Applied Crypto Group
D-INFK

Analysing Key Transparency

Thore Göbel

Master Thesis Midway

Outline

1. Motivation

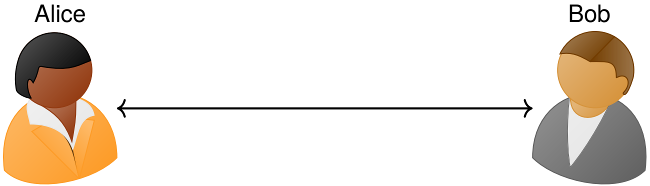
2. KT Architecture

3. Formal Model + Formal Verification

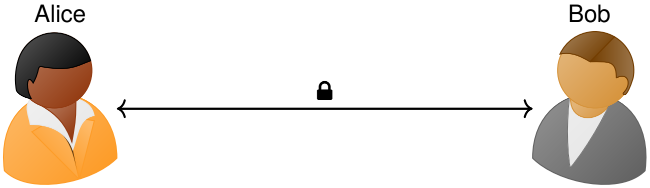
Outline

1. Motivation
2. KT Architecture
3. Formal Model + Formal Verification

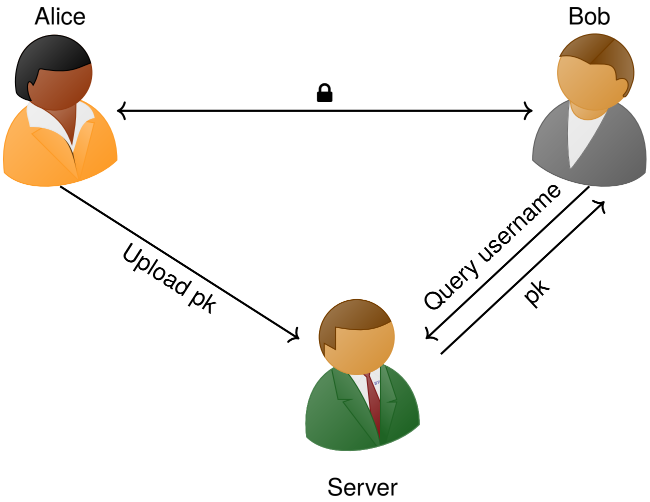
Internet Messaging



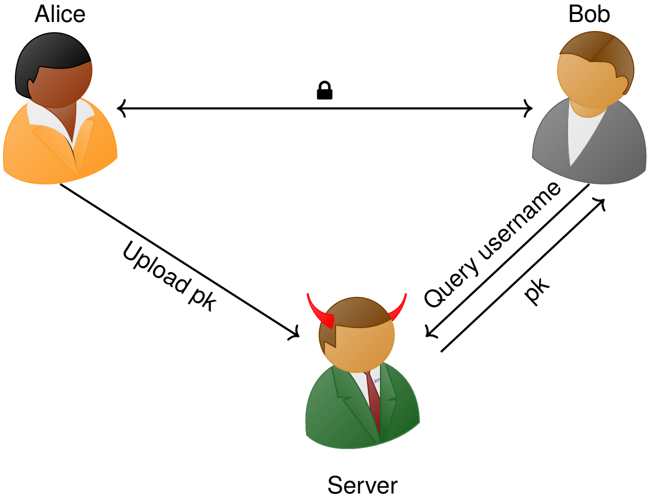
Internet Messaging



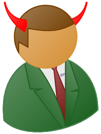
Internet Messaging



Internet Messaging

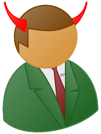


A Key Server is just a Database



Username	Public Key
Alice	pkA
Bob	pkB

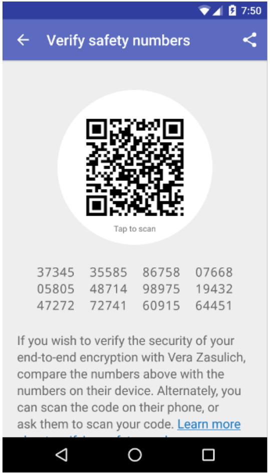
A Key Server is just a Database

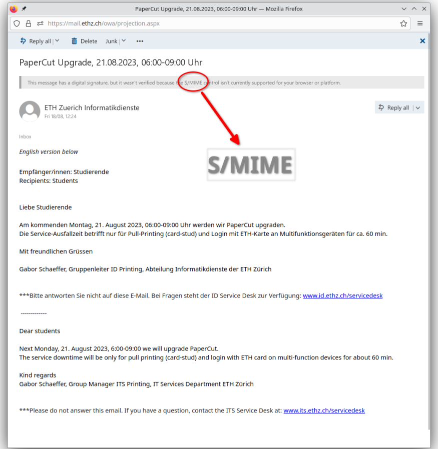


Username	Public Key
Alice	pkEve
Bob	pkB

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

Existing Solutions: Out-of-Band || Certificates





Goals

Goal 1: consistency (not correctness) of username–pk–bindings

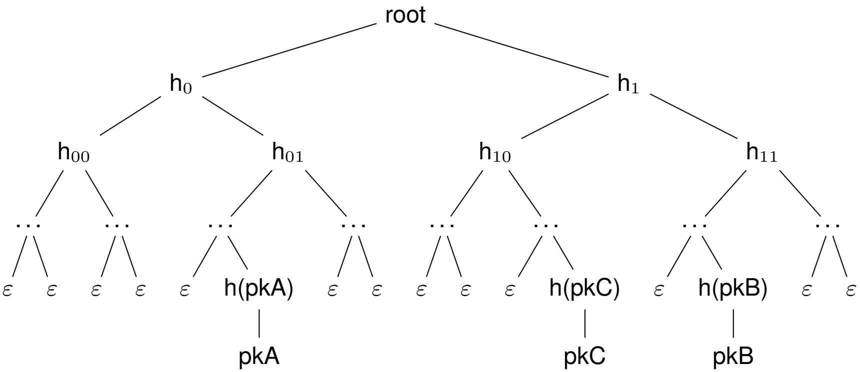
Goal 2: make server behaviour detectable ("transparent")

Goal 3: make verification automatic

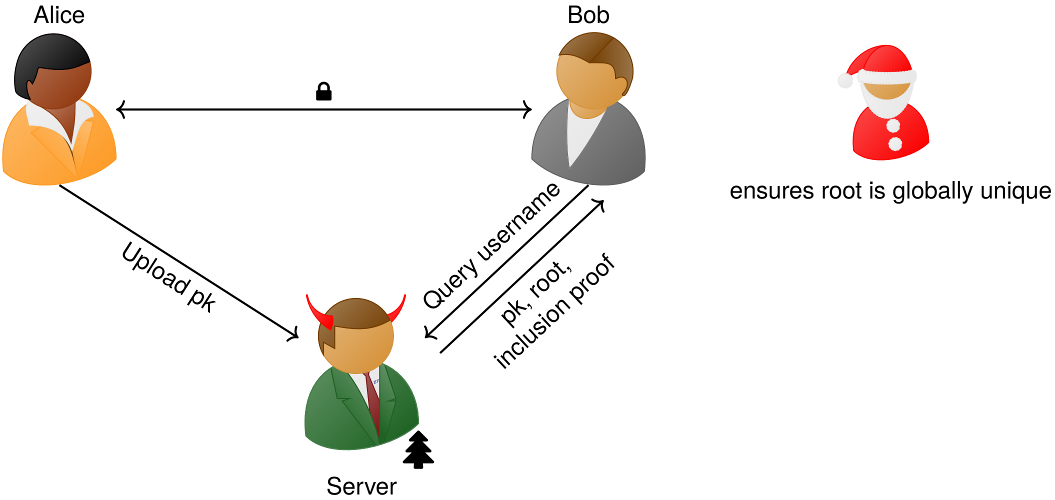
Outline

1. Motivation
2. KT Architecture
3. Formal Model + Formal Verification

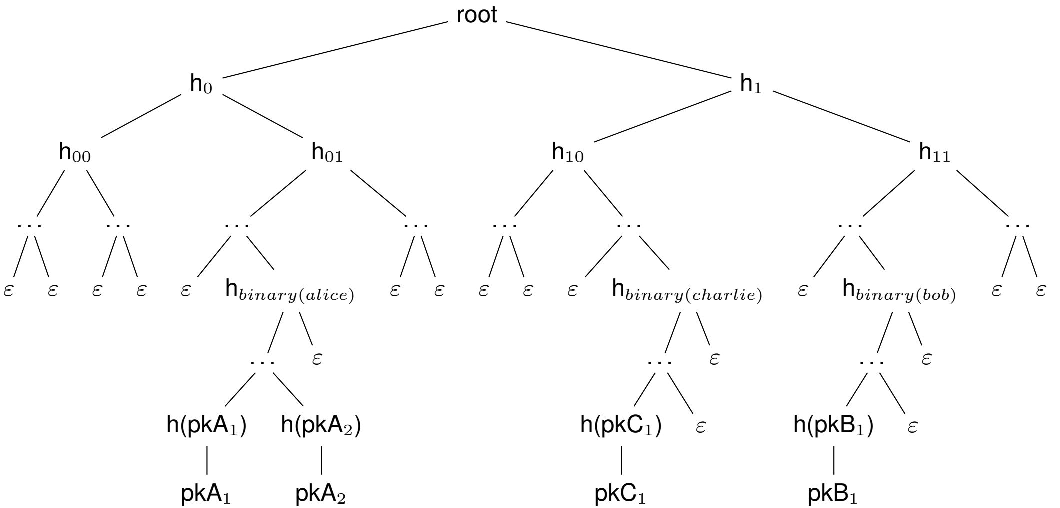
Idea: Trees


root
h_0
h_1
h_{00}
h_{01}
h_{10}
h_{11}
...
ε
h(pkA)
h(pkC)
h(pkB)
pkA
pkC
pkB

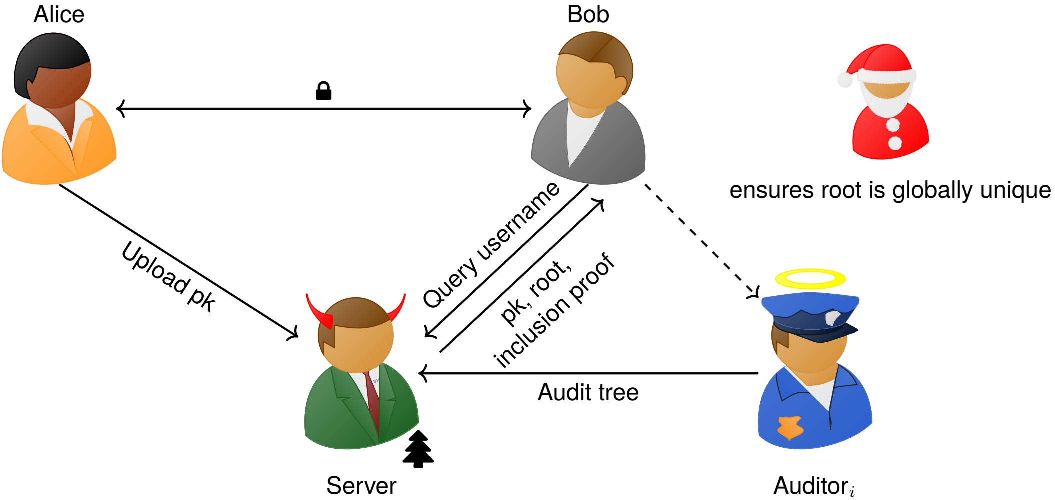
Idea: Trees


Alice
Bob
Upload pk
Query username
pk, root,
inclusion proof
Server
ensures root is globally unique

Insertions + Updates



Insertions + Updates



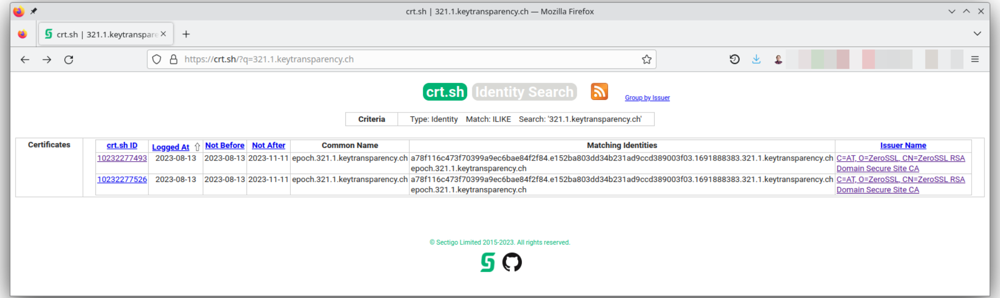
Key Transparency in the Real World

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

Except Keybase, none of this is fully rolled out.

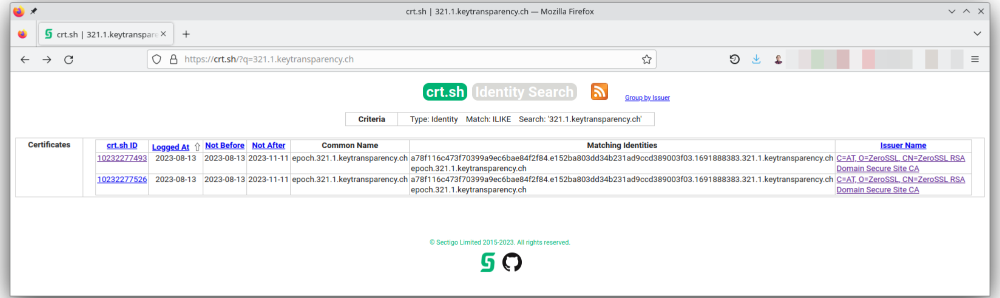
Committing to the Tree Root

`hash[0:32].hash[32:64].timestamp.epochid.1.keytransparency.ch.` (crt.sh)



Committing to the Tree Root

`hash[0:32].hash[32:64].timestamp.epochid.1.keytransparency.ch.` (crt.sh)



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

Goals

Goal 1: consistency of username–pk–bindings? Yes: tree root.

Goal 2: make server behaviour detectable? Yes: tree root divergence, append-only-ness, inclusion proofs.

Goal 3: make verification automatic? Yes.

Outline

1. Motivation

2. KT Architecture

3. Formal Model + Formal Verification

Verifiable Key Directory

A *Verifiable Key Directory (VKD)* is a set of 7 algorithms:

- $(\mathsf{Dir}_t, \mathsf{st}_t, \Pi^{\mathsf{Upd}})/\perp \leftarrow \mathsf{VKD.PublishUpdate}(\mathsf{Dir}_{t-1}, \mathsf{st}_{t-1}, S_t)$
- $0/1 \leftarrow \mathsf{VKD.VerifyUpdate}(t, \mathsf{com}, \mathsf{com}', \Pi^{\mathsf{Upd}})$ // ProtonVKD.VerifyEpoch
- $0/1 \leftarrow \mathsf{VKD.Audit}(t_1, t_n, (\Pi_t^{\mathsf{Upd}})_{t=t_1}^{t_{n-1}})$

Verifiable Key Directory

A *Verifiable Key Directory (VKD)* is a set of 7 algorithms:

- $(\mathsf{Dir}_t, \mathsf{st}_t, \Pi^{\mathsf{Upd}})/\bot \leftarrow \mathsf{VKD.PublishUpdate}(\mathsf{Dir}_{t-1}, \mathsf{st}_{t-1}, S_t)$
- $0/1 \leftarrow \mathsf{VKD.VerifyUpdate}(t, \mathsf{com}, \mathsf{com}', \Pi^{\mathsf{Upd}})$ // ProtonVKD.VerifyEpoch
- $0/1 \leftarrow \mathsf{VKD.Audit}(t_1, t_n, (\Pi_t^{\mathsf{Upd}})_{t=t_1}^{t_{n-1}})$
- $(\mathsf{val}, \alpha, \pi) \leftarrow \mathsf{VKD.Query}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$ // ProtonVKD.GetProof
- $0/1 \leftarrow \mathsf{VKD.VerifyQuery}(t, \mathsf{label}, \mathsf{val}, \alpha, \pi)$ // ProtonVKD.VerifyProofInEpoch

Verifiable Key Directory

A *Verifiable Key Directory (VKD)* is a set of 7 algorithms:

- $(\mathsf{Dir}_t, \mathsf{st}_t, \Pi^{\mathsf{Upd}})/\bot \leftarrow \mathsf{VKD.PublishUpdate}(\mathsf{Dir}_{t-1}, \mathsf{st}_{t-1}, S_t)$
- $0/1 \leftarrow \mathsf{VKD.VerifyUpdate}(t, \mathsf{com}, \mathsf{com}', \Pi^{\mathsf{Upd}})$ // $\mathsf{ProtonVKD.VerifyEpoch}$
- $0/1 \leftarrow \mathsf{VKD.Audit}(t_1, t_n, (\Pi_t^{\mathsf{Upd}})_{t=t_1}^{t_{n-1}})$
- $(\mathsf{val}, \alpha, \pi) \leftarrow \mathsf{VKD.Query}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$ // $\mathsf{ProtonVKD.GetProof}$
- $0/1 \leftarrow \mathsf{VKD.VerifyQuery}(t, \mathsf{label}, \mathsf{val}, \alpha, \pi)$ // $\mathsf{ProtonVKD.VerifyProofInEpoch}$
- $\left((\mathsf{val}_i, t_i)_{i=1}^{n}, \Pi^{\mathsf{Ver}}\right) \leftarrow \mathsf{VKD.KeyHistory}(\mathsf{st}_t, \mathsf{Dir}_t, t, \mathsf{label})$
- $0/1 \leftarrow \mathsf{VKD.VerifyHistory}(t, \mathsf{label}, (\mathsf{val}_i, t_i)_{i=1}^{n}, \Pi^{\mathsf{Ver}})$ // $\mathsf{ProtonVKD.SelfAudit}$

Message Sequence Diagrams

VKD.VerifyUpdate / ”VKD.VerifyEpoch”

Client		**Server**
Knows: $chainhash_{t-1}, \mathsf{pk}_{CA}, \mathsf{pk}_{CT}$		Knows: $cert_t, SCT_t, roothash_t$
	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\overbrace{(tbscert_t, tbscertsig_t)}^{cert_t}, \overbrace{(tCrtEntry, tCrtEntrysig)}^{SCT_t}, roothash_t}$	
Checks: $\{tCrtEntrysig\}_{\mathsf{pk}_{CT}} = tCrtEntry$	// Check SCT	
Checks: $snd(tCrtEntry) = h(\mathsf{pk}_{CA})$		
Checks: $trd(tCrtEntry) = tbscert_t$		
Checks: $\{tbscertsig_t\}_{\mathsf{pk}_{CA}} = tbscert_t$	// Check certificate	
$subject \leftarrow snd(tbscert_t)$	// Extract fields	
$notbefore \leftarrow trd(tbscert_t)$		
$chainhash_t \leftarrow fst(subject)$		
$isstime_t \leftarrow snd(subject)$		
Checks: $subject = (chainhash_t, isstime_t, t, ”1”)$	// Check subject	
Checks: $chainhash_t = h(chainhash_{t-1} \mid\mid roothash_t)$	// Check epoch chaining	
Checks: $\lvert notbefore - isstime_t \rvert \leq 24h$	// Check recentness	
Checks: $\lvert currentTime - isstime_t \rvert \leq 24h$	// Check recentness	
Checks: Tree correctly constructed/append-only		
Assert: $UpdateVerified(t, roothash_t, isstime_t)$		
Knows: $cert_t, \mathsf{pk}_{CA}, \mathsf{pk}_{CT}, cert_t, SCT_t, chainhash_t, roothash_t$		Knows: $cert_t, SCT_t, roothash_t$

Tamarin Prover

- Symbolic model, perfect crypto
- Dolev-Yao adversary, plus malicious Key Server
- Reason over an unbounded number of protocol executions

Formal Security Properties

Consistency:

$$
\begin{aligned}
&\forall A, B, pkB, epochid \;.\; QueryVerified(A, B, pkB, epochid) \\
&\quad \implies \Big(\nexists C, pkB' \;.\; QueryVerified(C, B, pkB', epochId) \wedge pkB \neq pkB'\Big) \\
&\quad \vee \Big(\nexists V, roothash \;.\; HonestAudit(V, epochid, roothash)\Big) \\
&\vee \Big(\exists V, W, roothash_v, roothash_w \;.\; HonestAudit(V, epochid, roothash_v) \\
&\qquad \wedge HonestAudit(W, epochid, roothash_w) \wedge roothash_w \neq roothash_w\Big)
\end{aligned}
$$

Formal (Security) Properties

Append-only-ness:

$$\begin{aligned}\forall V, epochid, roothash_t, roothash_{t-1}\ .\ & HonestAudit(V, epochid - 1, roothash_{t-1}) \\ & \wedge HonestAudit(V, epochid, roothash_t) \\ \implies & PublishedTree(roothash_{t-1}) \sqsubset PublishedTree(roothash_t)\end{aligned}$$

Formal Security Properties

Key detection:

$$\forall A, epochid_i, key, k \ . \ SelfAuditCompleted(A, epochid_i, key)@k$$
$$\implies \left(\exists epochid_j \ . \ InsertKey(A, epochid_j, key) \land epochid_j < epochid_i\right)$$
$$\lor SuspiciousKeyDetected(A, key)@k$$

Questions?