[USENIX Security ’15, Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, Michael J. Freedman]

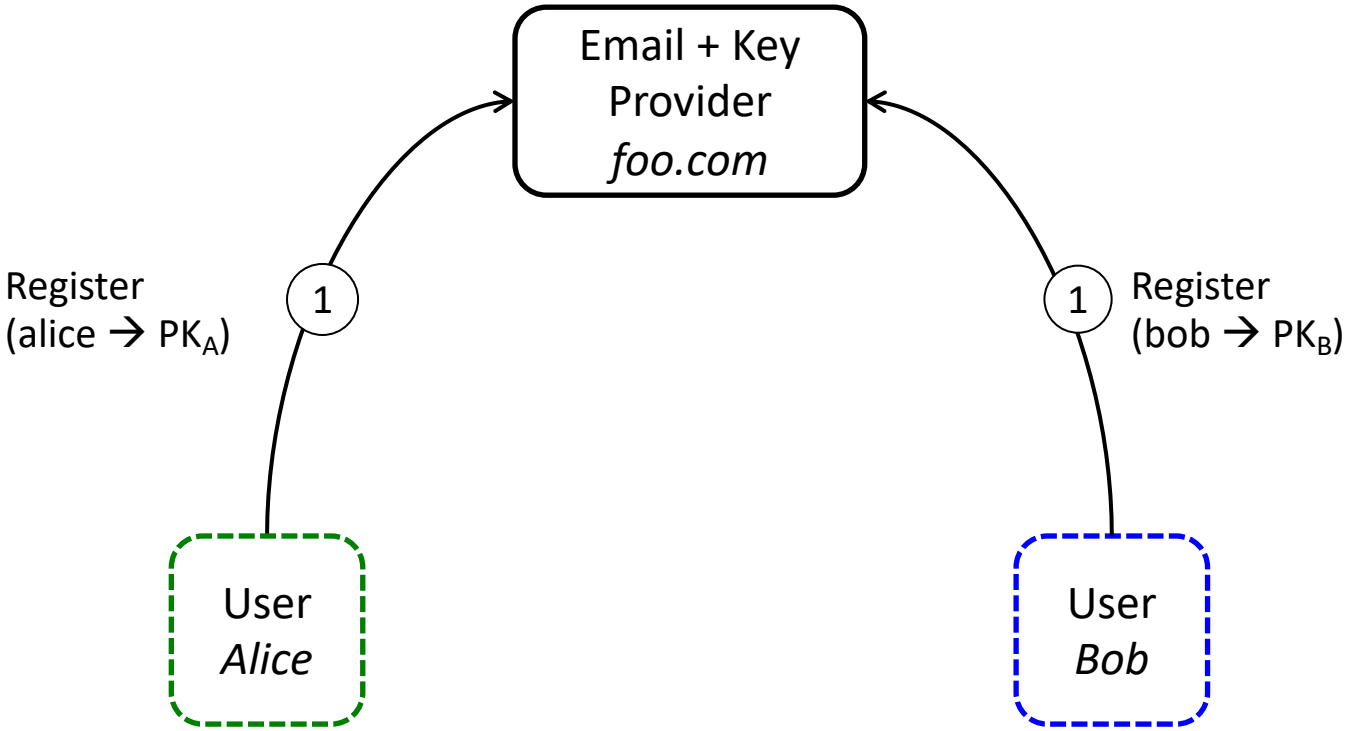
CONIKS (KEY TRANSPARENCY)

Slides adapted from Marcela Melara

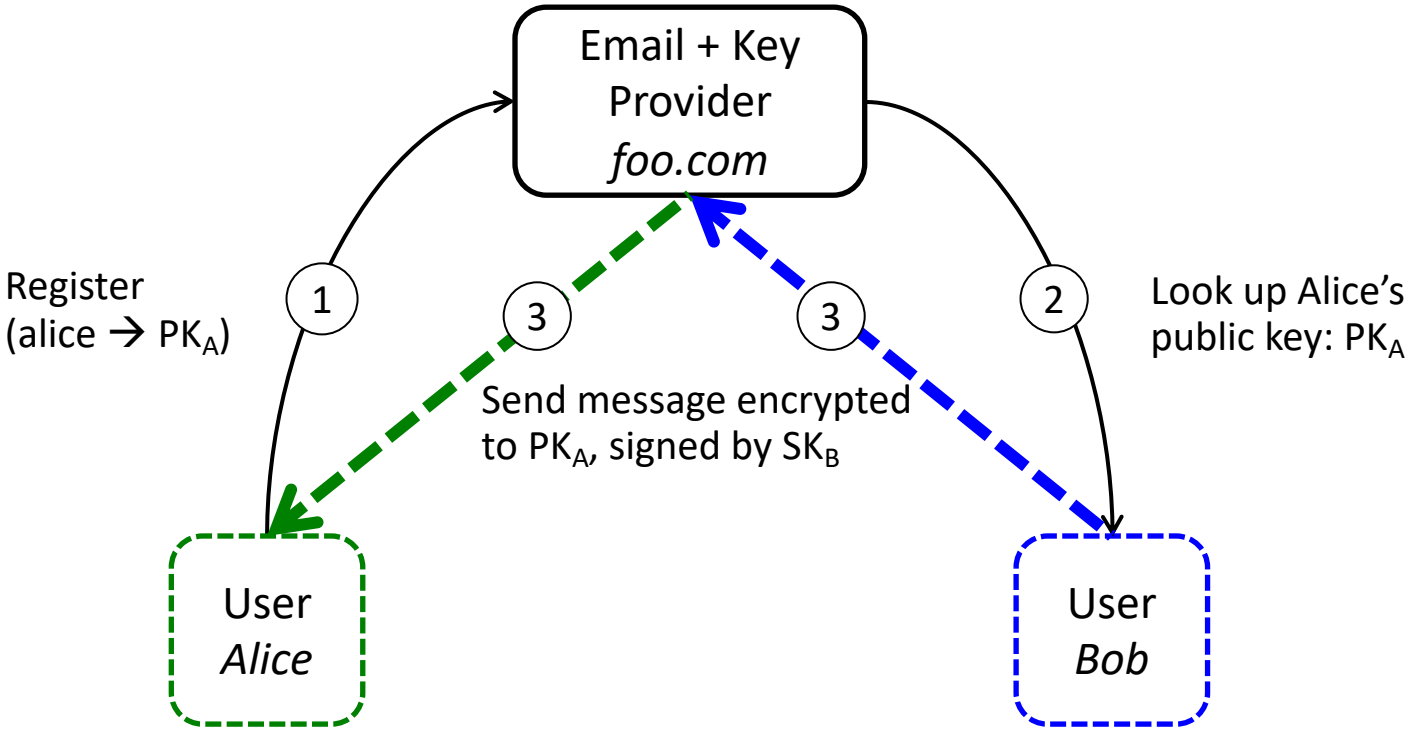
The problem of the PKI (Public key infrastructure)

- A long standing problem has been to distribute public keys securely in the presence of attackers
- Use cases:
  - Secure messaging: Alice needs to send an encrypted message to Bob and needs Bob's public key to encrypt the message
  - Web surfing and https: when Alice's browser contacts amazon.com over https, we need Amazon's PK
- Man in the middle attacker can intercept PK and return incorrect PK

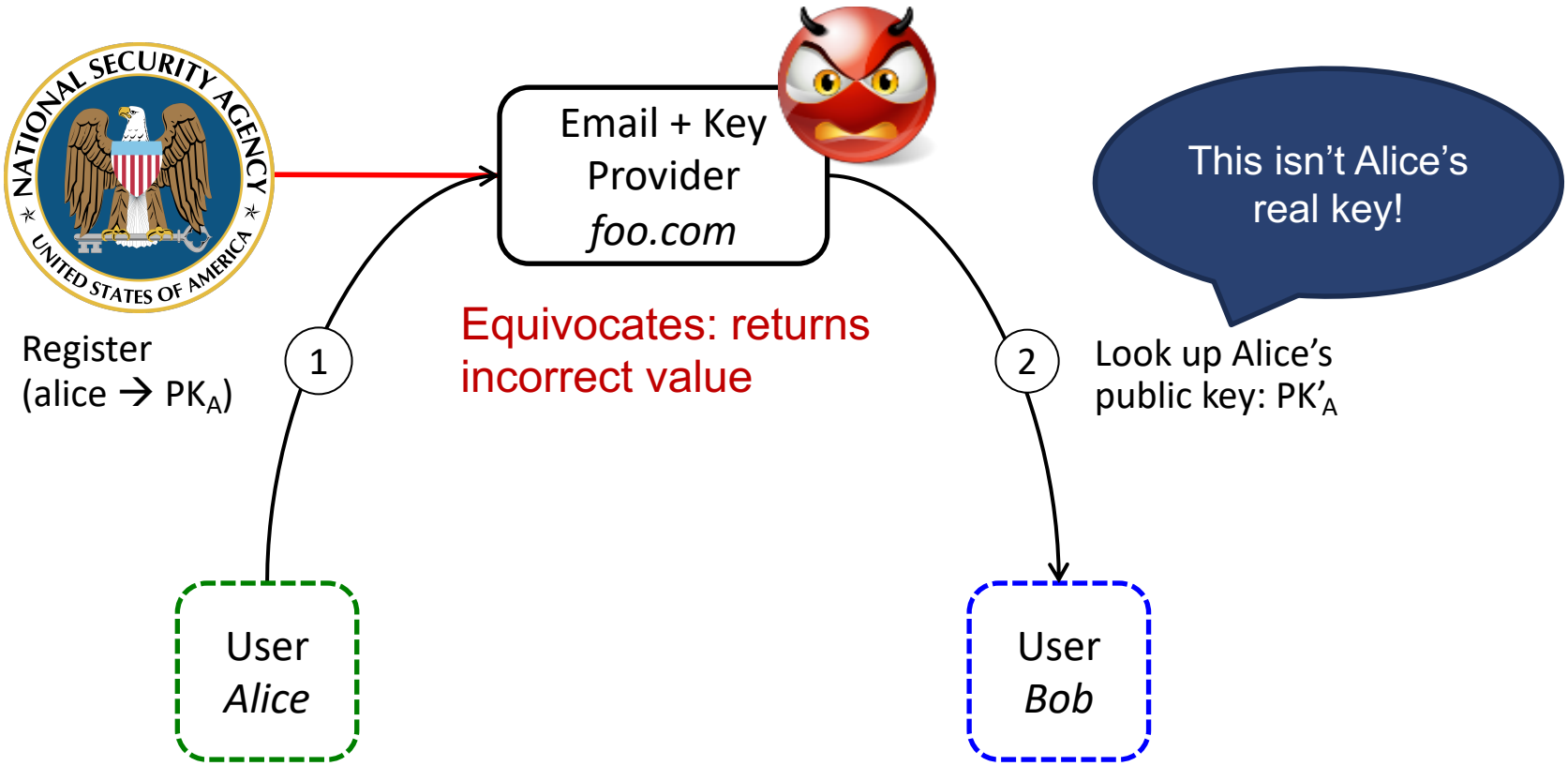
Centralized Key Management



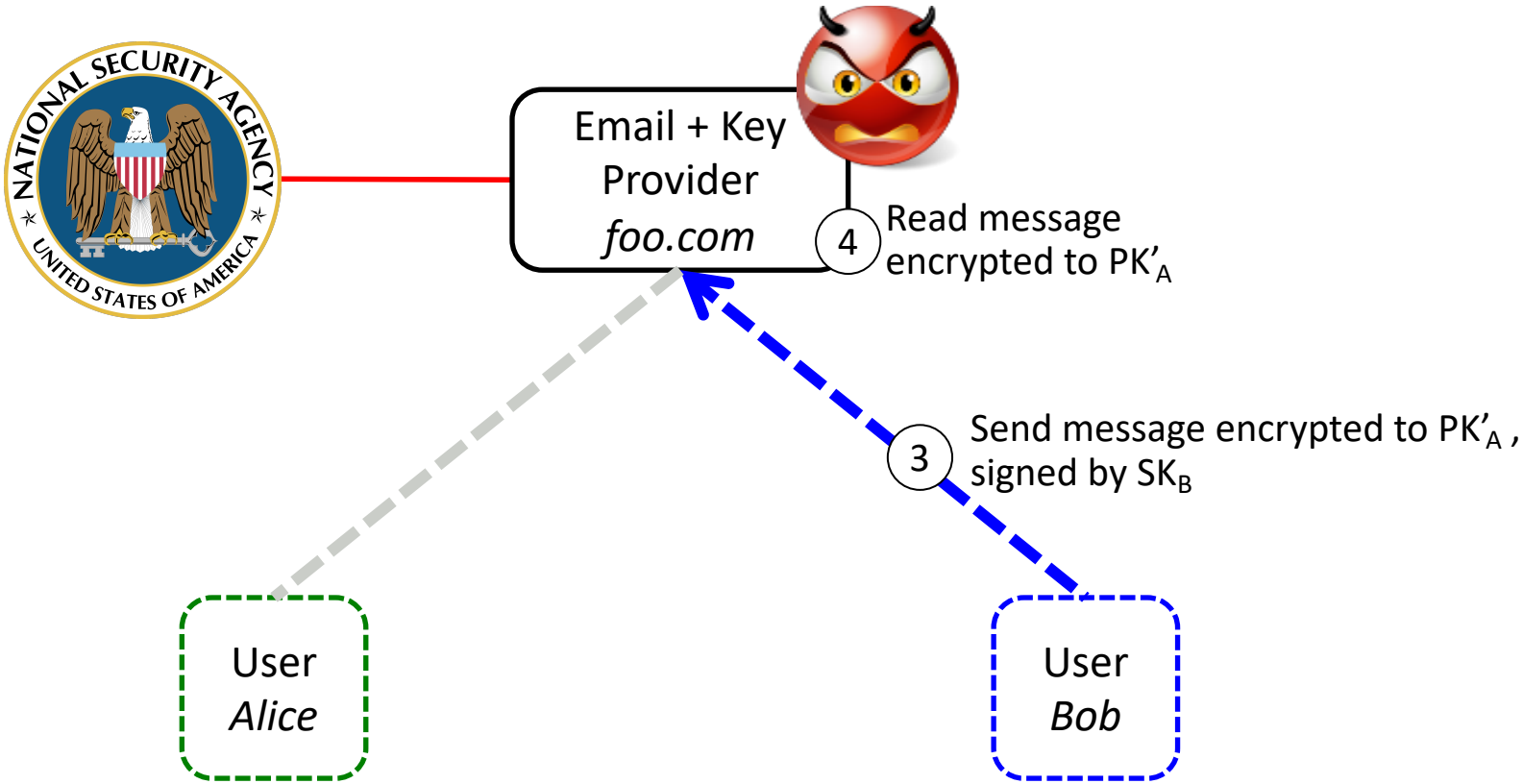
Centralized Key Management



Still, Key Server Compromise



Still, Key Server Compromise



KT’s Approach: Consistent Identities

- Users expect consistency of online identities.
- Certificates make no statement about correctness.

Consistency

No unexpected key changes

Alice’s key today = Alice’s key yesterday

Non-Equivocation

Key seen by Alice = Key seen by Bob

Solution: a promising new generation

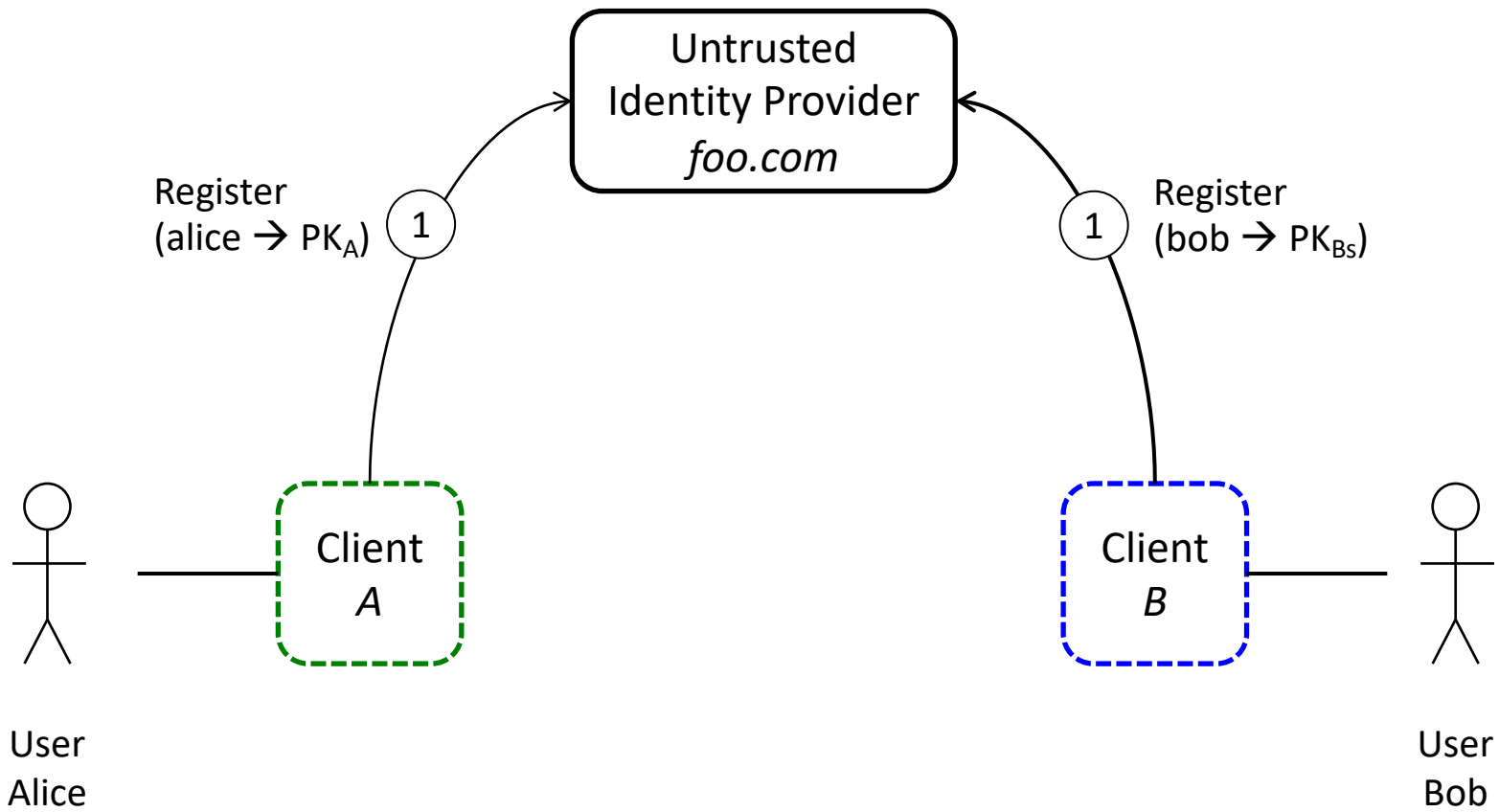
- CONIKS
  - An alternative Public Key Infrastructure (PKI)
  - Embedded in Google's project called Key Transparency or Trillian
    - The key data structure is the Log-backed Verified Map
- Certificate Transparency
  - Similar technology to Key Transparency but aimed at certificates
  - Currently mandated by Chrome, supported by Firefox and others

These are essentially blockchains, with decentralized security but hosted centrally

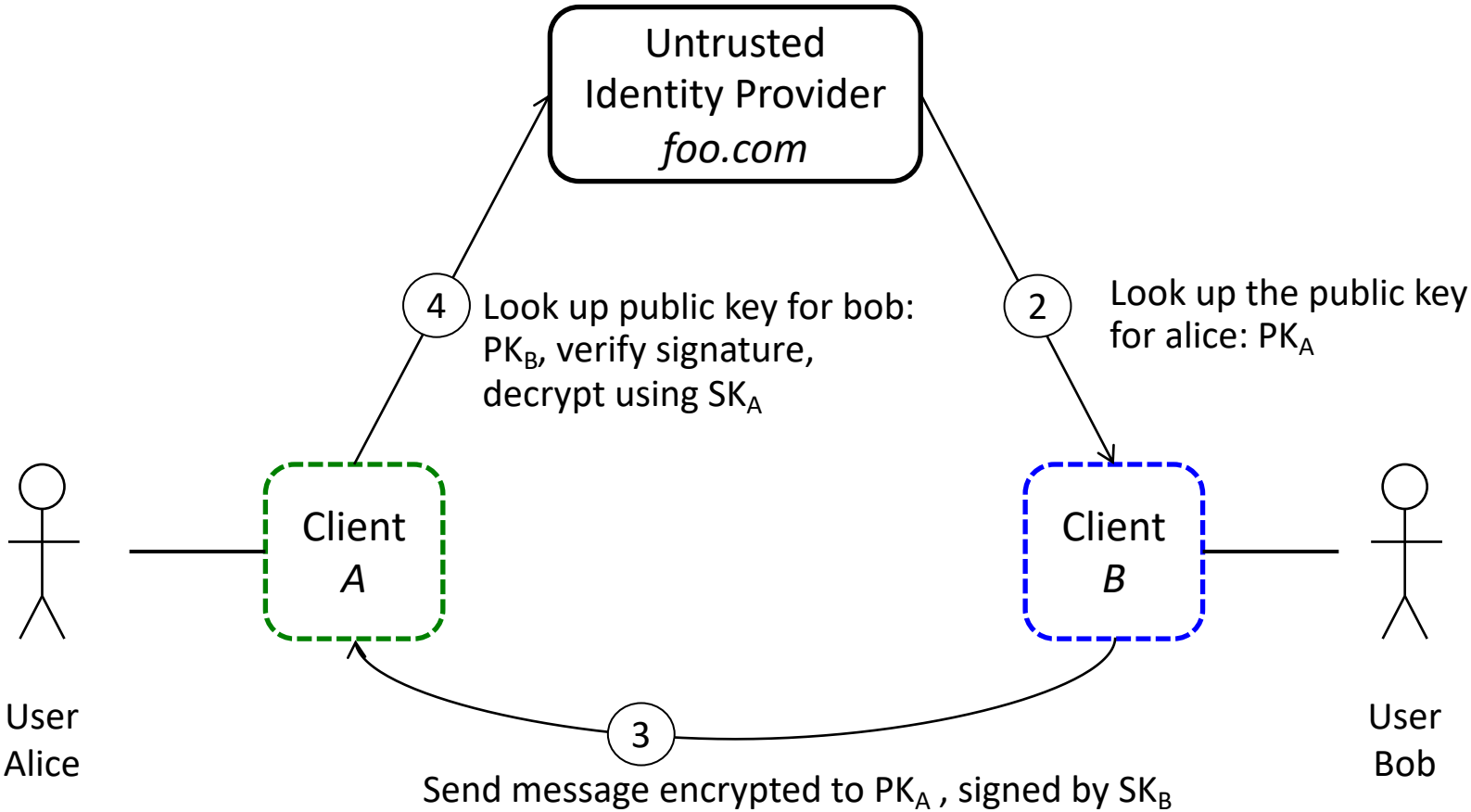
CONIKS [Melara et al. 2014]

- End-User Key Management Service.
- Consistent name-to-key bindings.
- Verifiable key directories → Untrusted identity providers.
- Clients verify consistency in-band.
  - →automated key management

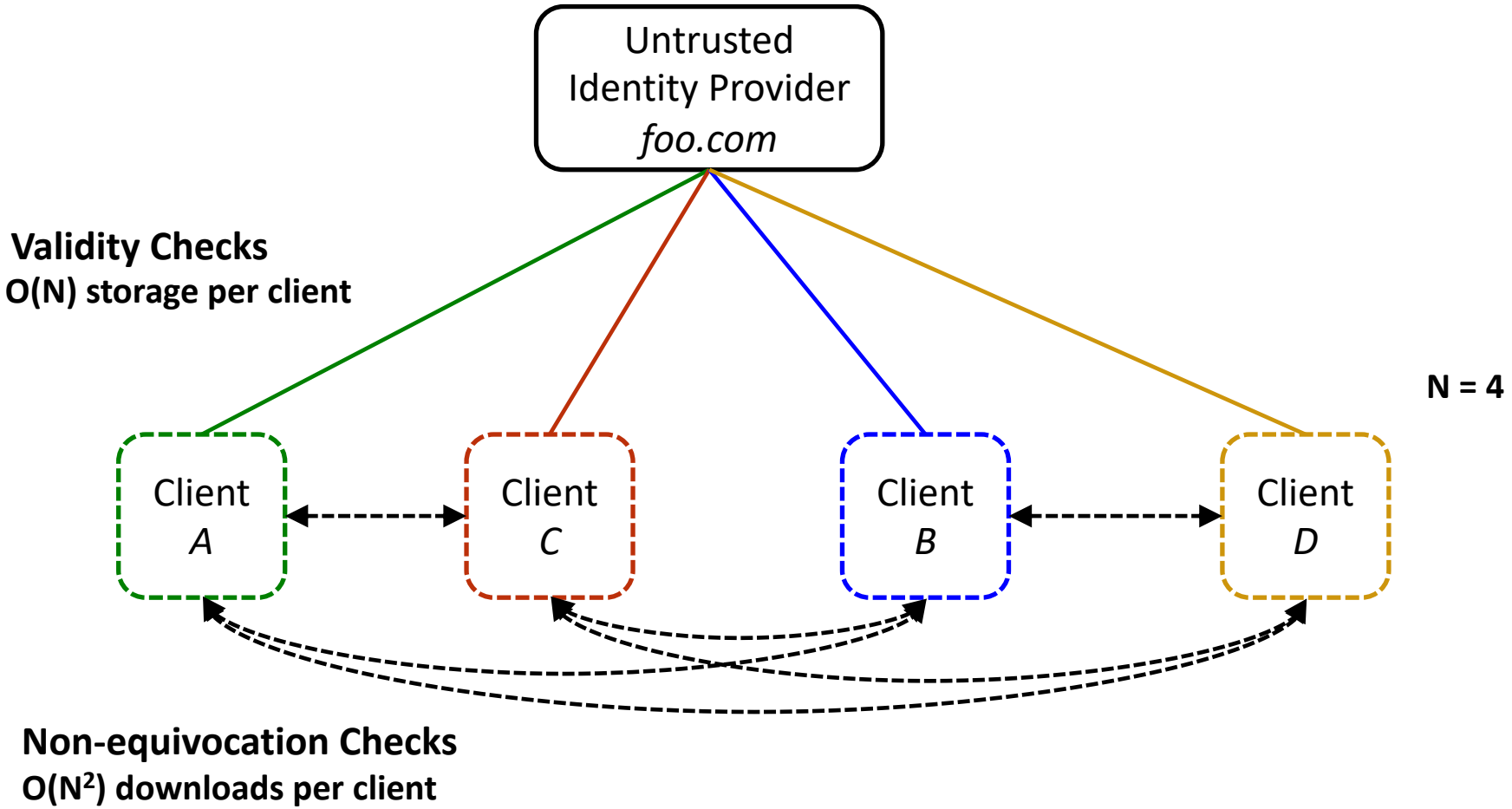
CONIKS Overview



Using CONIKS for E2E Encryption



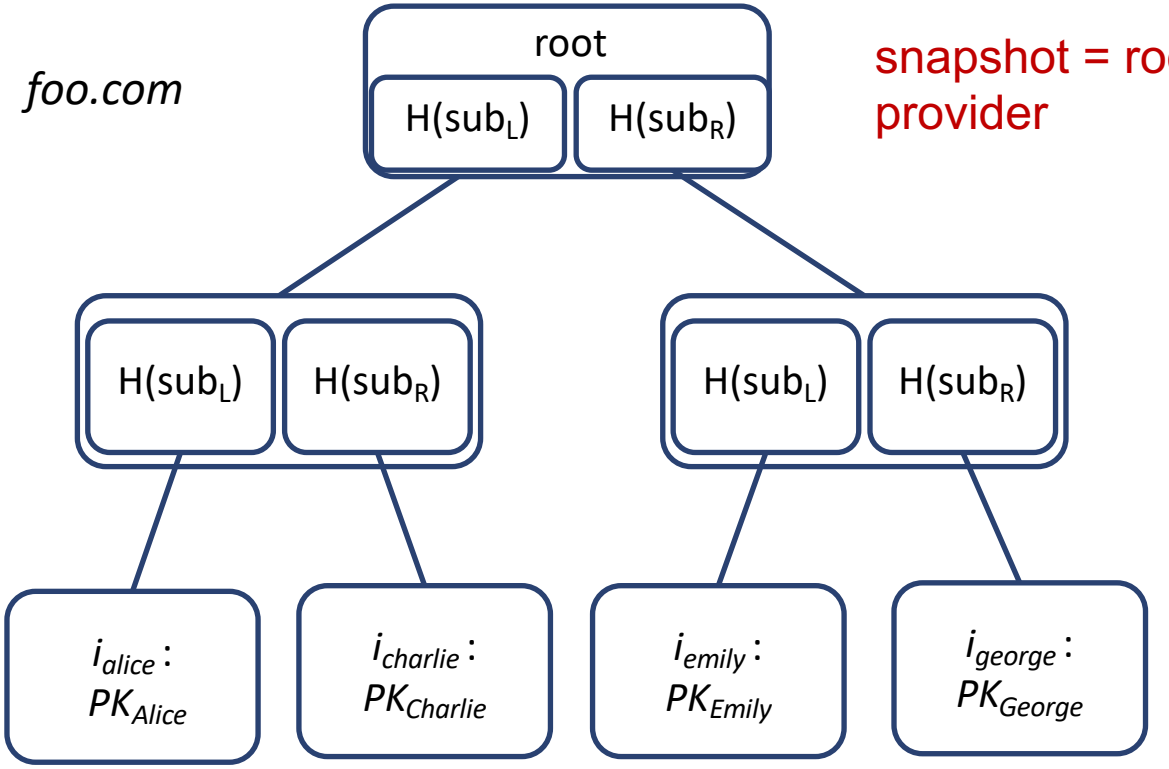
Strawman Design



CONIKS Design

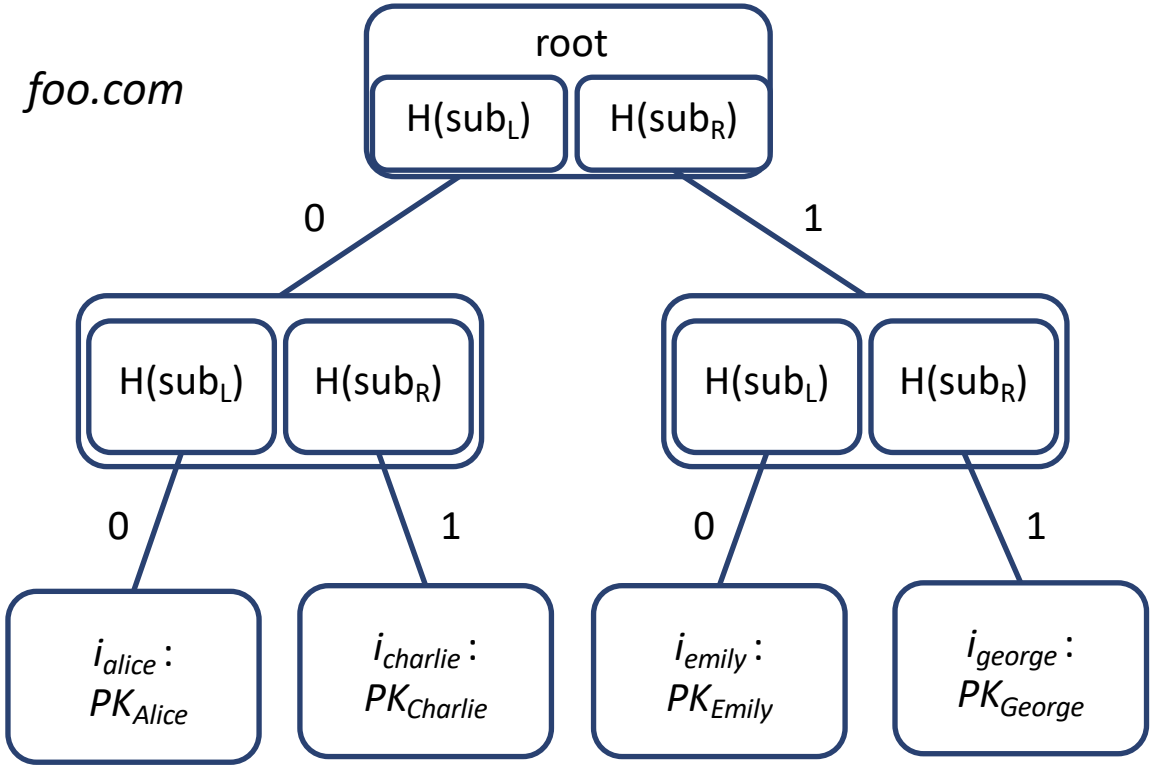
- Divide time into epochs.
- Providers generate snapshots of directory.
  → Clients do not check individual bindings.
- Providers distribute snapshots to other providers.
  → Build publicly verifiable history
- Non-repudiation: Snapshots are digitally signed.

Efficient Snapshots



snapshot = root signed by provider

Efficient Snapshots



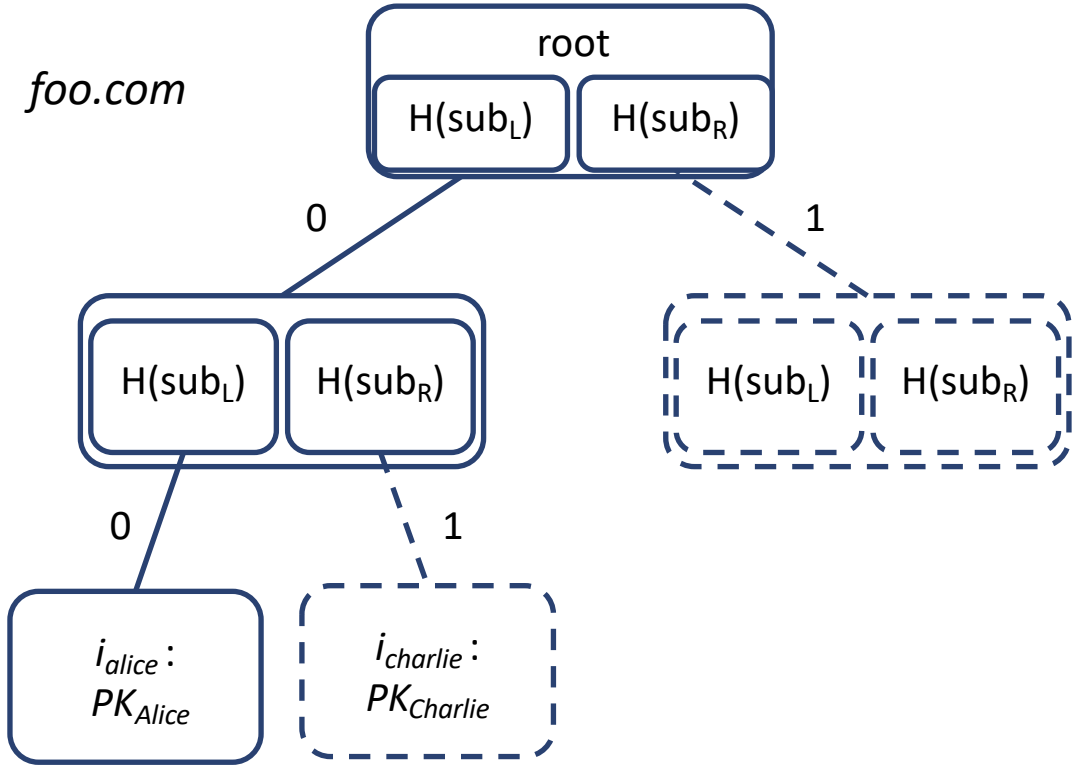
username -> index; arrange usernames based on index in the leaves

Checking Consistency

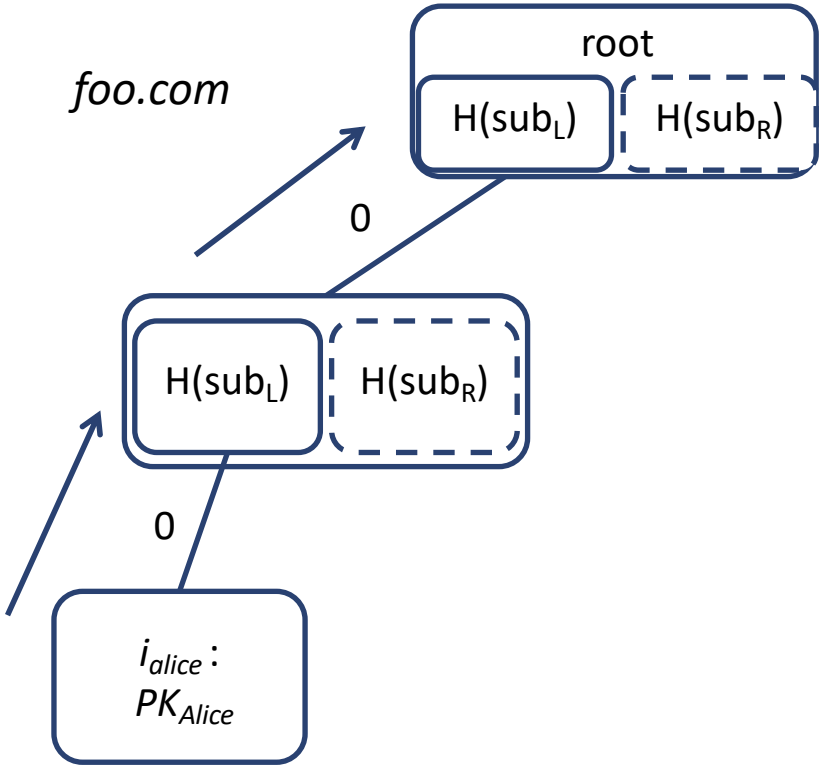
- Use snapshots to check for consistency.
- Clients need to ensure bindings are included in snapshots.
  - Lookups: prove inclusion of bindings in directory.
- Clients check validity, non-equivocation.
- Providers cross-verify each other for non-equivocation.

Proving Inclusion: Authentication Paths

A user can ask the server to prove inclusion of a binding (i->PK) against a signed snapshot: this can be the owner of the binding, or a user wanting to fetch the PK for i



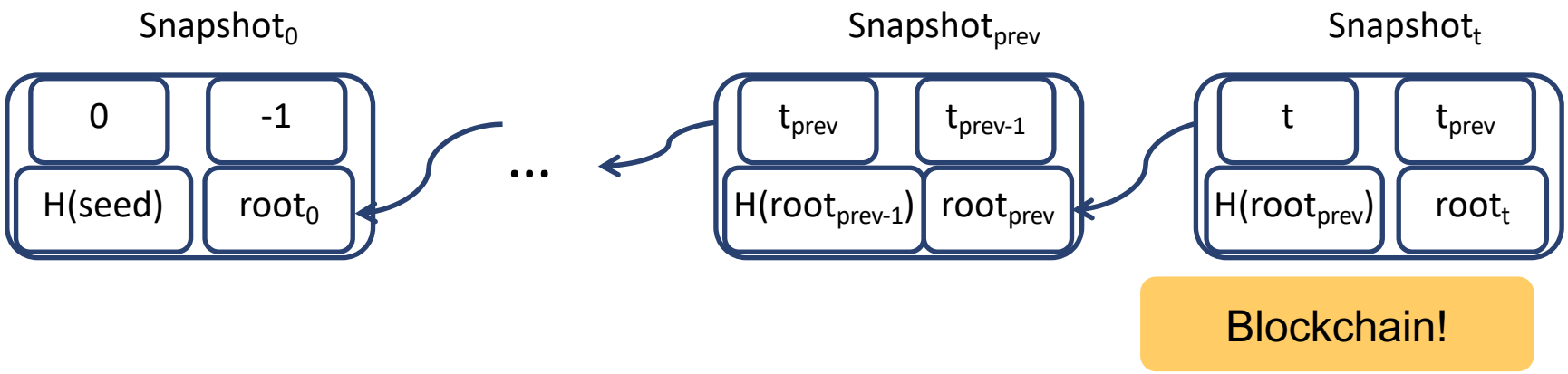
Proving Inclusion: Authentication Paths



What we have so far

- A client, say Alice can check that her binding is correct in a signed snapshot
- Another client fetching Alice's key can check Alice's binding against the signed snapshot
- What is missing?
  - Provider can equivocate about the signed snapshot (e.g., give different signed snapshots
  - How to address this?

Checking Non-Equivocation: Snapshot History



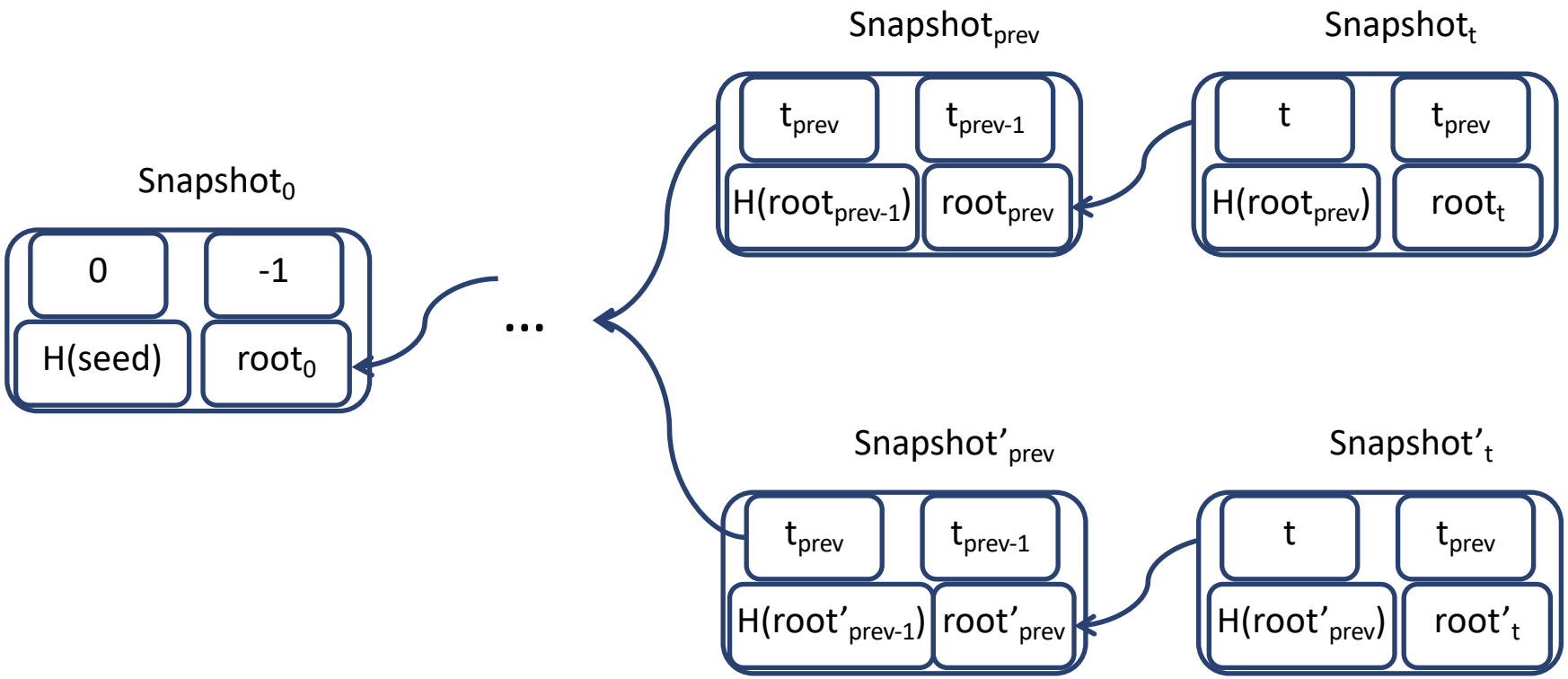
Trillian calls this: Log-backed Verified Map

The provider gives the signed last snapshot to other providers, who check that it is well-formed (e.g., it contains a hash of the previous snapshot) and cross-check that they all received the same snapshot

Why do we need to chain the hashes?

- Otherwise, providers need to exchange the whole history to check for consistency
- With a hash chain forks are not possible
- Why?
  - Collusion resistance of the hash

Checking Non-Equivocation: Snapshot History



The server can try a fork attack, but after fork the provider must maintain these forked hash chains for the rest of time, and not allow clients seeing one branch of the hash chain to communicate with anyone seeing the other branch.

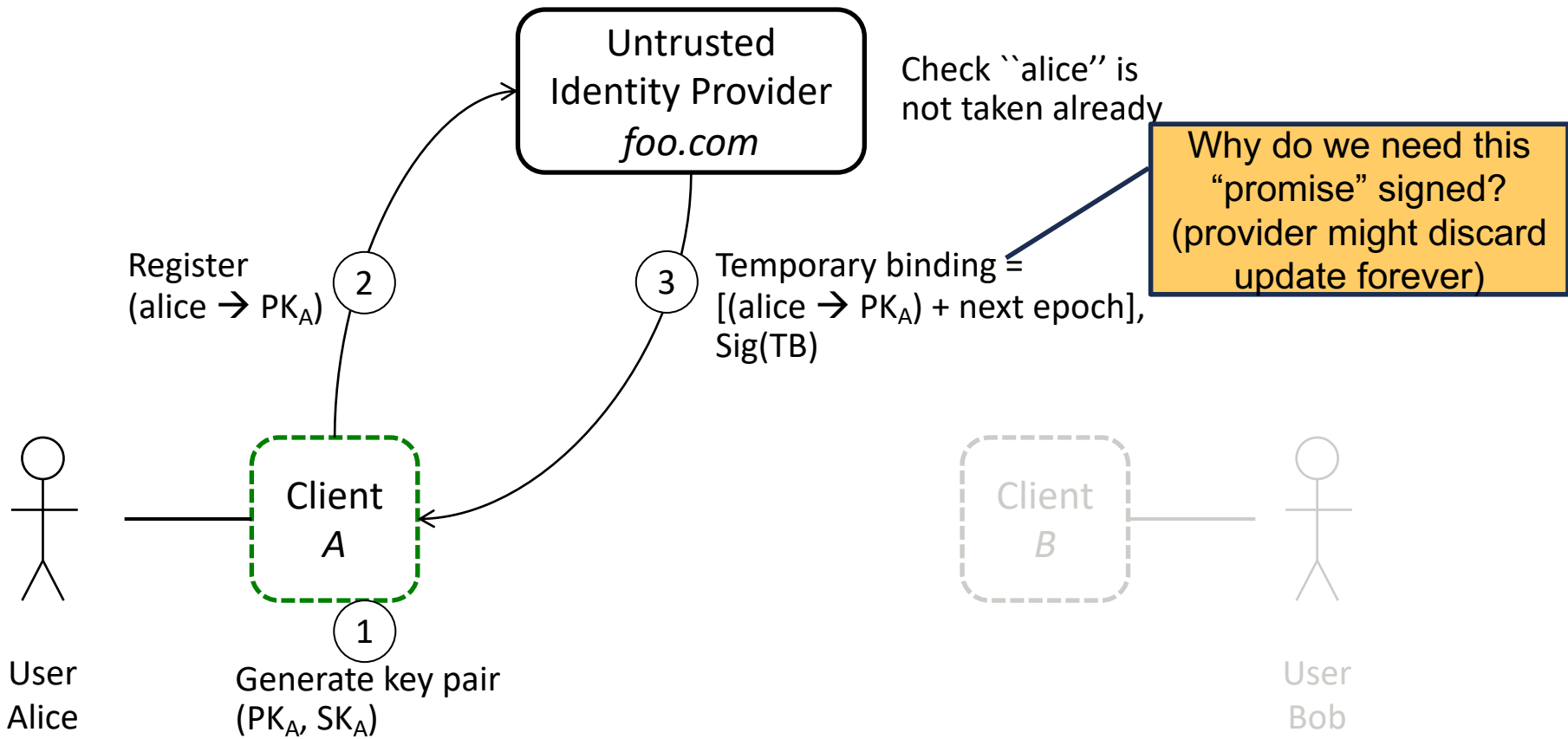
CONIKS Protocols

- Registration: New bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.

CONIKS Protocols

- Registration: New bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.

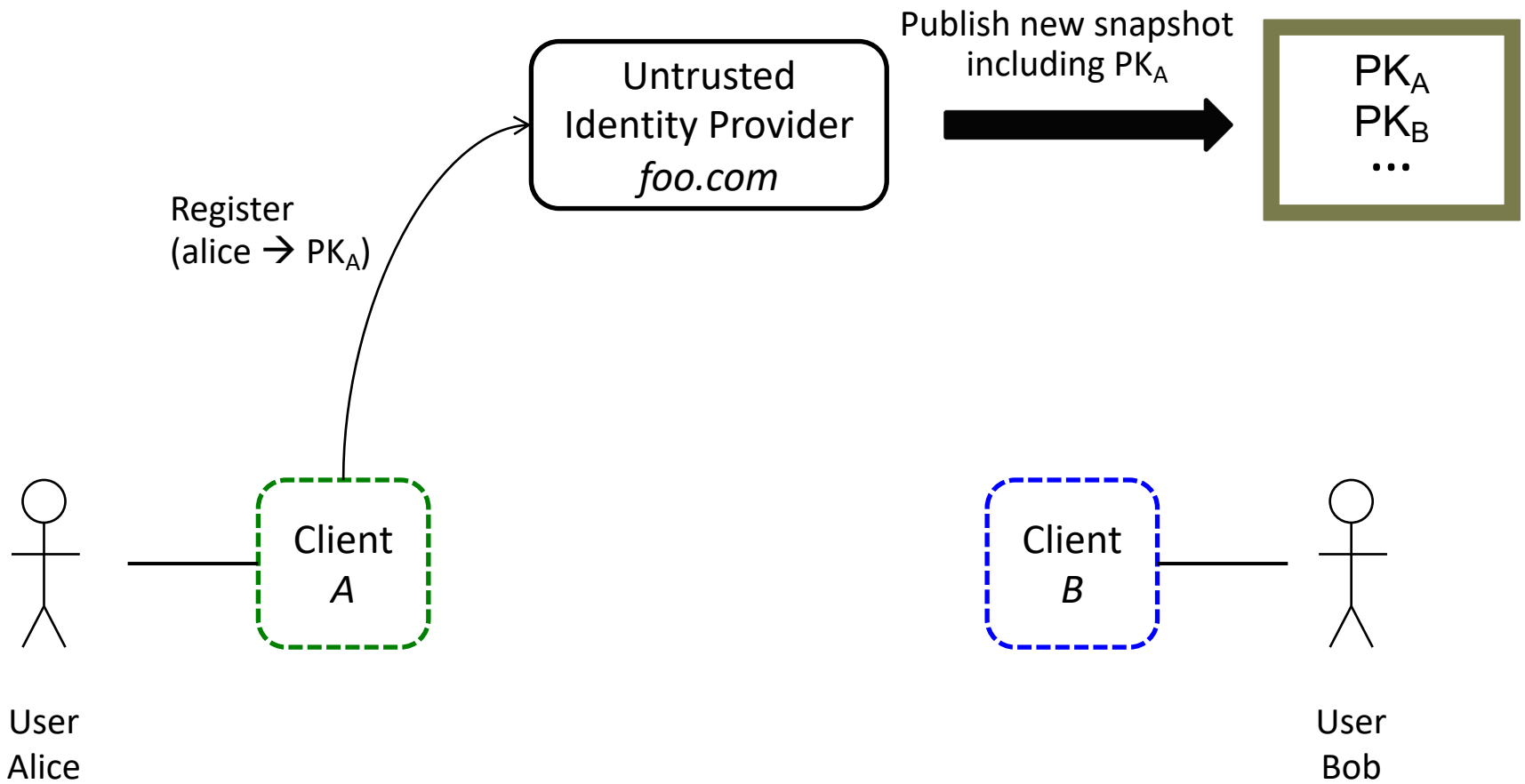
Registration Protocol



CONIKS Protocols

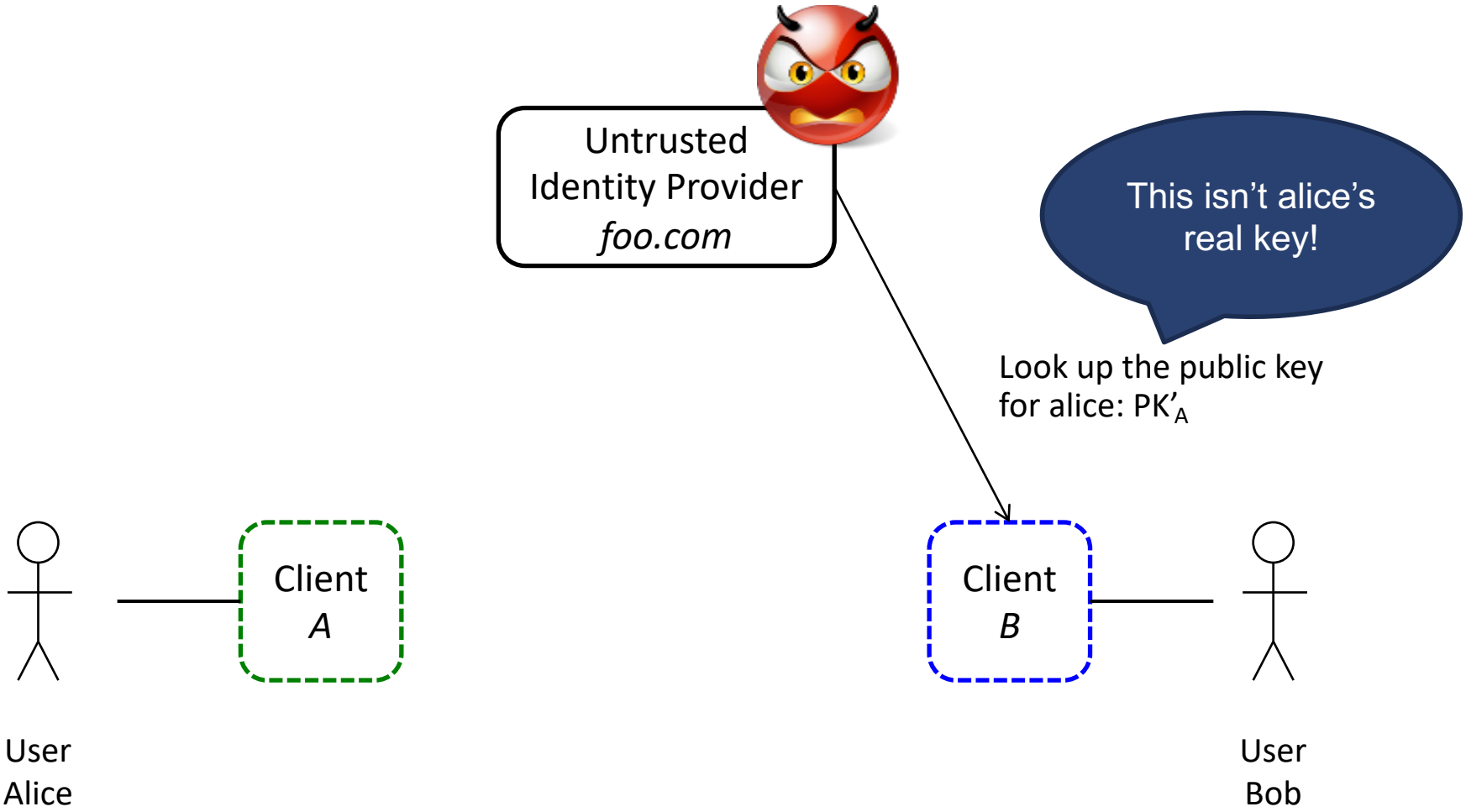
- Registration: new bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.

Lookups without Inclusion Proofs



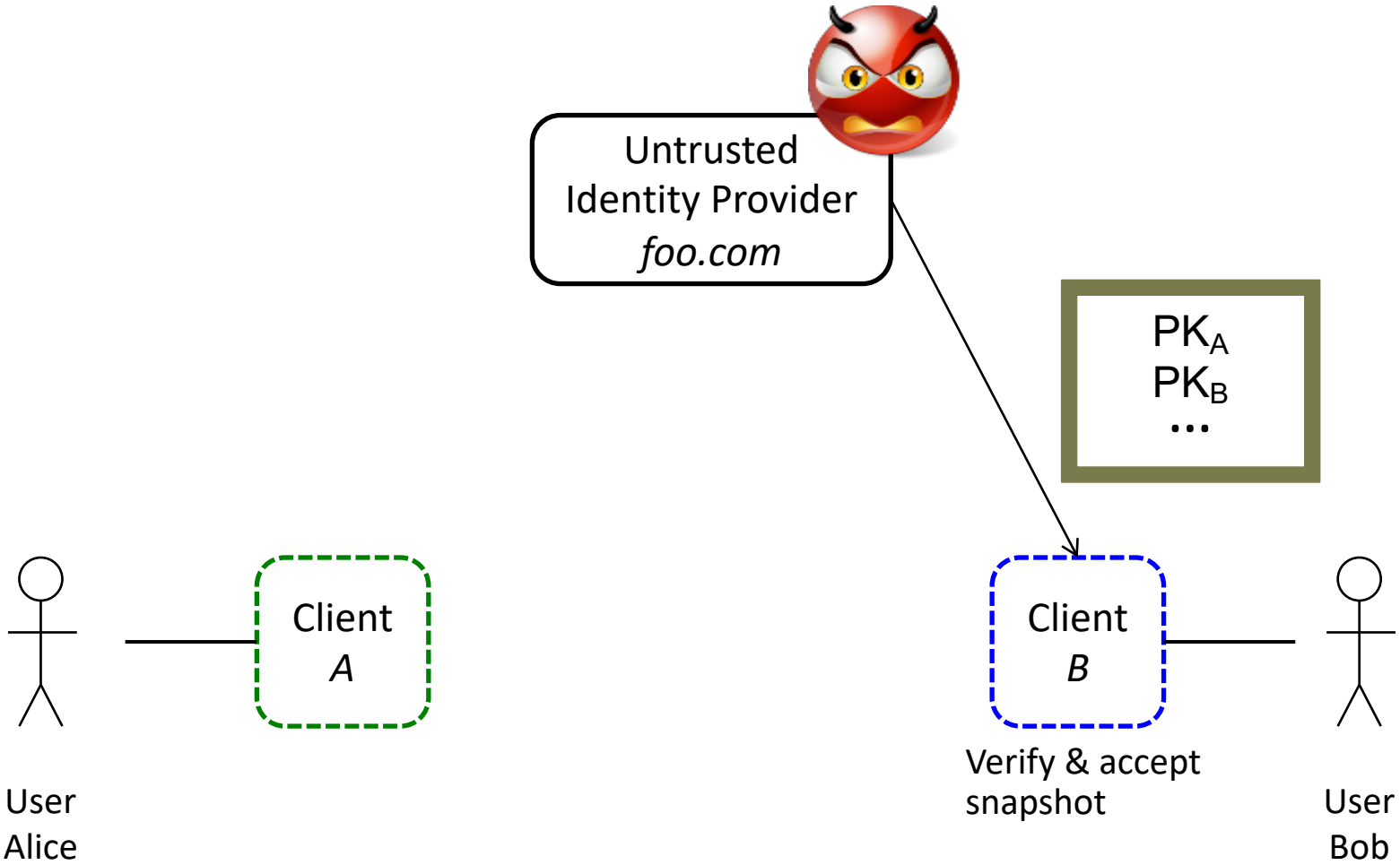
Provider regularly generates and publishes snapshots

Lookups without Inclusion Proofs



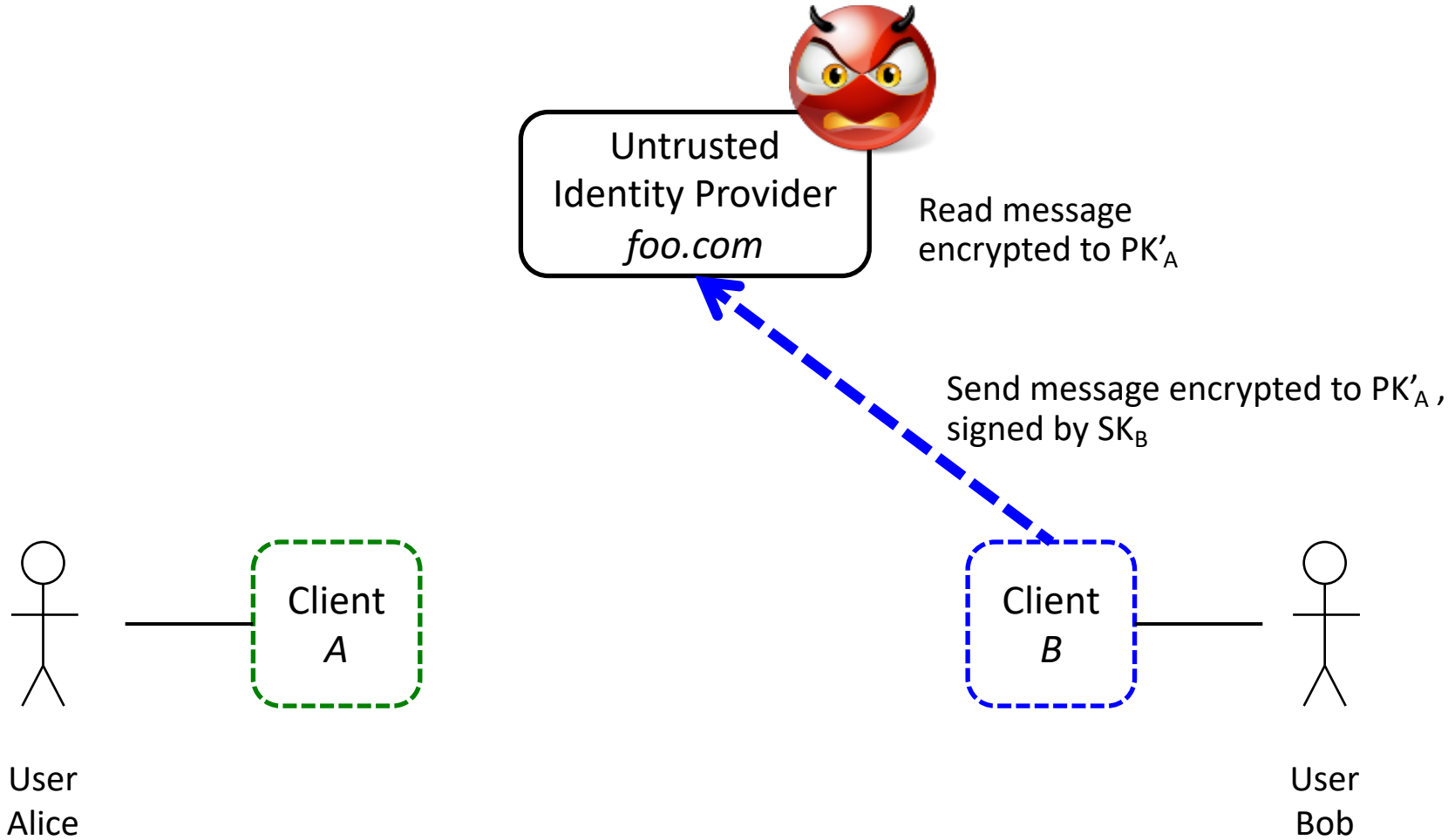
Return Fake Key

Lookups without Inclusion Proofs



No proof fake key is inconsistent with snapshot

Lookups without Inclusion Proofs

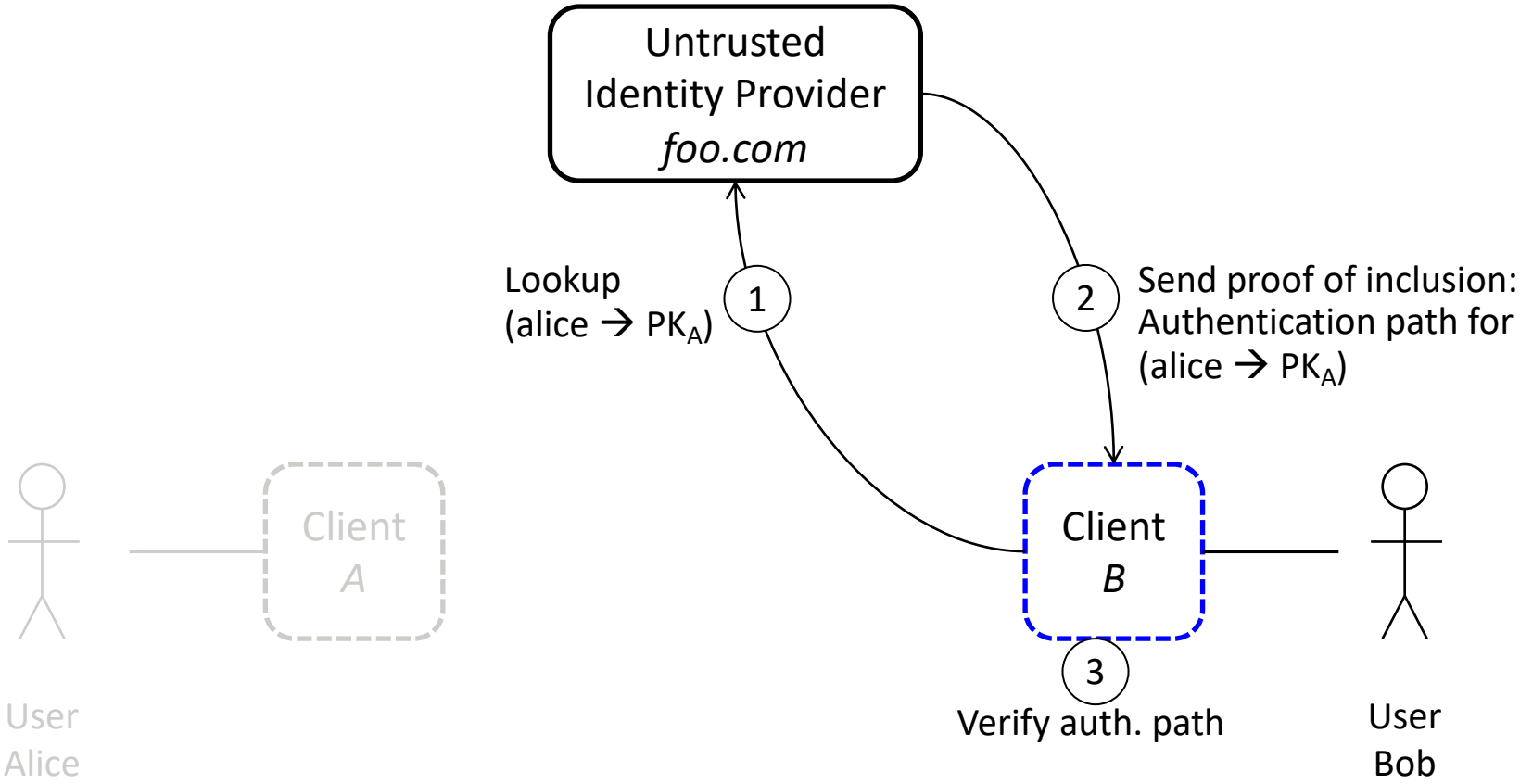


Provider can read Bob's message

Lookup Protocol

- Ensures bindings are consistent with snapshots.
- Prevents malicious providers from publishing fake keys and not leaving any evidence of the misbehavior.

Lookup Protocol

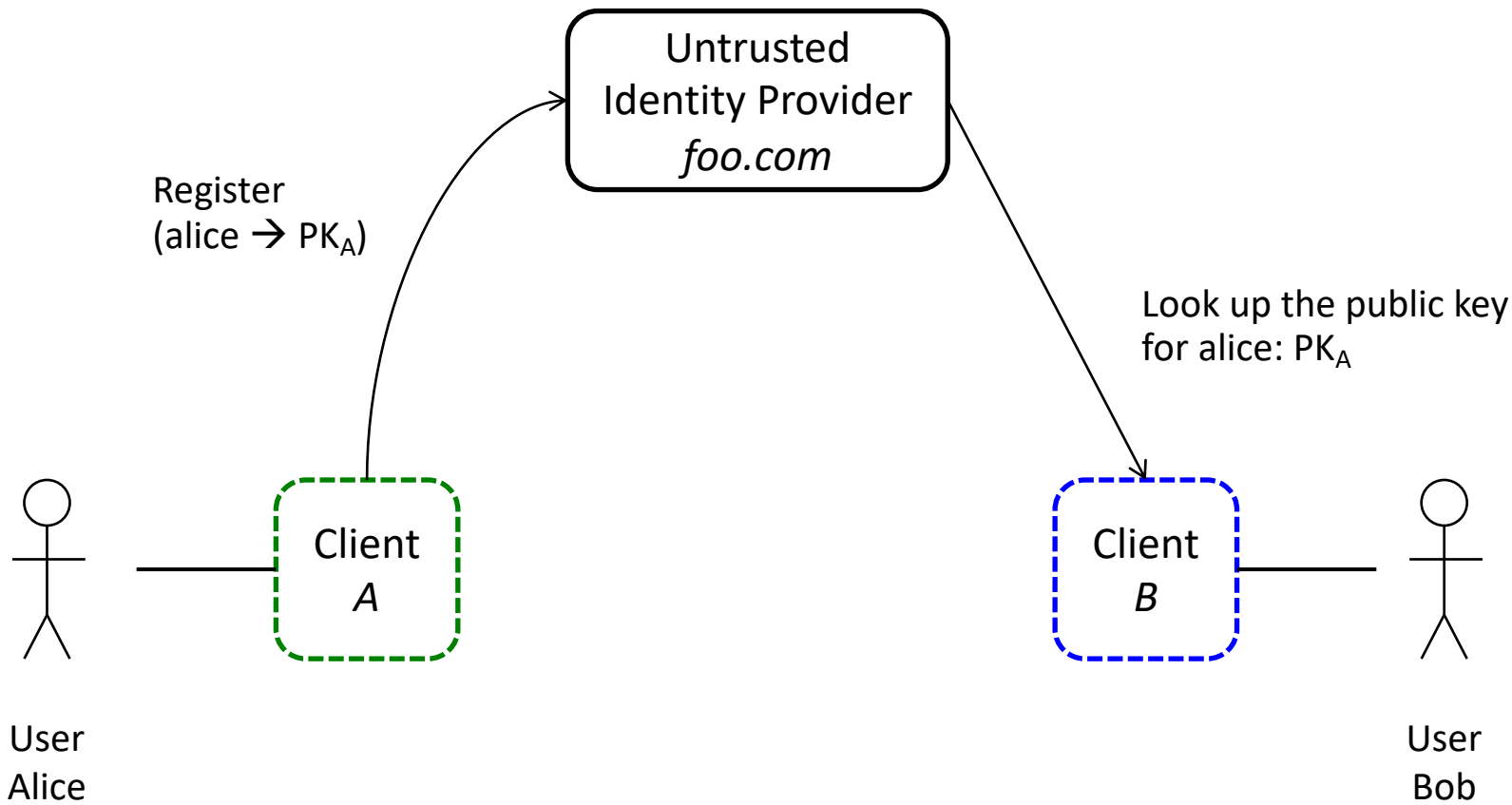


CONIKS Protocols

- Registration: new bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.

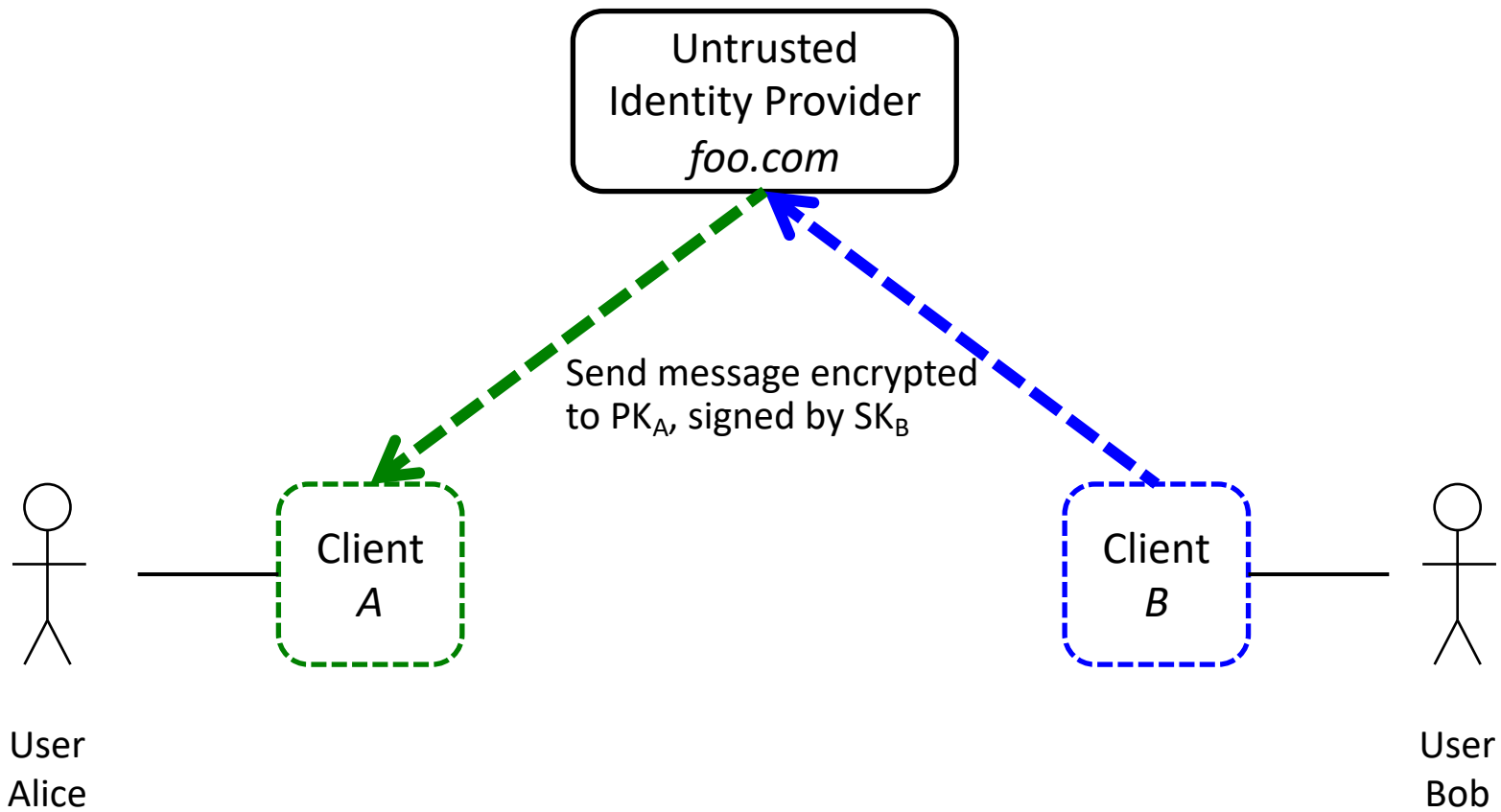
What can happen without monitoring?

Communication without Monitoring



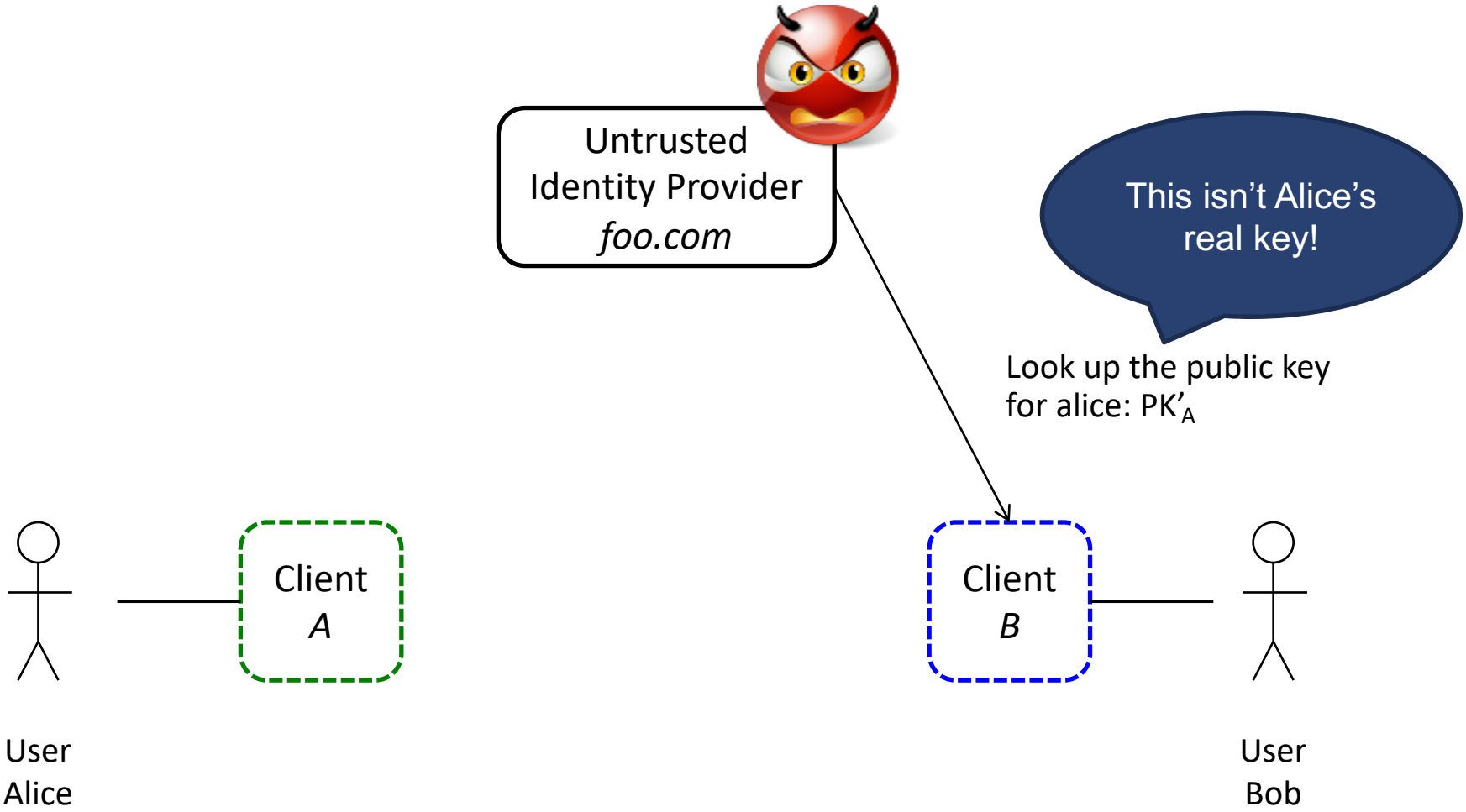
Epoch 1: Key has not Changed

Communication without Monitoring



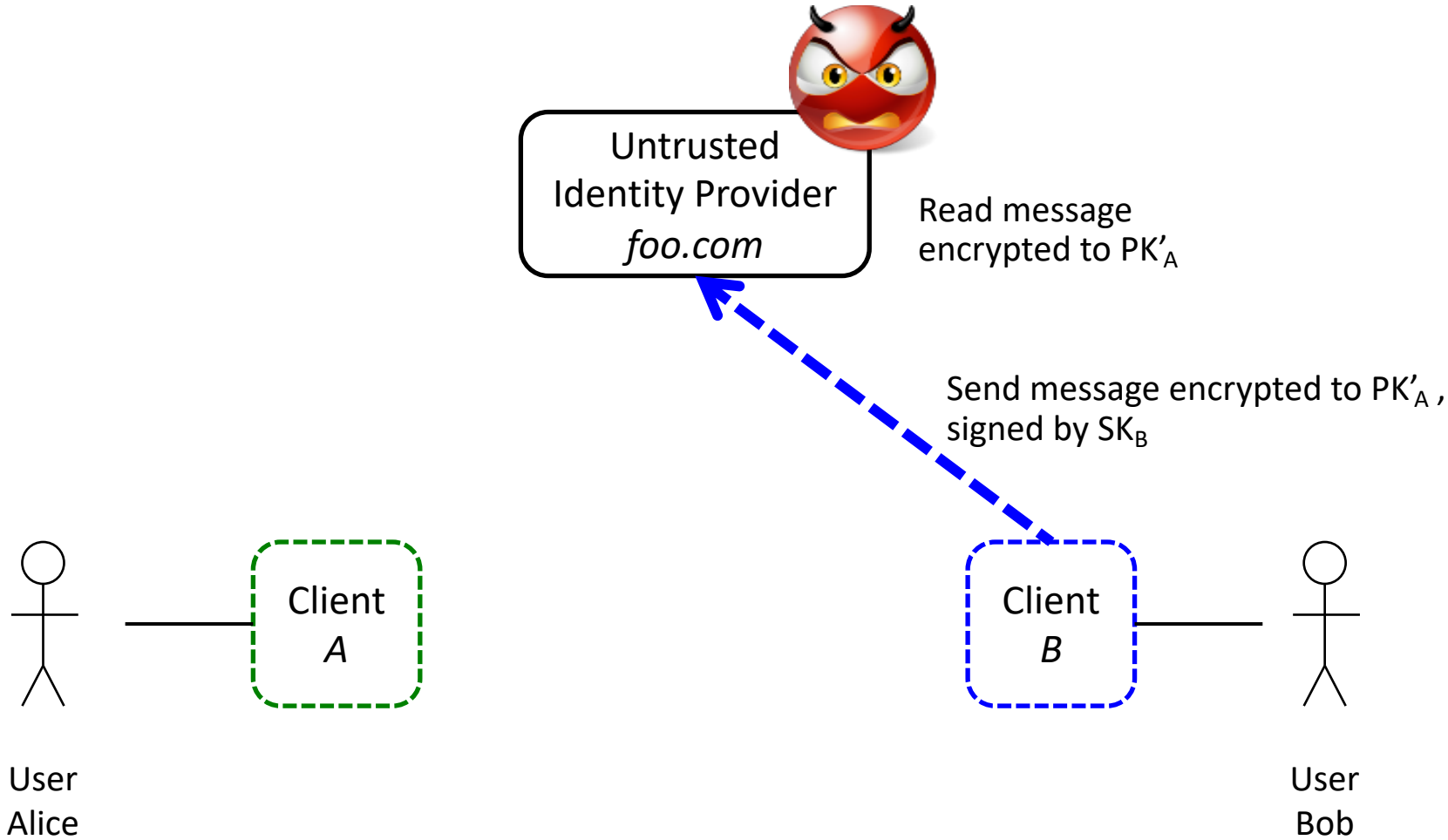
Epoch 1: Provider cannot read Bob's message

Communication without Monitoring



Epoch 2: Key has been Changed

Communication without Monitoring



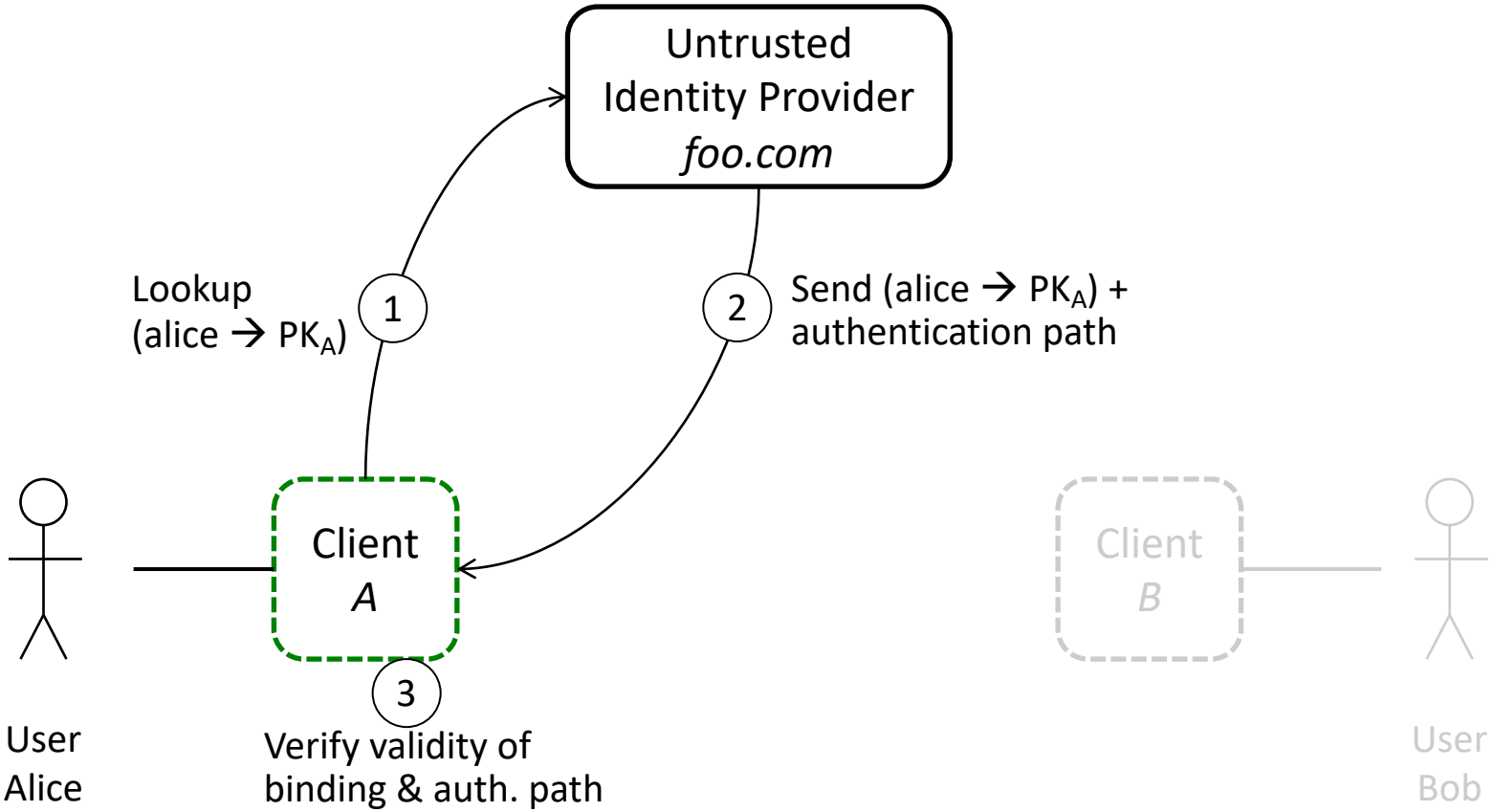
Epoch 2: Provider can read Bob's message

Monitoring Protocol

- Ensures public keys do not change unexpectedly.
- Prevents malicious providers from getting away with replacing existing keys.

Monitoring Protocol

Every user is responsible for monitoring their binding



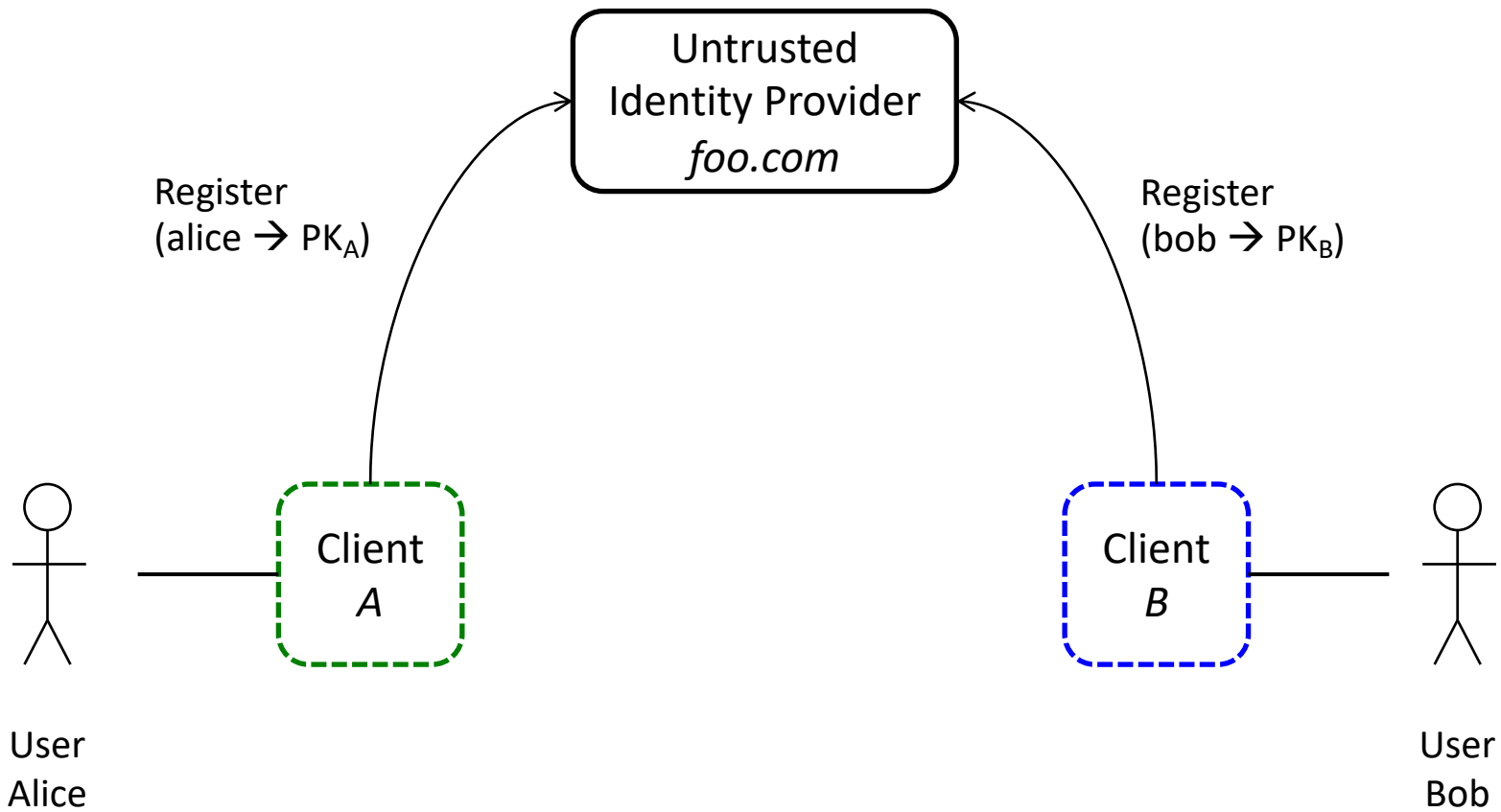
How often does each user need to check his/her binding?

CONIKS Protocols

- Registration: new bindings/updated keys.
- Lookups: Clients check bindings are included in directory.
- Monitoring: Clients check for validity of bindings.
- Auditing: Clients & providers check for non-equivocation.

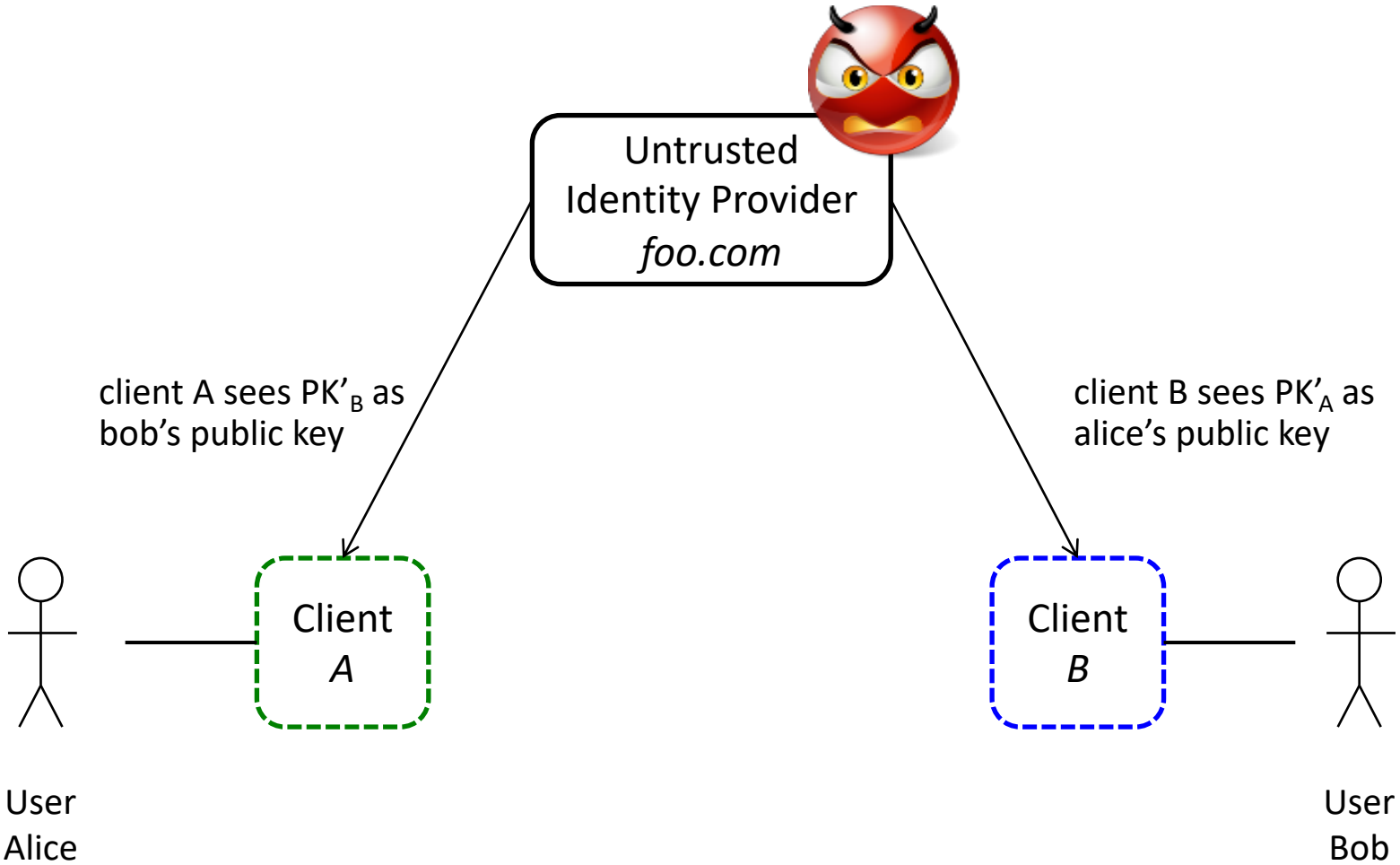
What can happen without auditing?

Communication without Auditing



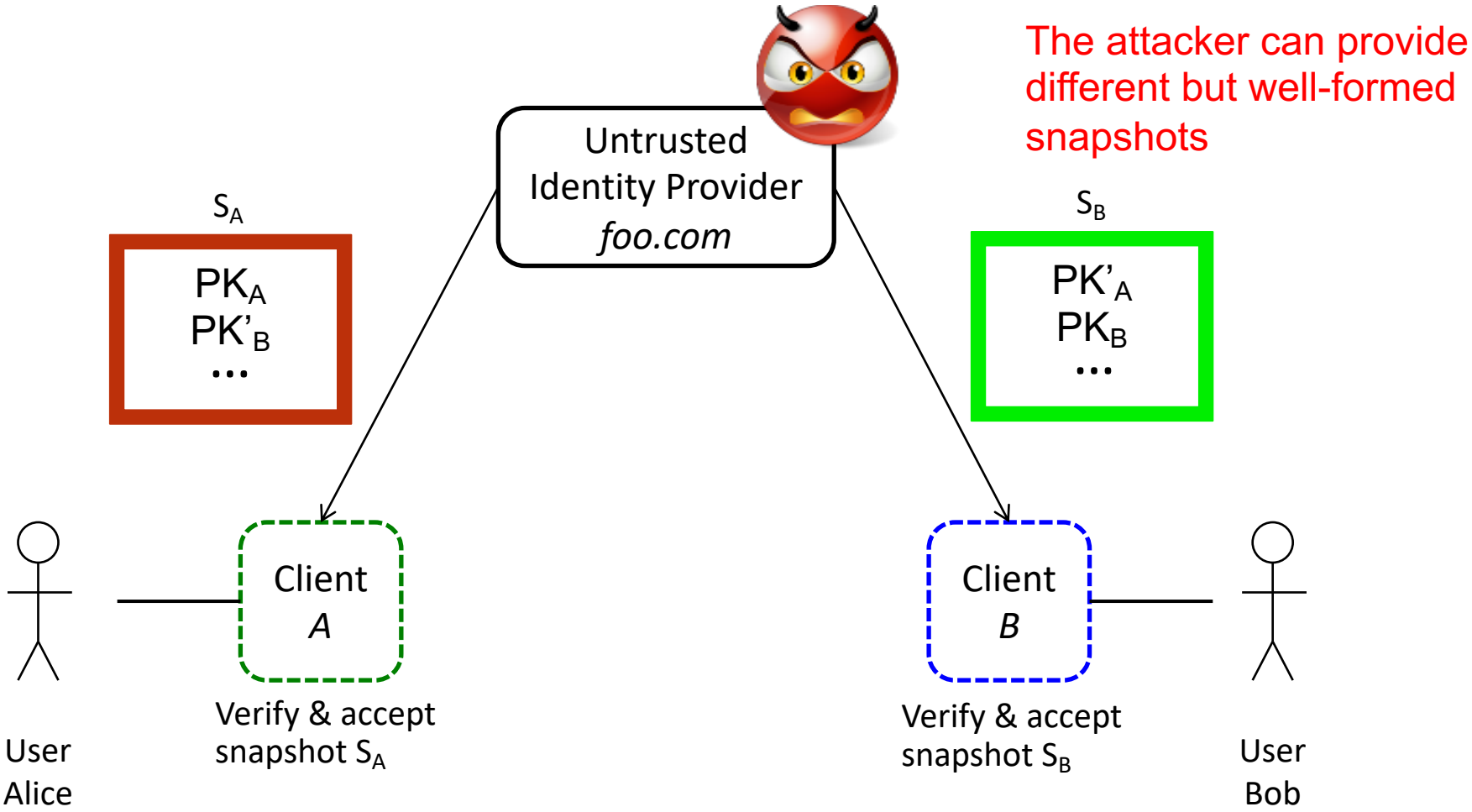
Clients register legitimate Keys

Communication without Auditing



Provider creates two versions of its Directory

Communication without Auditing

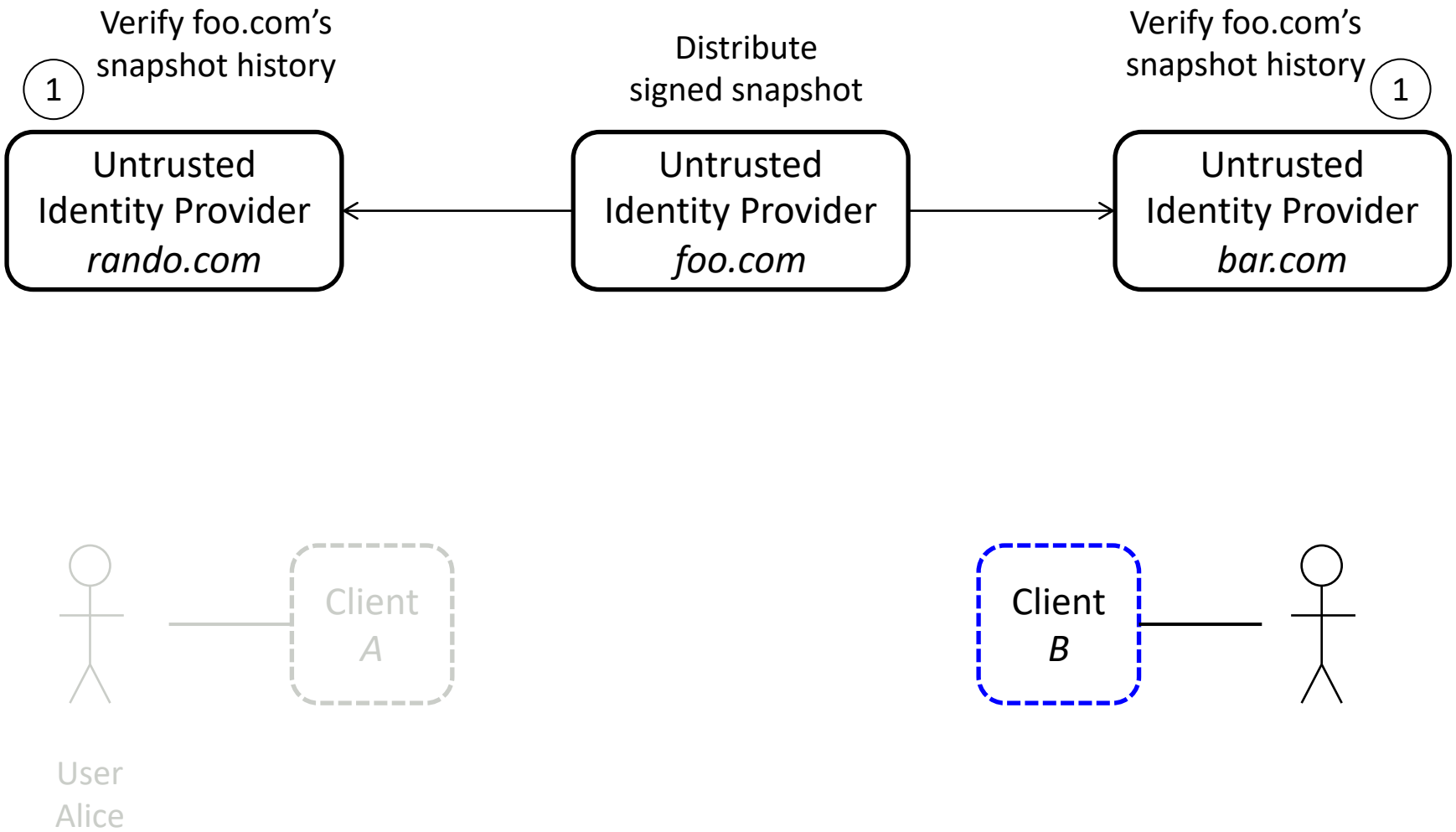


Provider presents two different but valid snapshots

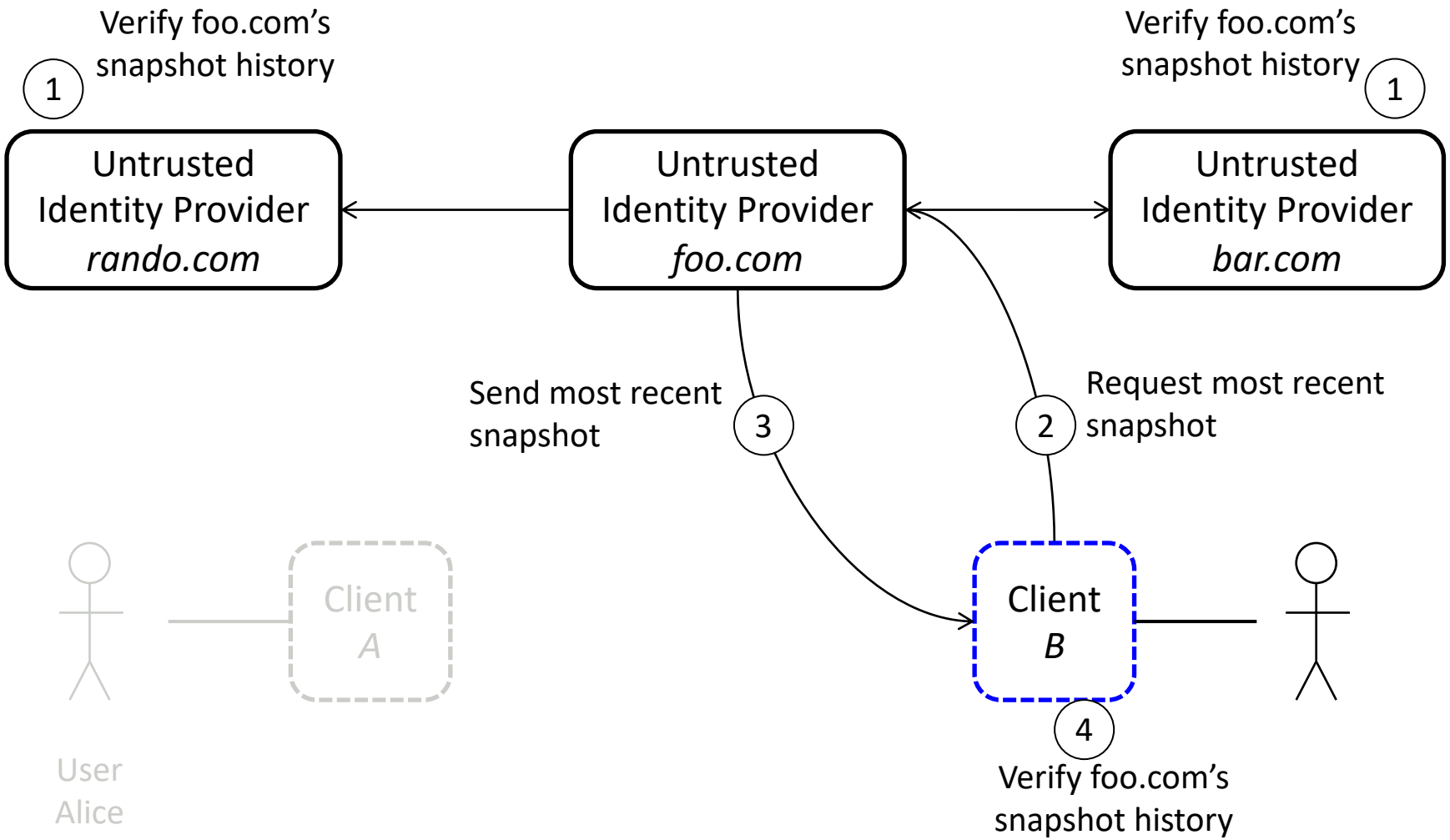
Auditing Protocol

- Ensures clients have consistent views of directories.
- Clients query random other providers for snapshots observed from given provider.
- Prevents malicious providers from getting away with publishing inconsistent versions of key directory.

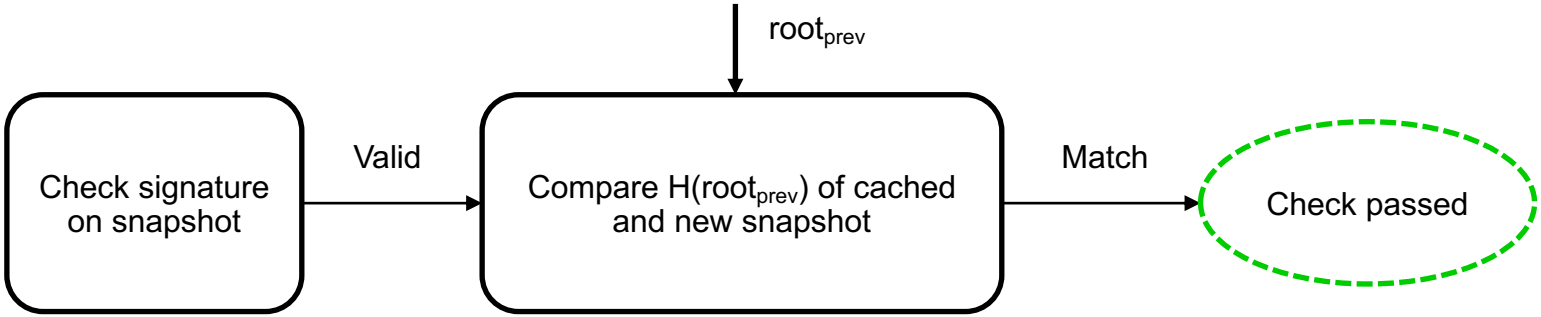
Auditing Protocol



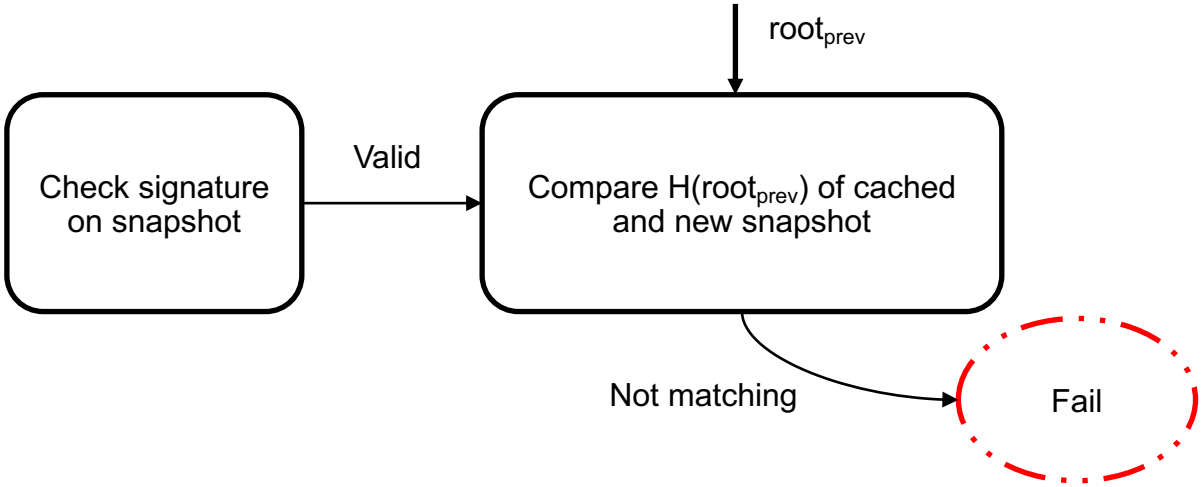
Auditing Protocol



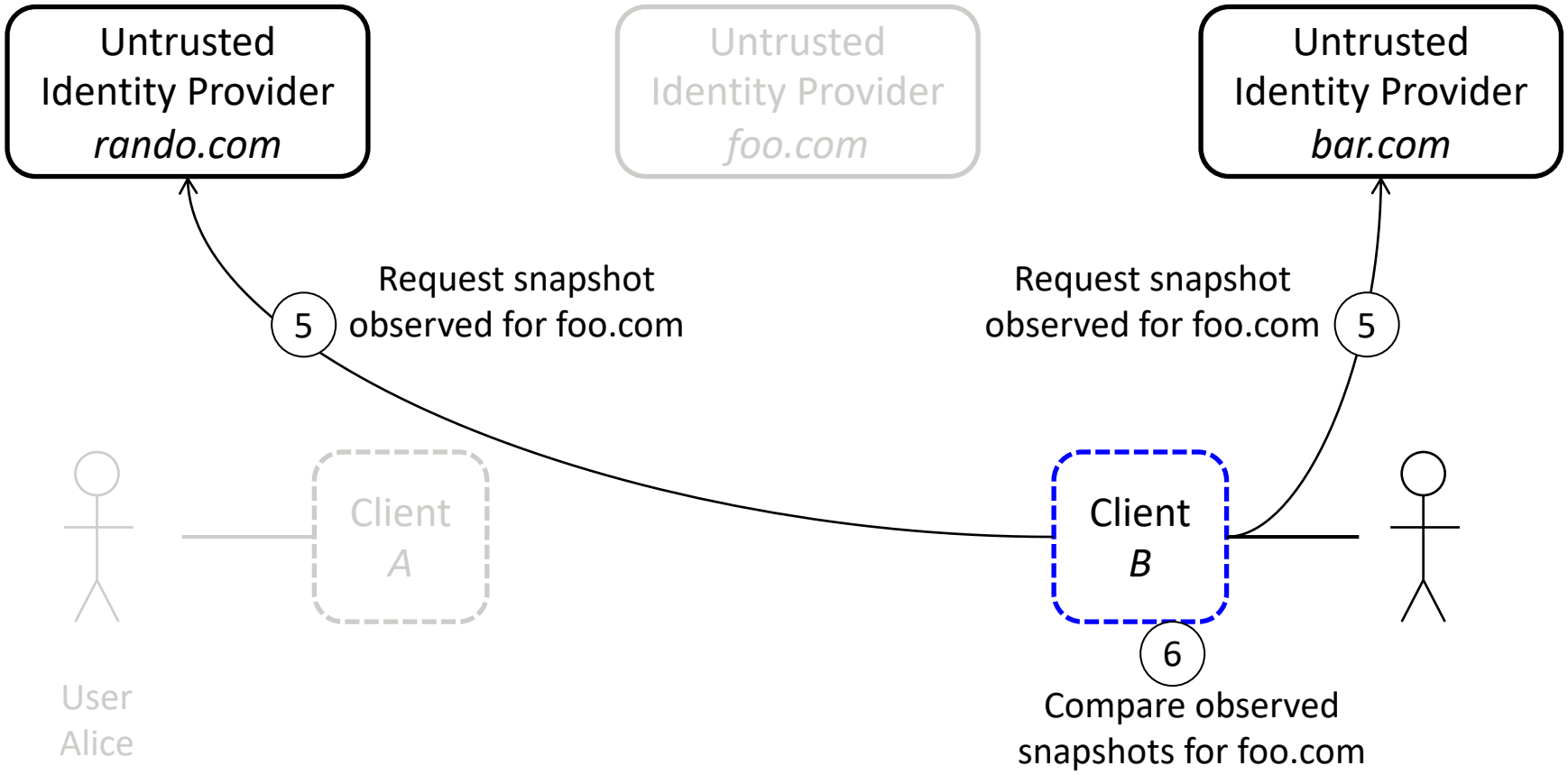
Checking Snapshot History



Checking Snapshot History



Auditing Protocol



Is it clear why foo cannot equivocate?

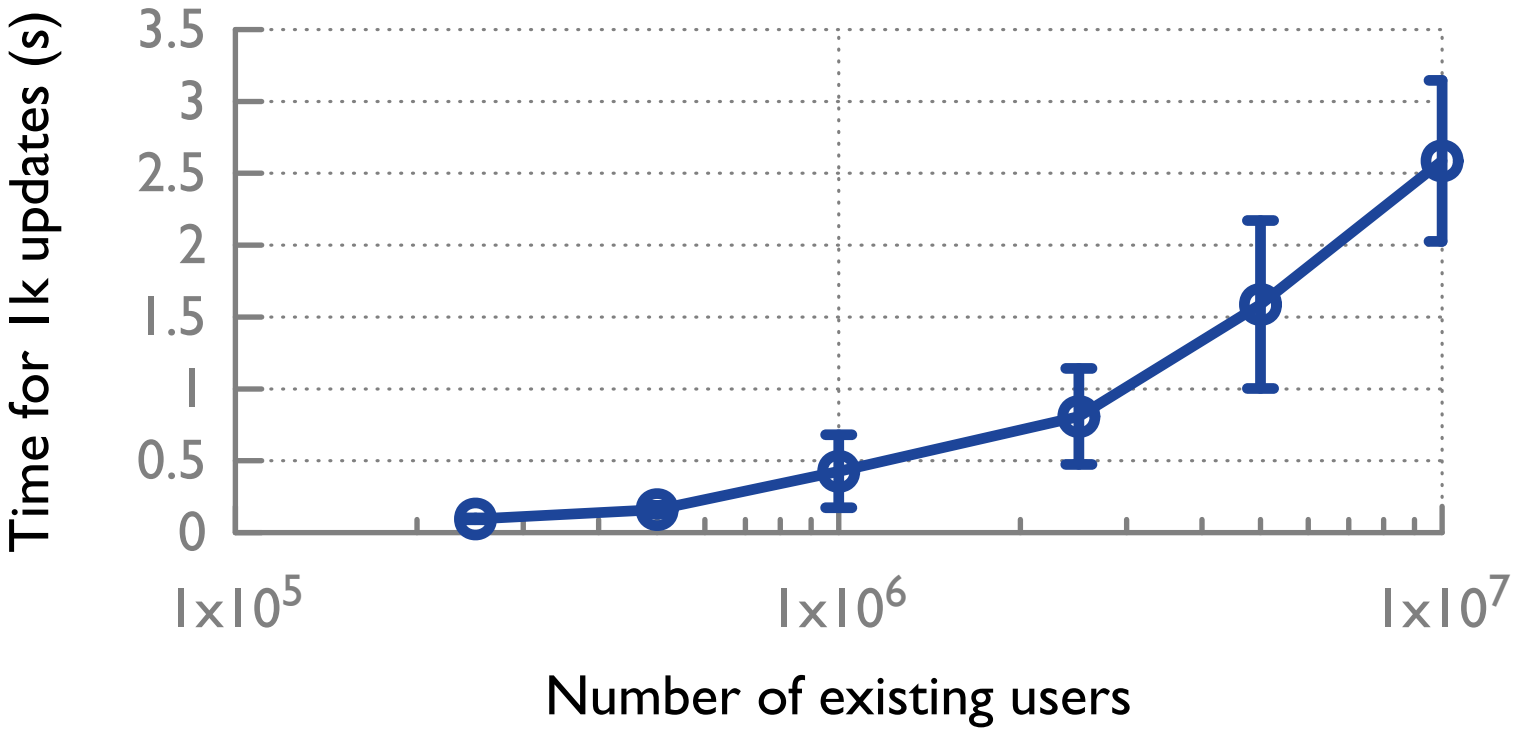
Privacy problems with this protocol?

- Auditors and the provider see contents of directory
- Authentication paths reveal other entries in the directory

Prototype Implementation

- CONIKS Chat: Secure chat service.
- Stand-alone provider alongside XMPP server.
- Supports registration, lookups and basic auditing.

Server performance



Client Overheads

	Download Requirements	**Storage Requirements**
Lookup (per binding)	< 1.4KB	0B
Monitoring (epoch)	< 800B	~ 300B
Monitoring (day)	< 20KB	~ 300B
Auditing (epoch, per snapshot)	~ 100B	~ 100B
Auditing (day, per snapshot)	~ 2.5KB	~ 100B

- $N \approx 4$ bill. users, $n \approx 2$ mill. updates/epoch, $d = 24$ epochs/day.

Client Overheads

	Download Requirements	Storage Requirements
Lookup (per binding)	< 1.4KB	0B
Monitoring (epoch)	< 800B	~ 300B
Monitoring (day)	< 20KB	~ 300B
Auditing (epoch, per snapshot)	~ 100B	~ 100B
Auditing (day, per snapshot)	~ 2.5KB	~ 100B

- $N \approx 4$ bill. users, $n \approx 2$ mill. updates/epoch, $d = 24$ epochs/day.

Client Overheads

	Download Requirements	**Storage Requirements**
Lookup (per binding)	< 1.4KB	~ 300B
Monitoring (epoch)	< 800B	~ 300B
Monitoring (day)	< 20KB	~ 300B
Auditing (epoch, per snapshot)	~ 100B	~ 100B
Auditing (day, per snapshot)	~ 2.5KB	~ 100B

- $N \approx 4$ bill. users, $n \approx 2$ mill. updates/epoch, $d = 24$ epochs/day.

Client Overheads

	Download Requirements	**Storage Requirements**
Lookup (per binding)	< 1.4KB	~ 300B
Monitoring (epoch)	< 800B	~ 300B
Monitoring (day)	< 20KB	~ 300B
Auditing (epoch, per snapshot)	~ 100B	~ 100B
Auditing (day, per snapshot)	~ 2.5KB	~ 100B

- $N \approx 4$ bill. users, $n \approx 2$ mill. updates/epoch, $d = 24$ epochs/day.

CONIKS summary

- CONIKS is an efficiently verifiable, privacy-preserving key management service for end-user public keys.
- End-to-end security can be practical for end-users.
- Some industry adoption: e.g. Google’s Key Transparency

Open questions/projects

Progress on any of:

- Scalability:
  - Users should not have to check their bindings every epoch
  - Provider should be able to scale to the whole world of people and IoT devices, wants to generate a snapshot every second
- Privacy:
  - Provider sees who talks to who
  - Auditors sees many bindings and changes