CONIKS
BRINGING KEY TRANSPARENCY TO END USERS

Marcela Melara

Aaron Blankstein, Joseph Bonneau*, Edward W. Felten, Michael J. Freedman

*Princeton University, *Stanford University/EFF*

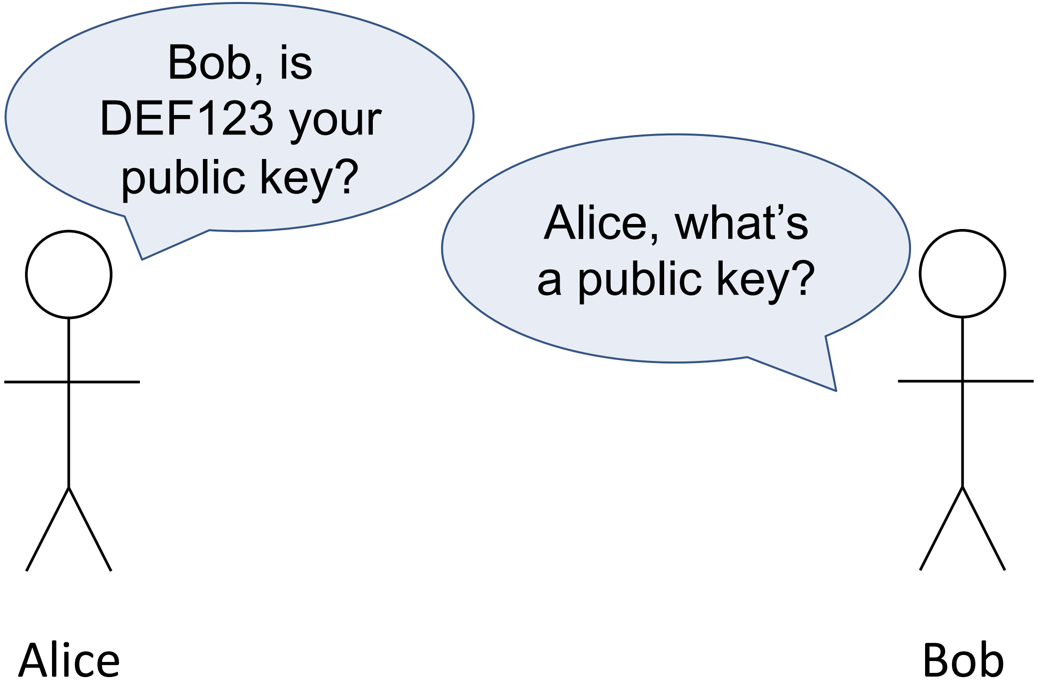
E2E Encrypted Communication Today

- Users' growing demand for E2E secure communication
- Known problem: Key management is difficult for users

Unsolved: How do users establish trust?

- Trust establishment = Learn & verify the other party’s key
- Goal: Establish secure communication channel

Out-of-Band Trust Est. = Unintuitive

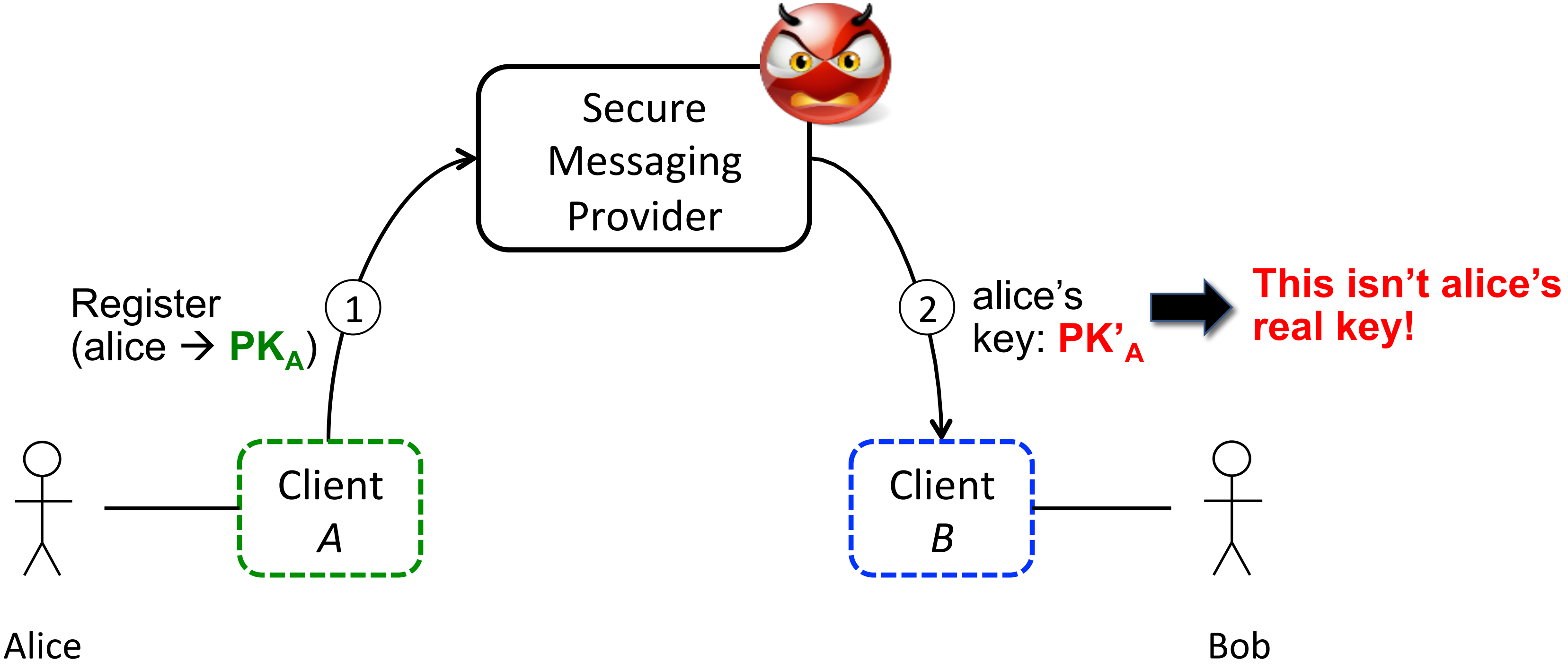


Requires users to reason about encryption/keys → unintuitive, error-prone!

Trust Est. by the Provider – Better?

- Clients query provider for others’ keys
- Users don’t worry about or see keys
- Caveat: Users must trust provider unconditionally

Malicious Provider can Equivocate



Equivocation = Presenting diverging views to different clients.

Pros/Cons of Existing Trust Establishment

	Users verify keys out of band	Providers establish trust for users
Security	✔	✖
Usability	✖	✔

Challenge: How can we get the best of both worlds?

Ideal Trust Establishment Properties

1. Security against equivocation attacks

2. Automation: Users don’t worry about trust establishment

Existing Approach: Verifying Correctness

- Correctness = Expected real-world person controls online name-to-public key binding
- Problem: Requires out-of-band communication

Our Approach: Verifying Consistency

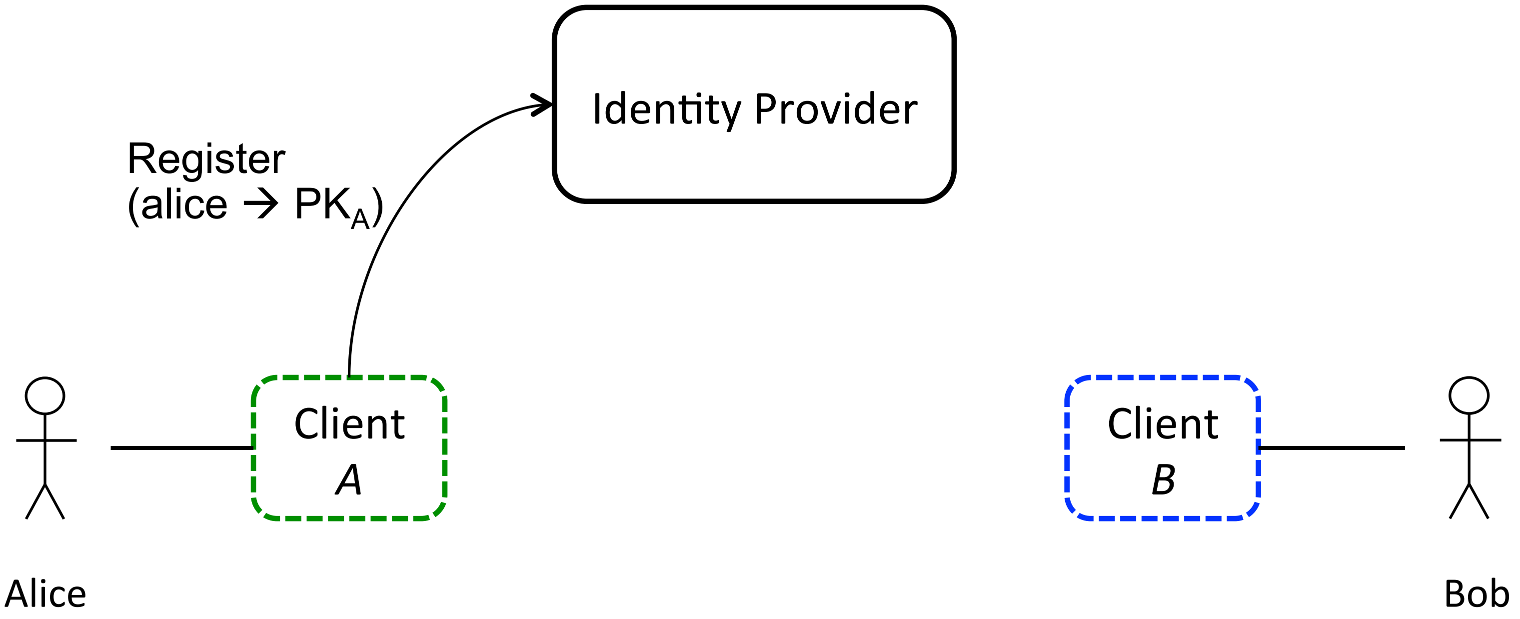
- Consistency =
  1. Alice's key today = Alice's key yesterday
  2. Alice's key seen by Alice = Alice's key seen by everyone else
- Benefit: Can be enforced via crypto

  → Providers manage consistent keys → Automation

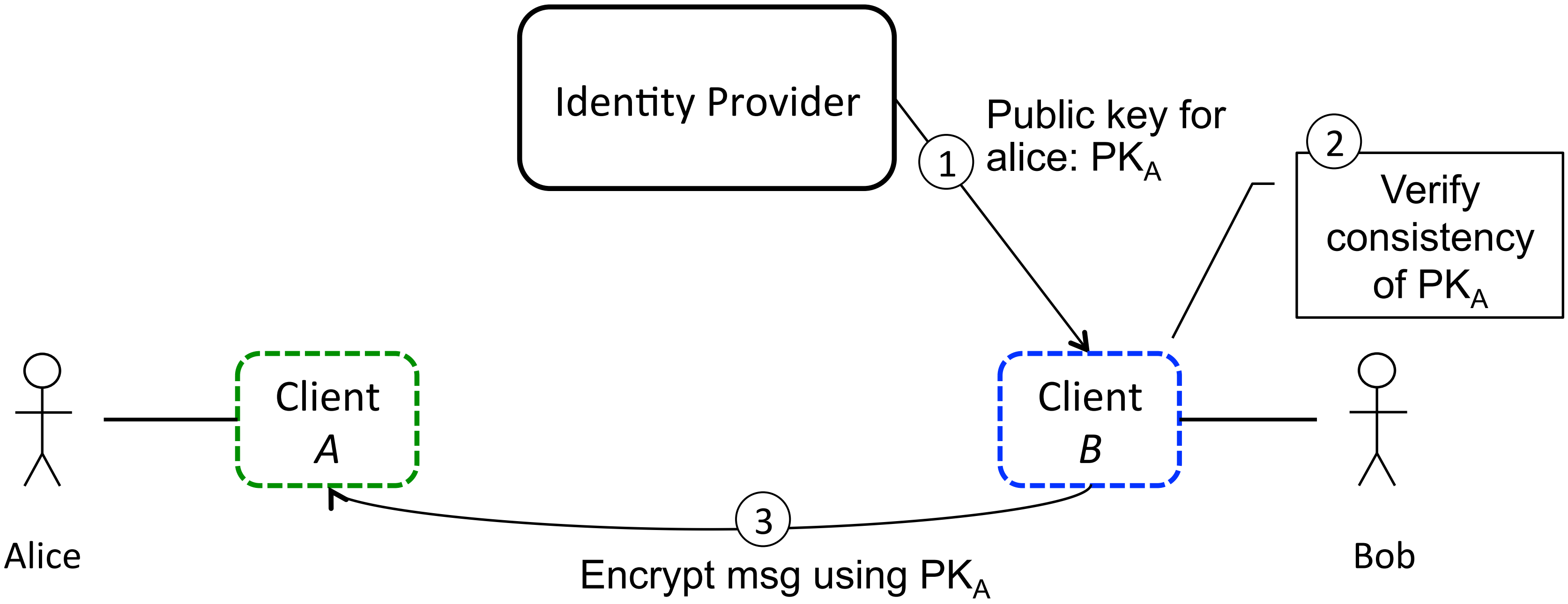
Solution: CONIKS

- Automated trust establishment with untrusted providers
- Clients verify consistency of bindings
- Goal: Make provider equivocation easily detectable

CONIKS – Registering a Key

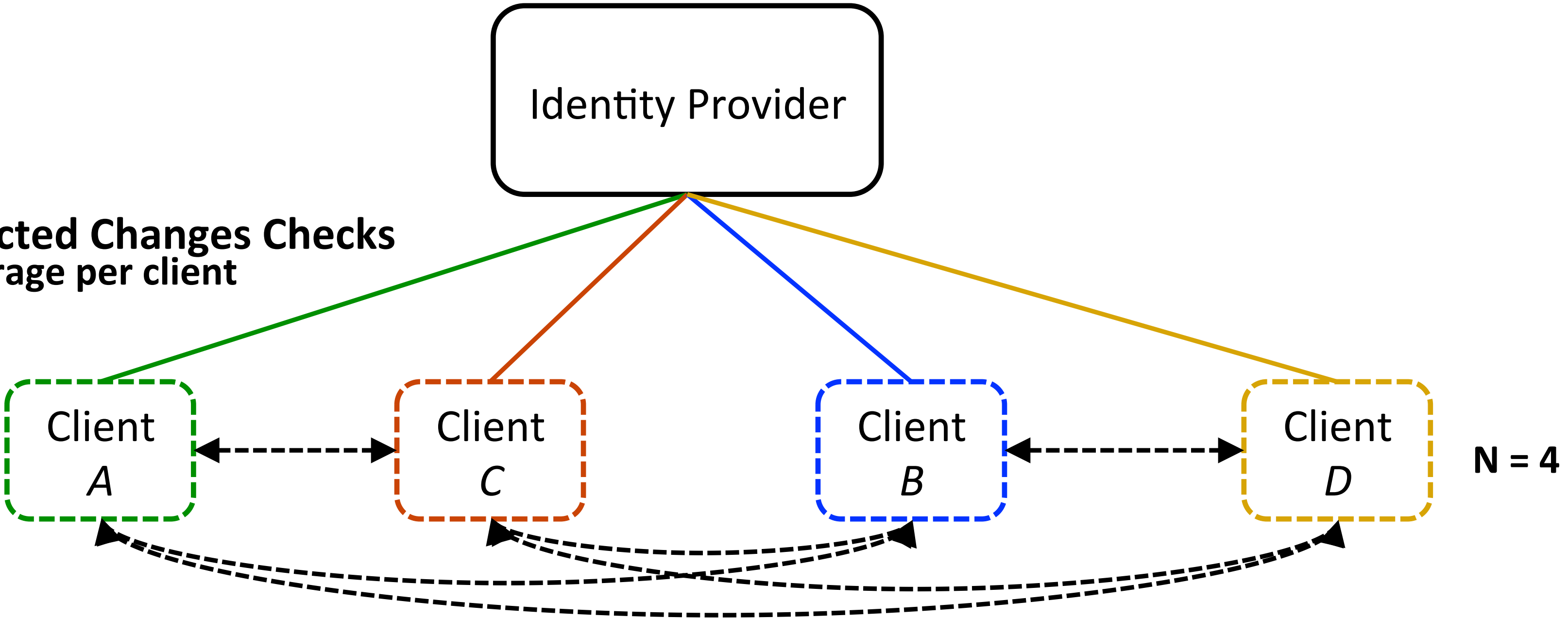


CONIKS – Learning a User's Key



Strawman Consistency Checks: Verify All Bindings

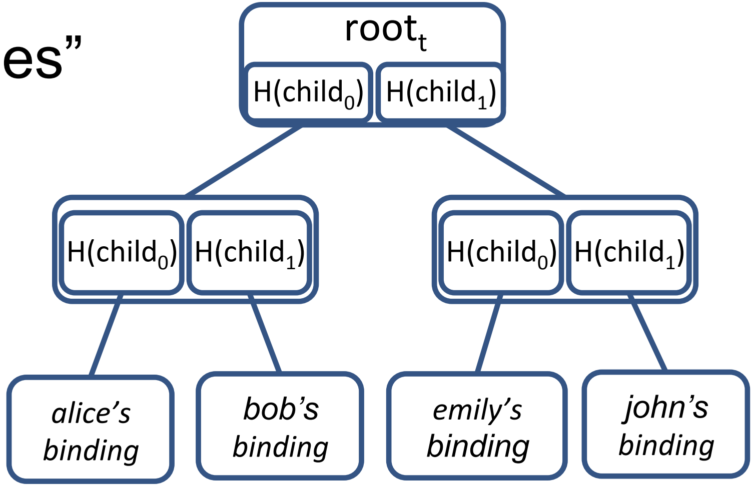
Unexpected Changes Checks
O(N) storage per client



Consistent View Checks
$O(N^2)$ downloads per client

CONIKS: Efficient Checks thru "Summaries"

- Providers generate directory "summaries"
  → Clients don't verify all bindings
- Bindings stored in Merkle prefix trees
  → Tree root = Summary of all bindings
  → Tamper-evident directory
- Non-repudiation: Signed tree root (STR)
  → Undeniable statement about tree contents



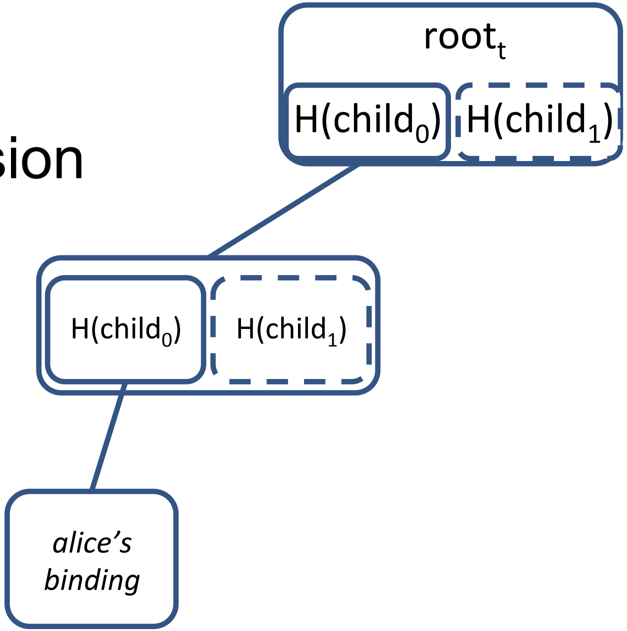
CONIKS – Main Security Properties

1. No Unexpected Key Changes: Expected Bindings included in Signed tree root

2. Non-equivocation = All clients see the same STR

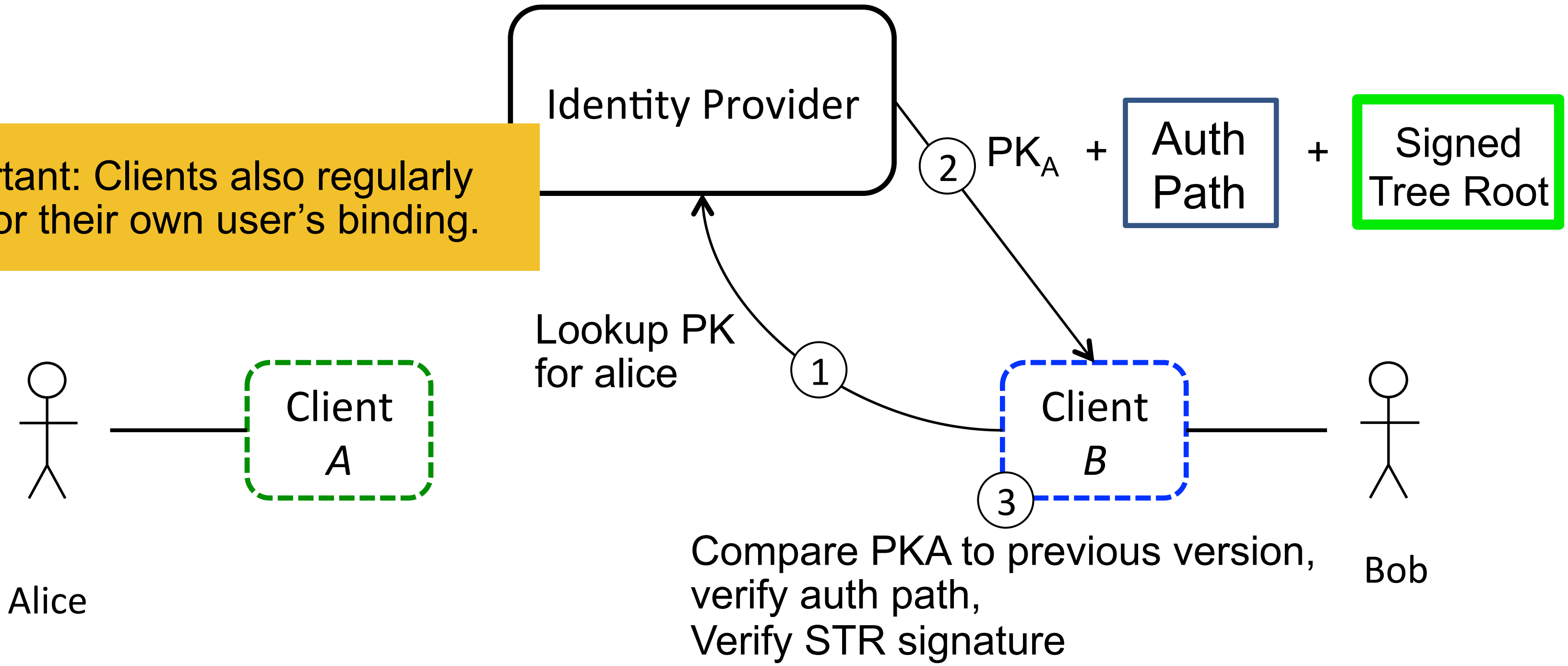
1. Expected Bindings incl. in STR – Auth Paths

- Why? Evidence for fake keys
- How? Authentication path = proof of inclusion
  → Pruned Merkle tree from binding to root
- Verification: recomputed root = STR
  → O(log n) for tree with n bindings

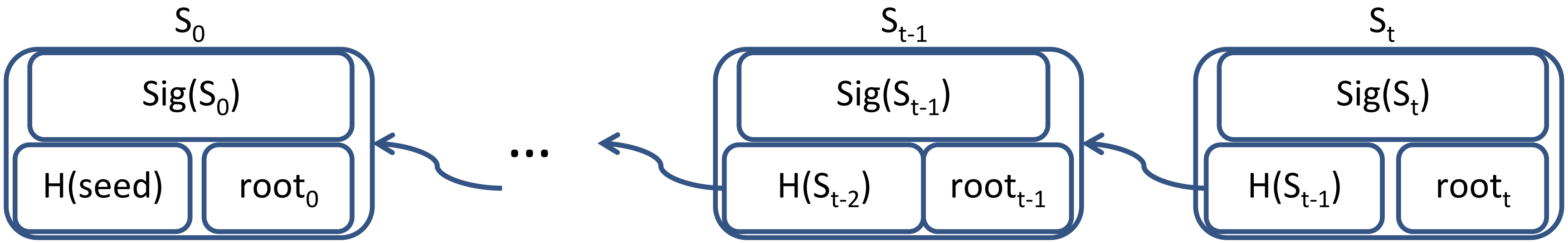


1. Checking Inclusion – Verify Auth Path

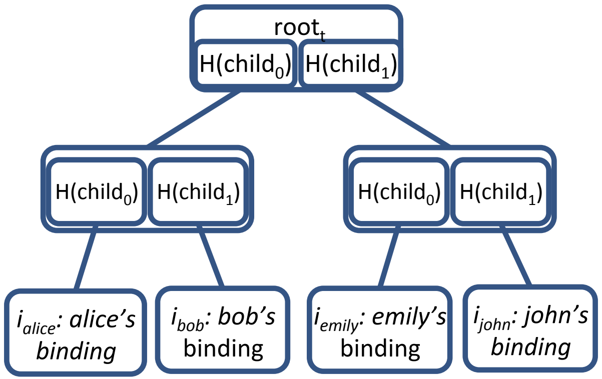
Important: Clients also regularly monitor their own user's binding.



2. Non-Equivocation – STR History

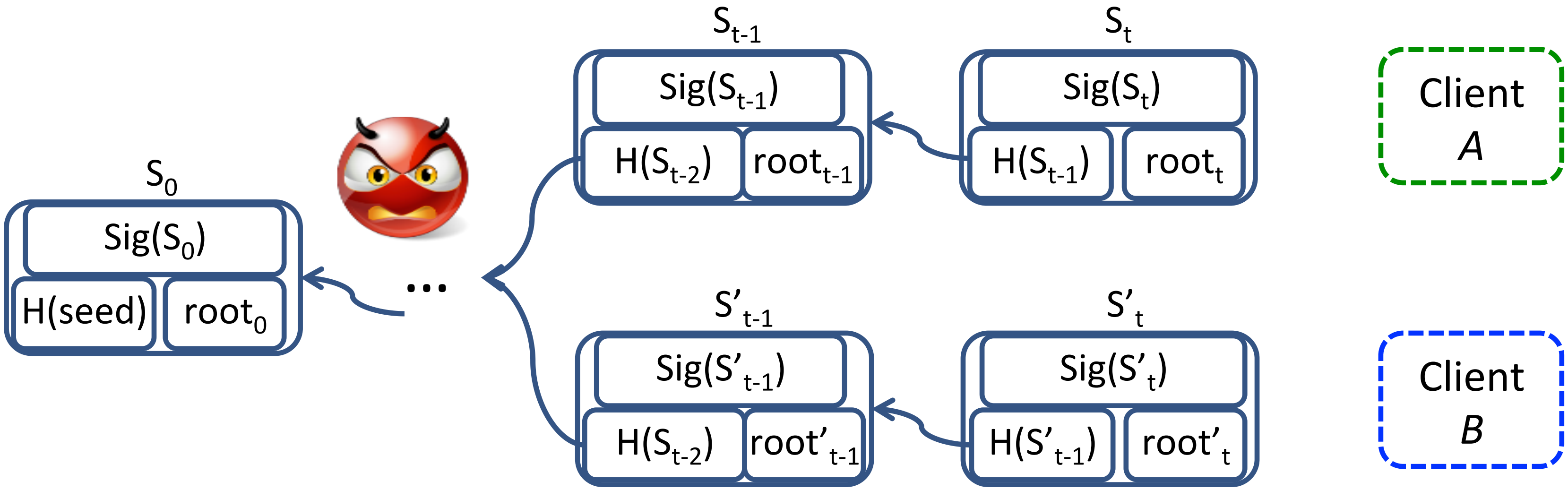


- Why? Detect provider attempt to MITM
- How? Building verifiable STR history
- Hash chain → commitment to all STRs
- Verification: previous STR is incl. in next STR

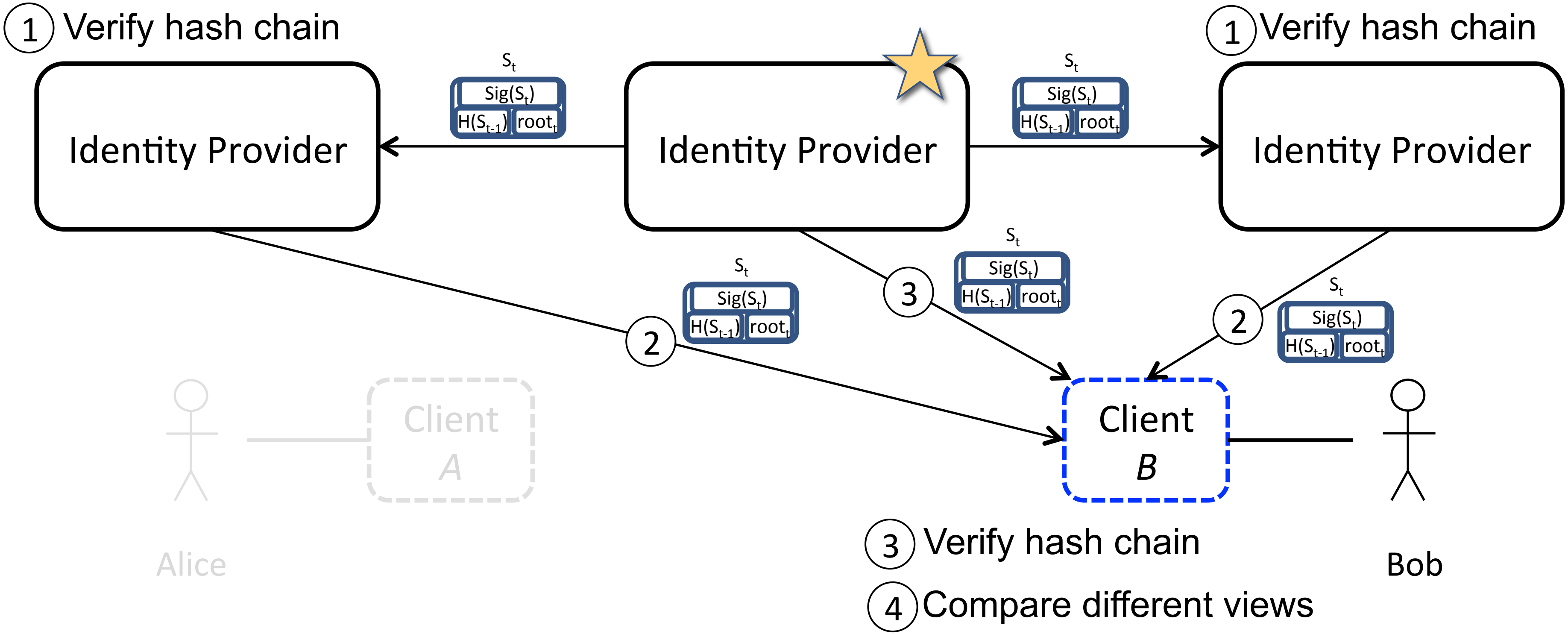


2. Non-Equivocation – Clients see same STRs

- Checking hash chain not enough:



2. Checking Non-Equivocation – Cross-Verification



Privacy Challenges in CONIKS

1. Don’t want to publish list of usernames

2. Don’t want to publish PKs associated with names

3. Don’t want to expose total # of users

→ Addressed through practical crypto tricks!

Main Performance Questions

- Does our server design scale to the size of a typical user base (thousands – billions)?
- Are CONIKS consistency checks efficient enough to run on today’s mobile devices?
- Does CONIKS integrate well with existing E2E services?

CONIKS' Performance is Practical!

- Server scales to tens of millions of users on single machine
  - Inserting 1K new bindings into 10M-user tree: 2.6ms
- Client consistency checks need little bandwidth/storage
  - Max. bandwidth requirements < 20kB per day
- Proof of concept: Integration with Pidgin OTR plug-in

Conclusion

- Main idea: Users should not have to manage keys, but service providers should not be trusted either.
- CONIKS: Security through consistency → more practical
- Yahoo & Google adopting CONIKS in their E2E systems

Q&A

More Info:

Website: www.coniks.org

Ref. Implementation: github.com/coniks-sys

We thank:

Yan Zhu (Yahoo)

Gary Belvin (Google)

Trevor Perrin (TextSecure)

David Gil (formerly Yahoo)