CONIKS

Louise, Niels

End-to-End encryption

- What is end-to-end encryption?
  - If A wants to send a message to B, A query's their service provider for B's public key and encrypts their message with it, sends the encrypted message to B, and B can decrypt the message using B's secret key
- What is the problem with E2EE?
  - Key management is difficult for users (and out-of-band trust is unintuitive)
  - If we query a provider to supply the public key, we need to trust the provider, which we cannot do

CONIKS' solution

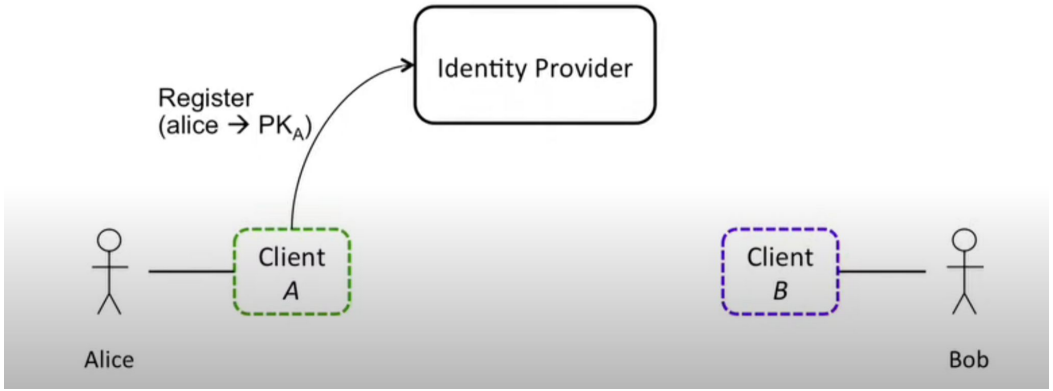
- CONIKS provides
  - Automated trust establishment with untrusted providers
  - Clients can verify their own consistency of key bindings
    - This is different from verifying correctness of keys
- Security Goal
  - Making provider key equivocation easily detectable
    - A provider may be able to register a false key in a clients name

Participants in CONIKS

- Identity providers
  - Manage namespaces, e.g. foo.com
  - Each namespace has name-to-key binding
    - Name: alice@foo.com
    - Key(s): PK1, PK2, …
- Clients
  - Monitor the consistency of their user's own bindings
- Auditors
  - Verify that identity providers are not equivocating by tracking changes in the key directory
  - Clients are often auditors for their own identity providers, but 3rd party auditors can participate as well

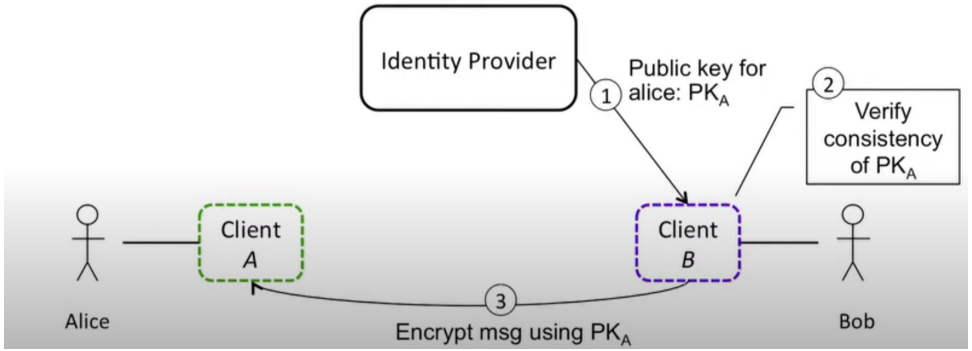
Registering a Key

- Alice registers her public key with her identity provider



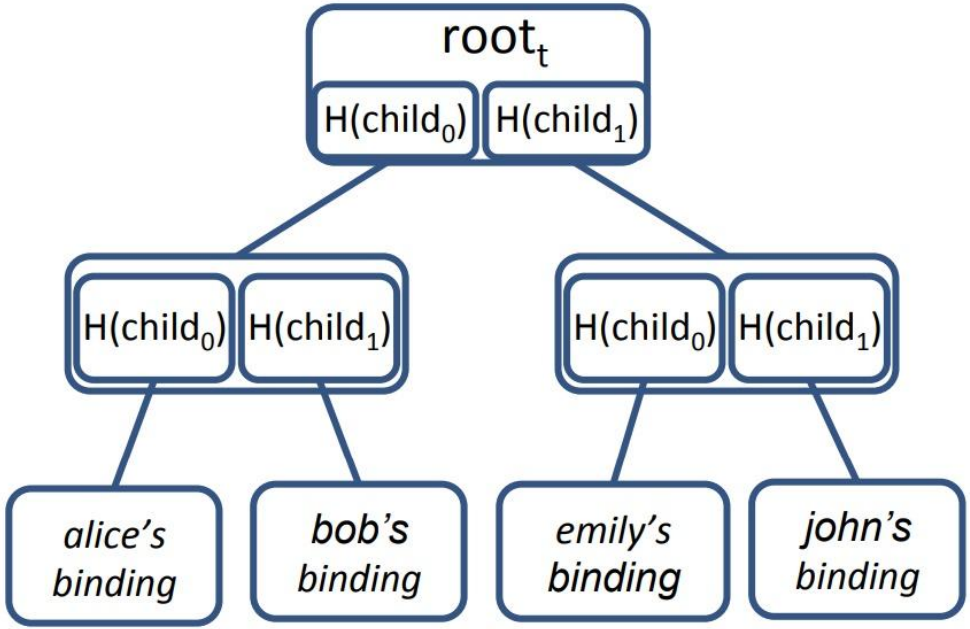
Learning a User's Key

- Bob is able to learn Alice's key by first receiving it from the identity provider
- Then verifies that it is consistent using the STR of the merkle tree



CONIKS structure

- How does CONIKS let users authenticate keys?
  - Use Merkle prefix trees!
  - Clients can verify consistency of name-to-key bindings using the root



Merkle Prefix Tree

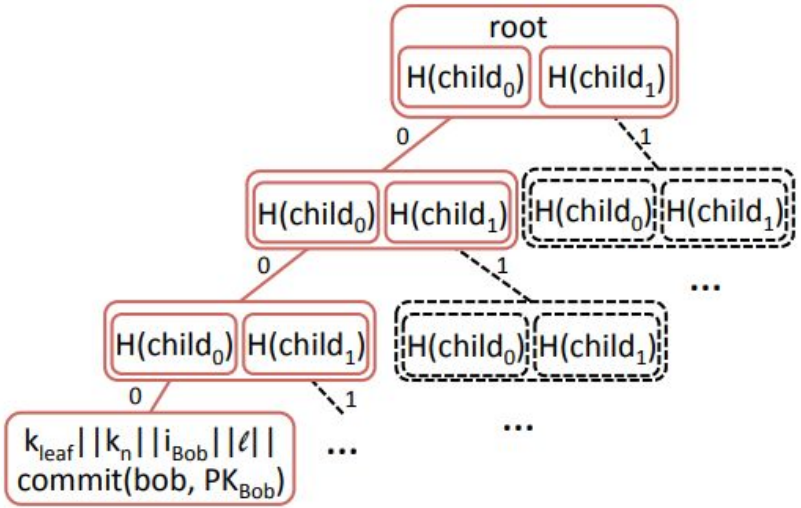
- Interior nodes
  - Hash of two child nodes
  - $h_{\text{interior}} = \mathrm{H}\left(h_{\text{child.0}} || h_{\text{child.1}}\right)$
- Leaf nodes
  - Represents all information needed about an individual user

$$\mathrm{h}_{\text{leaf}} = \mathrm{H}\left(k_{\text{leaf}} || k_n || i || \ell || \text{commit}(\text{name}_i || \text{keys}_i)\right)$$

prefix

Depth in tree

Cryptographic commitment to the name and key data associated with user



Security goals

- No Spurious Keys
  - No unexpected key changes, as the expected bindings are stored in the signed tree root (STR)

$$\mathrm{STR} = \mathrm{Sign}_{SK_d}\left(t||t_{prev}||\mathrm{root}_t||\mathrm{H}(\mathrm{STR}_{prev})||P\right)$$

Epoch number

Summary of the provider's current security policies

- Non-Equivocation
  - Identity providers cannot present diverging views of name-to-key bindings

Proof of inclusion

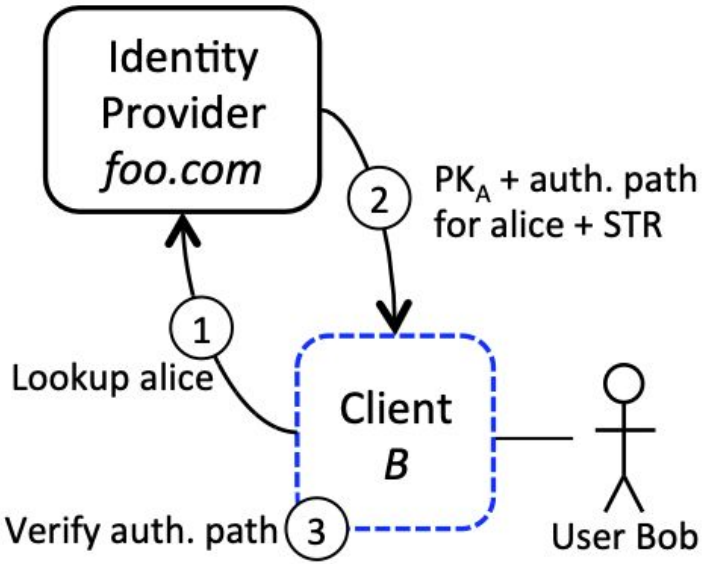
- Provider gives a complete authentication path between leaf and root
- So clients can verify that a particular index does indeed exist in the tree
    - By recomputing the root using the path and checking

Privacy

- Each authentication path reveals no information about whether any other names are present in the tree

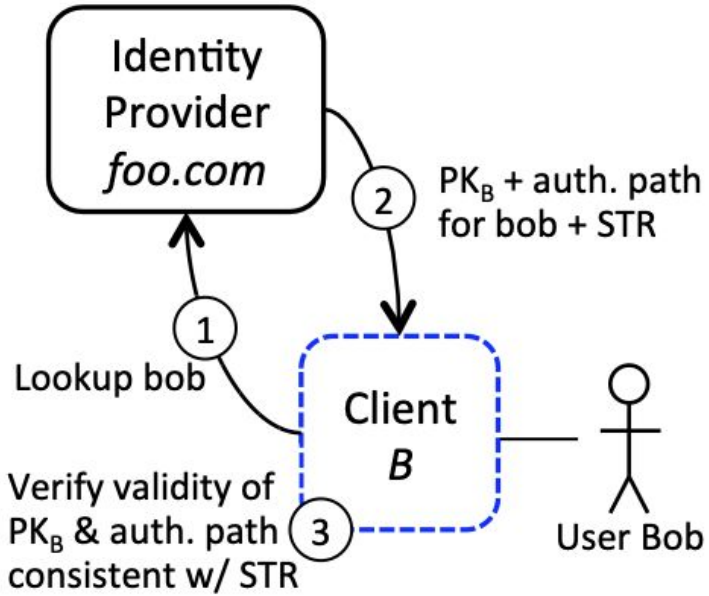
Key Lookup

- ID provider returns a proof of inclusion, i.e. the full authentication path + current STR
- Client can then check whether received binding is consistent with current STR
  - Show that the given binding is indeed in the tree (proof of inclusion)
  - Or show that the given binding is absent in the tree if it should be absent (proof of absence)



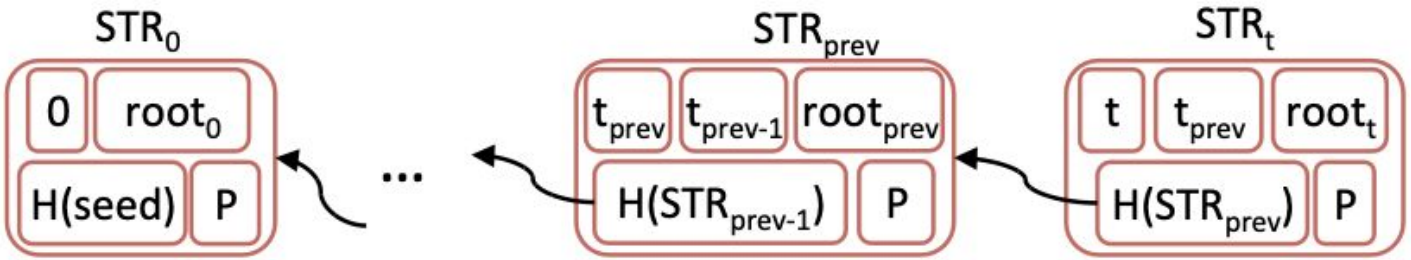
Monitoring protocol

- Perform key lookup and obtain proof of inclusion
- Check the represented public key is correct
  - Check if the key is consistent with keys in previous epoch
  - If keys haven't changed or have authorized change, then chill
  - If have unexpected key change, take action
  - Users can enforce authorization for every key change
- Verify authentication path



Non-equivocation

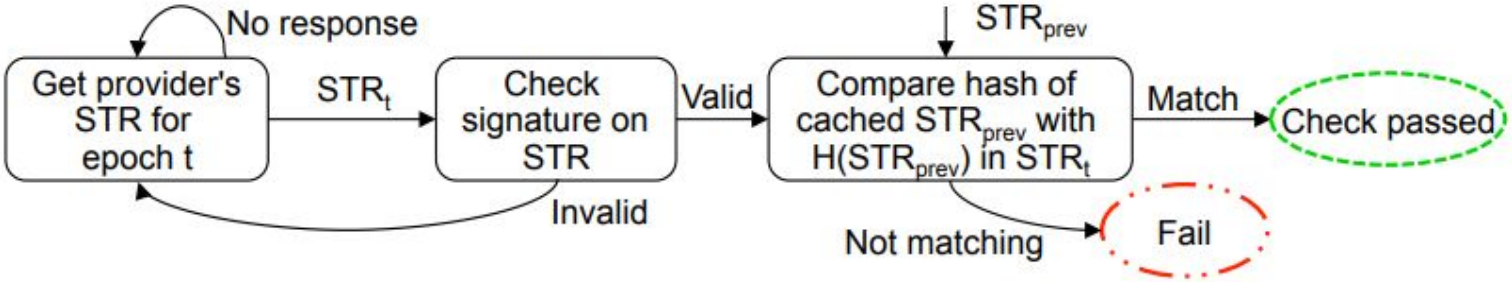
1. Verifiable Linear chain of STR history



2. Cross verification with multiple providers

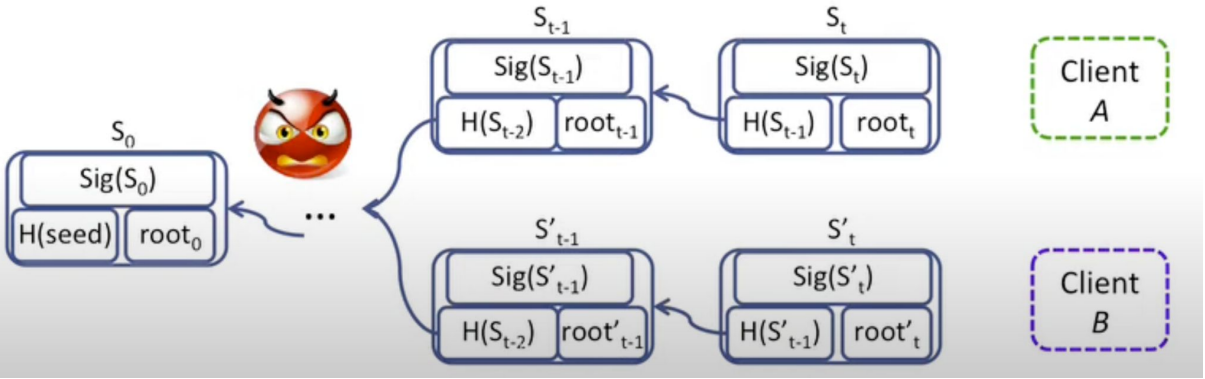
Auditing protocol

1. Verifiable Linear chain of STR history
   - Check if there is branching



Auditing protocol

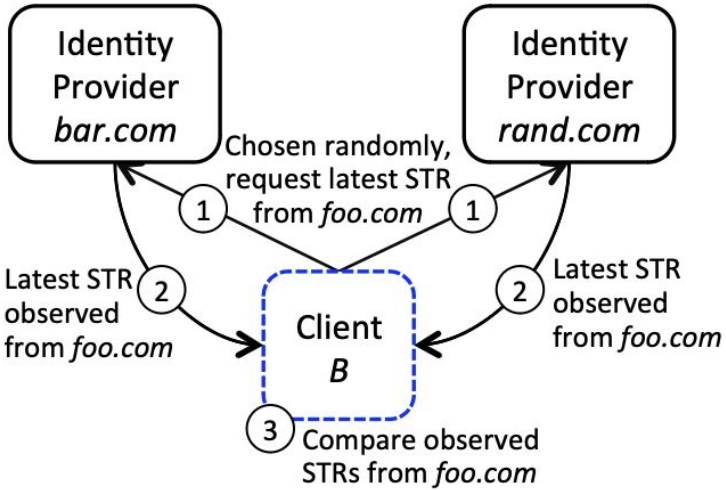
1. Verifiable Linear chain of STR history
   - Check if there is branching
   - But checking STR history is not enough: what if malicious act happened very early?



Auditing protocol

2. Cross verification with multiple providers

- If clients see two inconsistent STR, report to all auditors they know, or out-of-band whistleblowing (e.g. publish in social media)



Secure communication model with CONIKS

1. Periodically,client B checks the consistency of Bob's binding.
   - first performs the monitoring protocol
   - then it audits foo.com
2. Before sending Bob's message to client A, client B looks up the public key for the username alice at foo.com
   - It verifies the proof of inclusion for alice
   - It performs the auditing protocol for foo.com if the STR received as part of the lookup is different or newer than the STR it observed for foo.com in its latest run of step 1.
3. If client B determines that Alice's binding is consistent, it encrypts Bob's message using alice's public key and signs it using Bob's key. It then sends the message.

Works cited

- CONIKS: Bringing Key Transparency to End Users (iacr.org)
- USENIX Security '15 - CONIKS: Bringing Key Transparency to End Users - YouTube