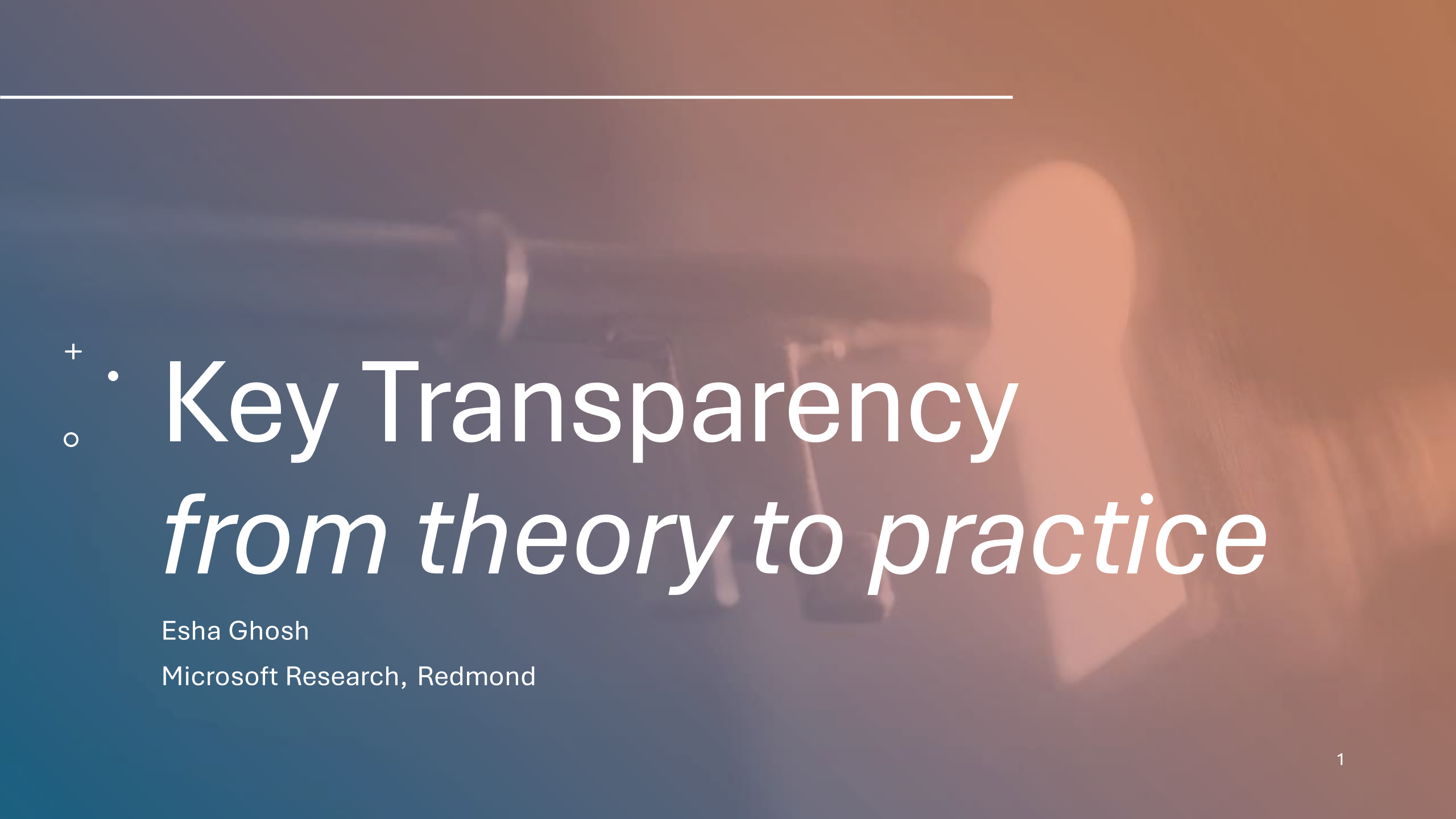


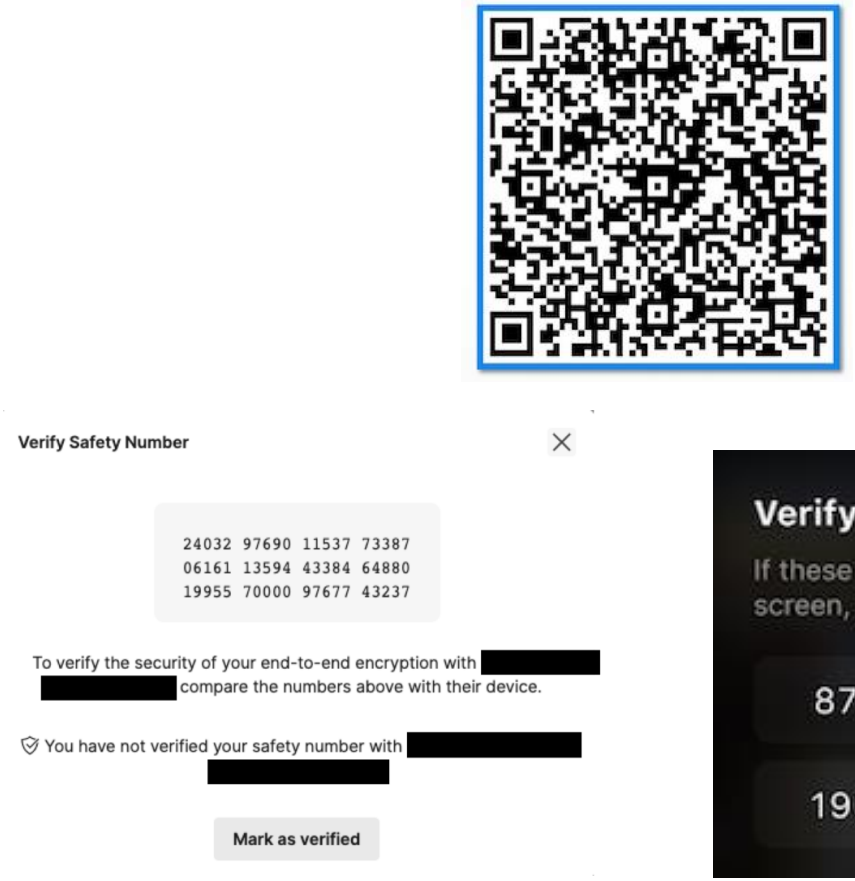
Key Transparency *from theory to practice*

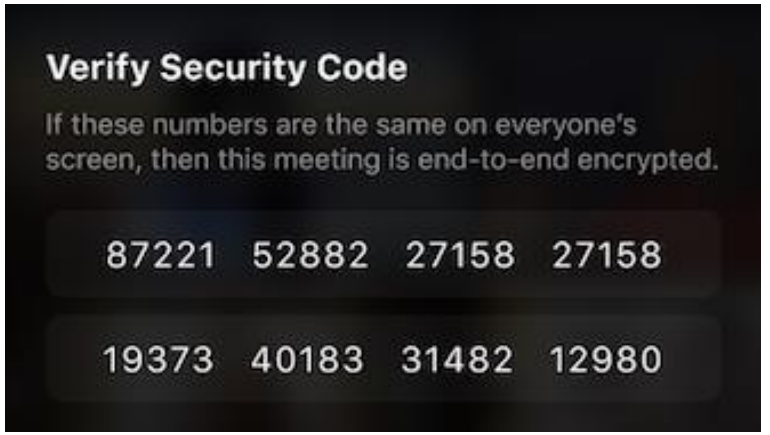
Esha Ghosh

Microsoft Research, Redmond

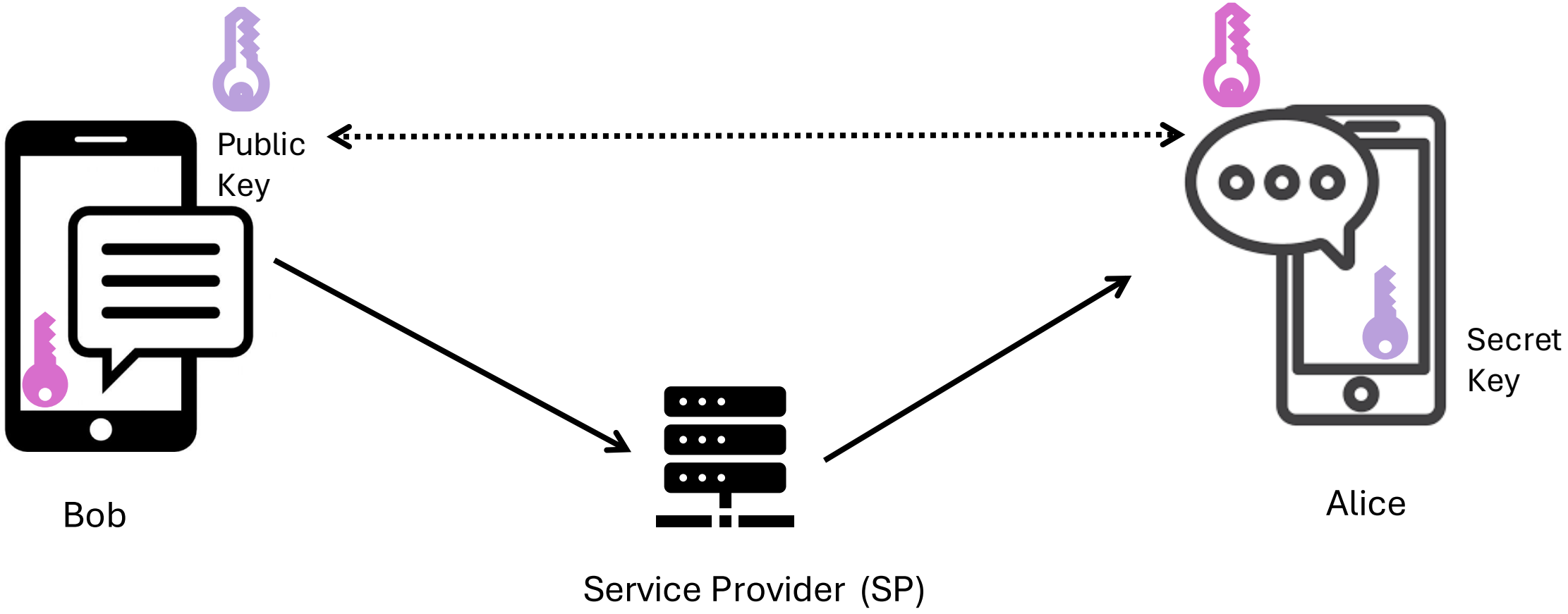
End-to-End Encrypted Communications

- Signal
- WhatsApp
- Messenger
- iMessage
- Skype
- Teams
- Zoom
- Webex
- …

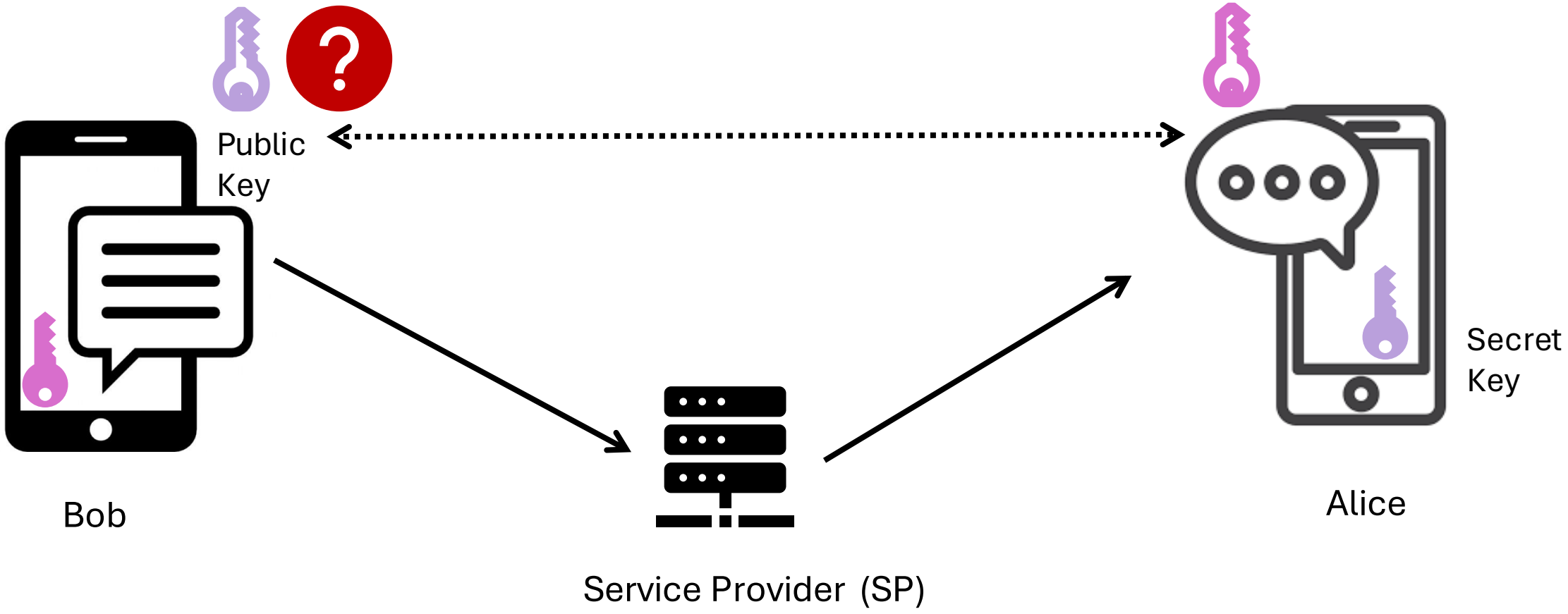




End-to-end encrypted messaging

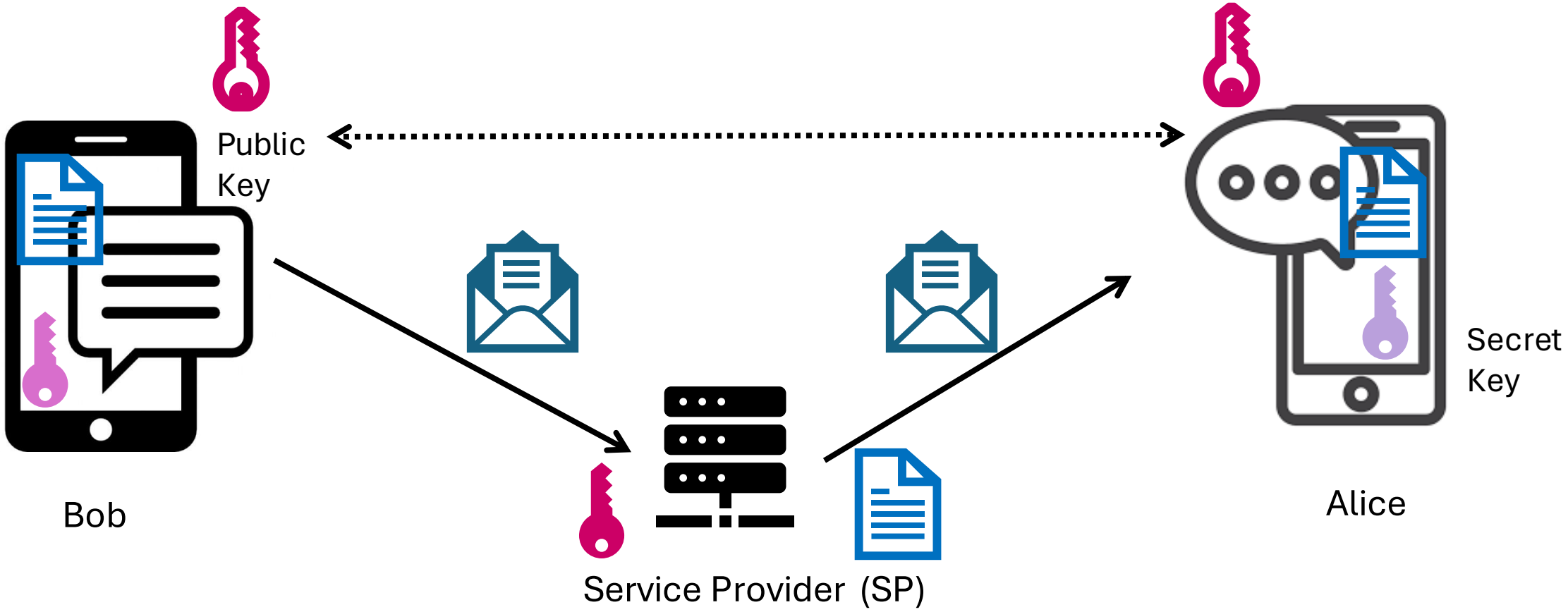


How does Bob get Alice's public key?

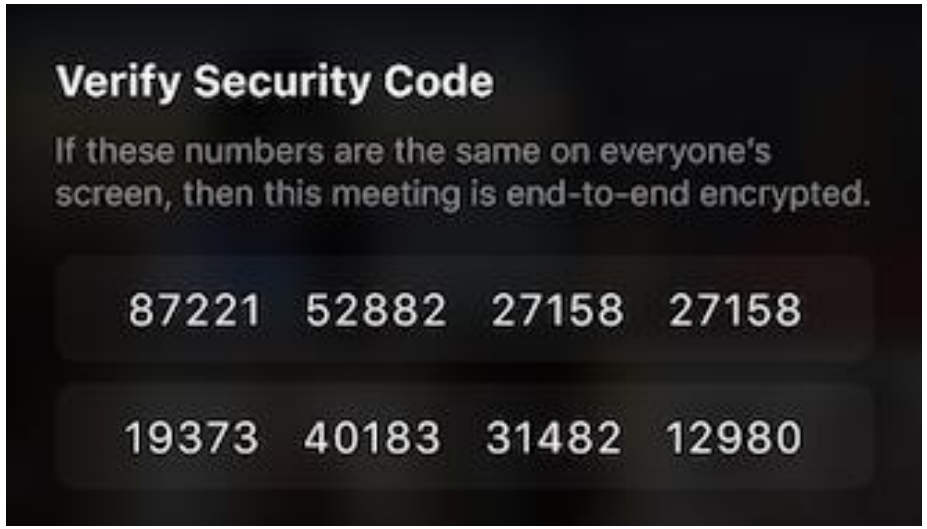


Service Provider manages public key directory and distribution

Undetectable Meddler-in-the-Middle

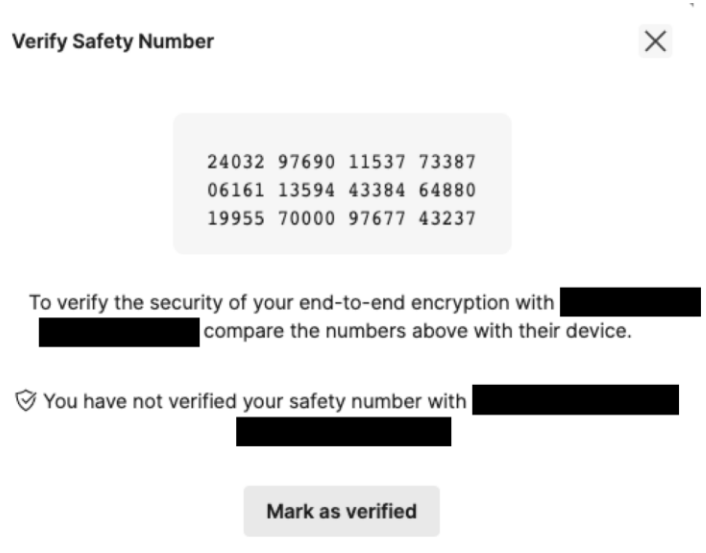


Traditional Approaches



Security Code

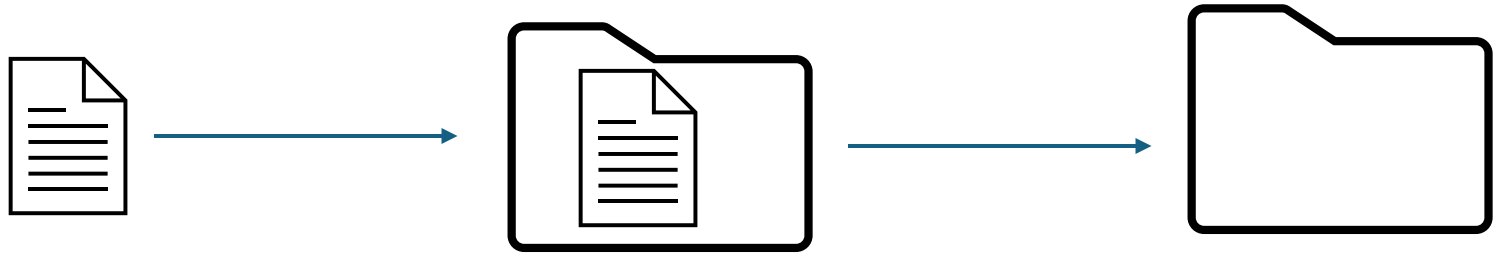
QR Code



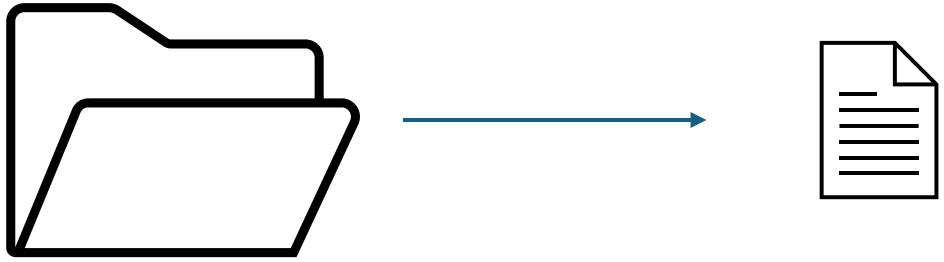
Safety String

Cryptographic commitments

Commit Phase



Opening Phase

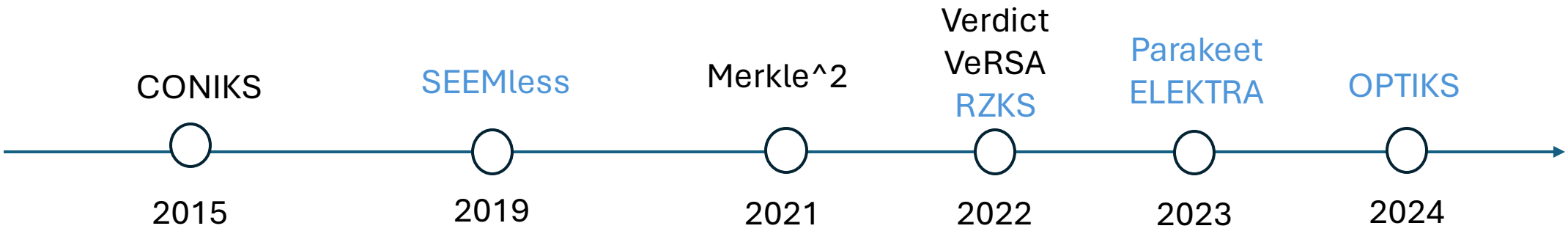


Key Transparency: Automating the checks

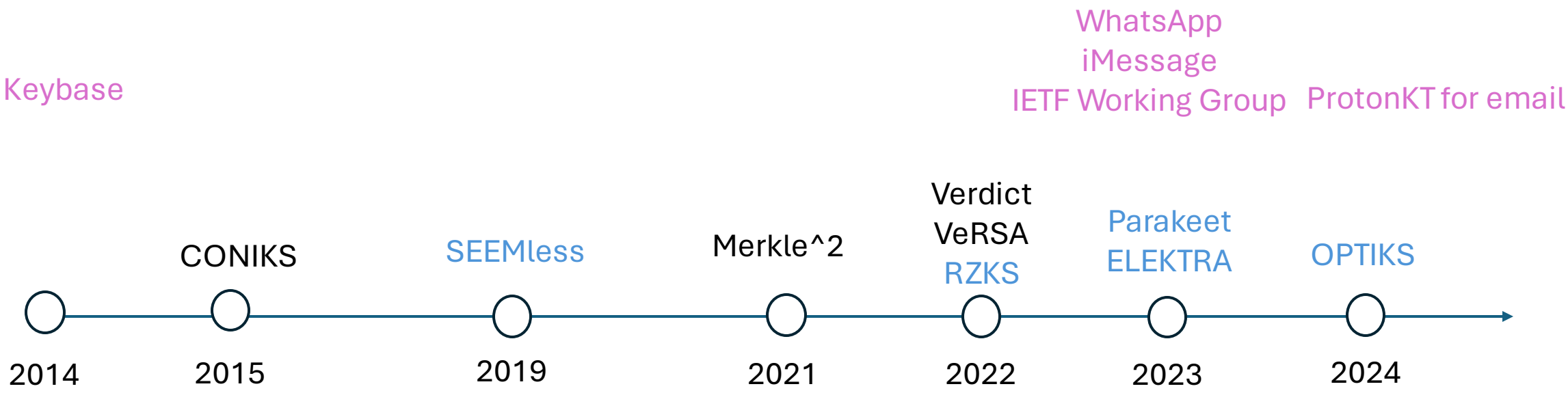
- Lifts the burden from users by automating the checks
- Requires the service provider to *commit* to its key directory periodically
- When Bob requests Alice's key, it comes with a proof with respect to the latest commitment
- Alice's client software can monitor her key in the directory:
  - *Regularly requests her own key* (and proof) and checks that it is correct.

Landscape

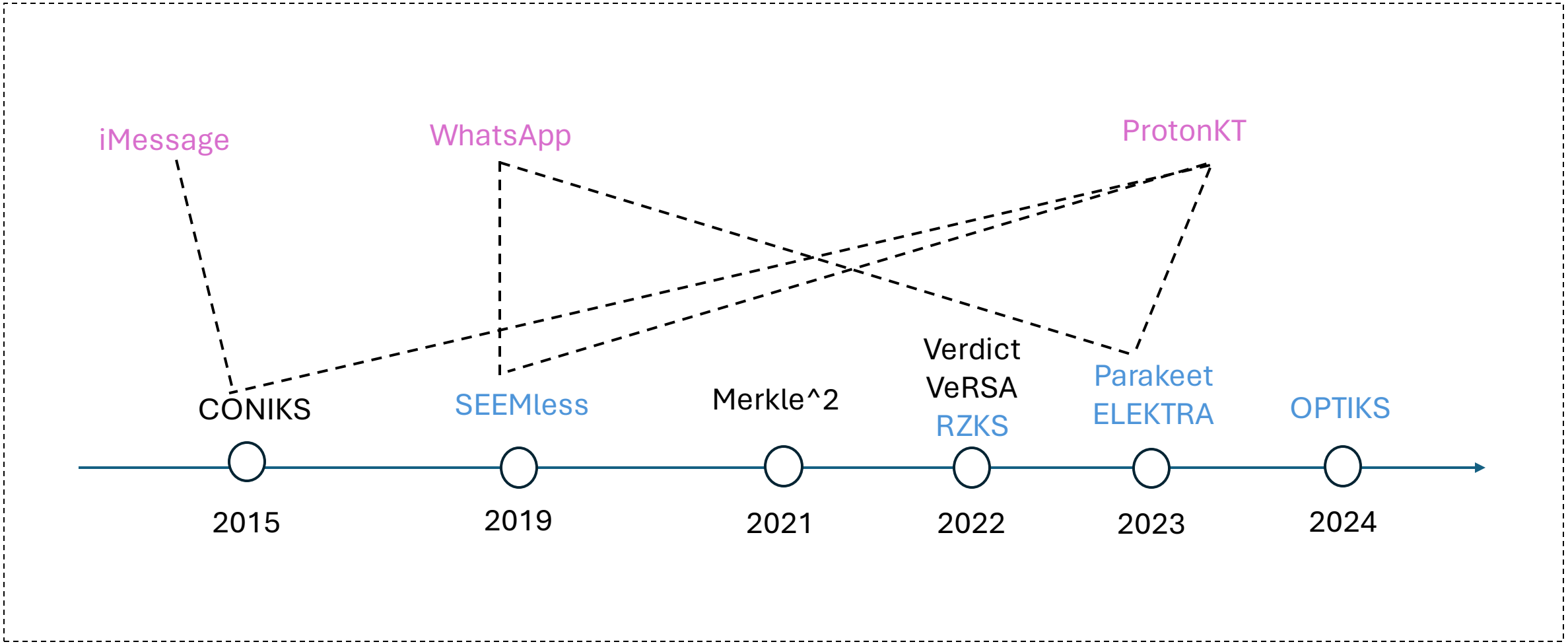
Academic Literature



Industry Deployments and Standardizations



Industry Deployments

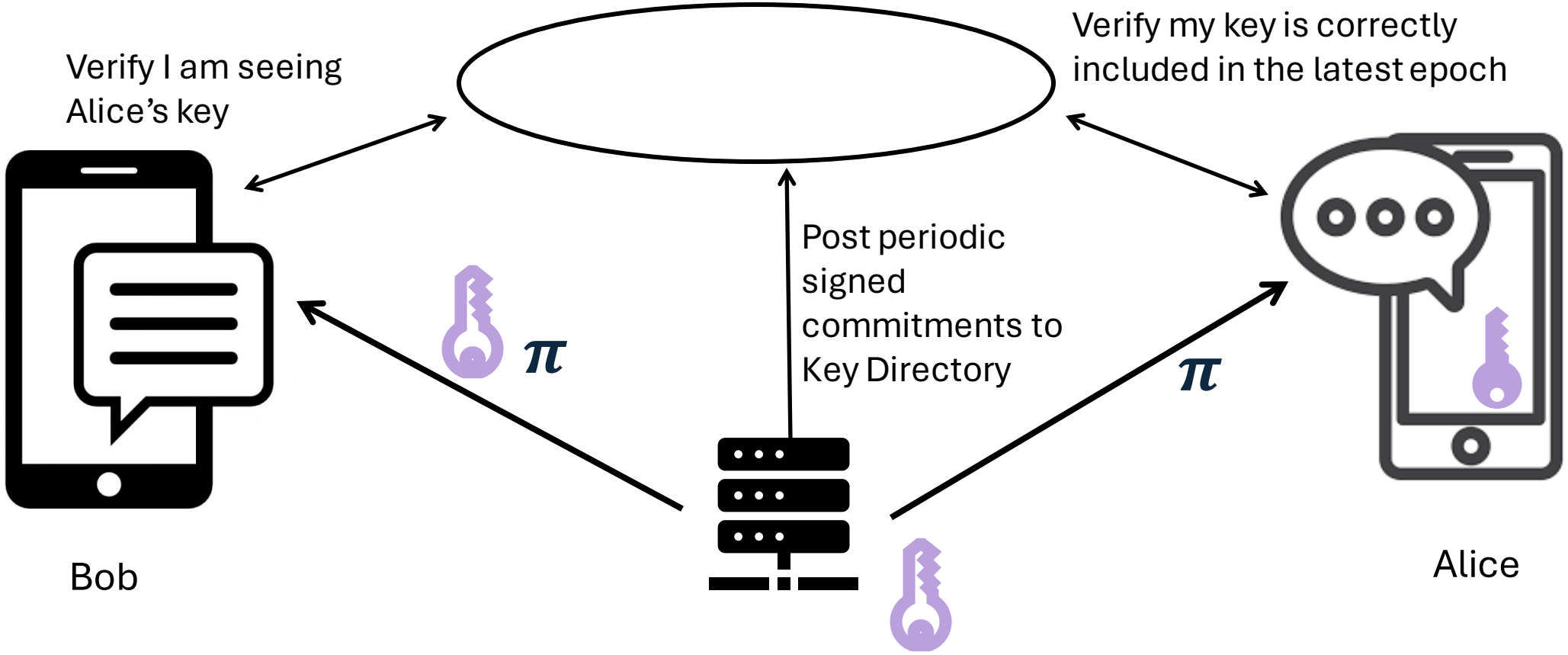


Overview

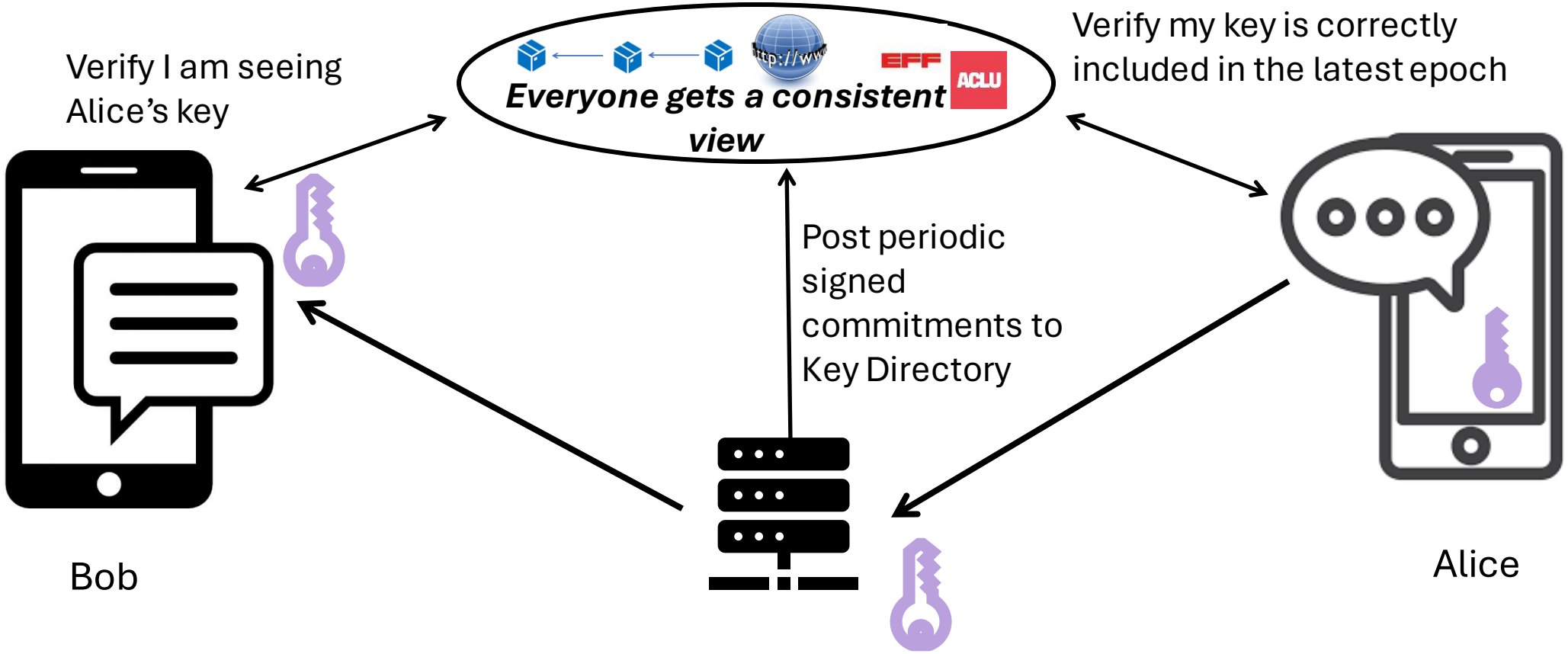
Directory Commitment

- A primitive to generate a short commitment to a collection of (username, public key) pairs where the labels form a set S
- The Service Provider (SP) generates the short commitment c to the collection
- SP can prove statements about membership of any element $x \in S$ $(x \notin S)$ later non-interactively with respect to c without revealing anything else about S

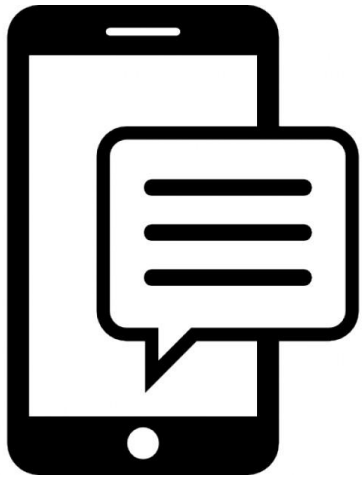
Key Transparency



Key Transparency



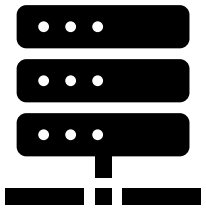
Registration/Update



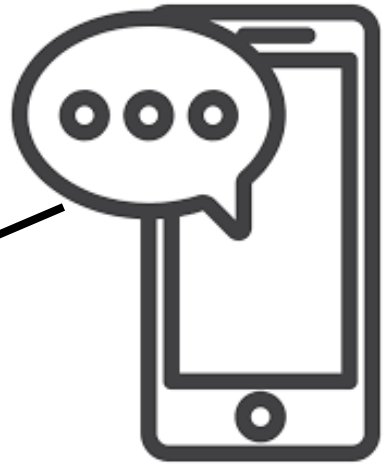
Bob

Register/Update device public key with username.

Username	Public Key
Alice	pk_A
Bob	pk_B

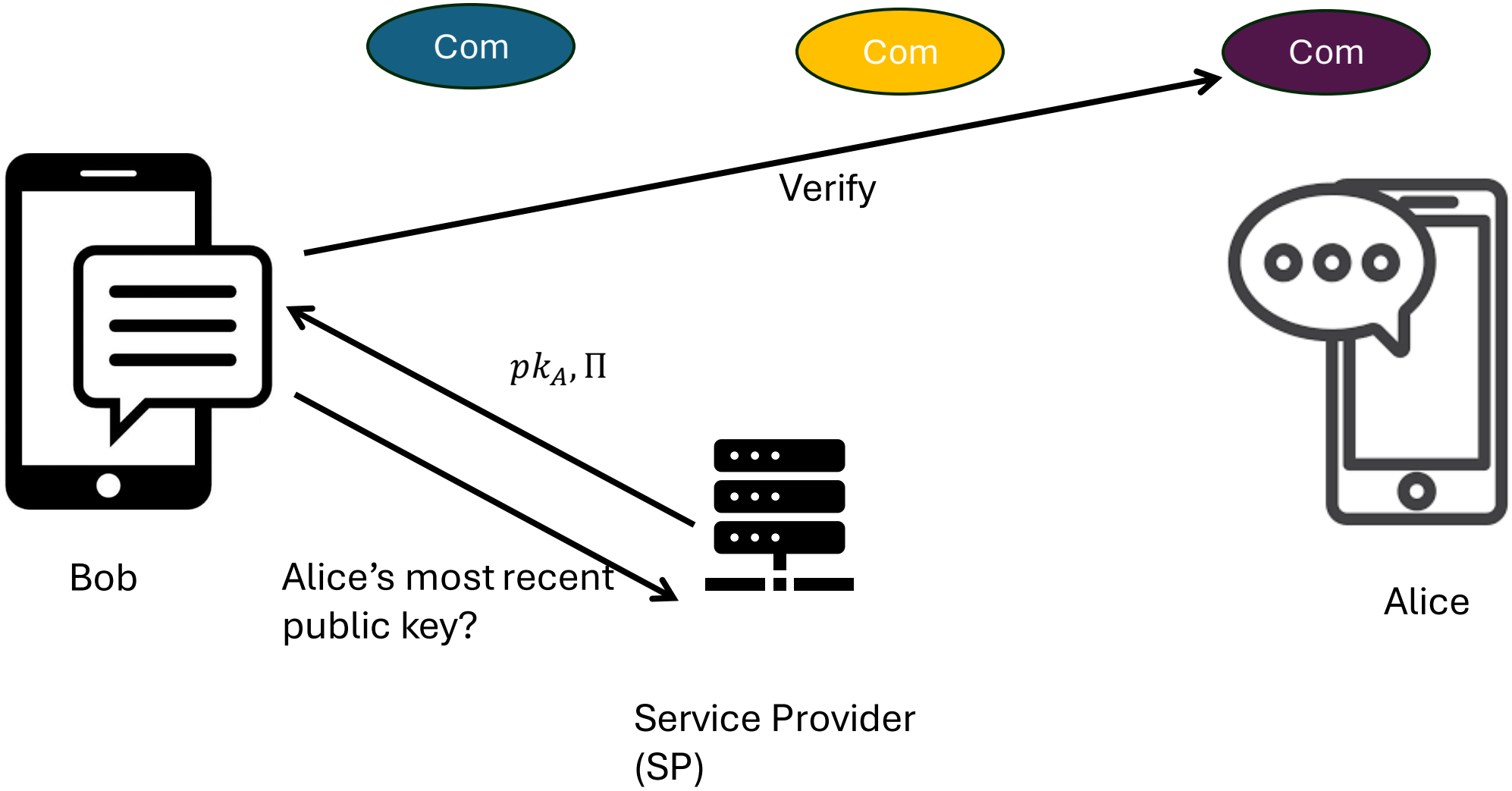


Service Provider (SP)

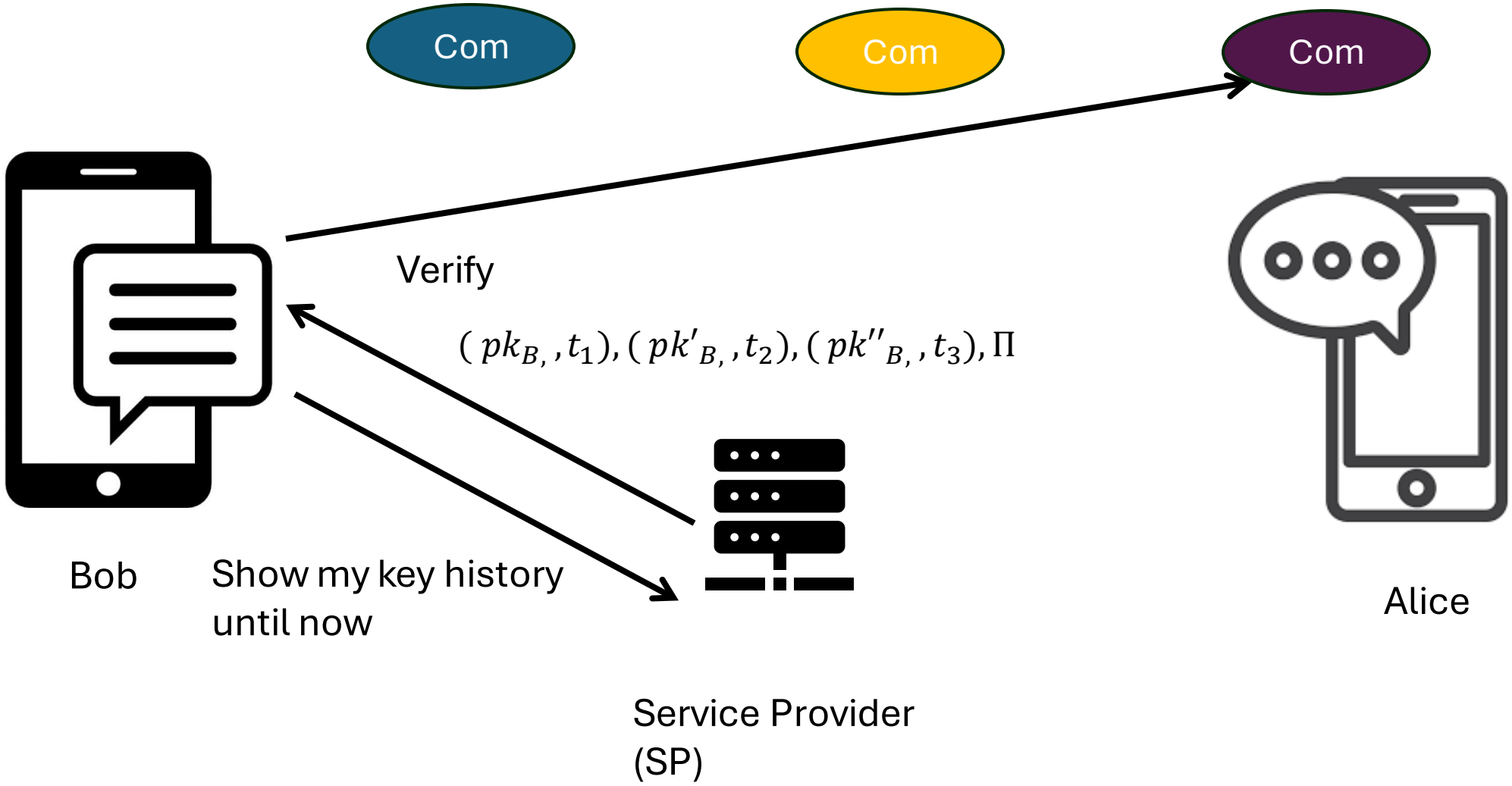


Alice

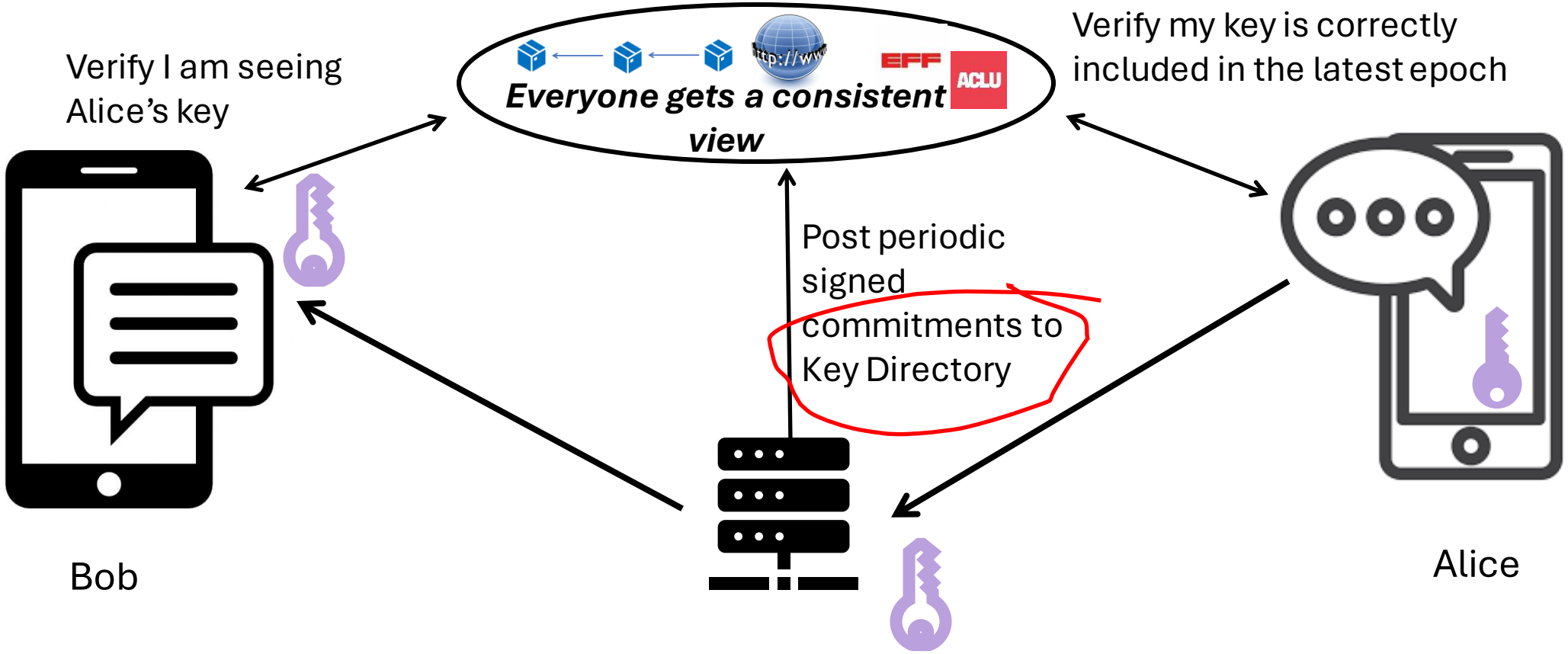
Query for other user's public key



Query for own key history



Key Transparency




Core Data Structure

Key Transparency

- Recall: commitment to a directory.
  - Map usernames to public keys
    - Must be a unique entry per username

- Sparse Merkle Tree construction
  - First used in Keybase
  - Approach underlies many current proposals: CONIKS, SEEMless, Merke^2, Parakeet, ELEKTRA, OPTIKS + All the industry deployments

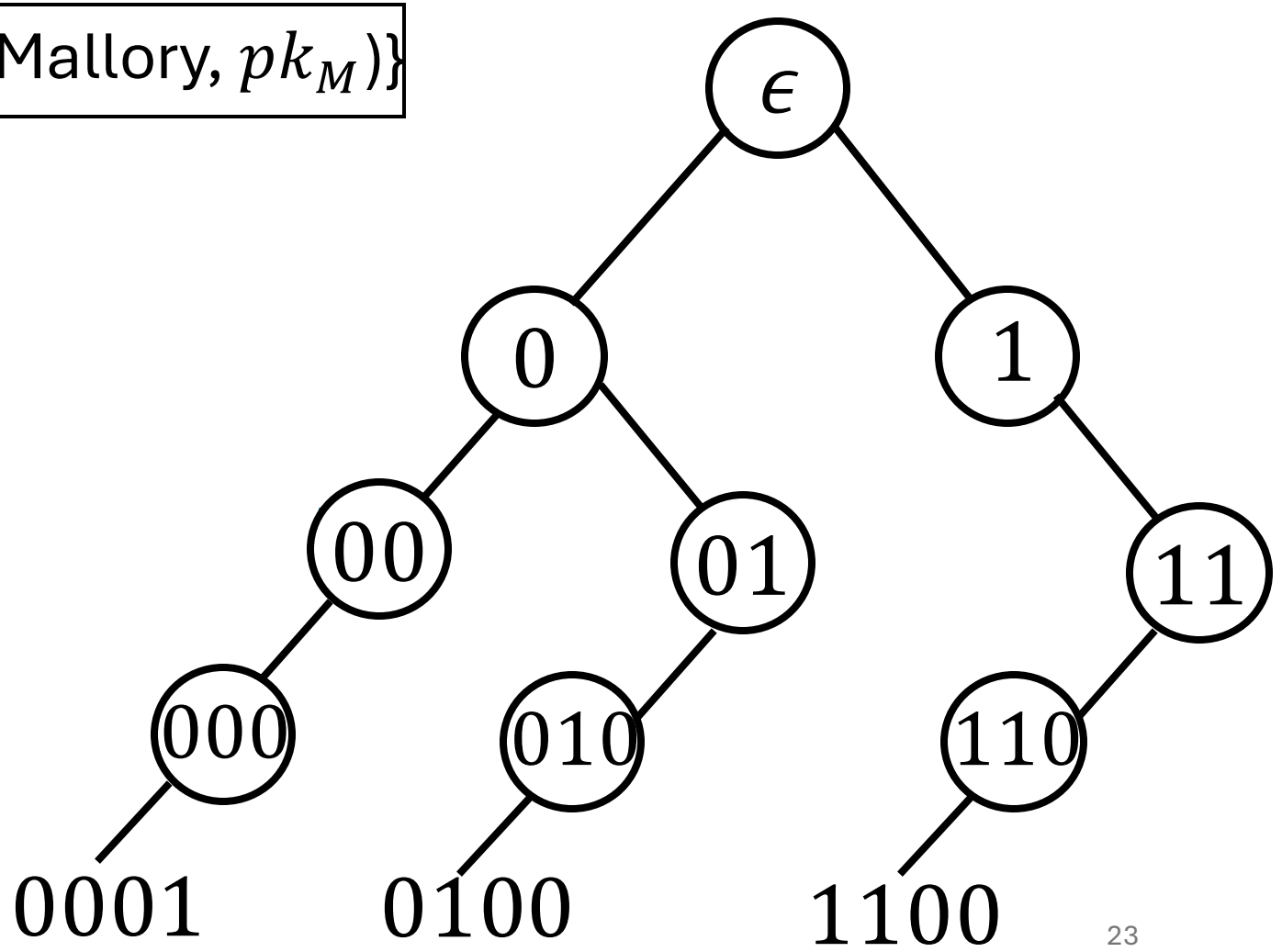
Sparse Merkle Tree construction

S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100



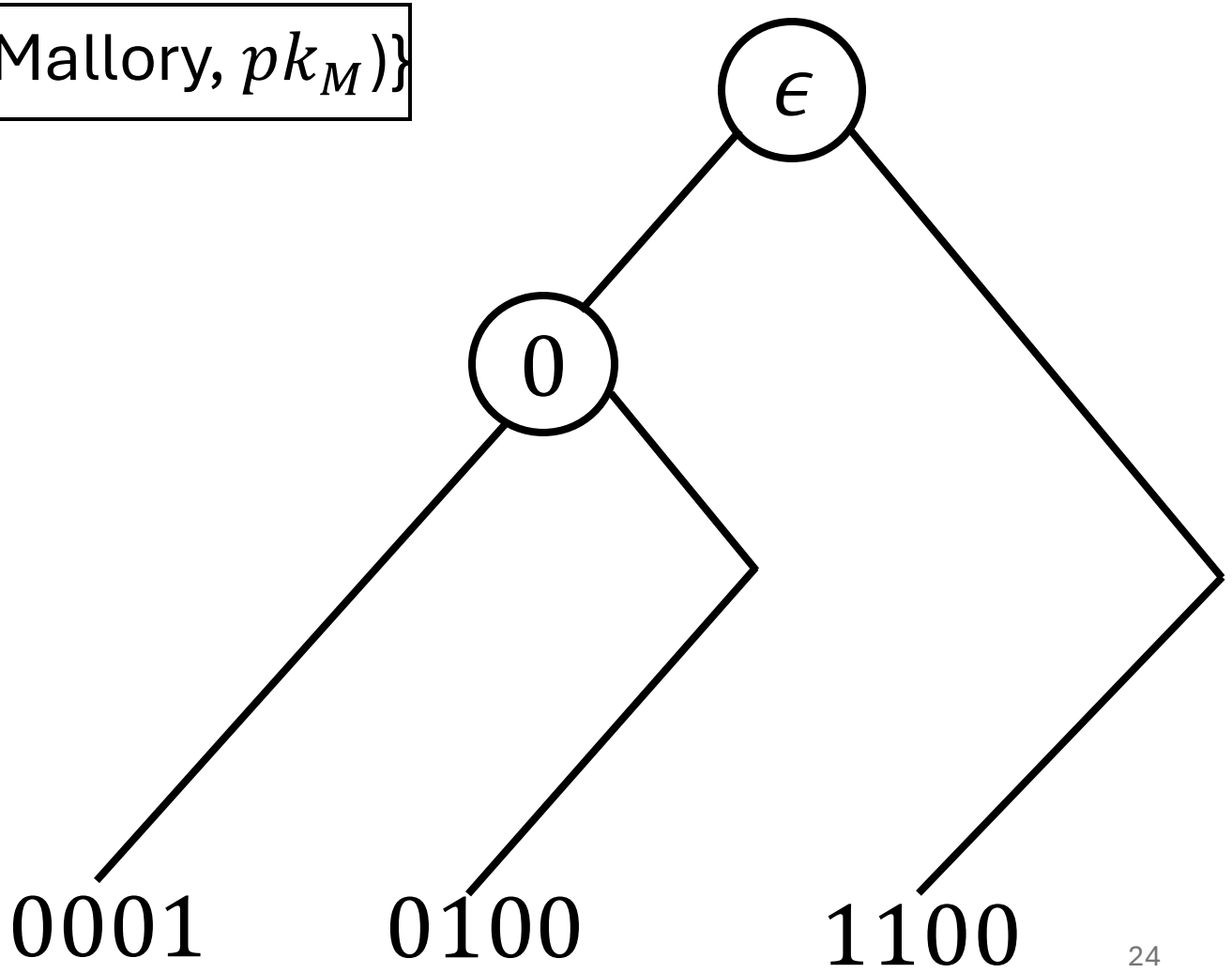
Sparse Merkle Tree construction

S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100



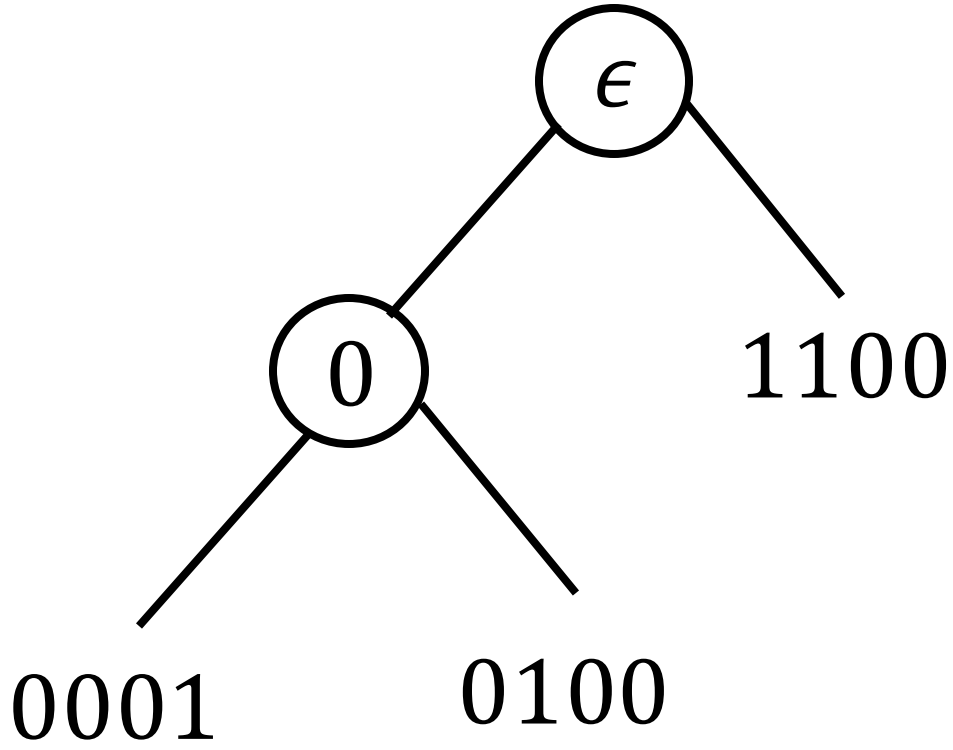
Sparse Merkle Tree construction

S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100



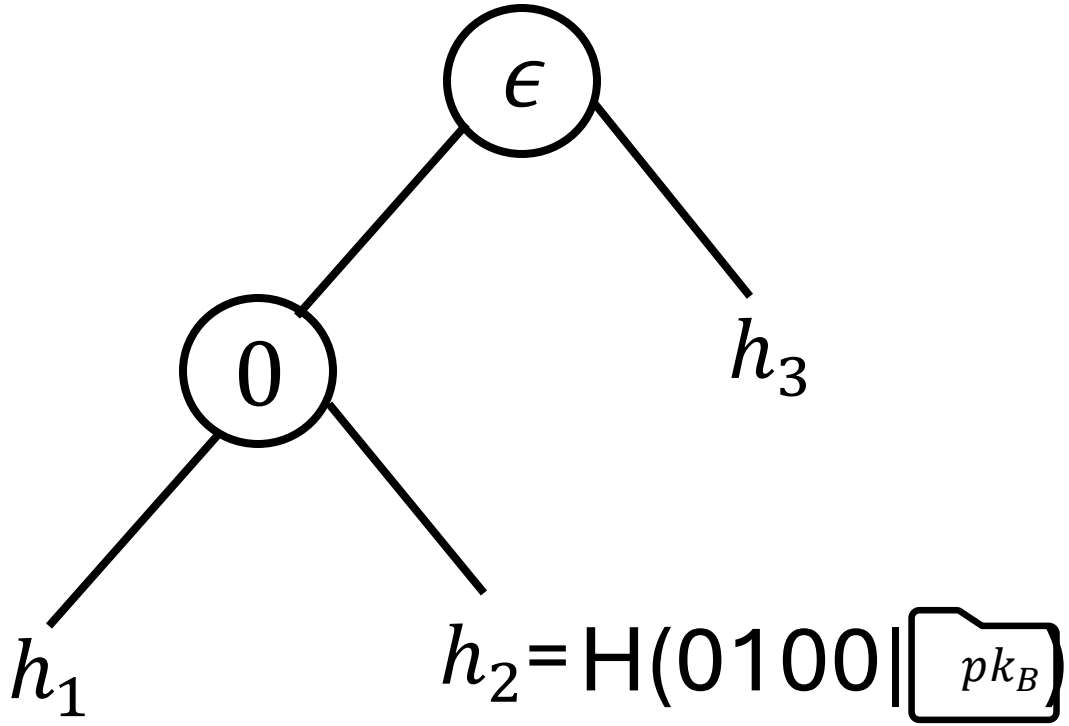
Sparse Merkle Tree construction

S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100



Sparse Merkle Tree construction

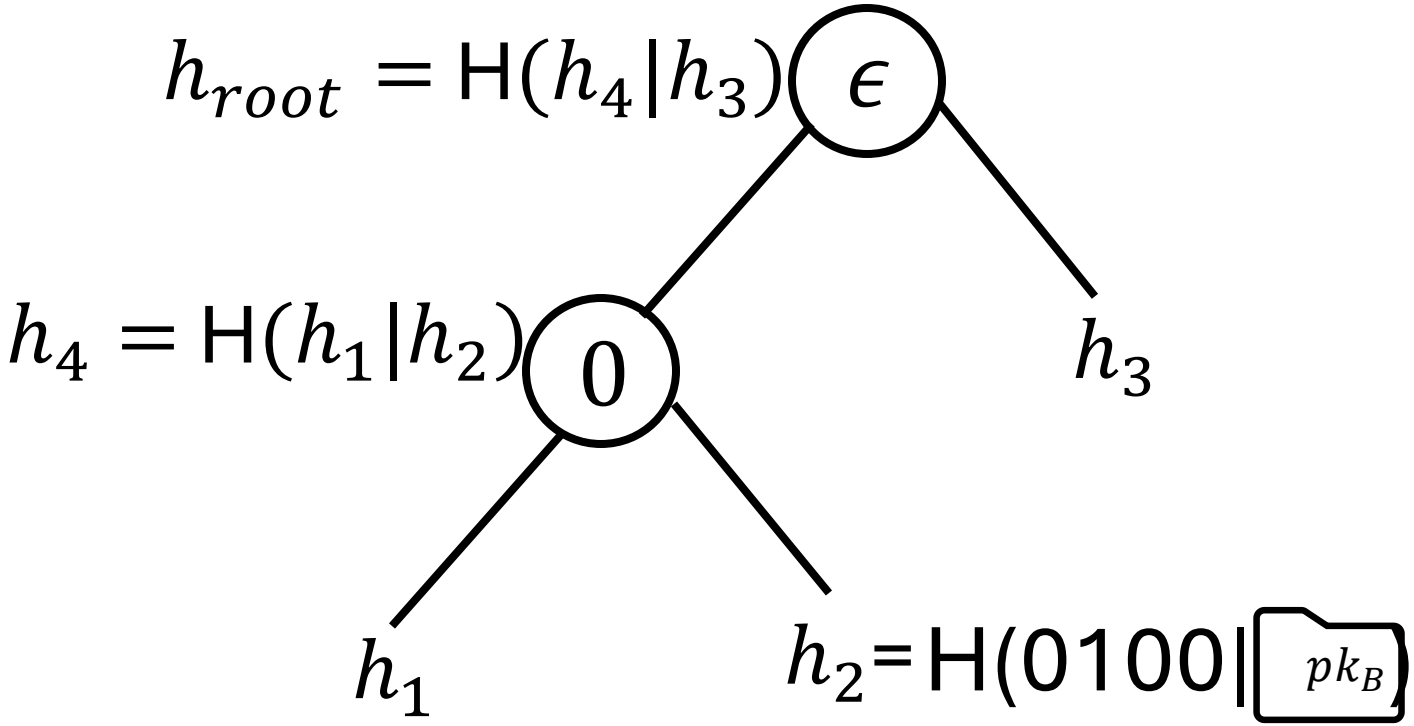
S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

com = h_{root}



Sparse Merkle Tree construction

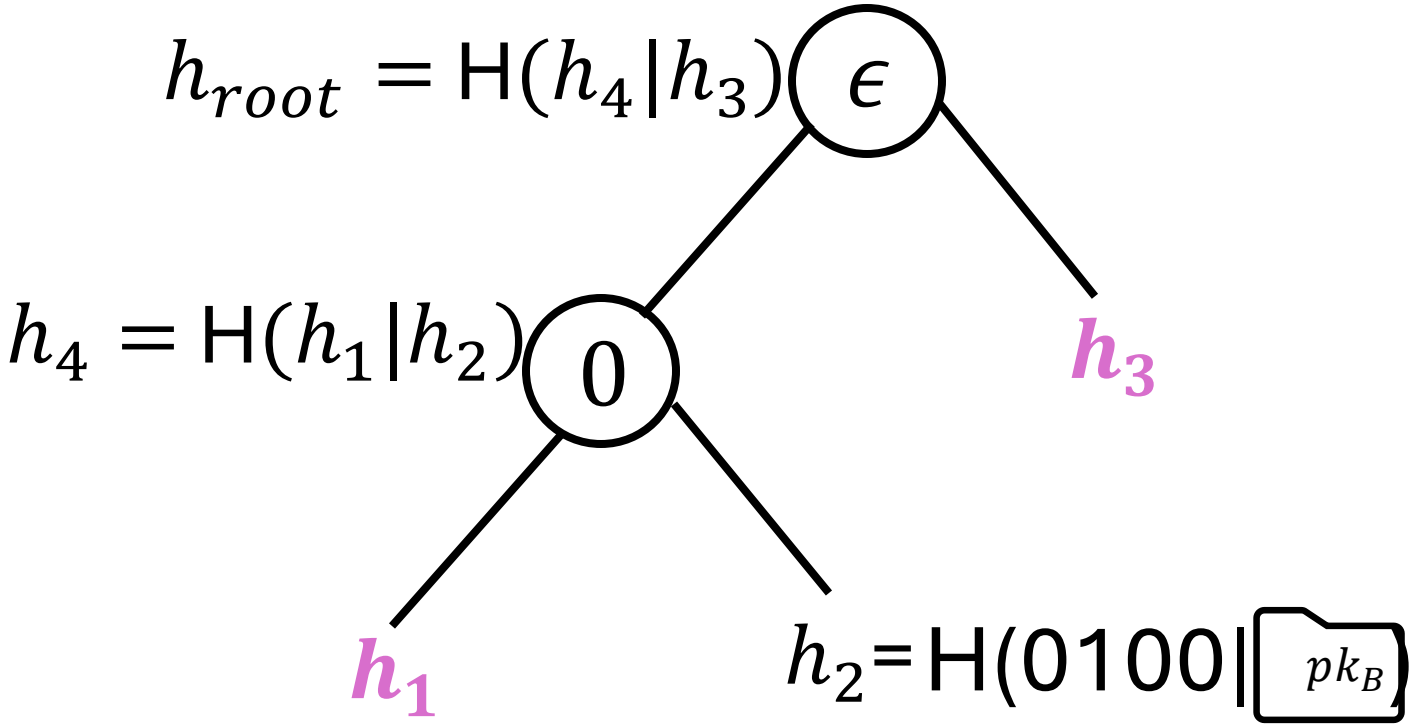
S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

com = h_{root}



Sparse Merkle Tree construction

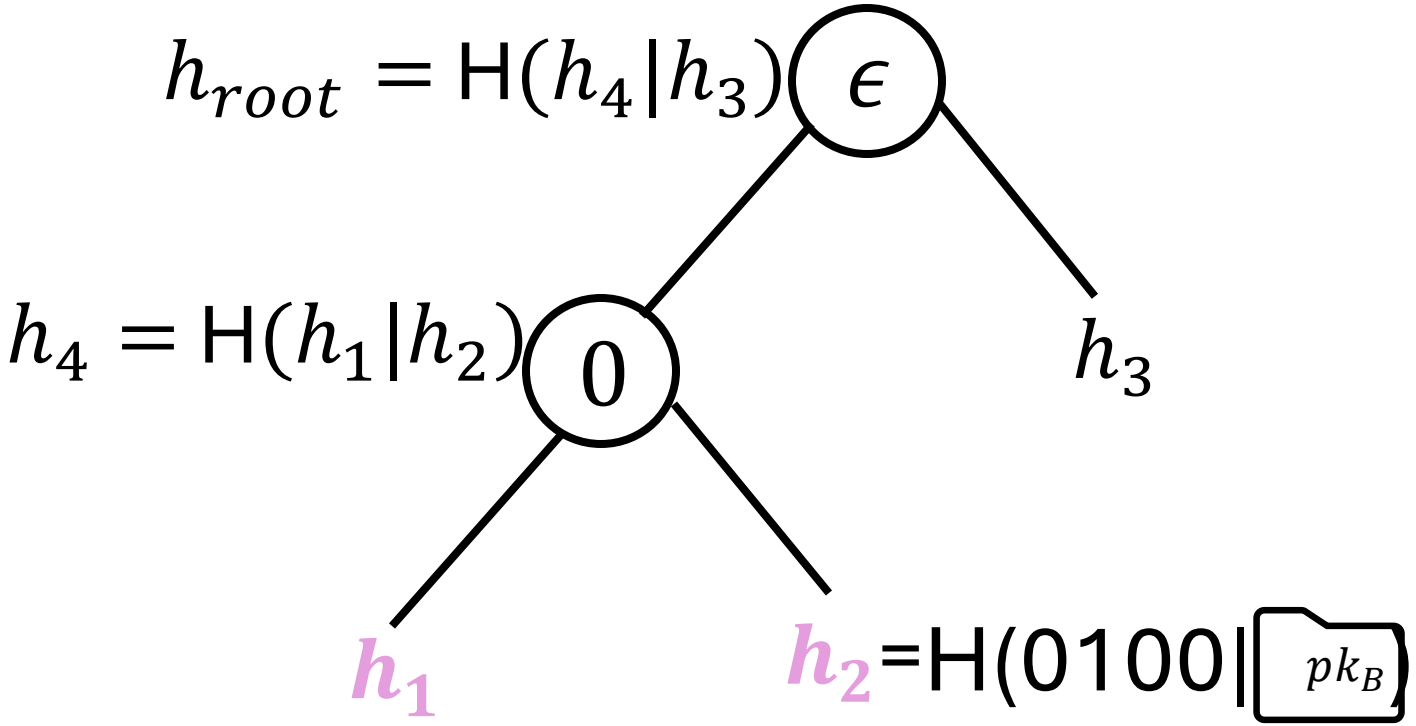
S = { (Alice, pk_A), (Bob, pk_B),(Mallory, pk_M)}

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

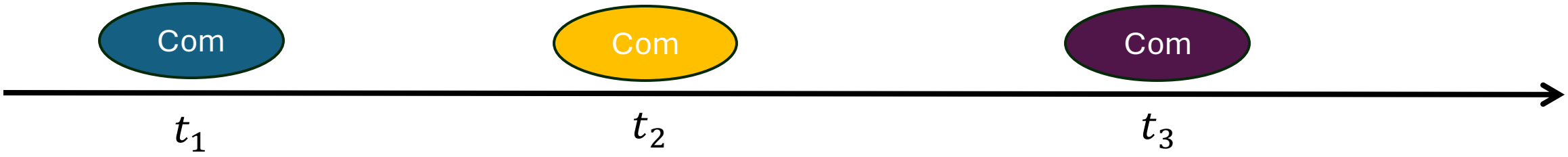
H(Carol) = 0011



com = h_{root}

Building up to a Key Transparency system

Periodic publish with Sparse Merke Tree



Username	Public Key
Alice	(pk_A, t_1)
Bob	(pk_B, t_1)

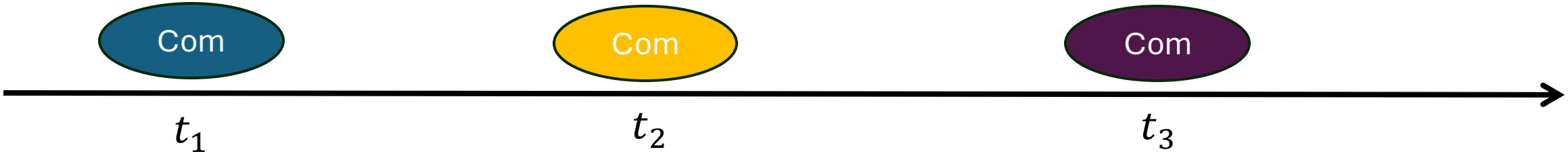
Username	Public Key
Alice	(pk'_A, t_2)
Bob	(pk_B, t_1)

Username	Public Key
Alice	(pk'_A, t_2)
Bob	(pk'_B, t_3)

Limitation: Client History Cost

- Alice's client has to check her key every epoch
- If Alice goes offline, she must check all missed commitments when she comes back online
- Limits how short epochs can be
- Note key changes aren't incorporated until the next epoch, so we want epochs to be *short*

Solution: Append-only Tree



Username	Public Key
Alice	(pk_A, t_1)
Bob	(pk_B, t_1)

Username	Public Key
Alice	$(pk_A, t_1), (pk'_A, t_2)$
Bob	(pk_B, t_1)

Username	Public Key
Alice	$(pk_A, t_1),(pk'_A, t_2)$
Bob	$(pk_B, t_1), (pk'_B, t_3)$

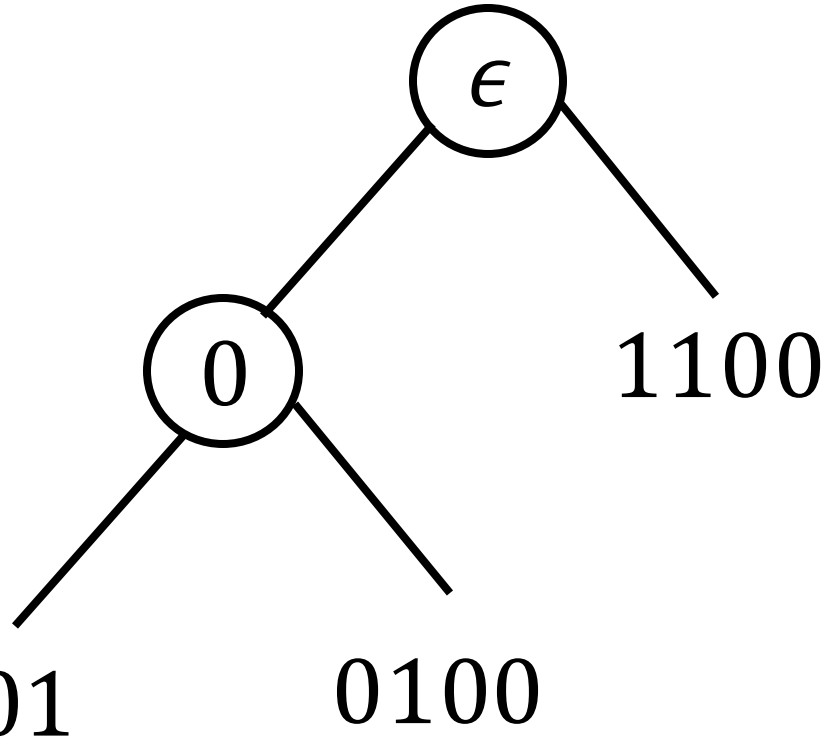
Solution: Append-only Tree

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

Requires: an honest auditor to check no commitment is ever deleted from the leaves



$$h_1 = H(0001|com_1, com_2)$$
$$com_1 = com(pk_A), com_2 = com(pk'_A)$$

Limitation: Privacy

- Brute force search reveals usernames

What if usernames are phone numbers or email addresses?

- Reveals information about when keys are updated
  - Even if they're never queried

Info about when user:
- Is Compromised
- Gets new device
- Travels out of country

Sparse Merkle Tree construction – Privacy?

S = { (Alice, pk_A), (Bob, pk_B),(Mallor

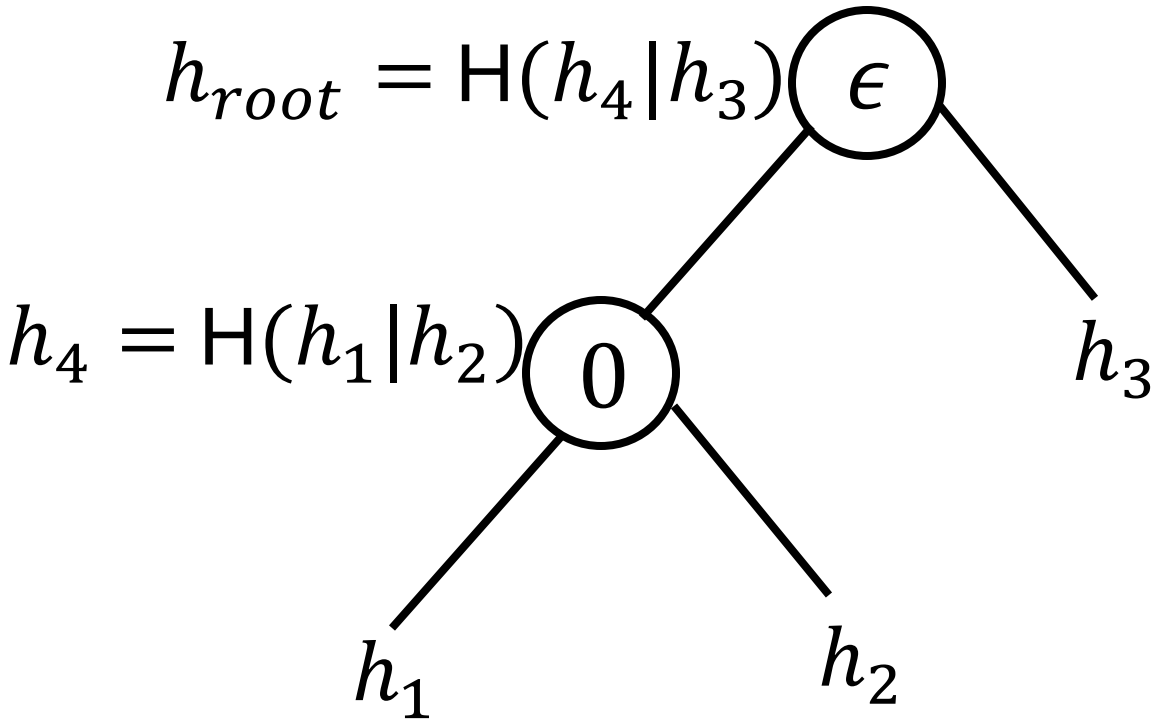
Querying Bob reveals Bob has a sibling, so it reveals info about H(Alice)!

H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

com = h_{root}



Sparse Merkle Tree construction – Privacy?

S = { (Alice, pk_A), (Bob, pk_B),(Mallor

Querying Bob reveals Bob has a sibling, so it reveals info about H(Alice)!

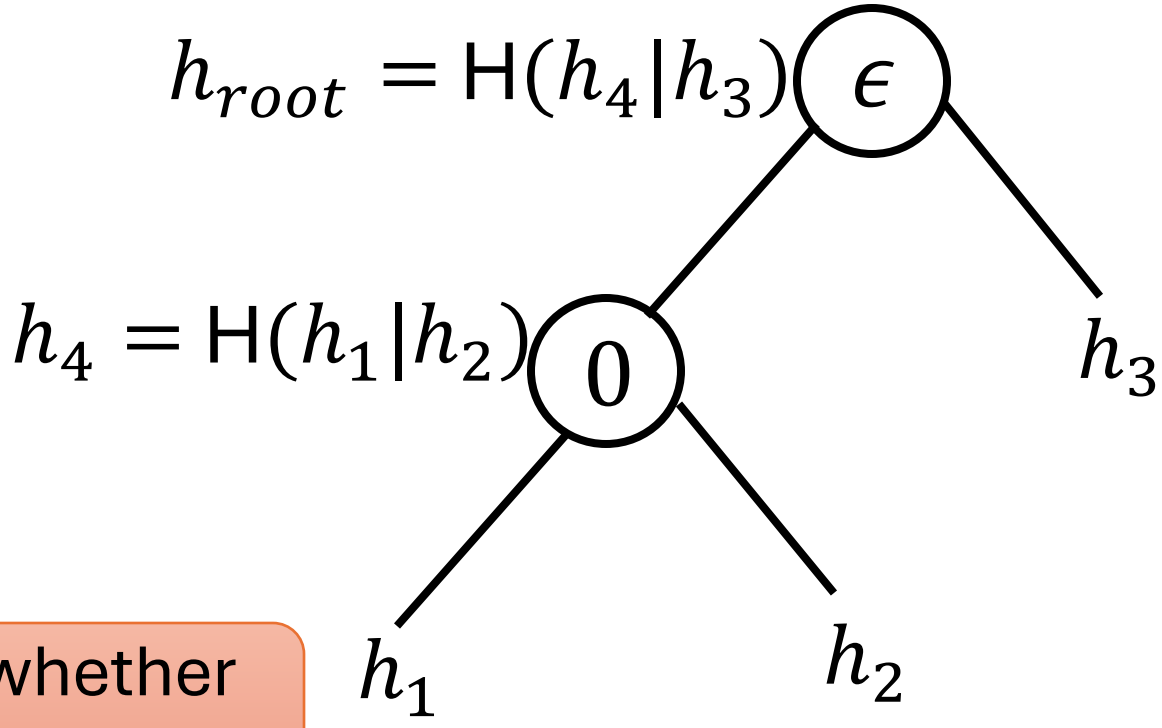
H(Alice) = 0001

H(Bob) = 0100

H(Mallory) = 1100

H(Carol) = 0011

com = h_{root}



Querying Bob reveals whether Alice has changed her key!

Sparse Merkle Tree construction – Privacy?

S = { (Alice, pk_A), (Bob, pk_B),(Mallor

Querying Bob reveals Bob has a sibling, so it reveals info about H(Alice)!

H(Alice) = 0001

H(Bob)

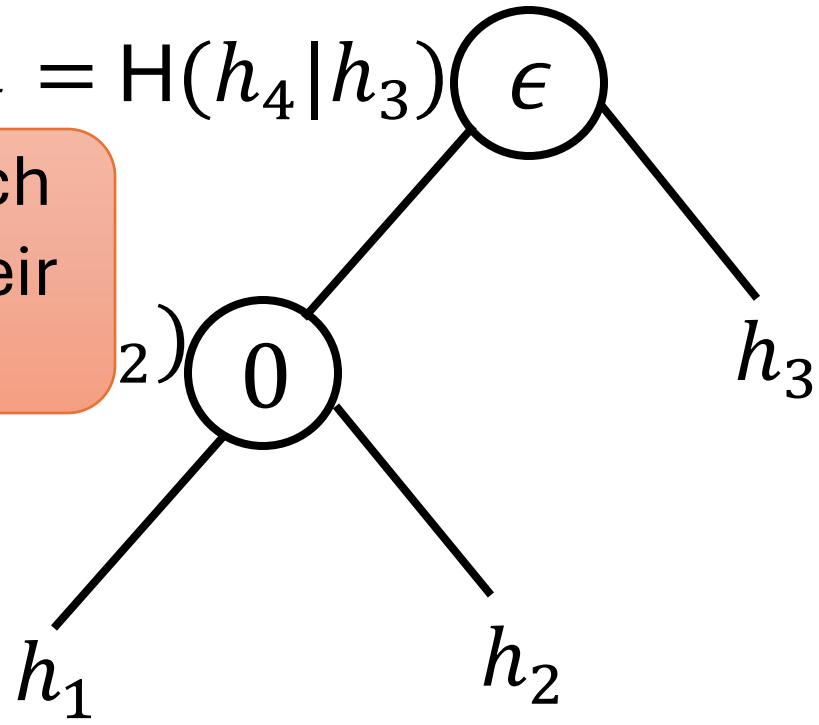
H(Mall

Auditors will learn exactly which user positions are updating their keys without querying!

H(Carol) = 0011

com = h_{root}

Querying Bob reveals whether Alice has changed her key!



Limitation: Privacy

- Brute force search reveals usernames
- Auditor learns when each leaf is updated
- Reveals information about when keys are updated
  - Even if they're never queried

What if usernames are phone numbers or email addresses?

Info about when user:
- Is Compromised
- Gets new device
- Travels out of country

Privacy

- Content Privacy
- Metadata Privacy
- Post Compromise Privacy

Content Privacy

Verifiable Random Function (VRF)

- KeyGen $\rightarrow k, pk$
- Evaluate: $\mathrm{F_k}(x) \rightarrow y$
- Prove$(k, x) \rightarrow y, \pi$
- Verify $(pk, x, y, \pi) \rightarrow accept/reject$

Content Privacy: Hiding Usernames

CONIKS [MBBFF15] : To hide usernames, add a VRF[MRV99]

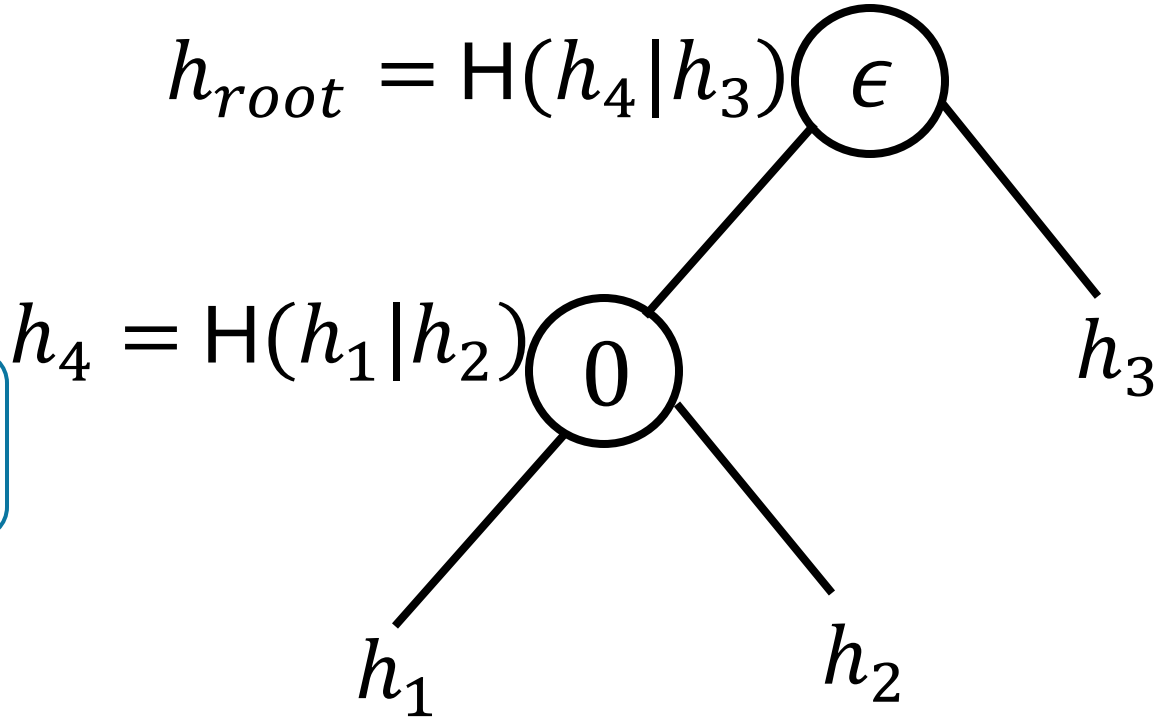
F_k(Alice) = 0001

F_k(Bob) = 0100

F_k(M[illegible]lory) = 1100

Service provider proves this is computed correctly

com = h_{root}, **VRF pk**



Content Privacy: Hiding Usernames

CONIKS [MBBFF15] : To hide usernames, add a VRF[MRV99]

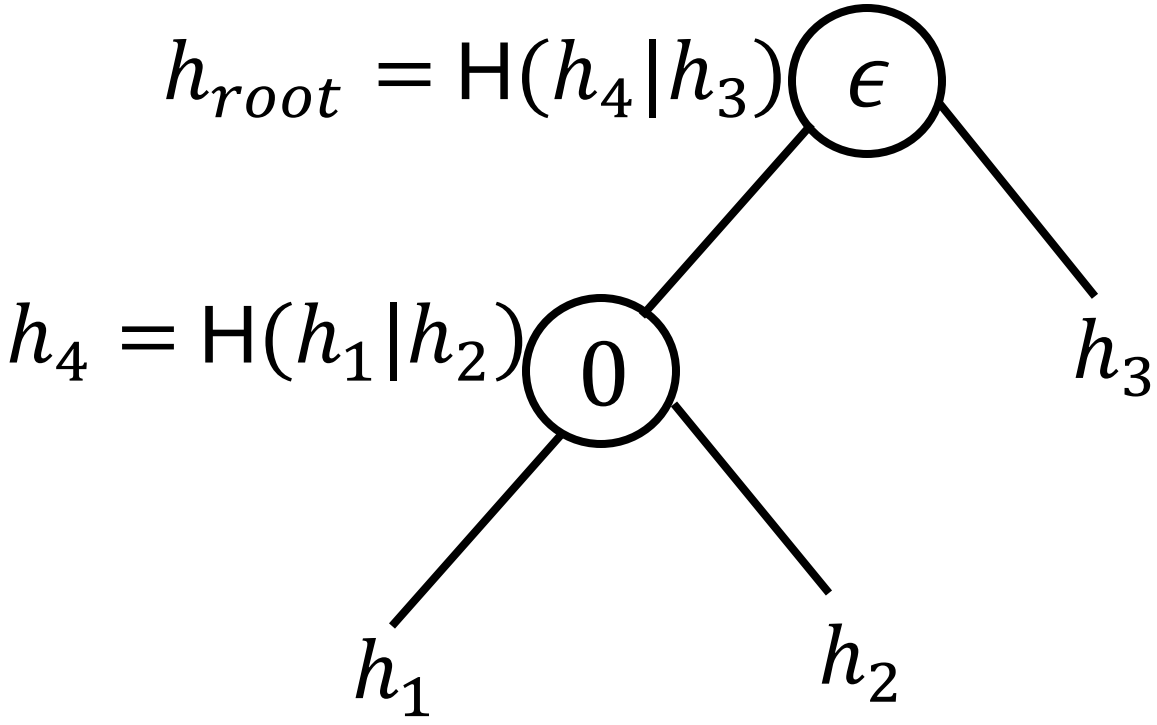
F_k(Alice) = 0001

F_k(Bob) = 0100

F_k(Mallory) = 1100

F_k(Carol) = ?

com = h_{root}, **VRF pk**



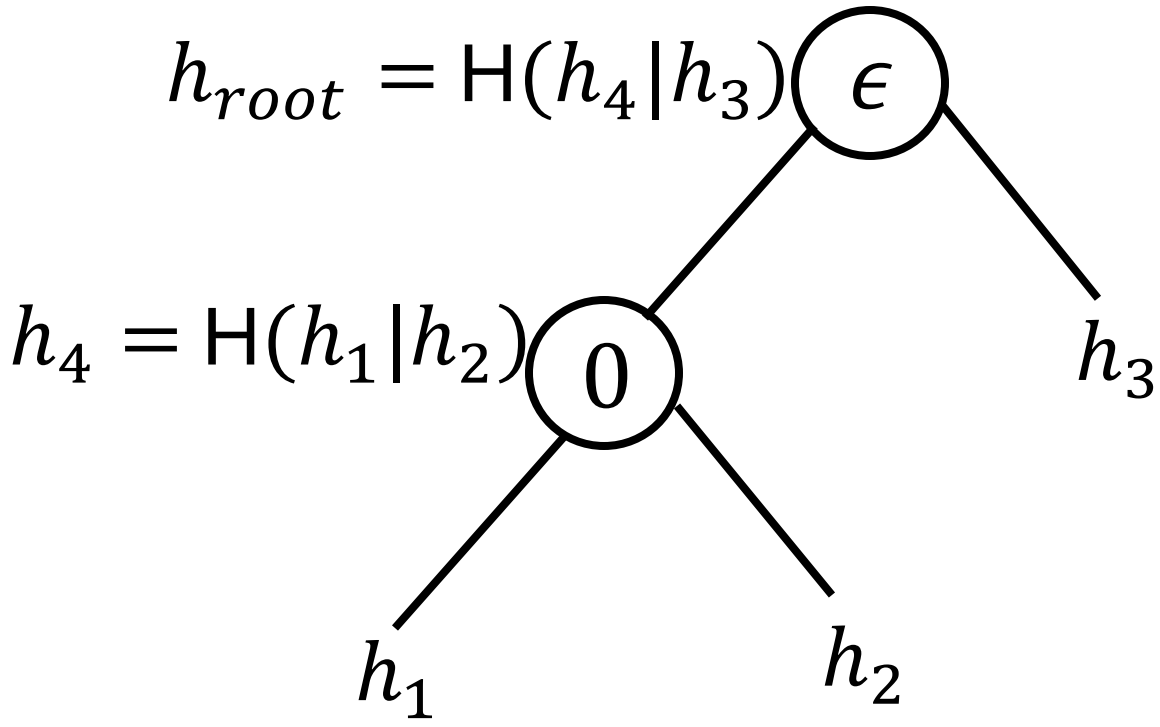
Metadata Privacy

Metadata Privacy: Hiding when user updates their keys

F_k(Alice) = 0001

F_k(Bob) = 0100

F_k(Mallory) = 1100

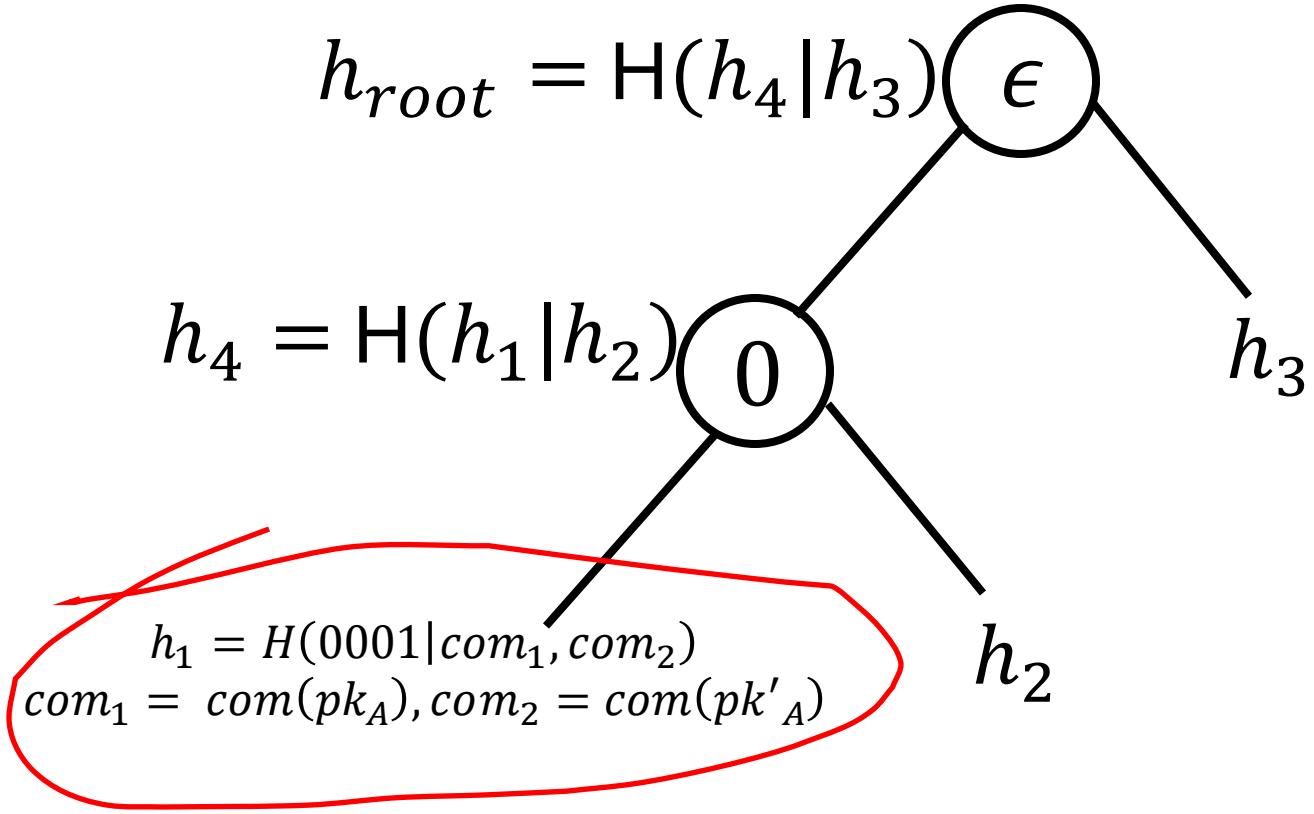


Metadata Privacy: Hiding when user updates their keys

F_k(Alice) = 0001

F_k(Bob) = 0100

F_k(Mallory) = 1100



Metadata Privacy: Hiding when user updates their keys

SEEMless [CDGM19]: label = username||version

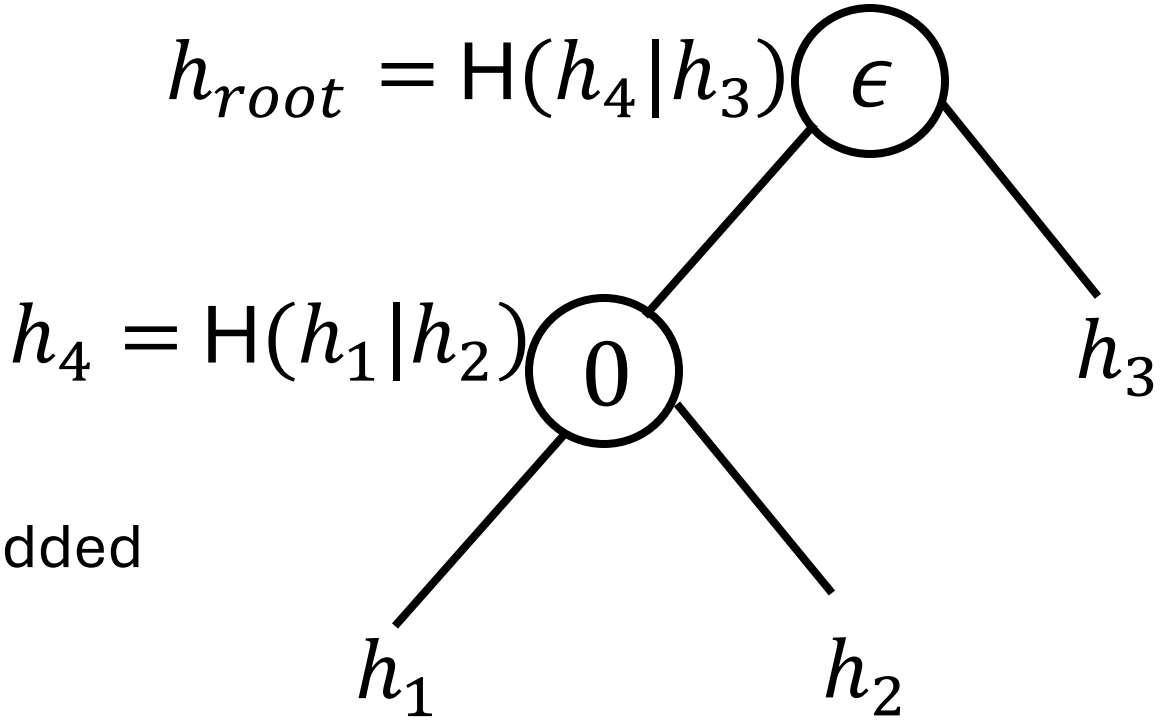
F_k(Alice|**v.1**) = 0001

F_k(Bob|**v.3**) = 0100

F_k(Mallory|**v.6**) = 1100

Tree includes all versions

- Leaf includes epoch it was added



Metadata Privacy: Hiding when user updates their keys

SEEMless [CDGM19]: label = username||version

F_k(Alice|**v.1**) = 0001

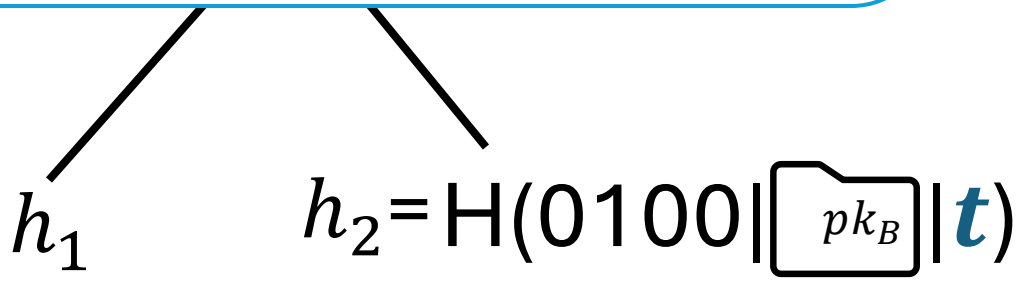
F_k(Bob|**v.3**) = 0100

F_k(Mallory|**v.6**) = 1100

Privacy:
- Updates add new pseudorandom nodes
- No changing nodes
- No link between previous and new nodes

Tree includes all versions
- Leaf includes epoch it was added

Auditor checks:
- Leaves of old tree are subset of new one
- New leaves have correct epoch t



Post Compromise Privacy

Post Compromise Privacy: Server's key leakage

Rotatable Zero Knowledge Sets [CDGGKMM22]

F_{k_1}(Alice|**v.1**) = 0001
F_{k_1}(Bob|**v.3**) = 0100
F_{k_1}(Mallory|**v.6**) = 1100

F_{k_2}(Alice|**v.1**) = 1001
F_{k_2}(Bob|**v.3**) = 0011
F_{k_2}(Mallory|**v.6**) = 1010

VRF pk_1

VRF pk_2

Verify Rotation: $(pk_1 pk_2, y_1, \ldots, y_n, y'_1, \ldots, y'_n, \pi_{rotation}) \rightarrow accept/reject$

A Simple Key Transparency System

OPTIKS-core [LCGKM24]

Commitment as described

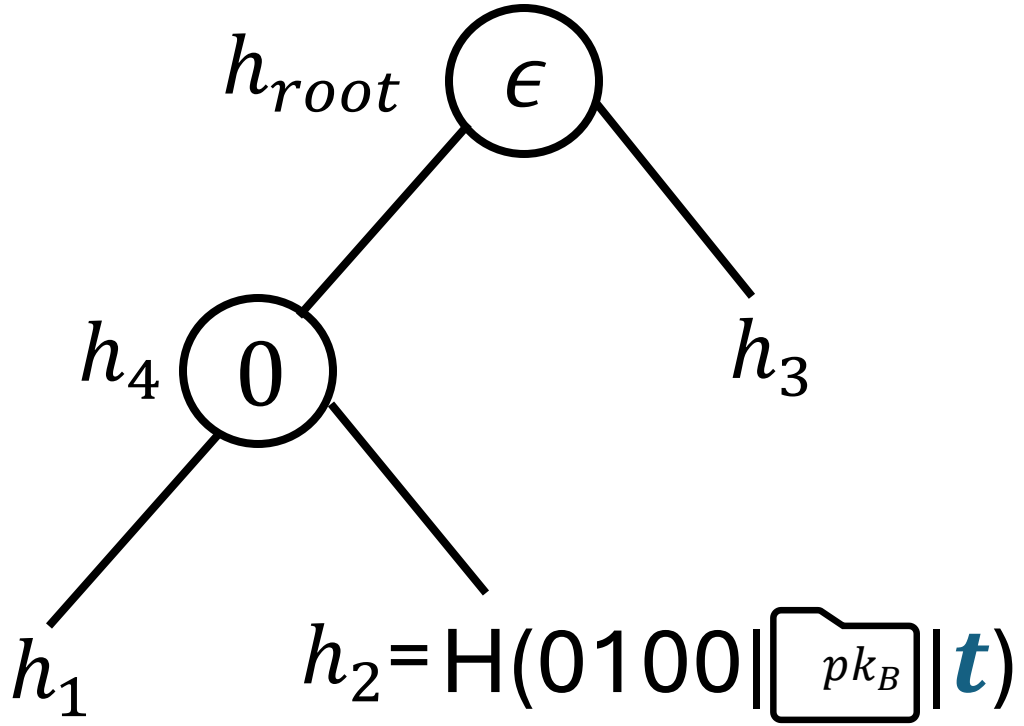
Lookup(Bob):

F_k(Bob|**v.1**) , π_{VRF}, π_{member}

F_k(Bob|**v.2**) ,π_{VRF}, π_{member}

F_k(Bob|**v.3**), π_{VRF}, π_{member}, $pk_B, open_3, t_3$

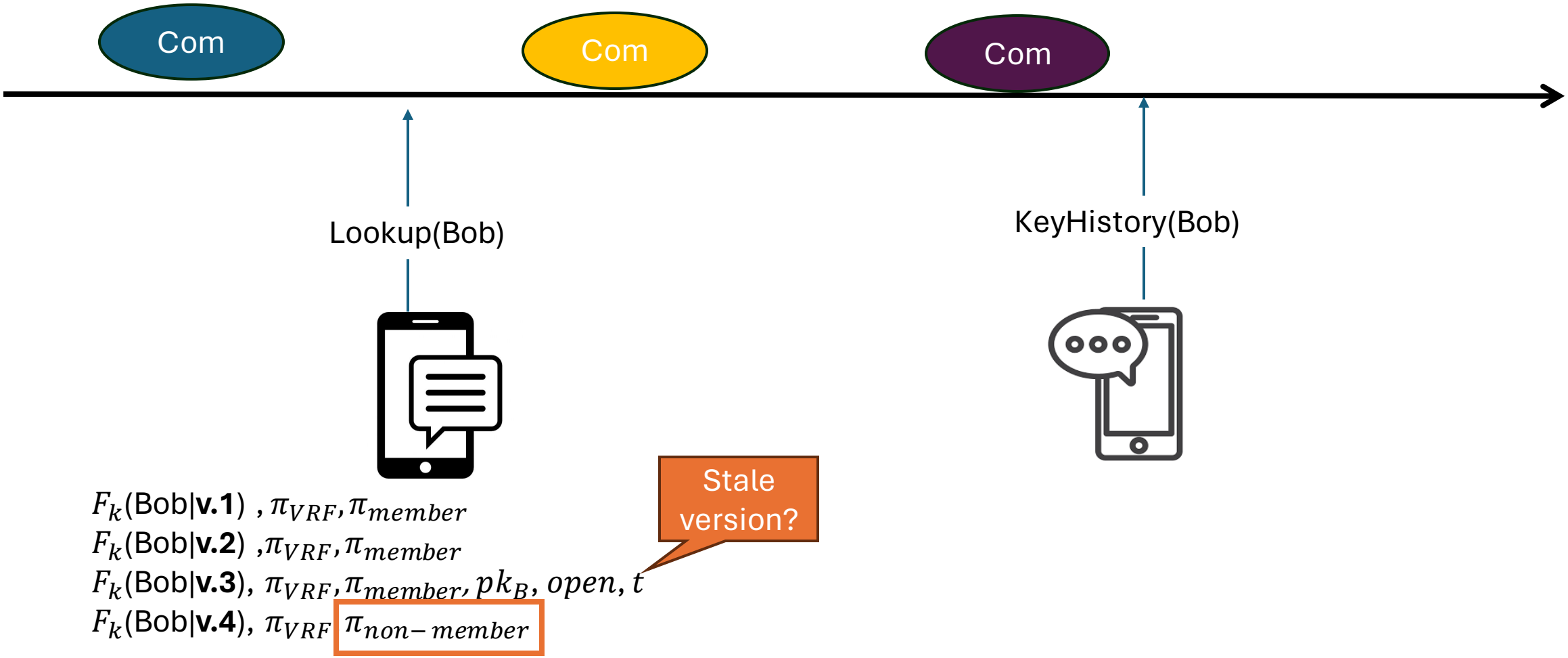
F_k(Bob|**v.4**), $\pi_{VRF}, \pi_{non-member}$



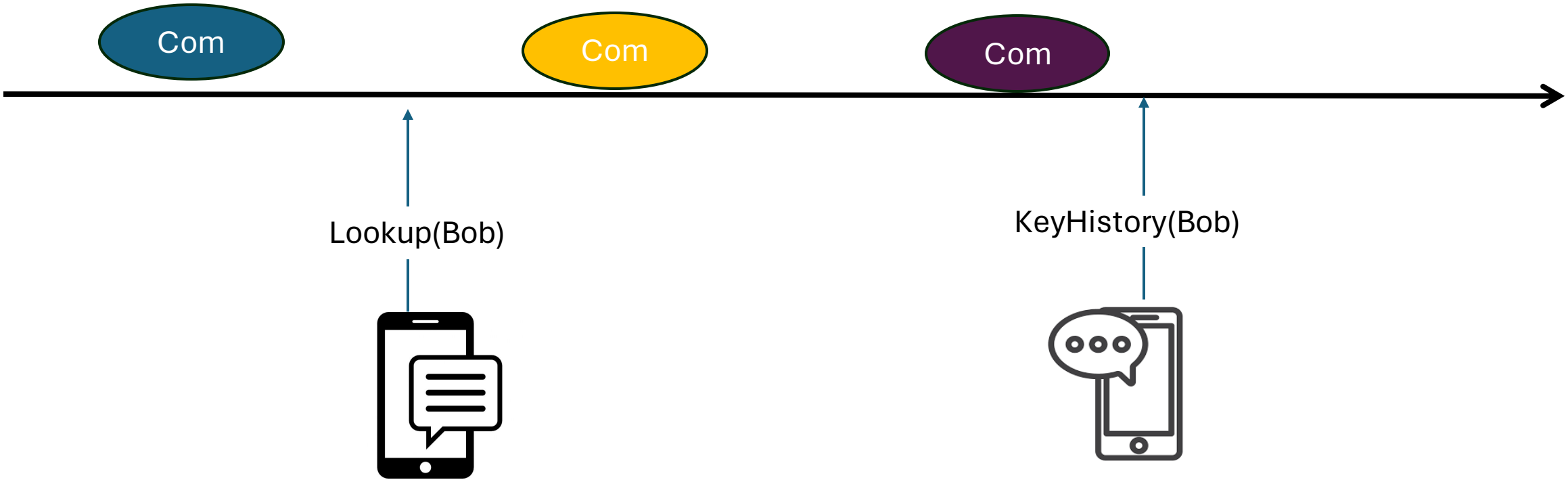
History(Bob):

- similar but with $(pk_B, open, t)$ for each version

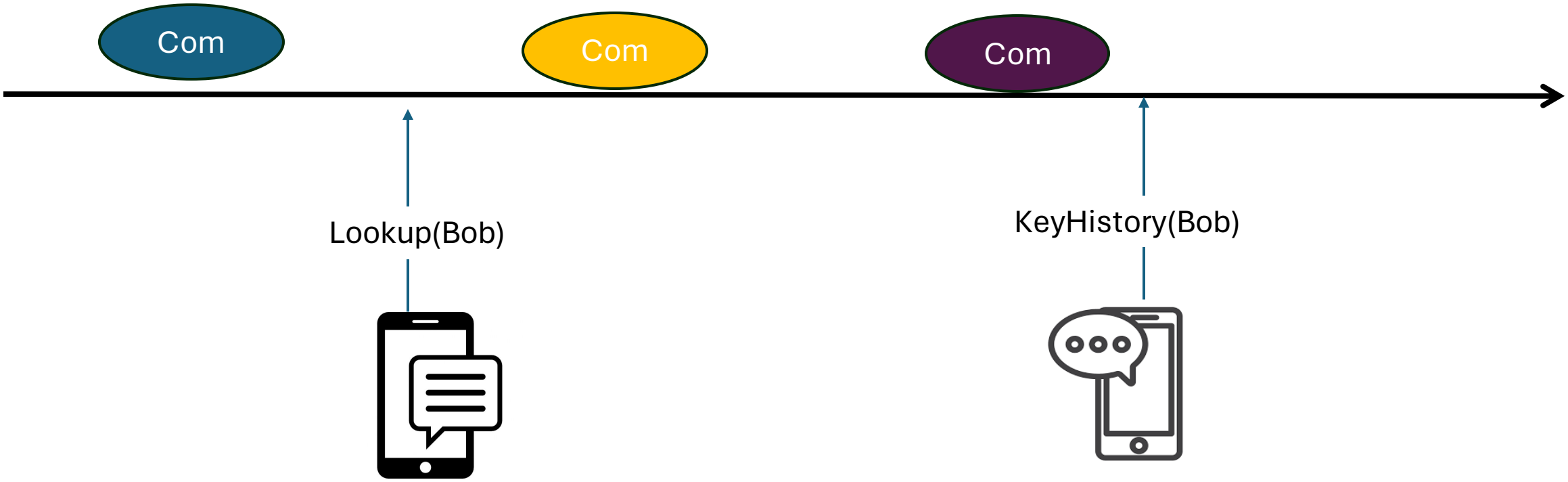
Soundness


Com
Com
Com
Lookup(Bob)
KeyHistory(Bob)
F_k(Bob|**v.1**) , π_{VRF}, π_{member}
F_k(Bob|**v.2**) , π_{VRF}, π_{member}
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{member}, pk_B, open, t$
F_k(Bob|**v.4**), π_{VRF} $\pi_{non-member}$
Stale version?

Soundness


Com
Com
Com
Lookup(Bob)
KeyHistory(Bob)
F_k(Bob|**v.1**) , π_{VRF}, π_{member}
F_k(Bob|**v.2**) , π_{VRF}, π_{member}
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{member}, pk_B, open_3, t_3$
F_k(Bob|**v.4**), $\pi_{VRF}, \pi_{non-member}$
F_k(Bob|**v.1**) , $\pi_{VRF}, \pi_{member}, pk_{B1}, open_1, t_1$
F_k(Bob|**v.2**) , $\pi_{VRF}, \pi_{non-member}$

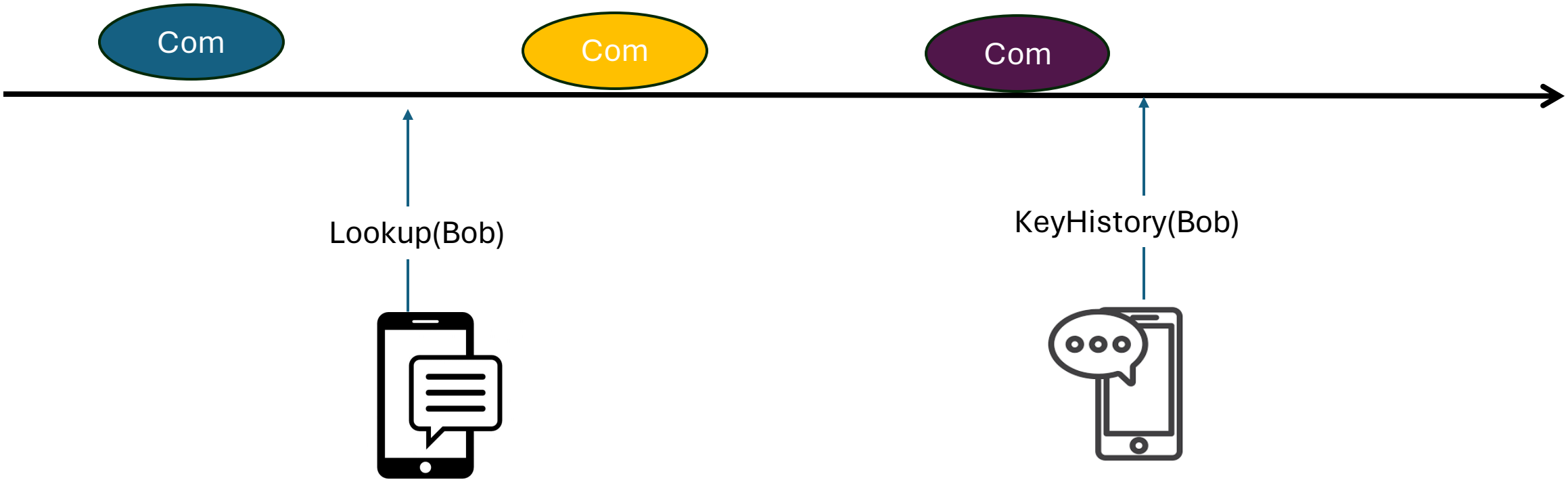
Soundness



F_k(Bob|**v.1**) , π_{VRF}, π_{member}
F_k(Bob|**v.2**) , π_{VRF}, π_{member}
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{member}, pk_B, open_3, t_3$
F_k(Bob|**v.4**), $\pi_{VRF}, \pi_{non-member}$

F_k(Bob|**v.1**) , $\pi_{VRF}, \pi_{member}, pk_{B1}, open_1, t_1$
F_k(Bob|**v.2**) , $\pi_{VRF}, \pi_{member}, pk_{B2}, open_2, t_2$
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{non-member}$

Soundness


Com
Com
Com
Lookup(Bob)
KeyHistory(Bob)
F_k(Bob|**v.1**) , π_{VRF}, π_{member}
F_k(Bob|**v.2**) , π_{VRF}, π_{member}
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{member}, pk_B, open, t$
F_k(Bob|**v.4**), $\pi_{VRF}, \pi_{non-member}$
F_k(Bob|**v.1**) , $\pi_{VRF}, \pi_{member}, pk_{B1}, open_1, t_1$
F_k(Bob|**v.2**) , $\pi_{VRF}, \pi_{member}, pk_{B2}, open_2, t_2$
F_k(Bob|**v.3**), $\pi_{VRF}, \pi_{member}, pk_{B3}, open_3, t_3$
F_k(Bob|**v.4**), $\pi_{VRF}, \pi_{non-member}$

Industry Deployments

Core Data Structure and Privacy

	Keybase	**WhatsApp**	**iMessage**	**ProtonKT**
Sparse Merkle Tree	✓	✓	✓	✓
Content Privacy (with VRF)	✗	✓	✓	✓
Metadata Privacy (unlinking key updates)	✗	✓	✗	✗
Post Compromise Privacy	✗	✗	✗	✗

Open Challenge: Auditor-Free Key Transparency

Auditing

Auditors check that commitment preserves previous history

- Necessary to reduce client monitoring

Proofs do not reveal information about individual users/updates

- Auditors do not need to be trusted for privacy

Auditors can be

- Users?
- 3rd parties?

Need Auditors?

Keybase	WhatsApp	iMessage	ProtonKT
✓	✓	[Additional assumption: tamper proof secret storage on server] ?	✓

Research Directions

Reducing auditing work with SNARKS (Verdict[TKPS22], Versa[TFZBT22])

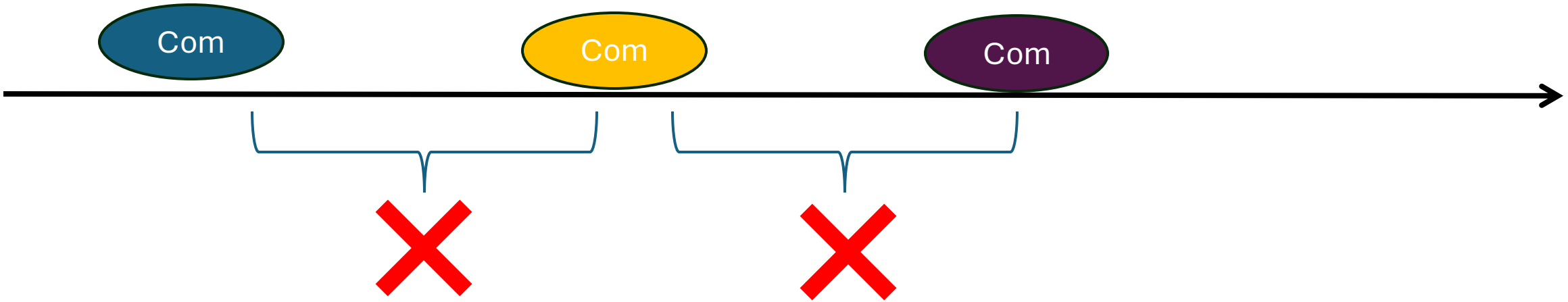
Client monitoring with weak consistency (Versa[TFZBT22]):

- Everytime Alice comes online she checks log commitments
- If Alice and Bob both do these checks, they either
  - get correct keys for one another
  - or detect misbehavior

Open problem: compress audit proofs and verification cost/optimized client auditing

Preliminary Result: Auditor-Free Client Monitoring Mode [GC24]

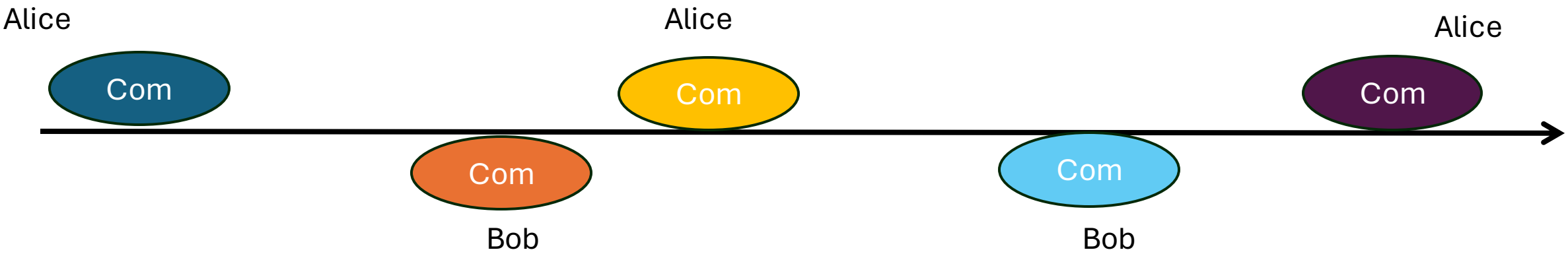
In the absence of a trusted auditor



No append-only checks between commitments

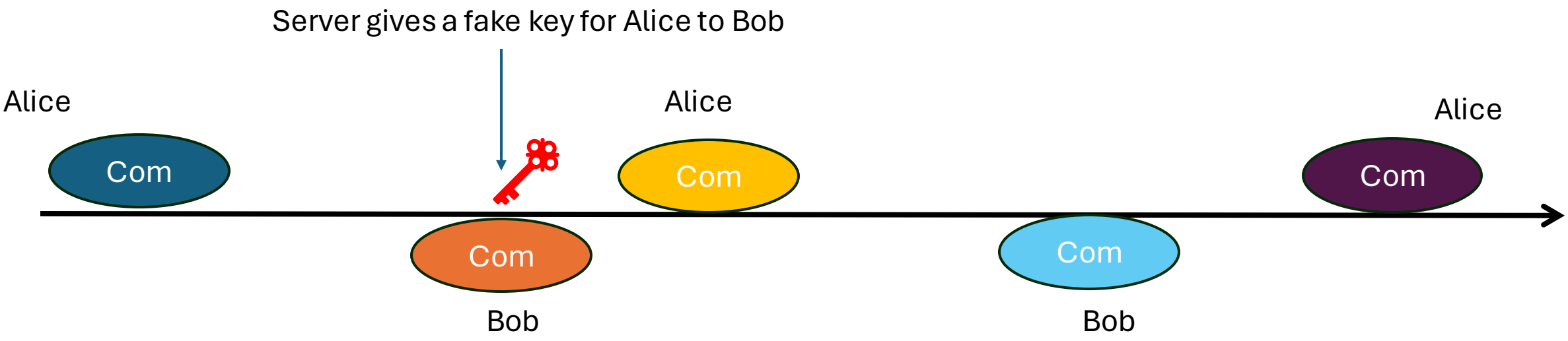
Can Alice and Bob check some additional proofs?

- A new algorithm: Monitor
- Alice and Bob performs monitor sporadically



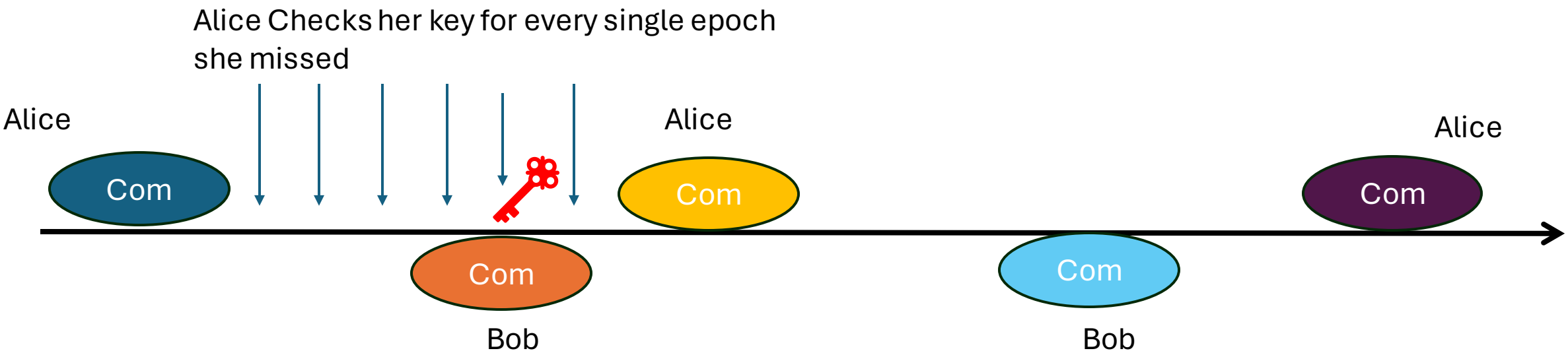
Can Alice and Bob check some additional proofs?

- A new algorithm: Monitor
- Alice and Bob performs monitor sporadically



We want this to be detectable even if no auditor in present

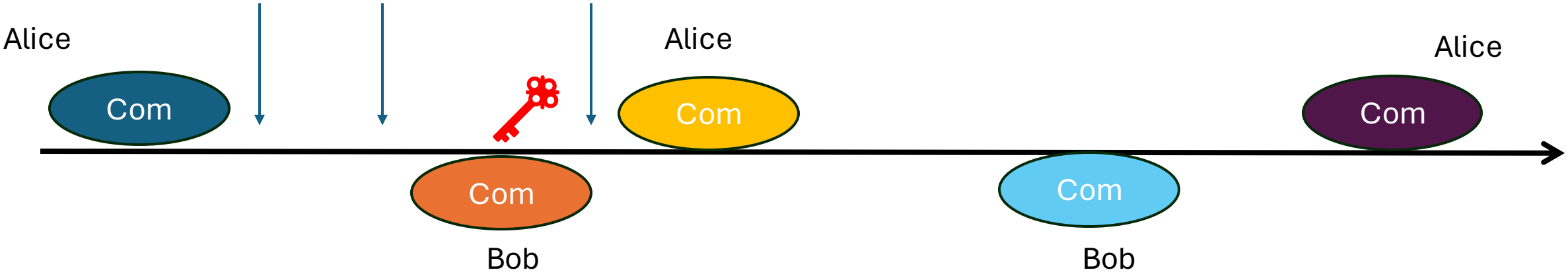
Inefficient Solution: Monitor Checks every epoch



Can we do better?

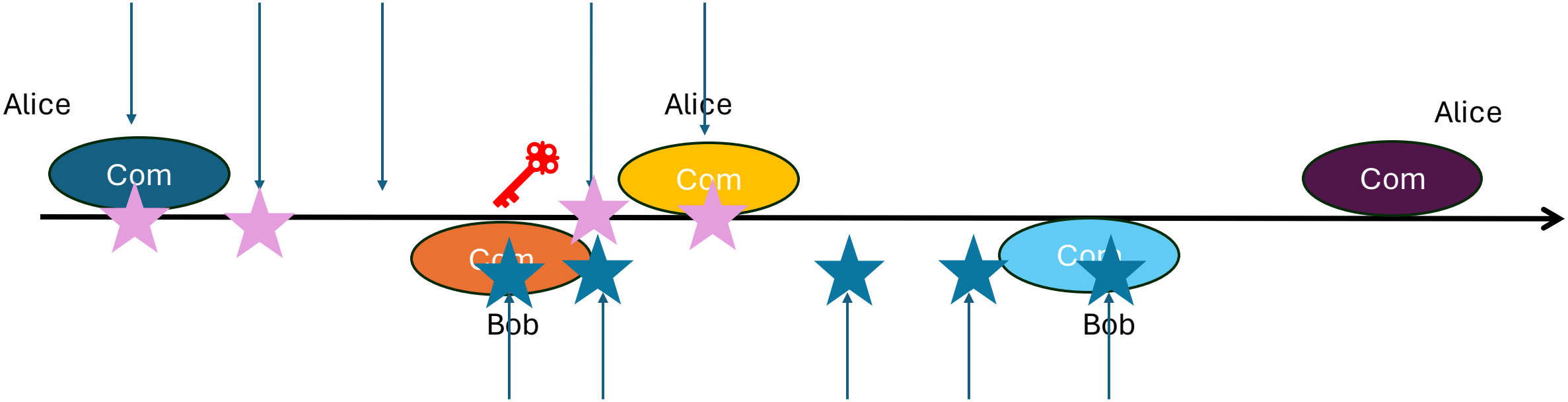
Not trivially, since Alice can easily miss the epoch in which the fake key was given out!

Can we achieve the detectability even when Alice checks log (number of missed epochs)?



Checkpointing

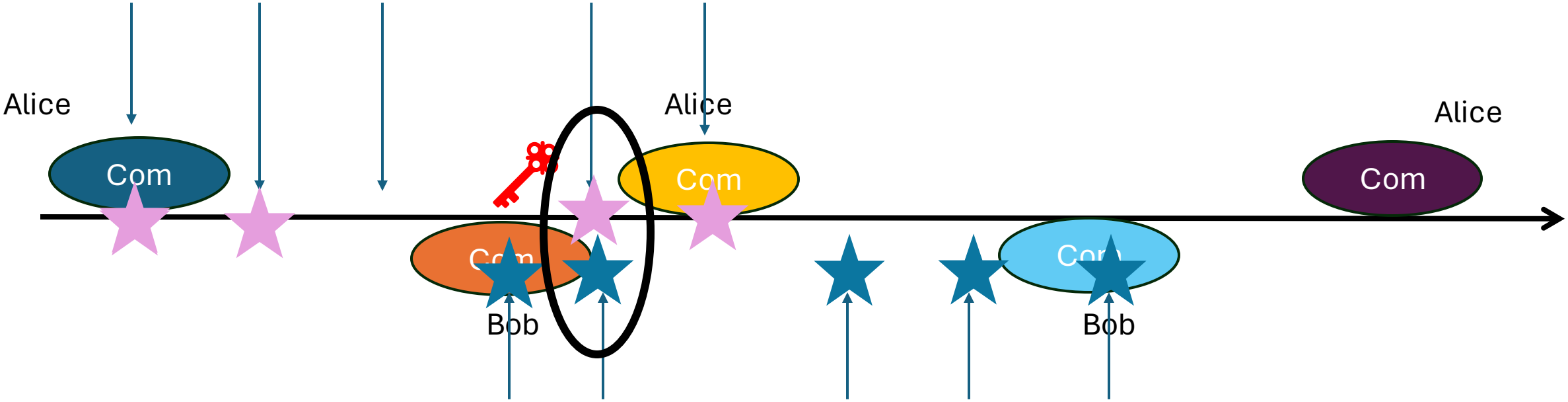
A deterministic algorithm that picks logarithmic number of points from on an integer interval [TFZBT22]



Checkpointing

Guarantee: Tv
intersection c

This checkpoint does not necessarily coincide with the bad epoch!



Solution

- At each checkpoint:
  - Run a OPTIKS type check for each of the keys
  - Check that the history of keys returned at each checkpoint are consistent

Full protocol specification in: eprint 2024/796

Challenges and Open Problems

Instantiating the Bulletin Board (BB)

- BB must be consistently viewed by all users
- Some alternatives:
  - 3rd party
  - Blockchain
  - Certificate Transparency Logs
  - Trusted hardware (e.g. Nimble[ABCJLSS23])
- Can we remove the BB?
  - Out-of-Band (OOB) gossip among users/their representatives
  - **Open problem: Can we use an in-band gossip?**
    - First suggested in [YGHZS22]
    - Apple may be doing this, but no analysis as far as we know
    - We have initial feasibility results, not yet efficient protocol

Stronger security

Service provider can replace user's key

- User will detect this but can't prevent it

Stronger security with multi devices (Keybase, ELEKTRA[LCGJKM23])

- Directory maps username to set of device keys
- Addition/removal must be signed by an existing device
- Service provider can't add/remove device by themselves

- Except with "reset"
  - added for usability
  - Rare, so can come with stronger UI messaging

Open problem: can we do any better?

Related Cryptographic Questions

Rotatable Zero Knowledge Sets/Verifiable Random Function

- Do we need GGM in order to achieve RZKS/RVRFs? Why or why not?
- Improve privacy of RVRF/RZKS

Better optimized SNARKs for compressing audit proofs?

Performant Post Quantum Secure VRF

Other Questions

How can we reduce DB cost?

What is the best way to do UI/explain KT?

How does the user trust client code?

- Code transparency? Verified client code?

What if there is a bug in the server code?

- Verified server code?

How can user tell account compromise from server misbehavior?

- What should the user do if they detect a fake key update?

Thanks!

esha.ghosh@microsoft.com

- SEEMless: eprint 2018/607 *(CCS)*
- Rotatable Zero Knowledge Sets: eprint 2022/1264 *(Asiacrypt)*
- Parakeet: eprint 2023/081 *(NDSS)*
- ELEKTRA: eprint 2024/107 *(CCS)*
- OPTIKS eprint 2023/1515 *(USENIX)*
- Client Auditability in OPTIKS: eprint 2024/796