s$.
Recall: $chainhash_i = h(chainhash_{i-1} \mid\mid roothash_i)$
Thus: $chainhash_r \sqsubset chainhash_s$
To reason about the leaves, we need to reason about how the tree(s) evolved, thus we need to reason about the chainhashes.

Lemma:

```
"All ch1 ch2 #i #j. Ch(ch1)@i & Ch(ch2)@j ==> ch1 << ch2"
```

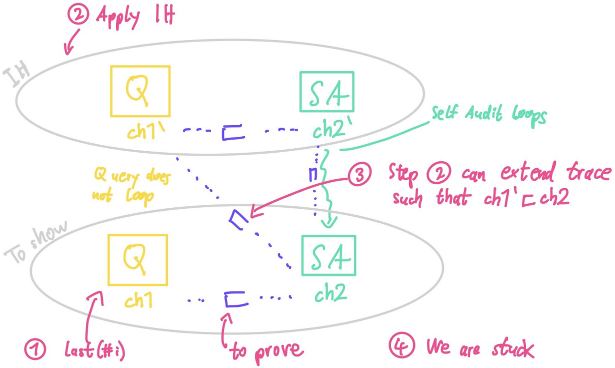
Induction hypothesis:

```
"All ch1 ch2 #i #j. Ch(ch1)@i & Ch(ch2)@j ==> ch1 << ch2 | last(#i) | last(#j)"
```

We cannot avoid induction because Self Audit and External Audit loop (over revisions and over epochs).

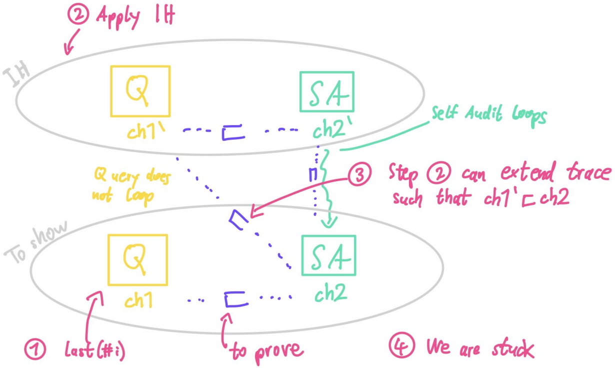
Case Split on Induction

IH: `"All ch1 ch2 #i #j. Ch(ch1)@i & Ch(ch2)@j ==> ch1 << ch2 | last(#i) | last(#j)"`



Case Split on Induction

IH: `"All ch1 ch2 #i #j. Ch(ch1)@i & Ch(ch2)@j ==> ch1 << ch2 | last(#i) | last(#j)"`



We would like:

`"All ch1 ch2 #j. Ch(ch2)@j & ch1 << ch2 ==> Ex #i. Ch(ch1)@i"`

But `ch1` is not guarded, i.e. invalid Tamarin syntax.