E2EE Meetings at Zoom

MIT 6.858 – 2022.05.05

Max Krohn

@maxtaco / max.krohn@zoom.us

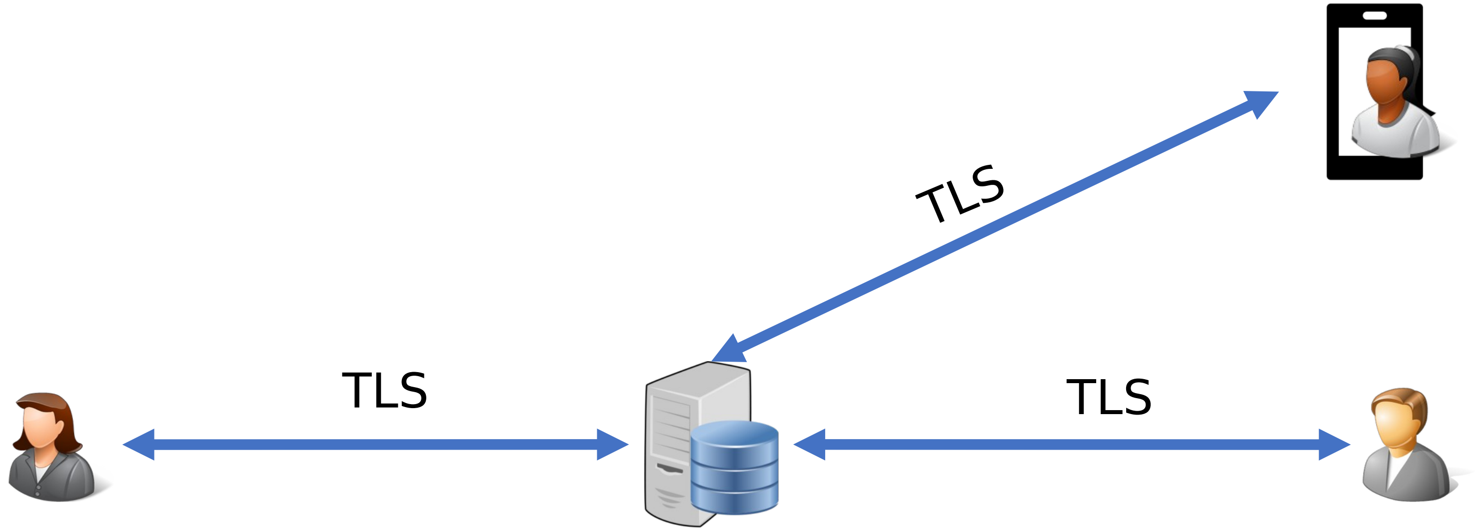
Backstory

- Co-founded Keybase in 2015
- Built some cool stuff
- May 2020: Bought by Zoom
- Ever since: using learnings and tech to improve Zoom

Agenda

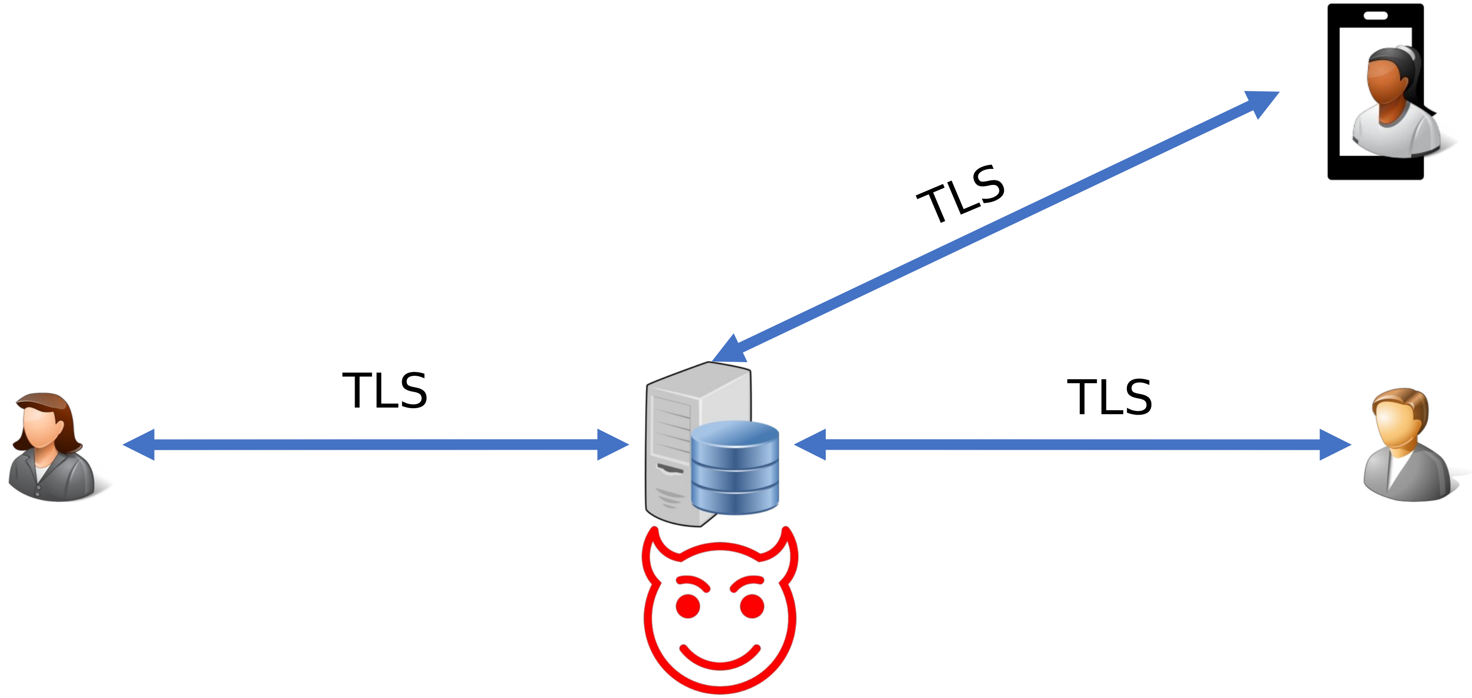
- Some quick background on videoconferencing
- End-to-end encryption for Zoom meetings (Phase 1)
- Persistent identity (Phases 2+)
- Open Q&A

What’s Out There?



Server down-samples
video streams

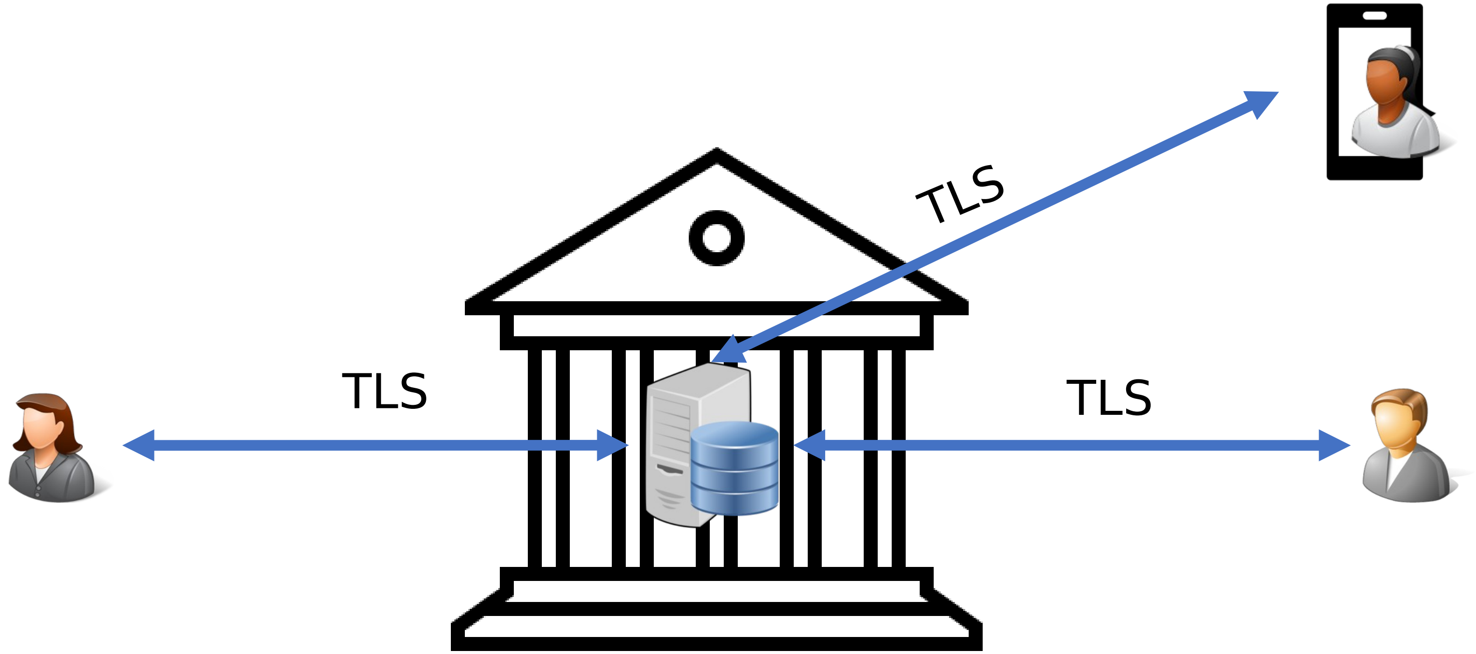
What's wrong with Point-to-Point Encryption?



Passive Server-Side Attacks

- Hard to detect
  - The attacker is just listening; if good, won’t leave a trace
- Many possible attackers
  - A bad employee (disgruntled, compromised, etc.)
  - A bad actor with access to the underlying cloud infrastructure
  - An attacker who gains access to infrastructure past the perimeter
- Extremely valuable targets
  - e.g., a Fortune 500 board meeting
  - Ukrainian officials + US Senators

What about On-Prem Solutions?



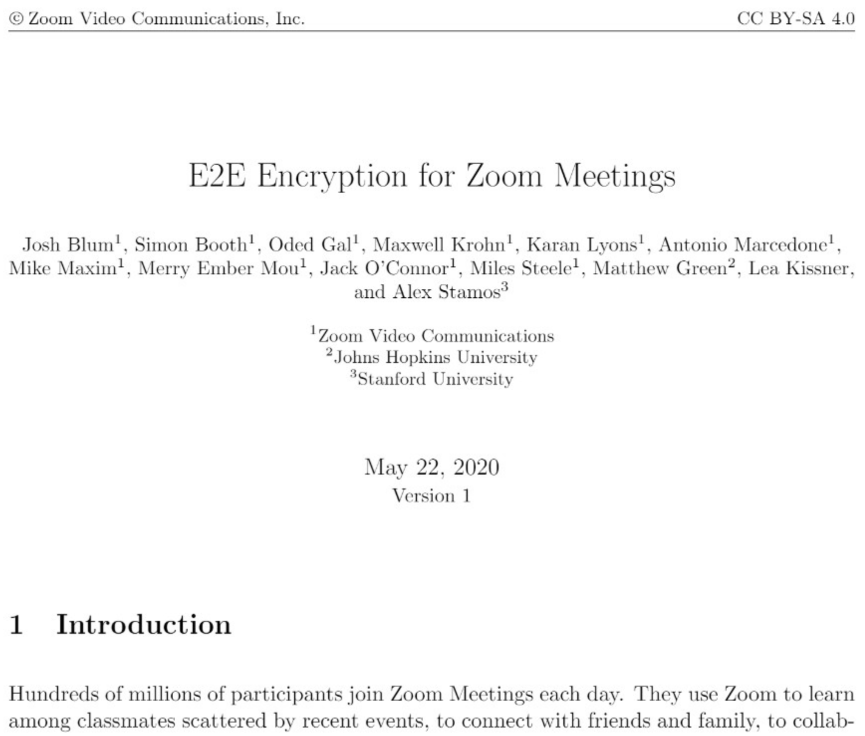
On-Prem Also Has Drawbacks

- High latency unless customer deploys global routers
- Company might not fully trust the people who change the fried network cards
- If buggy, software can become a backdoor into the prem
  - See: SolarWinds, Log4j, various Microsoft Exchange vulnerabilities

End-to-End Encryption

May 2020

July 2020



E2E Encryption for Zoom Meetings

Josh Blum[1], Simon Booth[1], Oded Gal[1], Maxwell Krohn[1], Karan Lyons[1], Antonio Marcedone[1], Mike Maxim[1], Merry Ember Mou[1], Jack O'Connor[1], Miles Steele[1], Matthew Green[2], Lea Kissner, and Alex Stamos[3]

[1]Zoom Video Communications
[2]Johns Hopkins University
[3]Stanford University

May 22, 2020
Version 1

1 Introduction

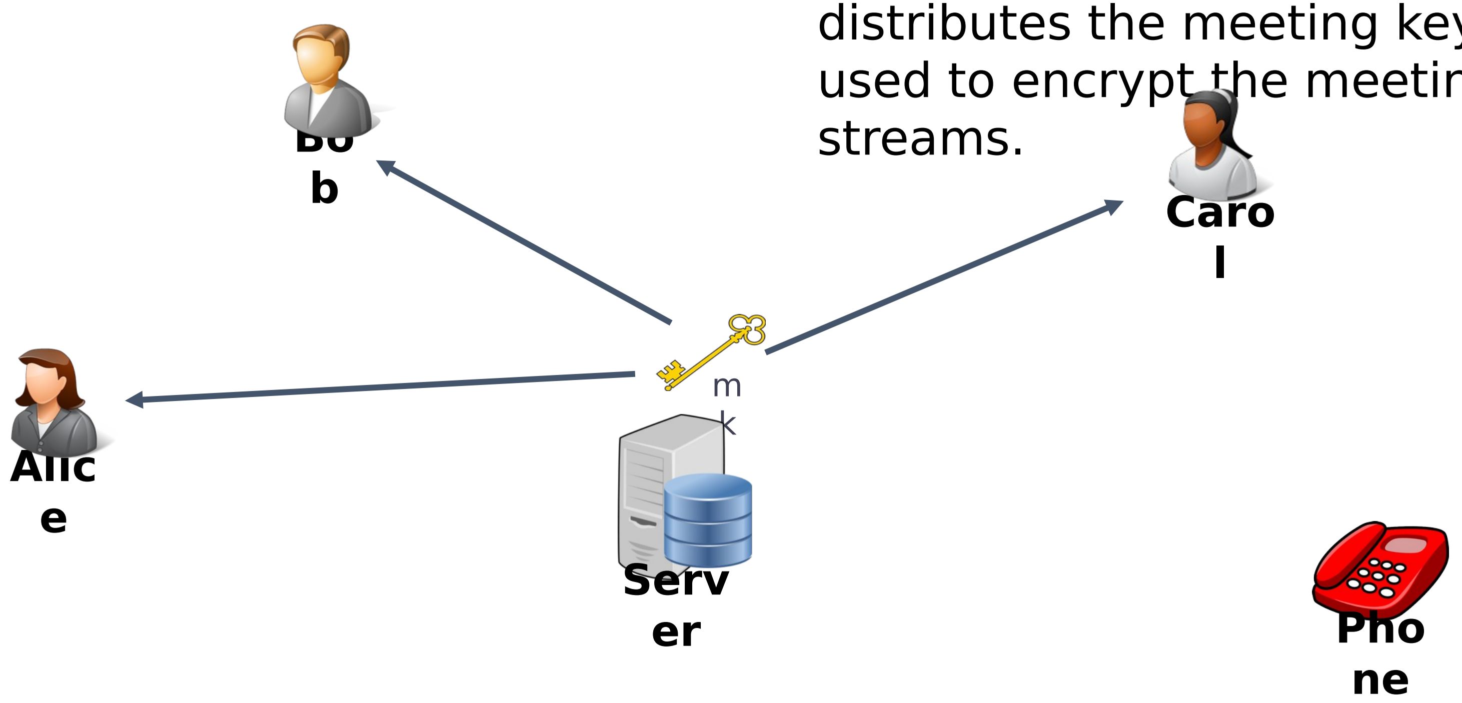
Hundreds of millions of participants join Zoom Meetings each day. They use Zoom to learn among classmates scattered by recent events, to connect with friends and family, to collab-

https://github.com/zoom/zoom-e2e-whitepaper

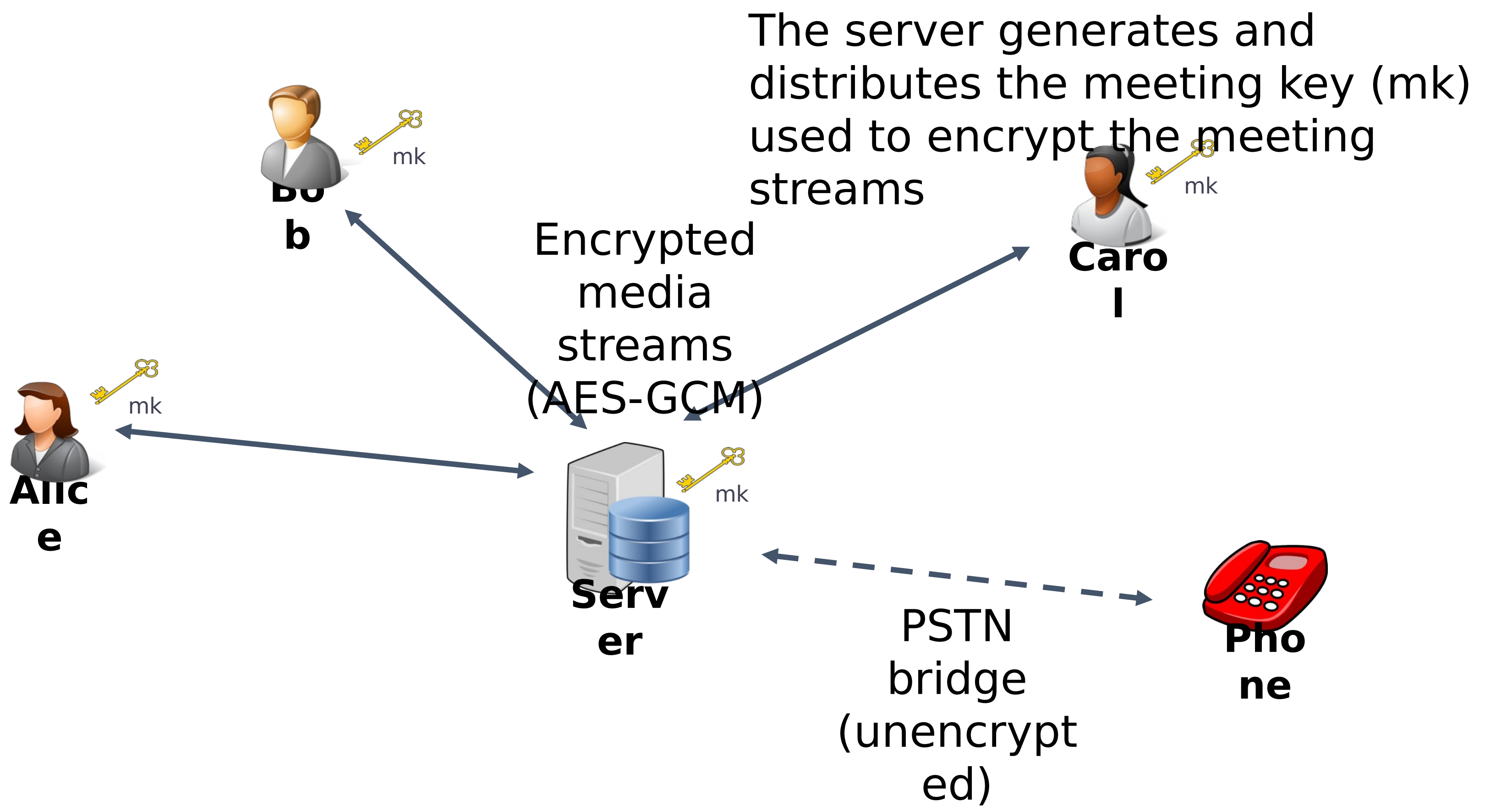
First E2EE Meeting

Enhanced Encryption (simplified)

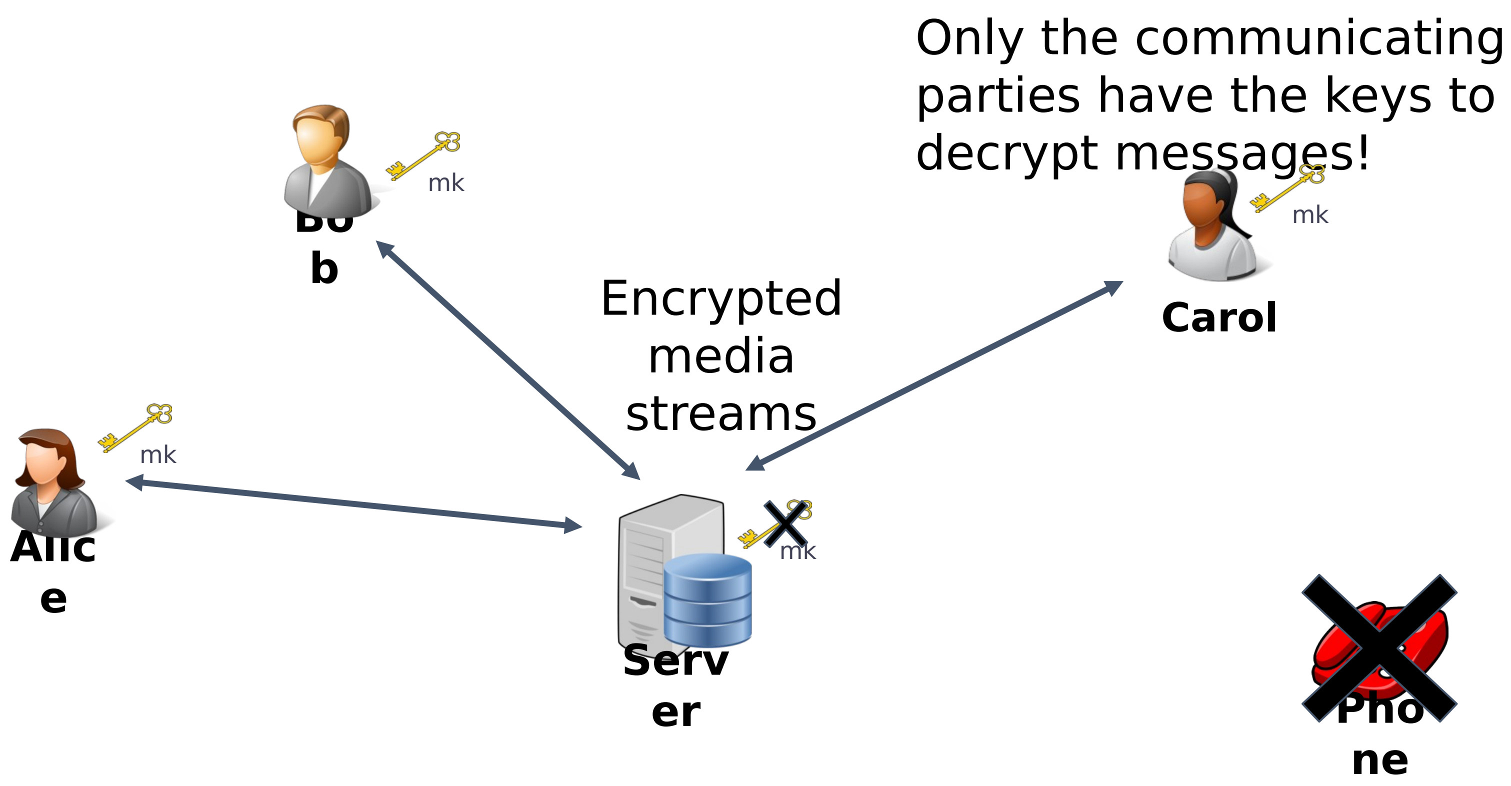
The server generates and distributes the meeting key (mk) used to encrypt the meeting streams.



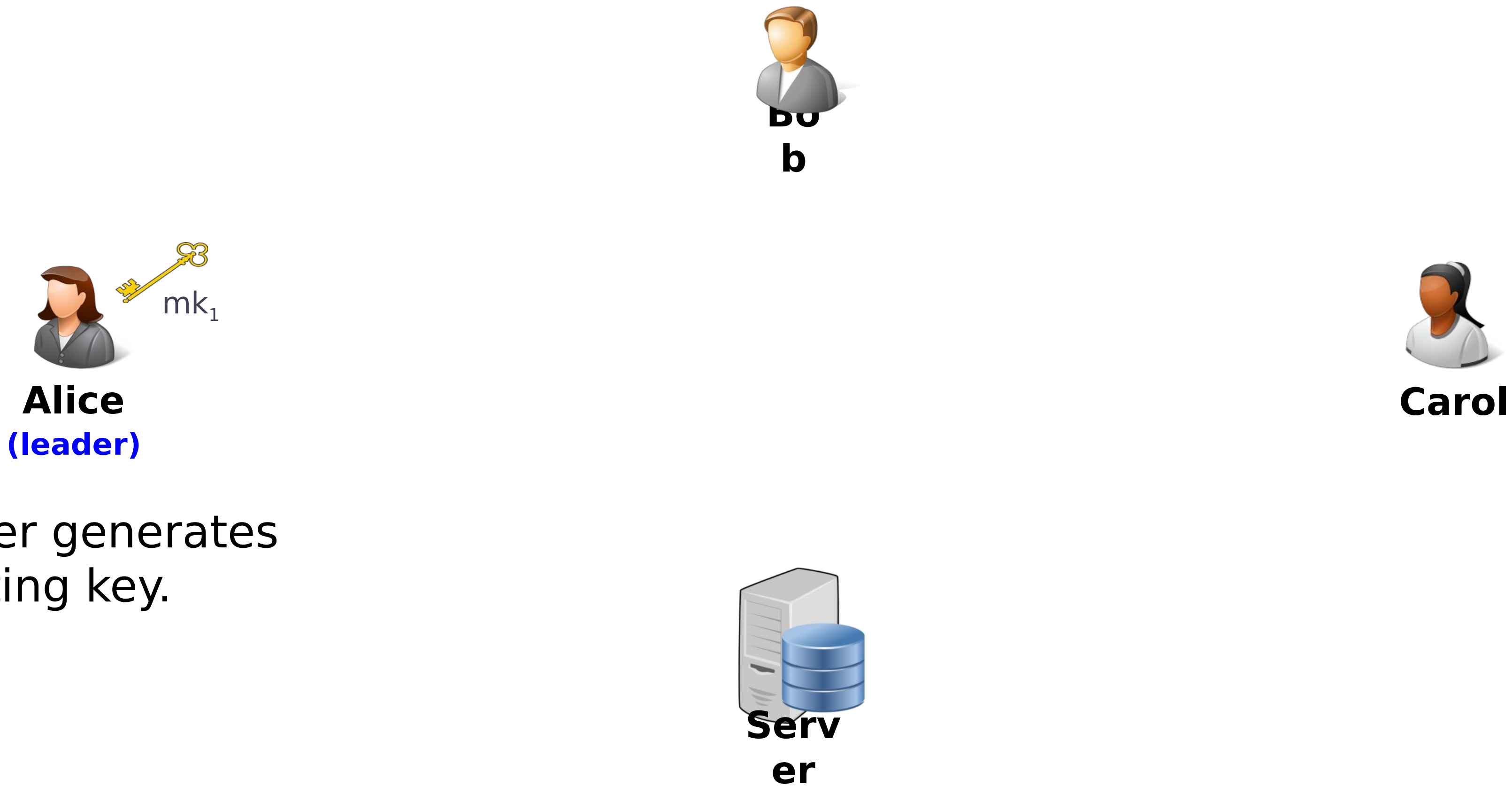
Enhanced Encryption (simplified)



End-To-End Encryption (simplified)



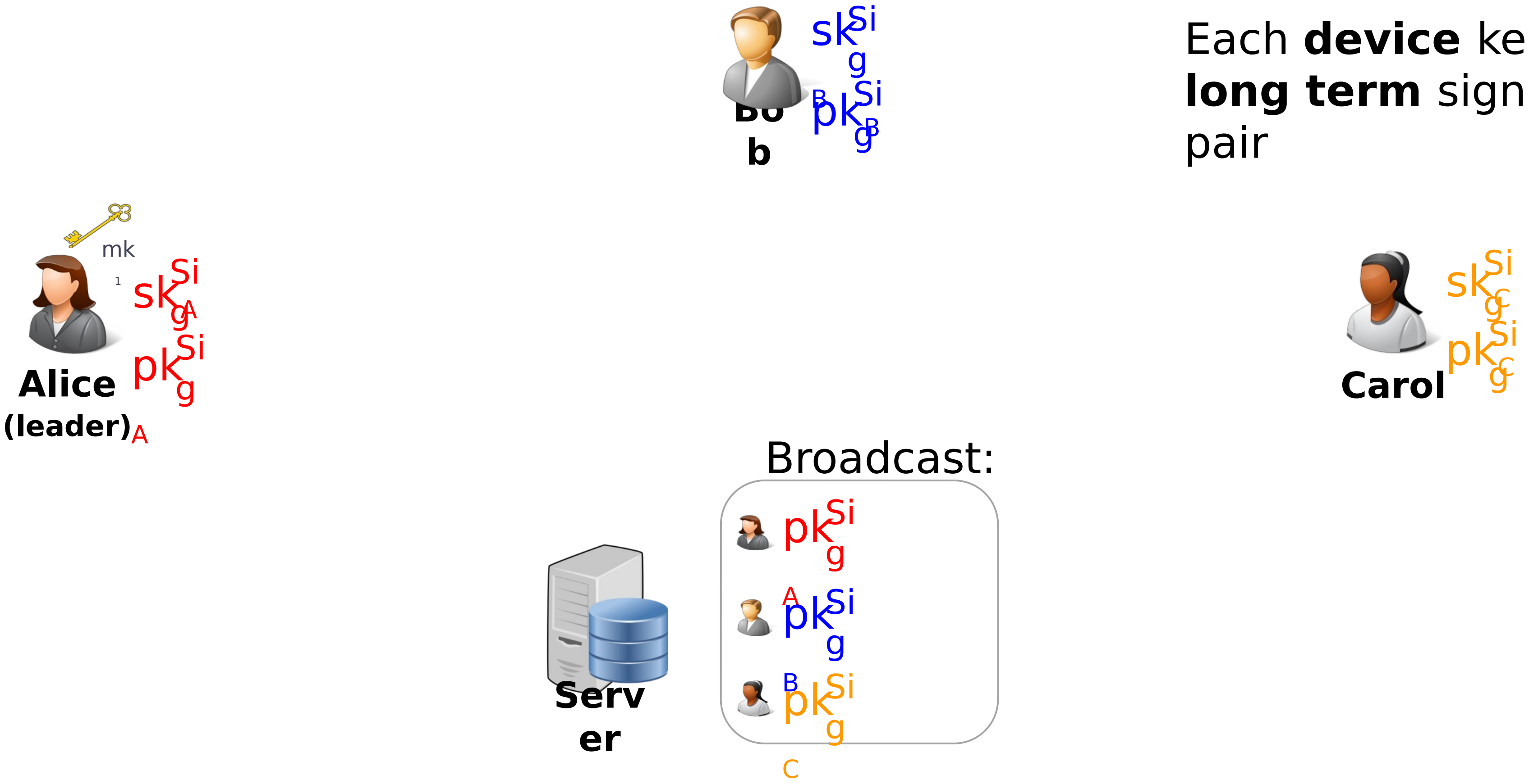
E2EE: Key Generation



The leader generates the meeting key.

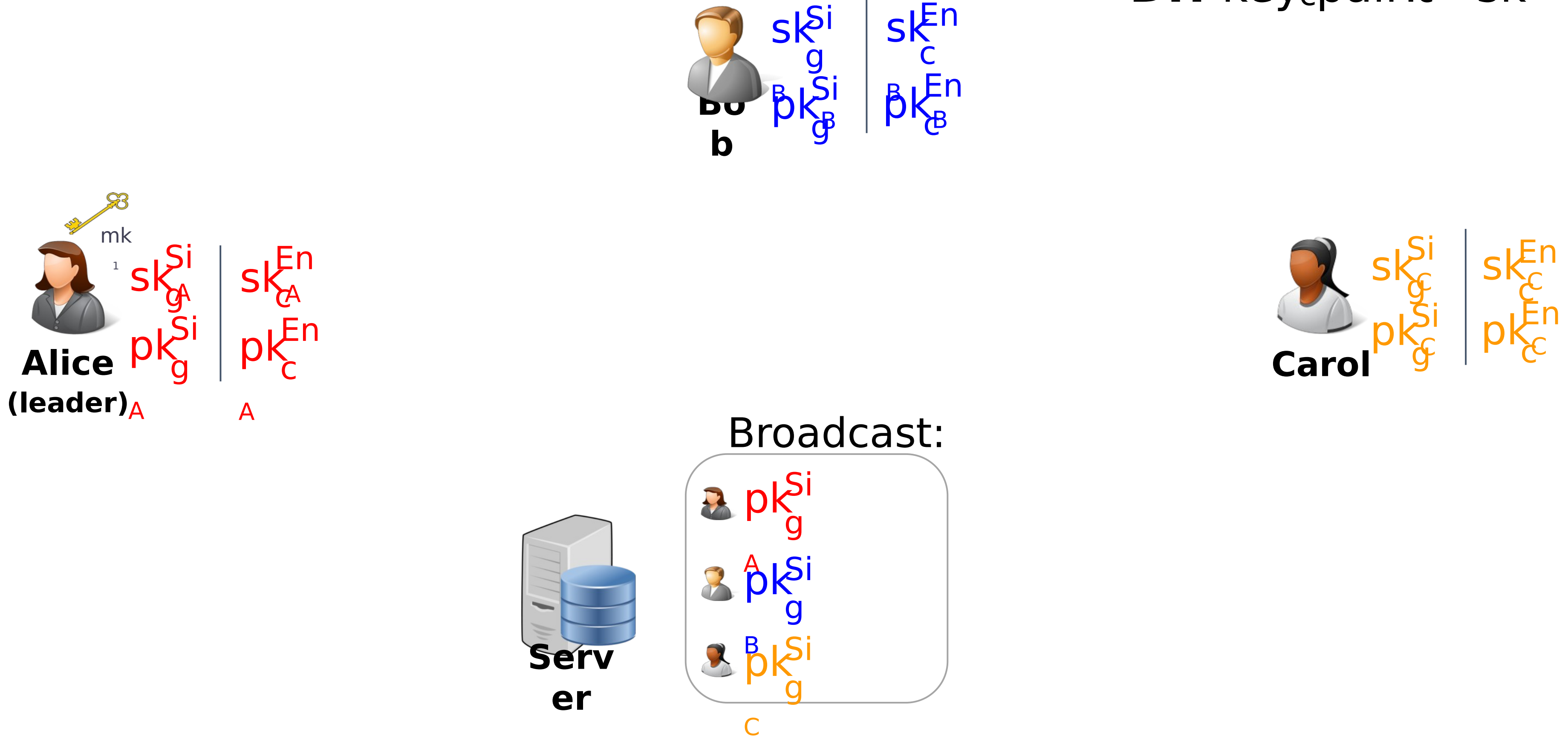
E2EE: Long term device keys

Each **device** keeps a **long term** signing key pair



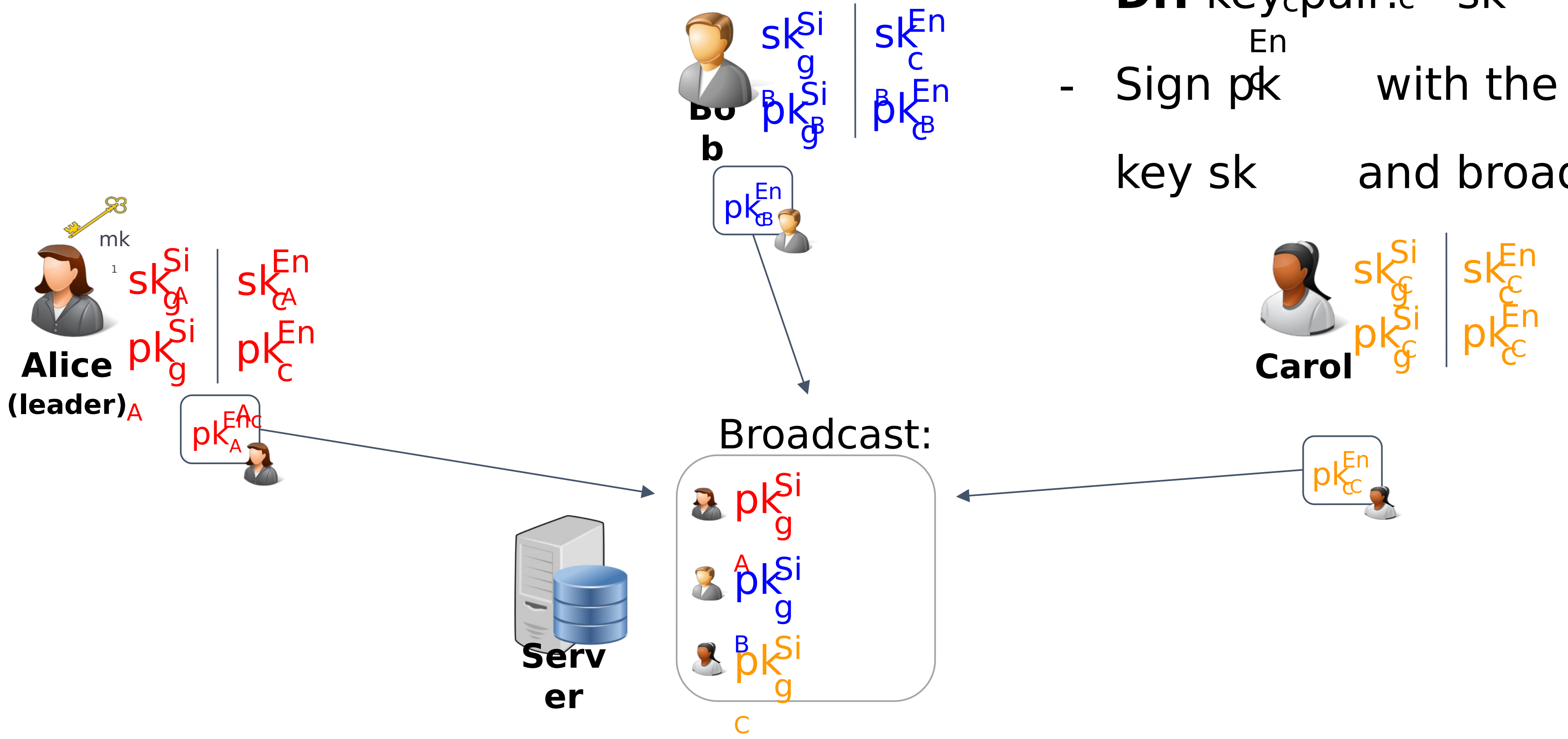
E2EE: Ephemeral DH keys

- Generate an **ephemeral** **DH** key pair: sk_c^{En}, pk_c^{En}



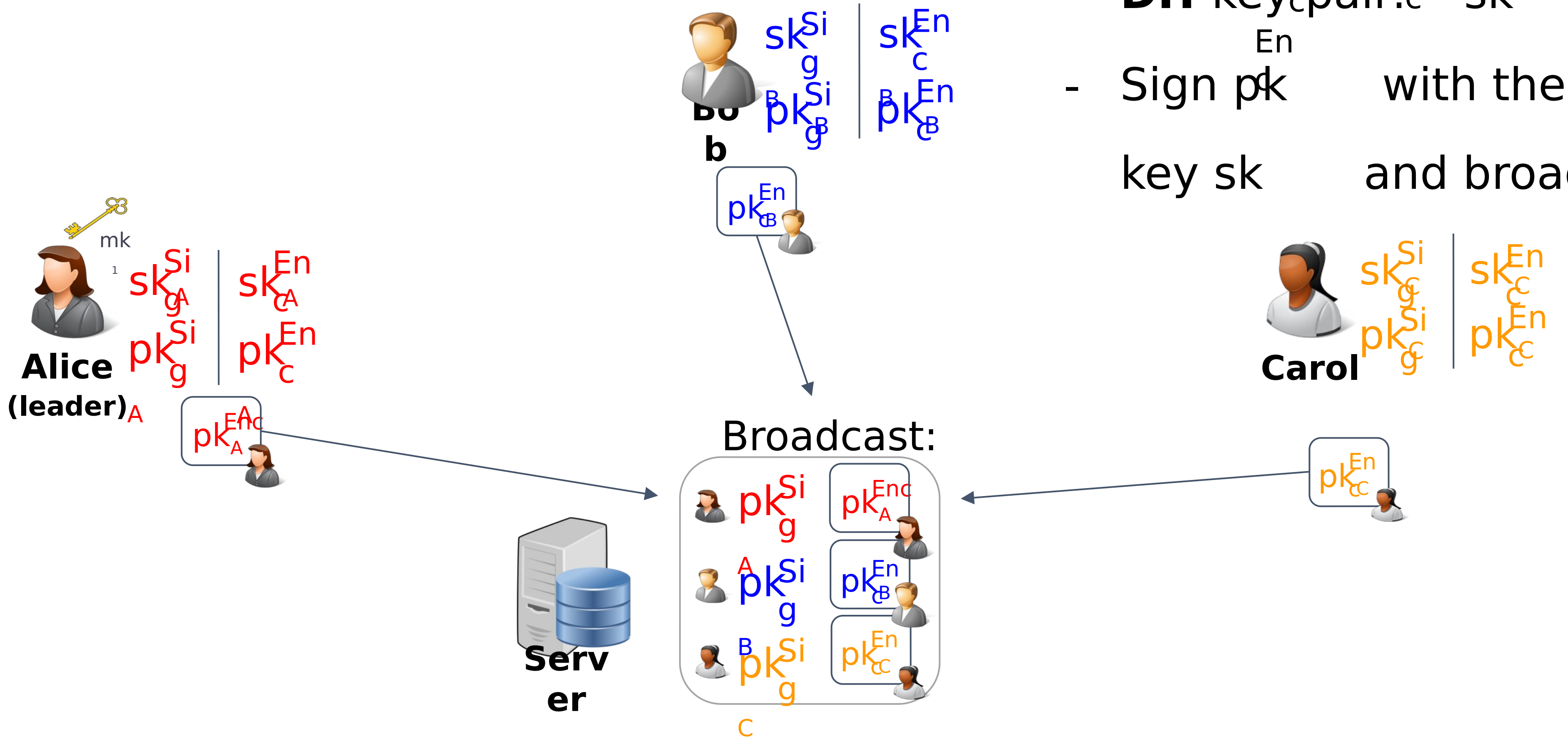
E2EE: Ephemeral DH keys

- Generate an **ephemeral DH** key pair: sk_c^{En}, pk_c^{En}
- Sign pk_c^{En} with the device key sk_g^{Si} and broadcast



E2EE: Ephemeral DH keys

- Generate an **ephemeral DH** key pair: sk_c^{Enc}, pk_c^{Enc}
- Sign pk_c^{Enc} with the device key sk_c^{Sig} and broadcast



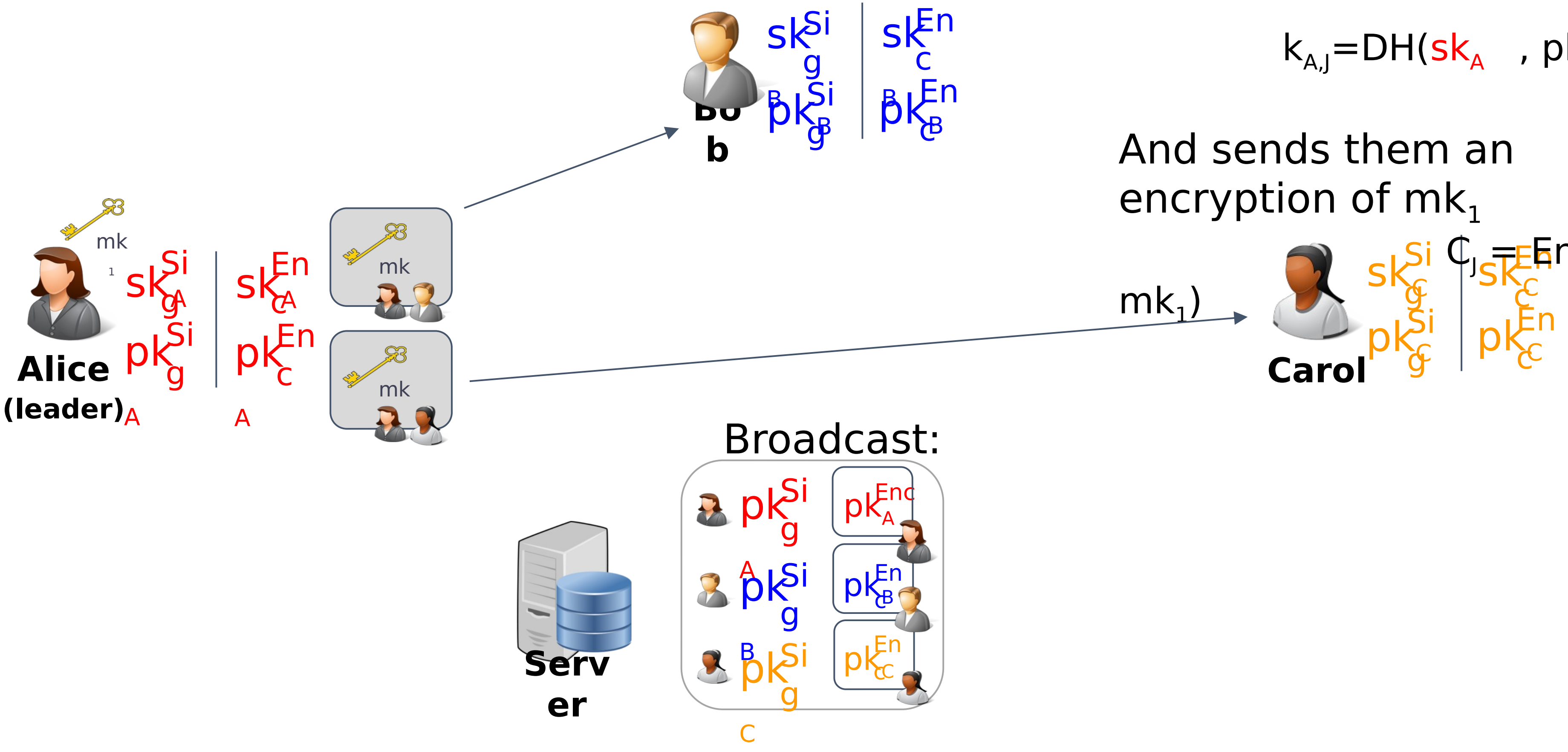
E2EE: Sharing Meeting Key

The leader A computes a shared key with each participant J

$k_{A,J}=DH(sk_A^{Enc}, pk_J^{Enc})$

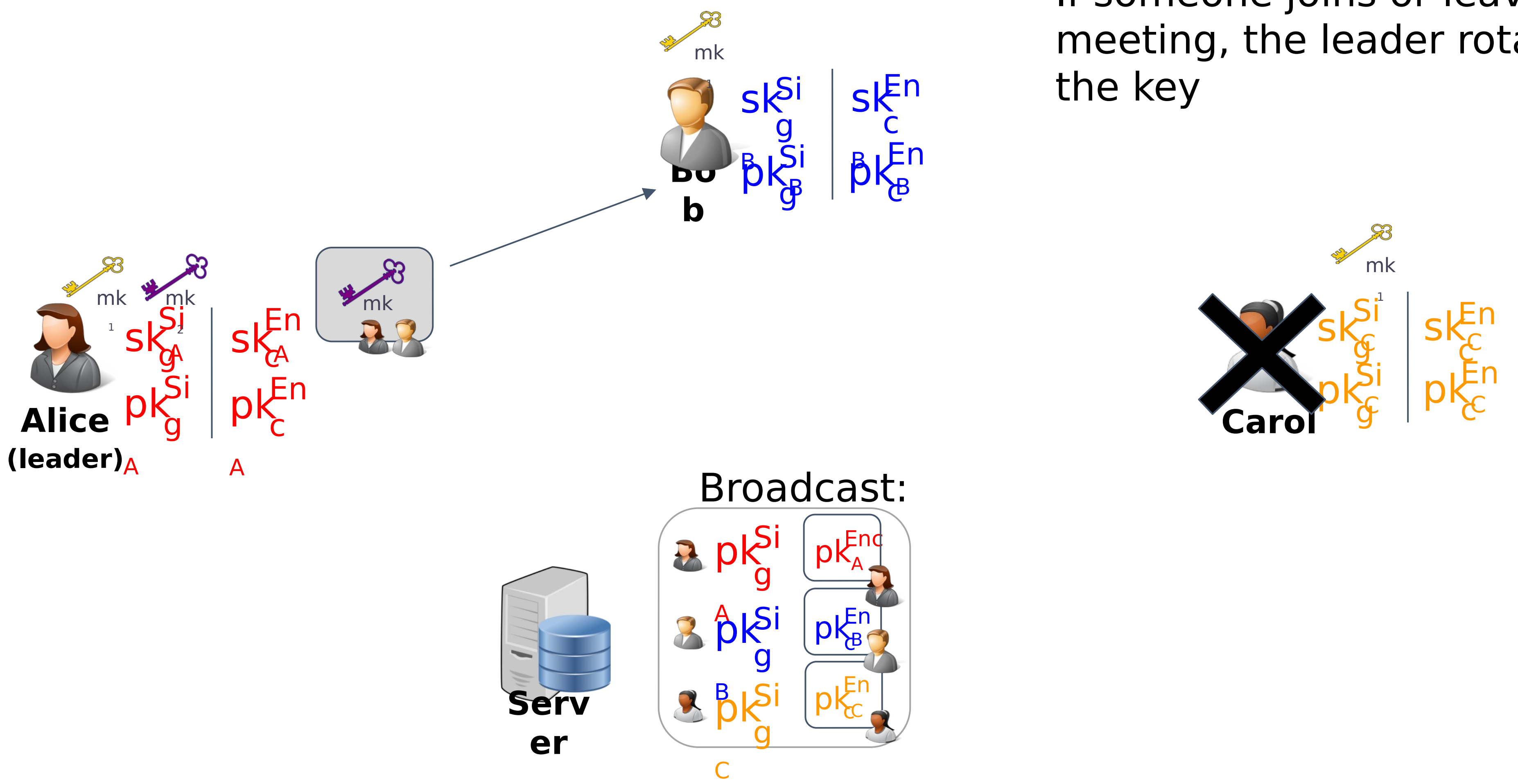
And sends them an encryption of mk_1

$C_J = Enc(k_{A,J}, mk_1)$



E2EE: Sharing Meeting Key

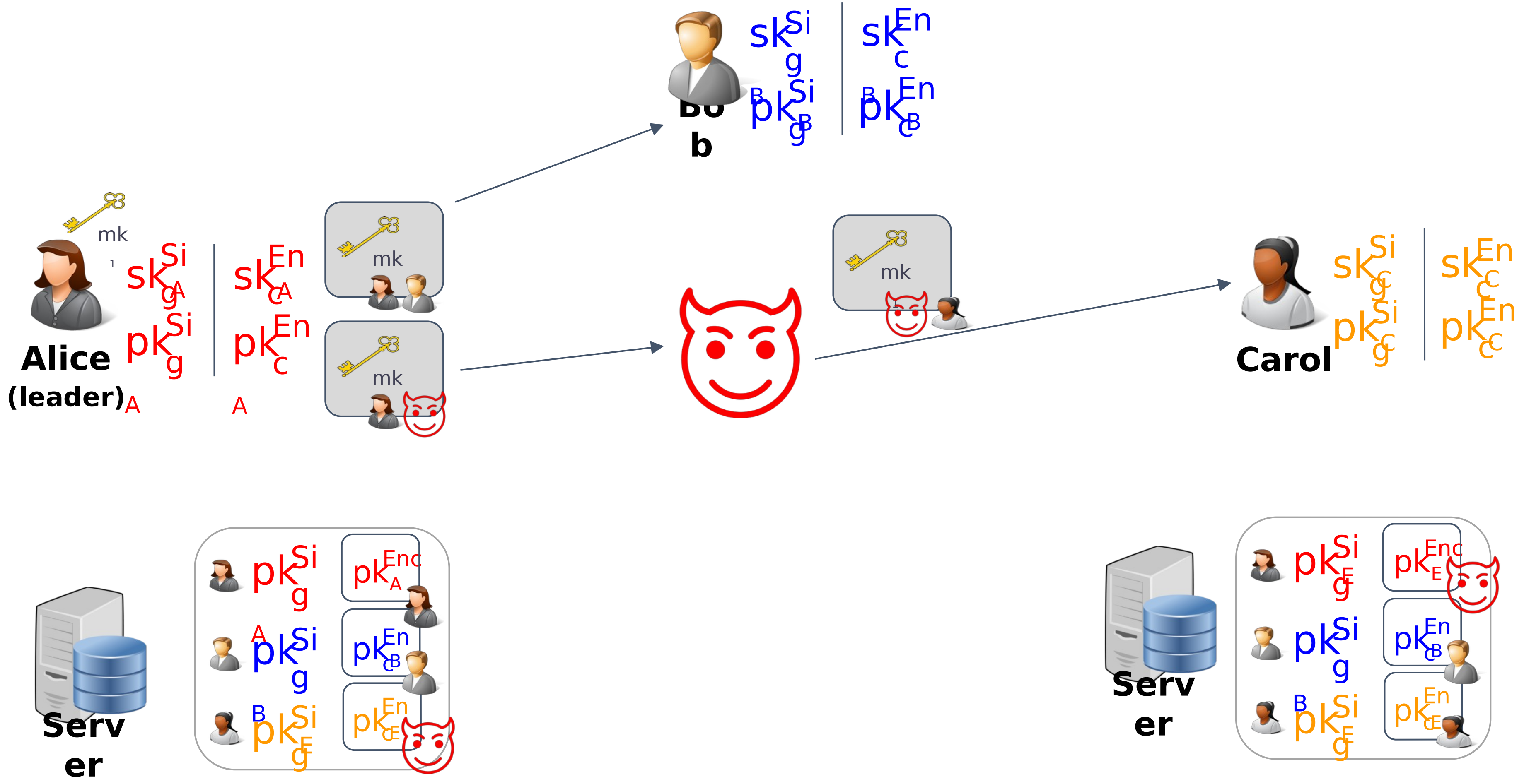
If someone joins or leaves the meeting, the leader rotates the key



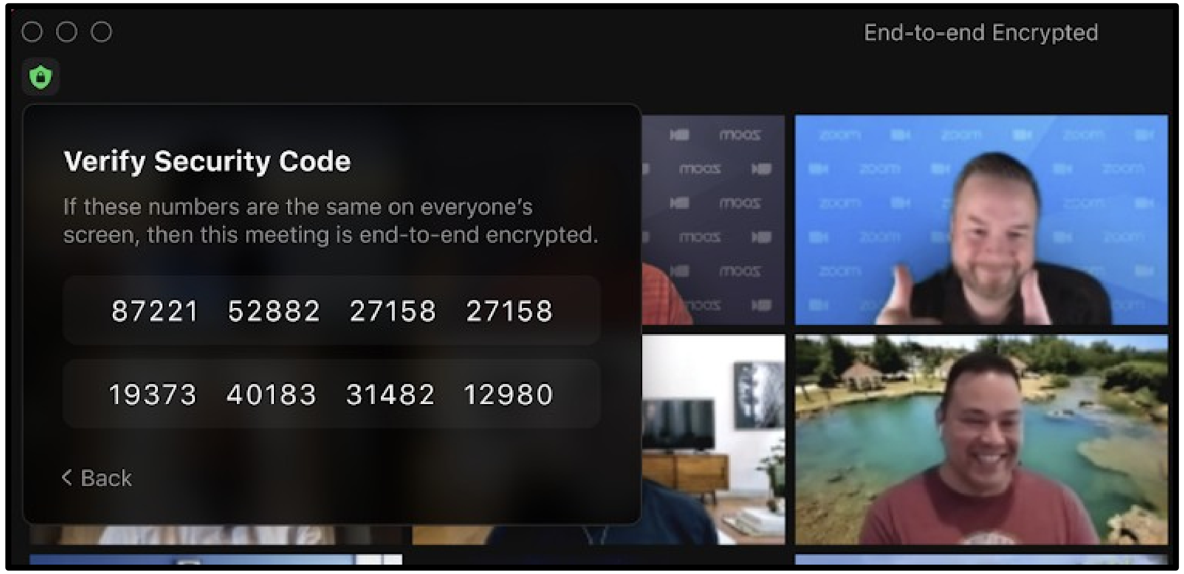
Heartbeats and Participant List

- At least every 10 seconds, the leader sends a “heartbeat” which includes:
  - The current key generation
  - A list of meeting participants who know this key
- In this phase, display names are arbitrary; you can change yours to “ ”

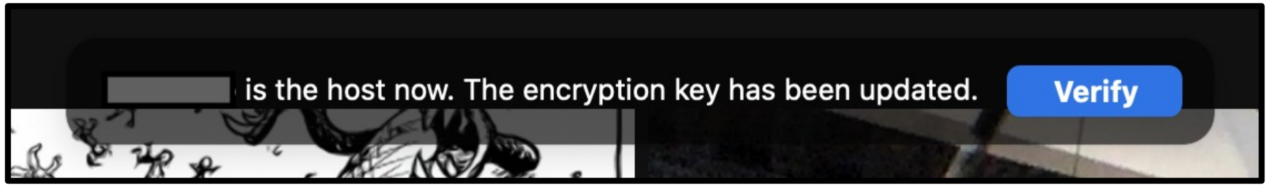
MiTM Attack



Preventing Meddler-in-the-Middle (MITM)

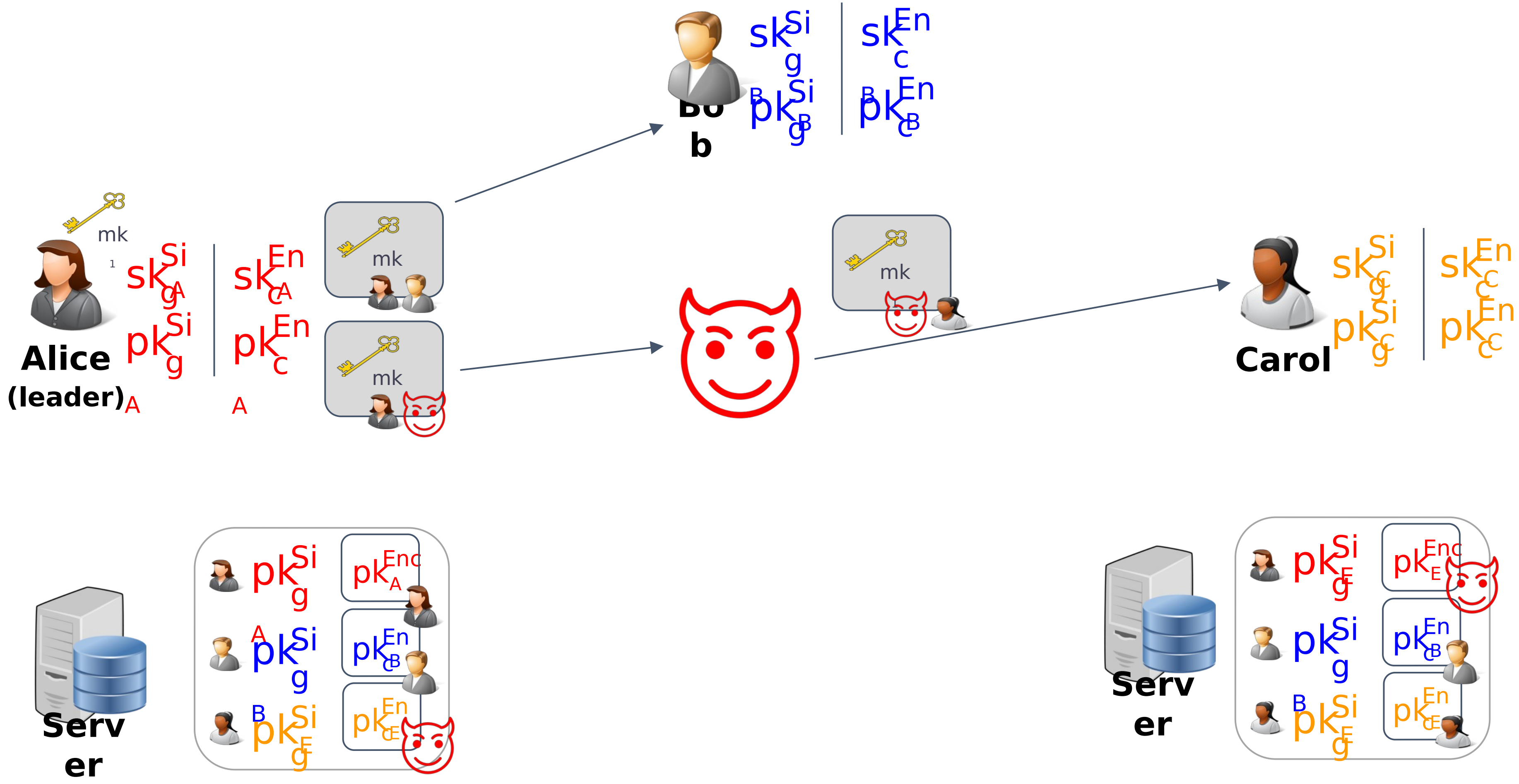


Meeting leader security code

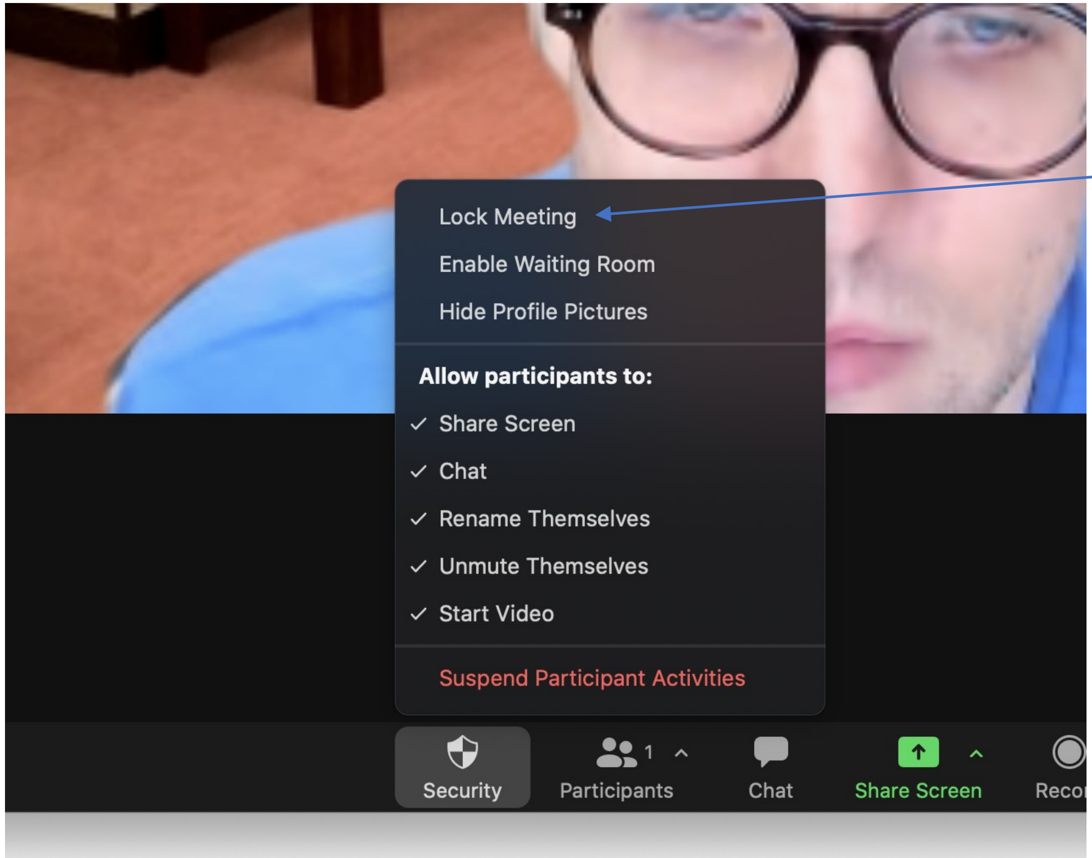


Leader change notification

MiTM Attack



Preventing MITM 2: “Lock The Meeting”



Meeting leader will not key in new participants once meeting is locked.

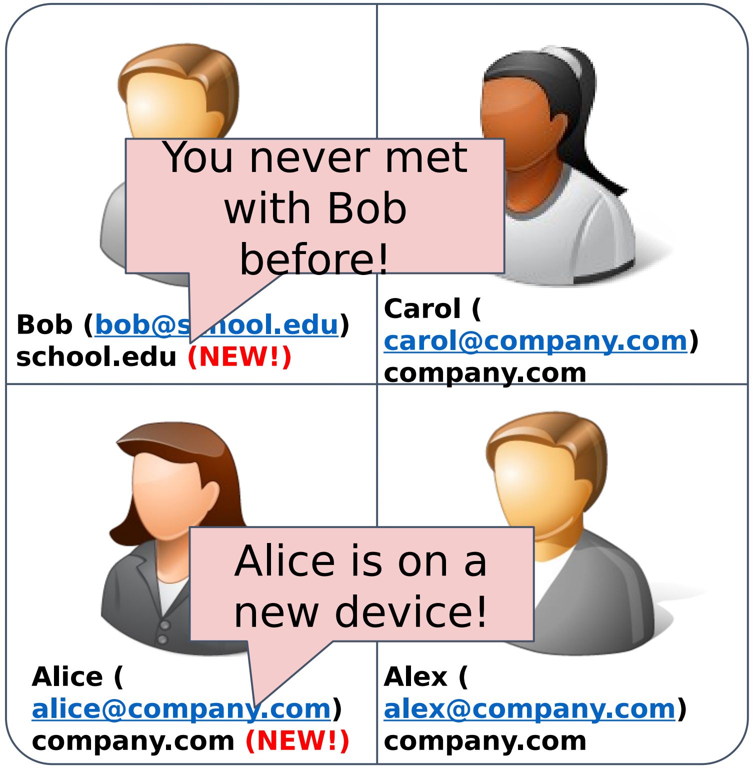
What attacks can you think of?

- Server injects spies into meetings
- Deep fakes
- Users not exchanging meeting codes or doing it incorrectly
- (Join and Drop) x 100
- Server tricks user into accessing E2EE via Web
- Bad binary

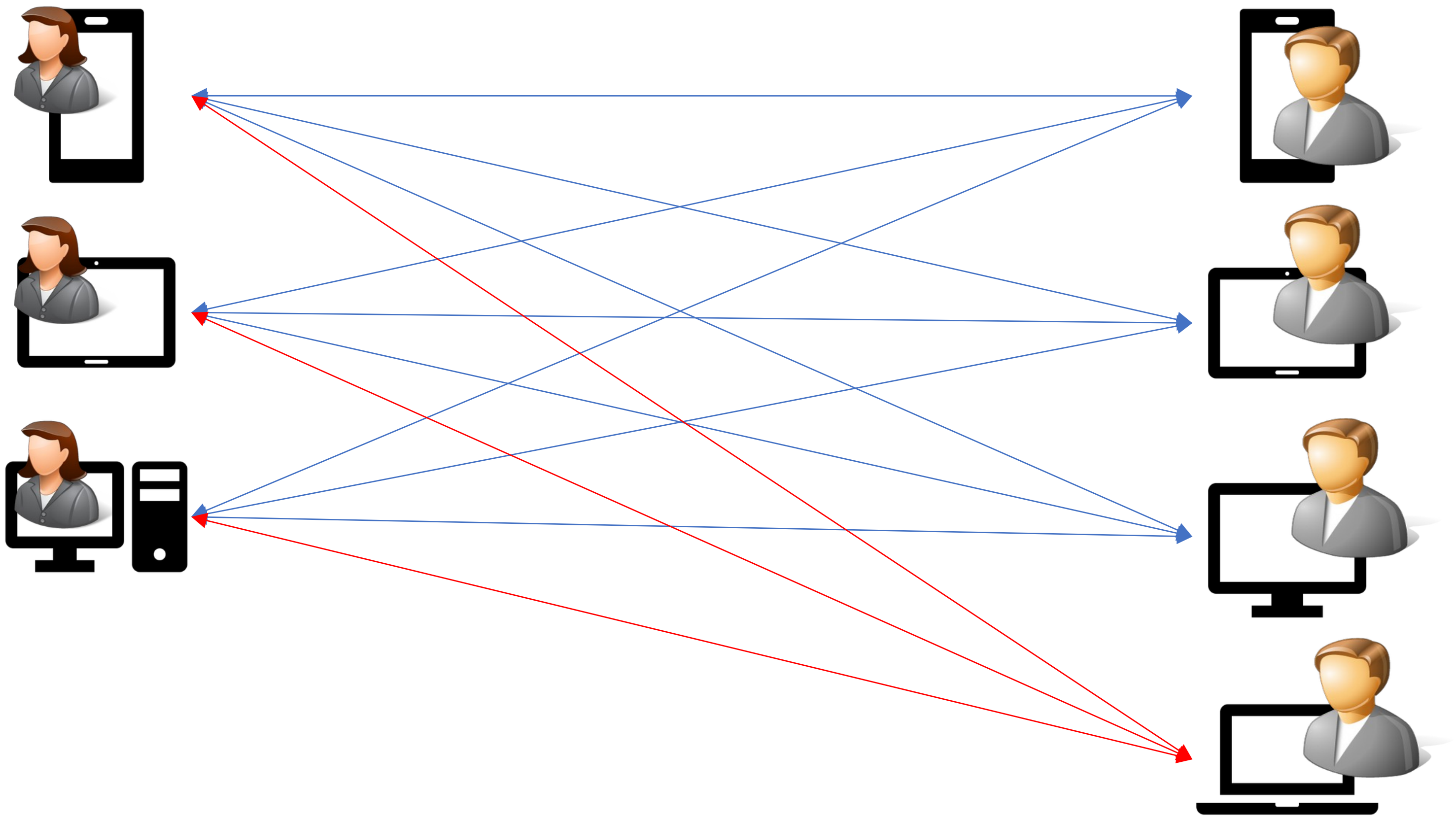
Phase 2+ of E2EE: Better Identity

- Participants remember each other's public keys (TOFU)
- Third-party IDPs can independently attest to these identities

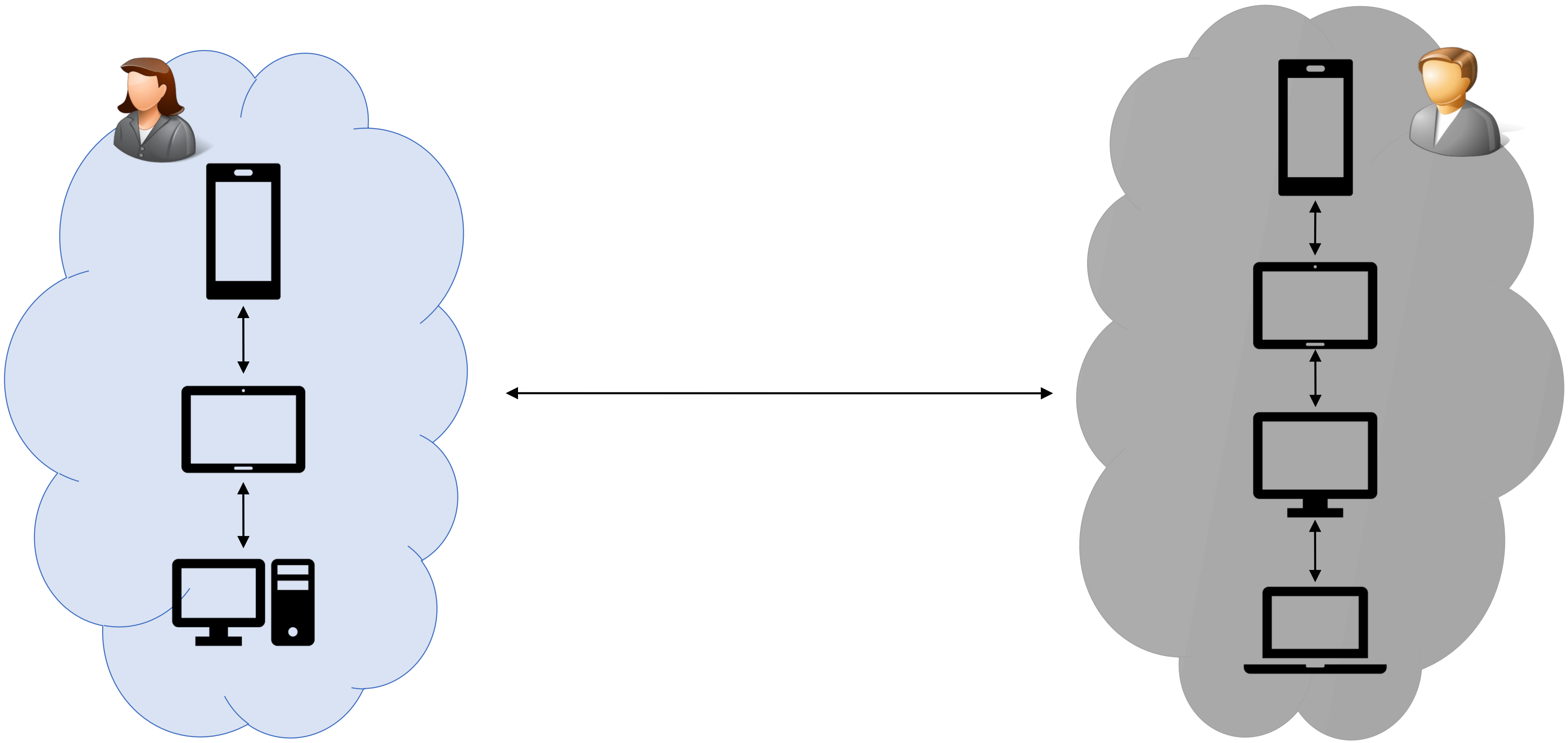
Participants Remember Public Keys (TOFU)



Problem: Quadratic Warnings

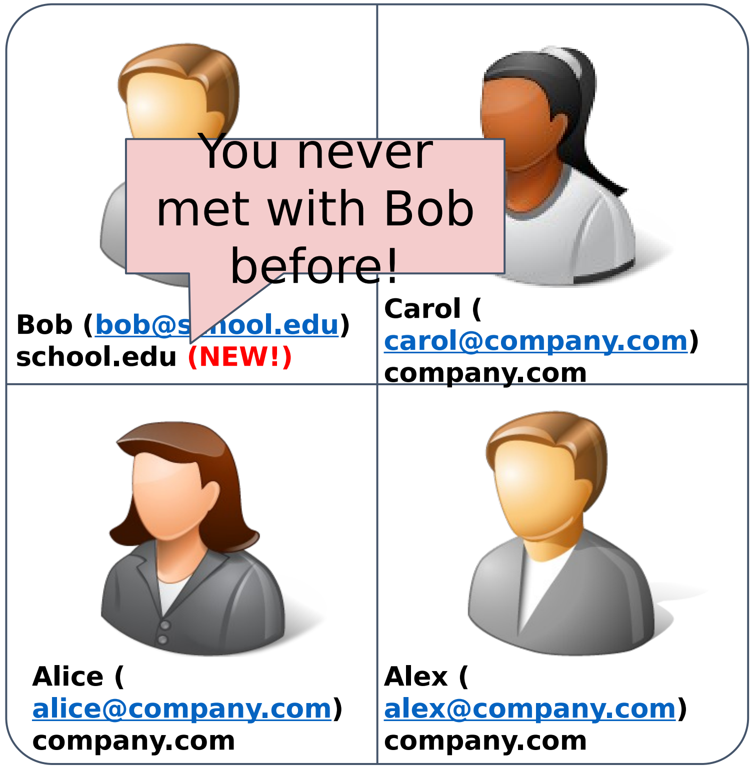


Solution: Personal Device Clouds

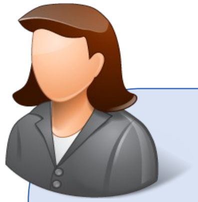


Contact Sync

- Better security and UX through personal device cloud model.
- Clients are alerted of new participants (not new devices)
- Contact records are **encrypted** and synced across devices, *without the server having plaintext access to these records.*

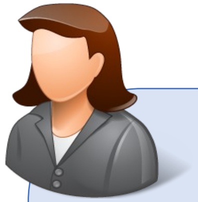


Solution: Personal Device Clouds



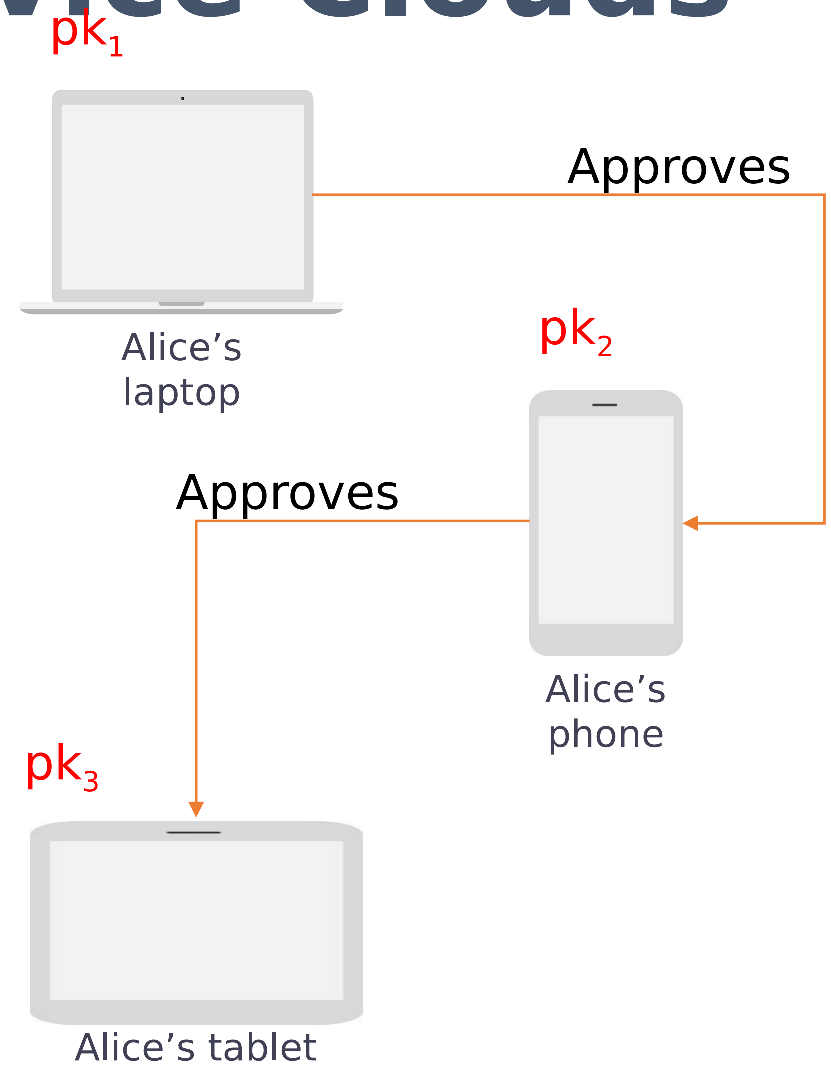
(**Display Name:** “Alice Jones”)
Email: alice@company.com

Solution: Personal Device Clouds



(**Display Name:** "Alice Jones")
Email: alice@company.com

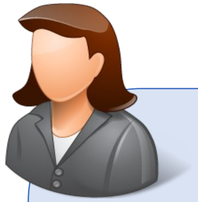
"Alice's laptop": pk_1
"Alice's phone": pk_2
"Alice's tablet": pk_3



Approval: Expanding the Circle of Trust

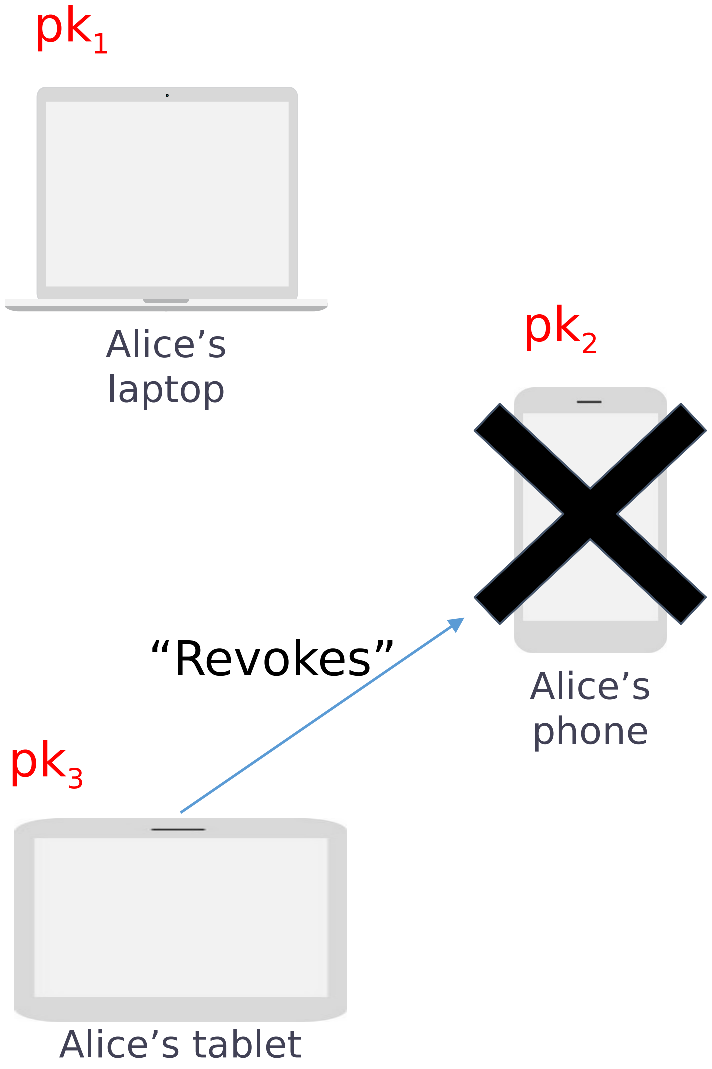
1. Delegates signing authority to new devices via signatures
2. Gossips private keys and secrets, so devices can sync data without server-trust

Personal Device Clouds: Revocation

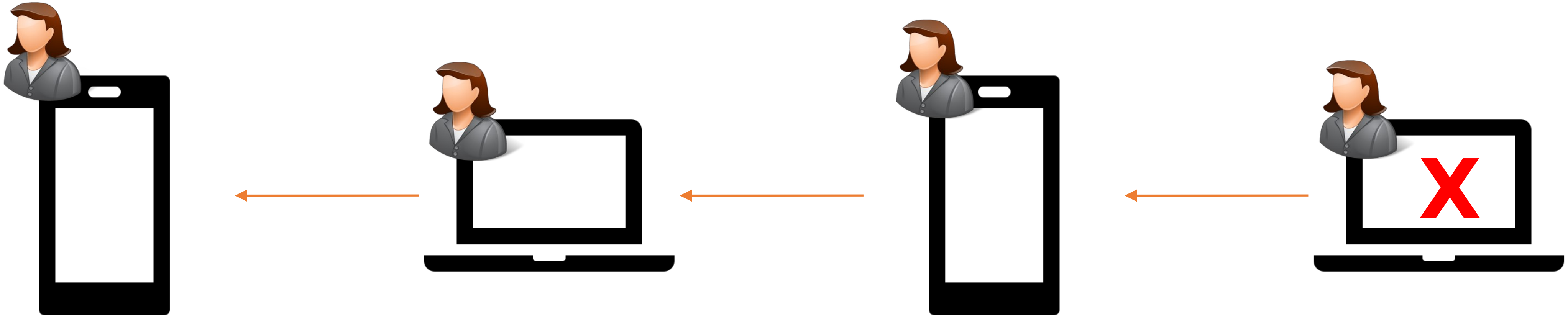


(**Display Name:** “Alice Jones”)
Email: alice@company.com

“Alice’s laptop”: pk$_1$
~~“Alice’s phone”: pk$_2$~~ **(REVOKED)**
“Alice’s tablet”: pk$_3$



Mechanism: “Signature Chains”



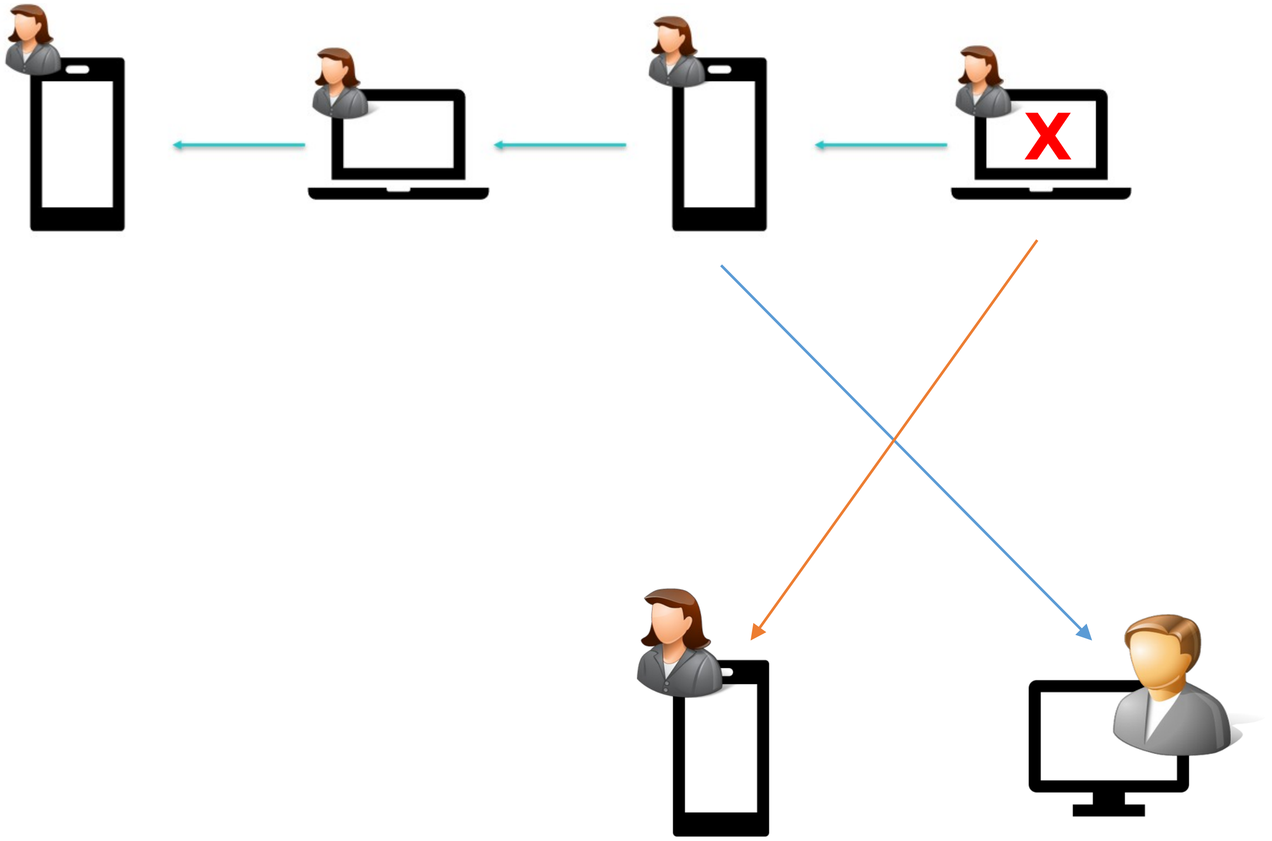
Link #1: provision phone (self-signed)

Link #2: provision laptop (self-signed)

Link #3: approve laptop (signed with phone)

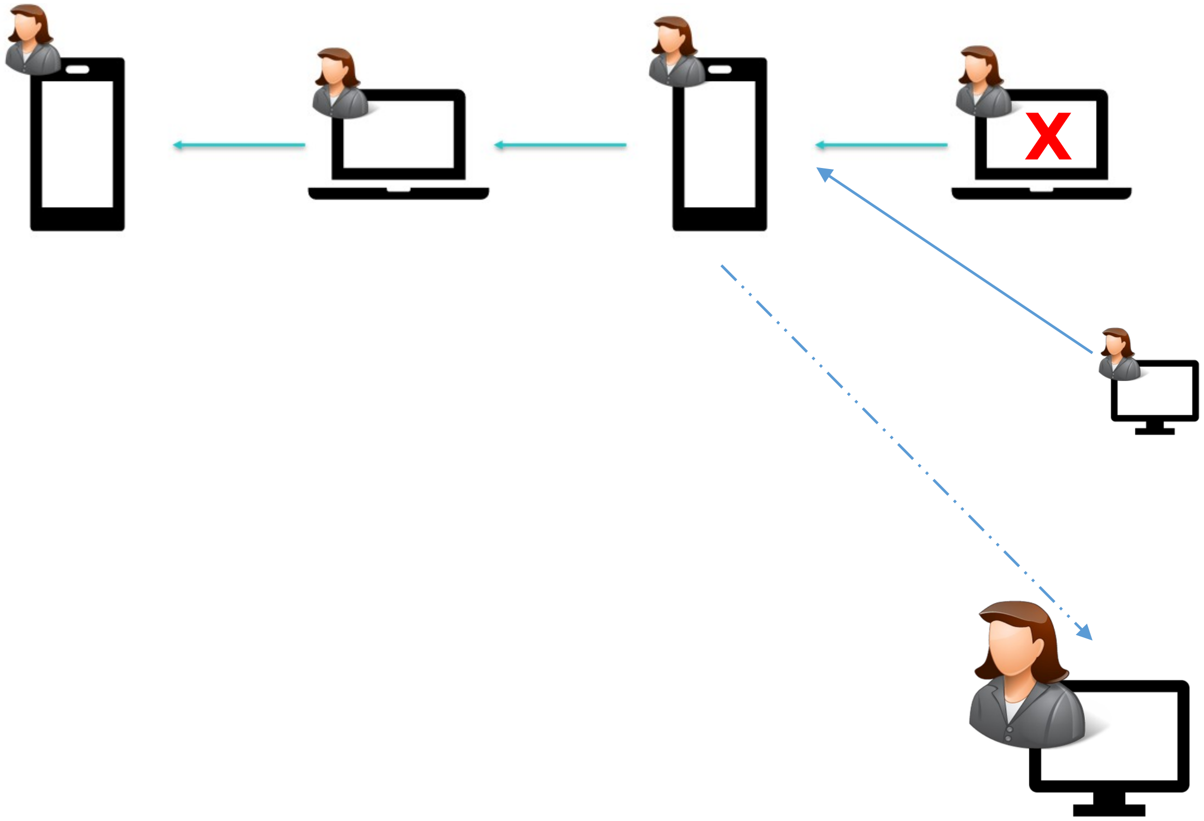
Link #4: revoke phone (signed with laptop)

Attacks Against Signature Chains



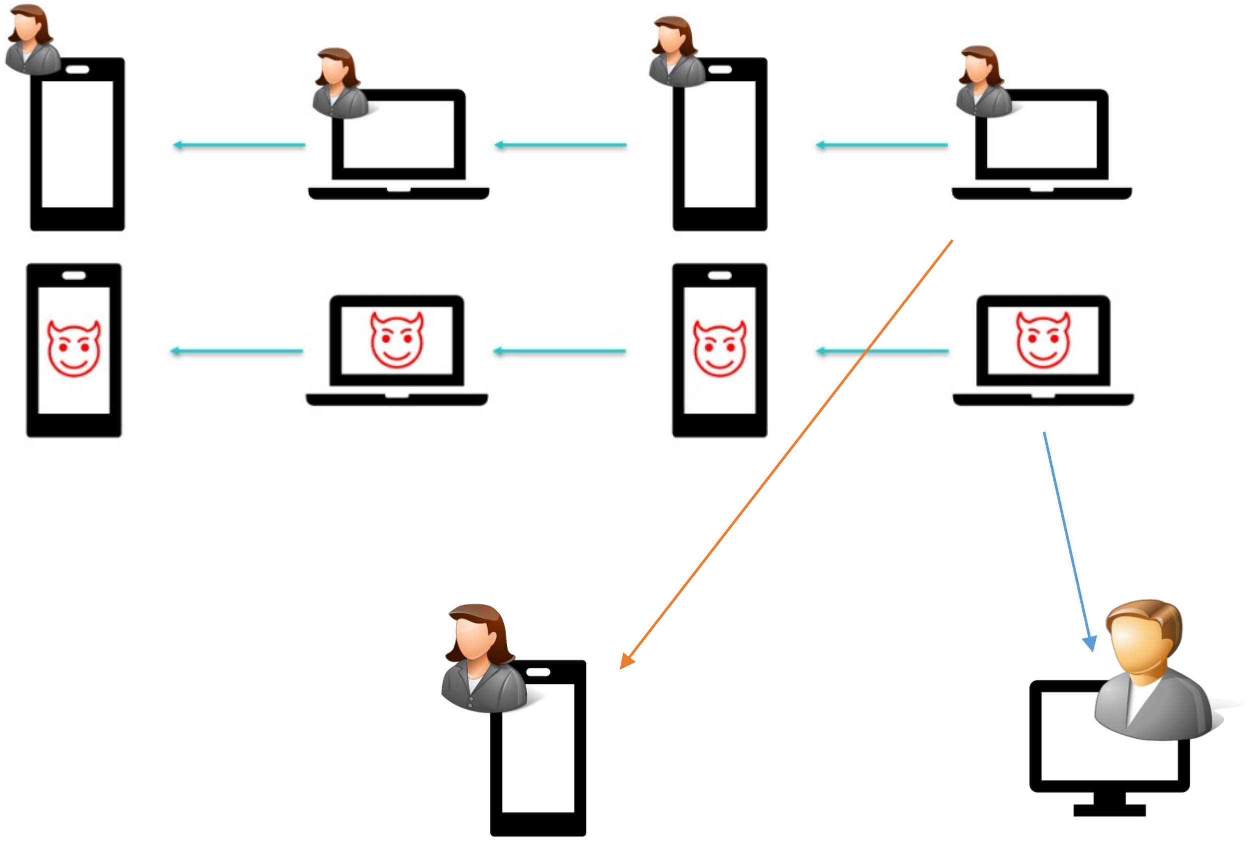
"Withholding Attack"

Attacks Against Signature Chains



“Forking Attack”

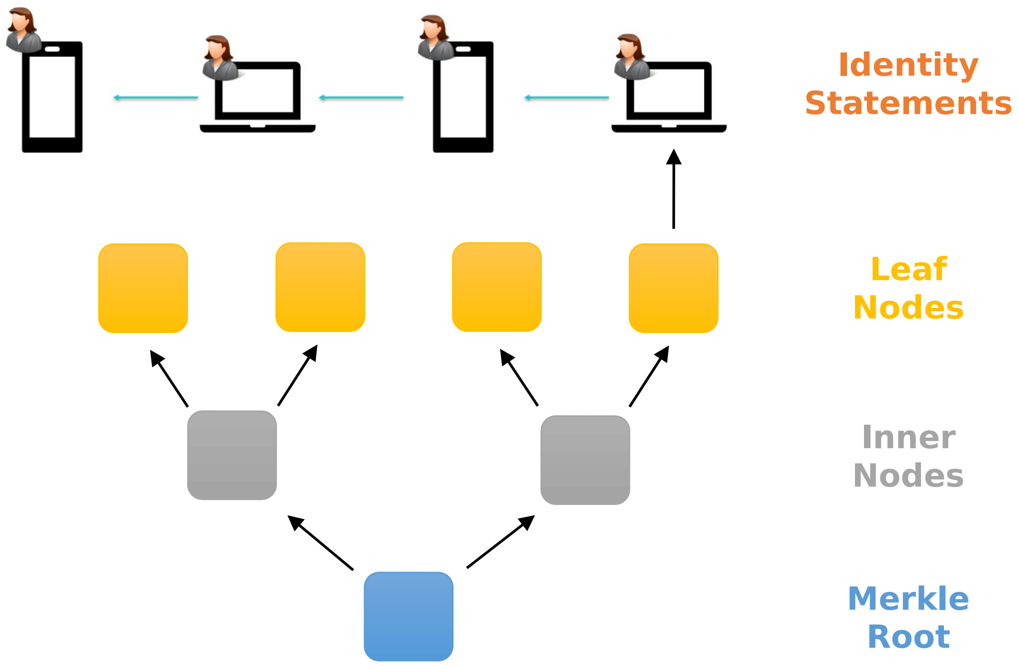
Attacks Against Signature Chains



"Substitution Attack"

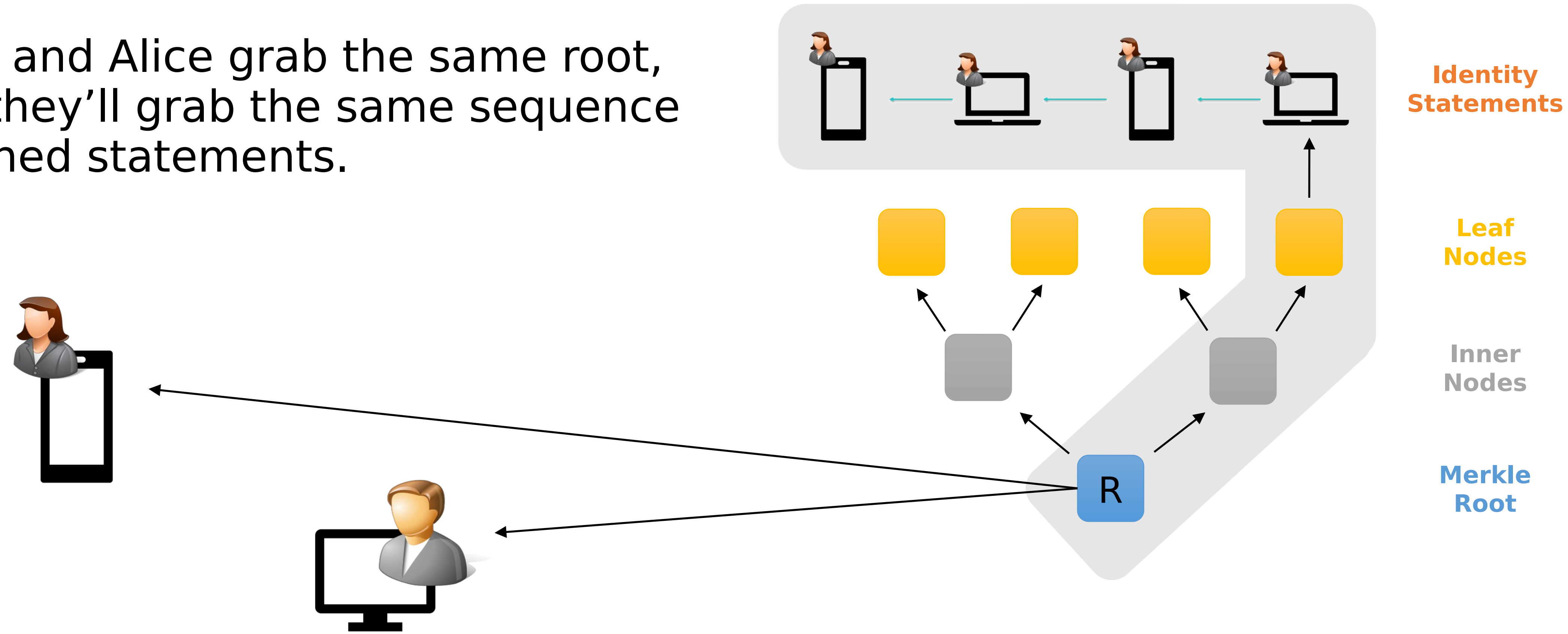
Solution: Transparency Tree

- Store all identities in an authenticated data structure (similar to Keybase, CONIKS, Key Transparency, SEEMless) which is:
  - Append only
  - Privacy preserving
  - Auditable
- This ensures all users have a consistent view of any identity, and that any impersonation attacks can be detected

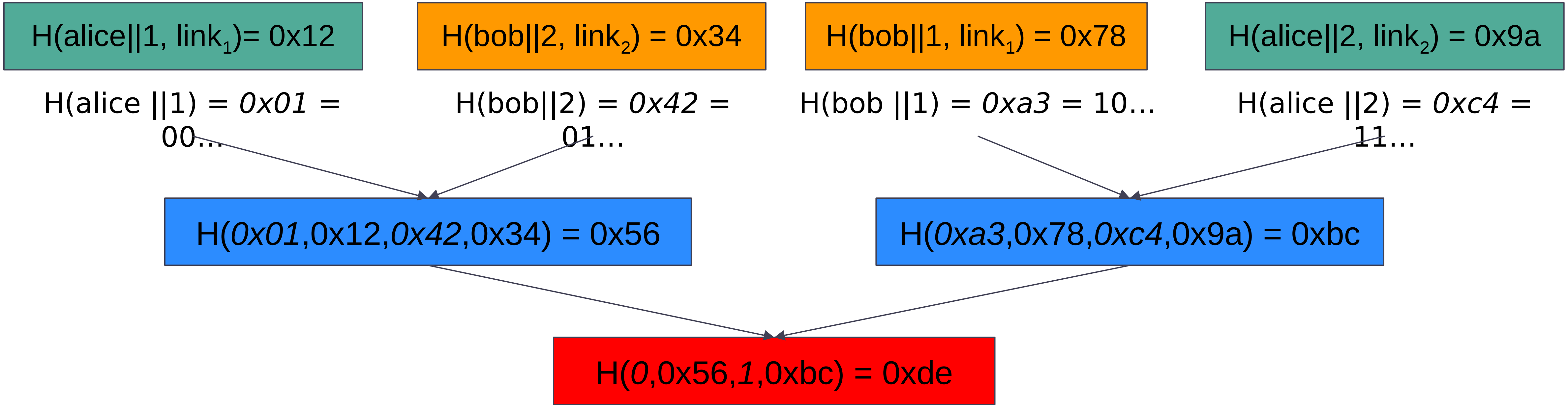


Solution: Transparency Tree

- If Bob and Alice grab the same root, then they'll grab the same sequence of signed statements.



Merkle Trees



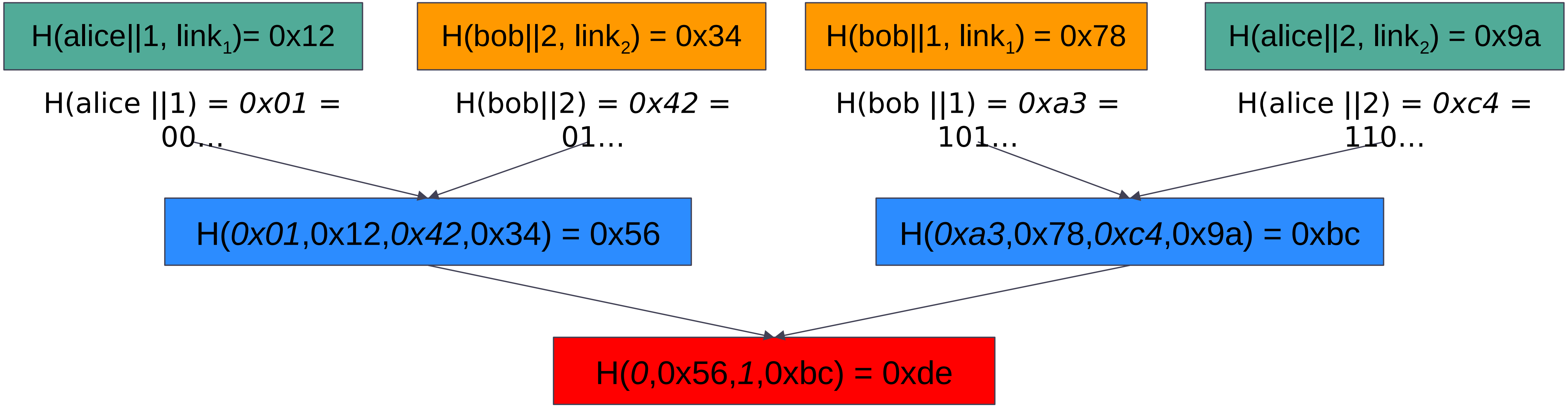
LookupLeaf(bob, 2)

1. Derive path from H(bob||2)
2. Query server for “bob,2”
3. Get back (link$_2$, 0x12, 0xbc) — note this is log(N) entries
4. Compute 0xde from path, leaf and sibling data
5. Check 0xde is correct out-of-band

Merkle Trees - LookupChain

- Query server for “alice”
- Get back:
  - R - root
  - (H(“alice||1”), $path_1$, $neighbors_1$, $link_1$)
  - (H(“alice||2”), $path_2$, $neighbors_2$, $link_2$)
  - (H(“alice||3”), $path_3$, $neighbors_3$, NULL)
- Verify:
  - Paths for links 1 and 2
  - That link 3 is excluded from the tree
  - R is the right root

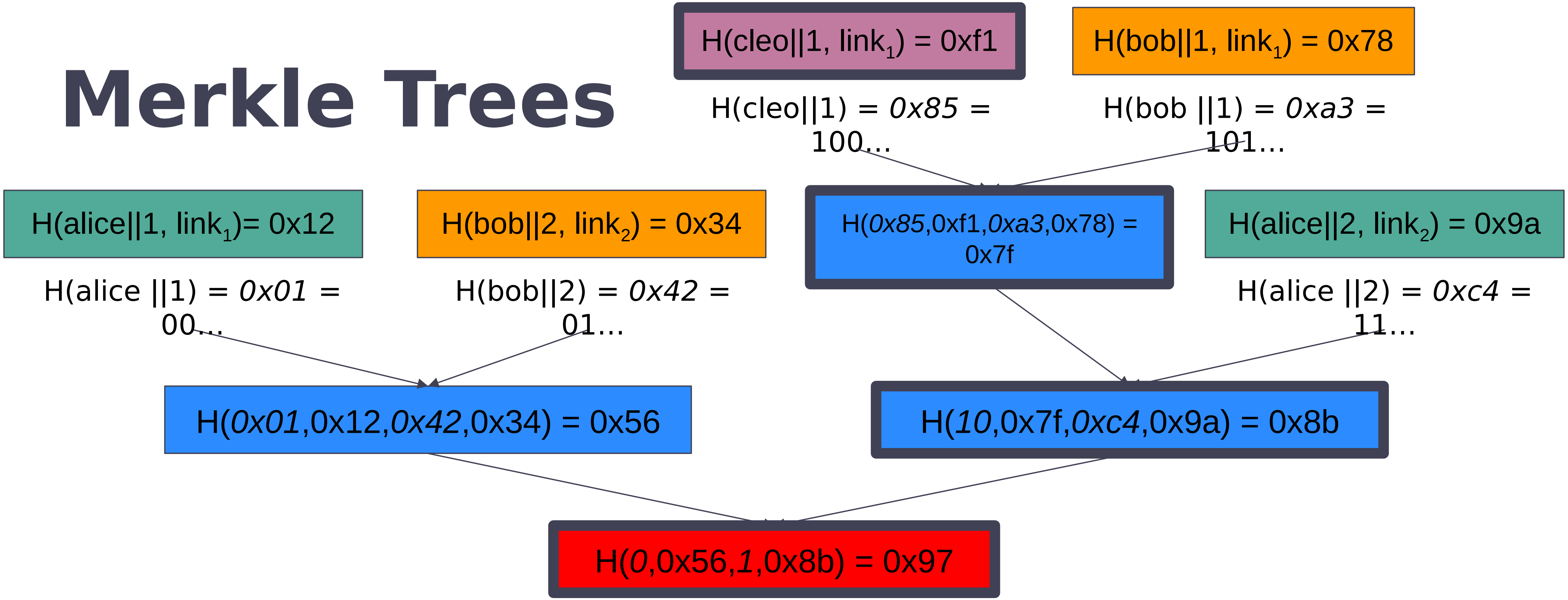
Merkle Trees – Exclusion



Say that:

- H(bob||1) = *0xa3* = 101…
- H(alice||2) = *0xc4* = 110…
- H(bob||3) = *0xff* = 111…

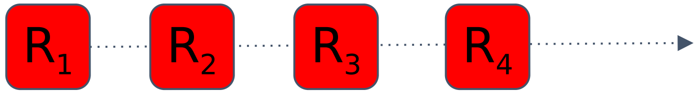
Merkle Trees



Update Protocol

- log(N) updates (bold boxes)
- Note how root hash changes

Merkle Trees - Audit



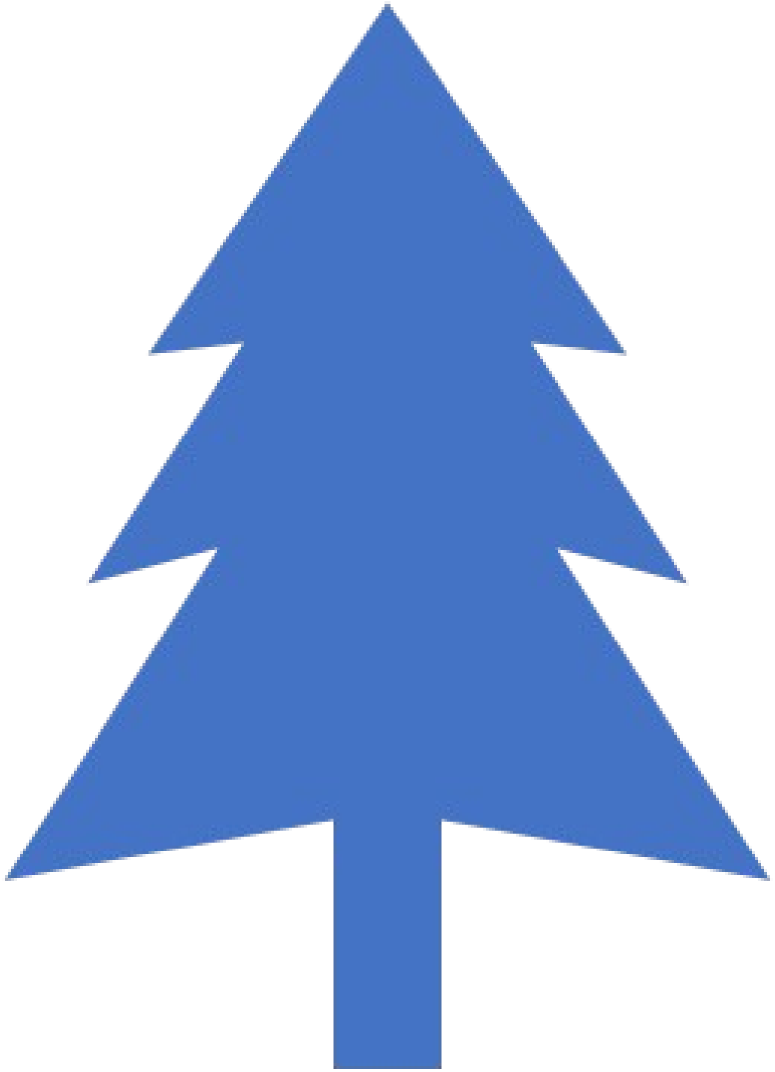
- Assume R_i is valid
  - how can you prove R_{i+1} obeys the “append-only property”?
- Audit one transition in log(N)
- Audit the whole tree in N*log(N)

Merkle Trees – Recap

- The good:
  - Can prove inclusion/exclusion of any path in log(N)
  - Can update in log(N)
  - Can verify the last a updates in a*log(N)
- What’s wrong?

What’s Wrong with Merkle Trees?

- Standard hash construction using H = sha256 leaks information about paths



Verifiable Random Functions

- **Gen**() → sk,pk
- **VRF**(sk, label) → out, proof
- **Verify**(pk,label,out,proof) →{0,1}

Security:

- **Uniqueness**
  - encompasses verifiability
- **Pseudorandomness**
  - encompasses secrecy of hash production
- **Collision Resistance**
  - Like regular hash functions

Example (simplified): Let (G,g) be a DDH group

- **Gen**() -> sk = x, pk = g^x
- **VRF**(x, label) → (H(label)^x, DH proof)
- **Verify**(g^x, label, H(label)^x, DH proof) → {0,1}

Want to learn more? Google "IETF VRFs"

Lookup Protocol

- Query server for “alice”
- Get back:
  - R - root
  - (VRF(sk, “alice||1”), $path_1$, $neighbors_1$, $link_1$)
  - (VRF(sk, “alice||2”), $path_2$, $neighbors_2$, $link_2$)
  - (VRF(sk, “alice||3”), $path_3$, $neighbors_3$, NULL)
- Verify:
  - VRF’s with pk for all 3 outputs
  - Paths for links 1 and 2 (as in previous Merkle tree)
  - That link 3 is excluded from the tree
  - R is the right root

Transparency Trees

H(cleo||1, link$_1$) = 0xf1

VRF(sk, cleo||1) = *0x85*

H(bob||1, link$_1$) = 0x78

VRF(sk, bob ||1) = *0xa3*

H(alice||1, link$_1$)= 0x12

VRF(sk, alice ||1) = *0x01*

H(bob||2, link$_2$) = 0x34

VRF(sk, bob||2) = *0x42*

H(*0x85*,0xf1,*0xa3*,0x78
)
= 0x7f

H(alice||2, link$_2$) = 0x9a

VRF(sk, alice ||2) = *0xc4*

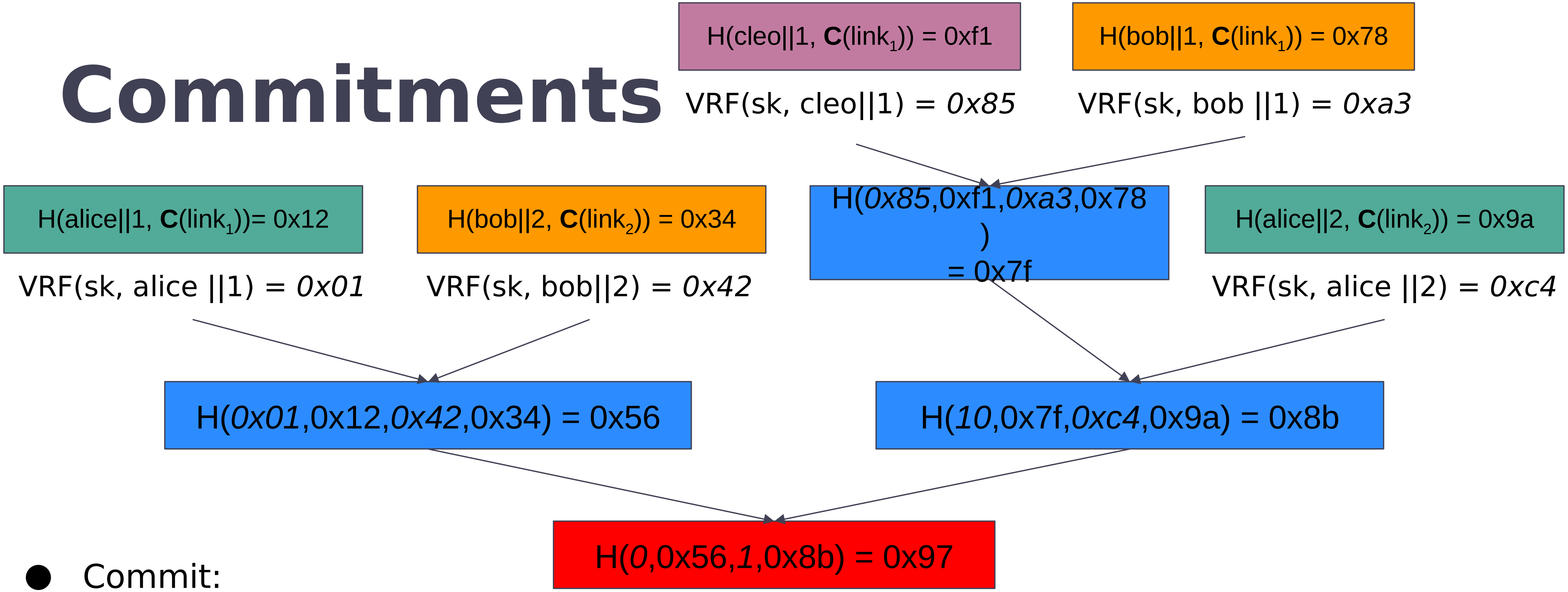
H(*0x01*,0x12,*0x42*,0x34) = 0x56

H(*10*,0x7f,*0xc4*,0x9a) = 0x8b

H(*0*,0x56,*1*,0x8b) = 0x97

There's still an issue!

Commitments



- Commit:
  - Pick random *r* (128 random bits)
  - C(x) = H(*r*,*x*)
- Reveal:
  - Provide *x* (which might be guessable) and *r*, which isn't.
- Server must now return the various r_i's along with paths

Important Note to System Builders

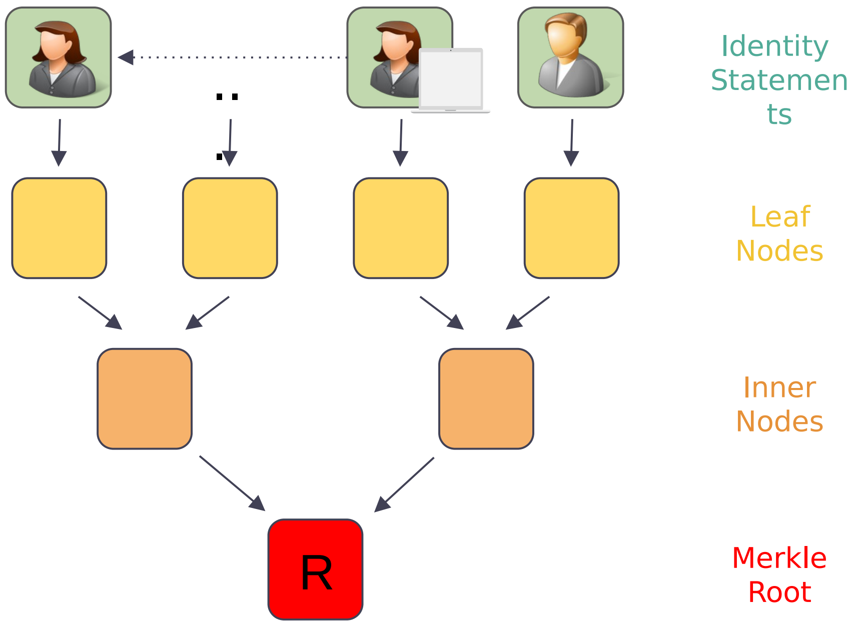
H(r,x) = SHA256(r||x)   ?????

- Beware of “length extension attacks”
  - Given H(r||x) and x, it’s possible to output H(r||x||y) without knowing r
- Can use SHA3(r||x)
- Or use HMAC-SHA256(r, x)

Zoom Transparency Tree - Recap

- Store all identities in an authenticated data structure (similar to Keybase, CONIKS, Key Transparency, SEEMless) which is
  - Append only
  - Privacy preserving (using VRF)
  - Auditable

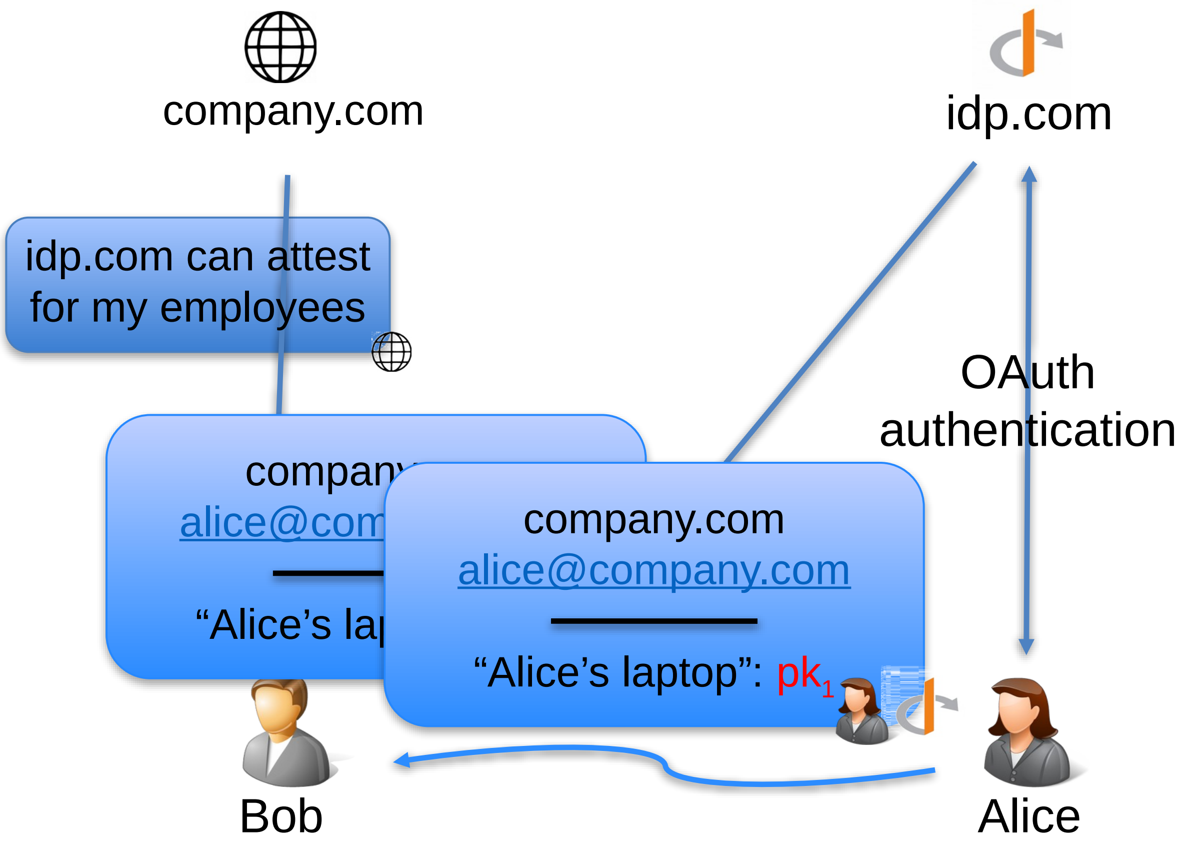
- This ensures all users have a consistent view of any Zoom identity, and that any impersonation attacks can be detected



Identity Provider Attestations

External Identity Providers can attest to their user's identities:

1. Alice authenticates to the IDP (OAuth2) and POSTs sigchain tail
2. Alice requests the IdP's signature over her sigchain tail
3. Participants check that the IdP is authorized by company.com by making a TLS request to the domain
4. Participants check the Identity and token



Recap

- Users have identities composed of multiple independent devices
- Users remember each other's keys (TOFU)
- The Zoom Transparency Tree will hold the server accountable and ensure consistency
- IdPs can independently attest to these identities

- **Applications go beyond meetings!**

Open Q&A

- Keybase
- Zoom
- Acquisitions
- Security startups
- Startups
- Big companies vs little companies

