Key Transparency: *Client Monitoring and 3rd Party Auditing*

Esha Ghosh

IETF 120

Deployment Modes [draft-ietf-keytrans-architecture-01]

3rd Party Auditor	Client Monitoring
Requires the transparency log to delegate part of its operation to a third-party auditor	No third-party auditor.
Users are able to run more efficiently as long as they can assume that the transparency log and the third party won't collude	Users are less efficient: it's less suited to use-cases where users look up a large number of ephemeral keys, but would work ideally in a use-case where users look up a limited number of keys repeatedly
Clients do not need persistent state (no signing keys or state between queries) between queries for security to hold	Clients require persistent state between queries for security to hold
WhatsApp, ProtonKT deployment	

Supporting both modes [Weak Consistency mode in Key Transparency: OPTIKS (iacr.org)]

3rd Party Auditor	Client Monitoring
Auditor Cost: $t \log N$ [For t updates per epoch, size of the directory is N	
Search: $v \log N$ Monitor: $v \log N$ [For a key with versions, size of the directory is N]	Search: $v \log N \log e$ Monitor: $v \log N \log e$ [For a key with versions, size of the directory is N, user comes online after e epochs]
Clients do not need persistent state between queries for security to hold	Clients require persistent state between queries for security to hold

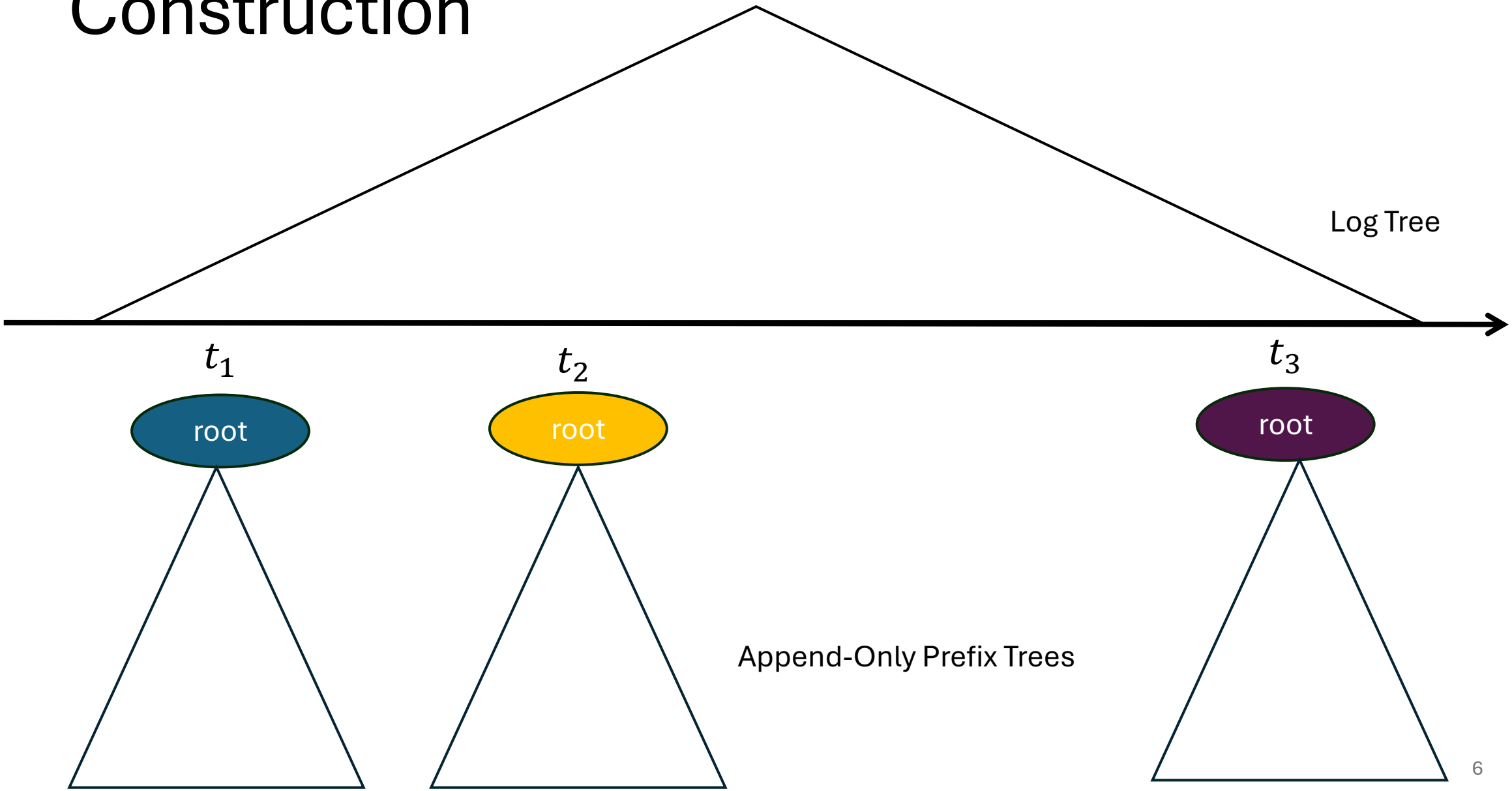
Comparison [Client Monitoring]

wcOPTIKS	draft-keytrans-mcmillion-protocol-00.txt
Search: $v \log N \log e$ Monitor: $v \log N \log e$ [For a key with versions, size of the directory is N, user comes online after e epochs]	Search: $\log v \log^2 N$ Monitor: $\log v \log^2 N$ [For a key with versions, size of the directory is N, user comes online after e epochs]

Comparison [3rd Party Auditing]

wcOPTIKS	draft-keytrans-mcmillion-protocol-00.txt
Auditor Cost: $t \log N$ [For t updates per epoch, size of the directory is N]	Audit Cost: $\log N$ [Only 1 update per epoch is allowed, size of the directory is N]
Search: $v \log N$ Monitor: $v \log N$ [For a key with versions, size of the directory is N, user comes online after e epochs]	Search: $\log v \log^2 N$ Monitor: $\log v \log^2 N$ [For a key with versions, size of the directory is N, user comes online after e epochs]
Clients do not need persistent state between queries for security to hold	Clients require persistent state between queries for security to hold

Construction



Prefix Tree Construction

[SEEMless [CDGM19]]: label = username||version

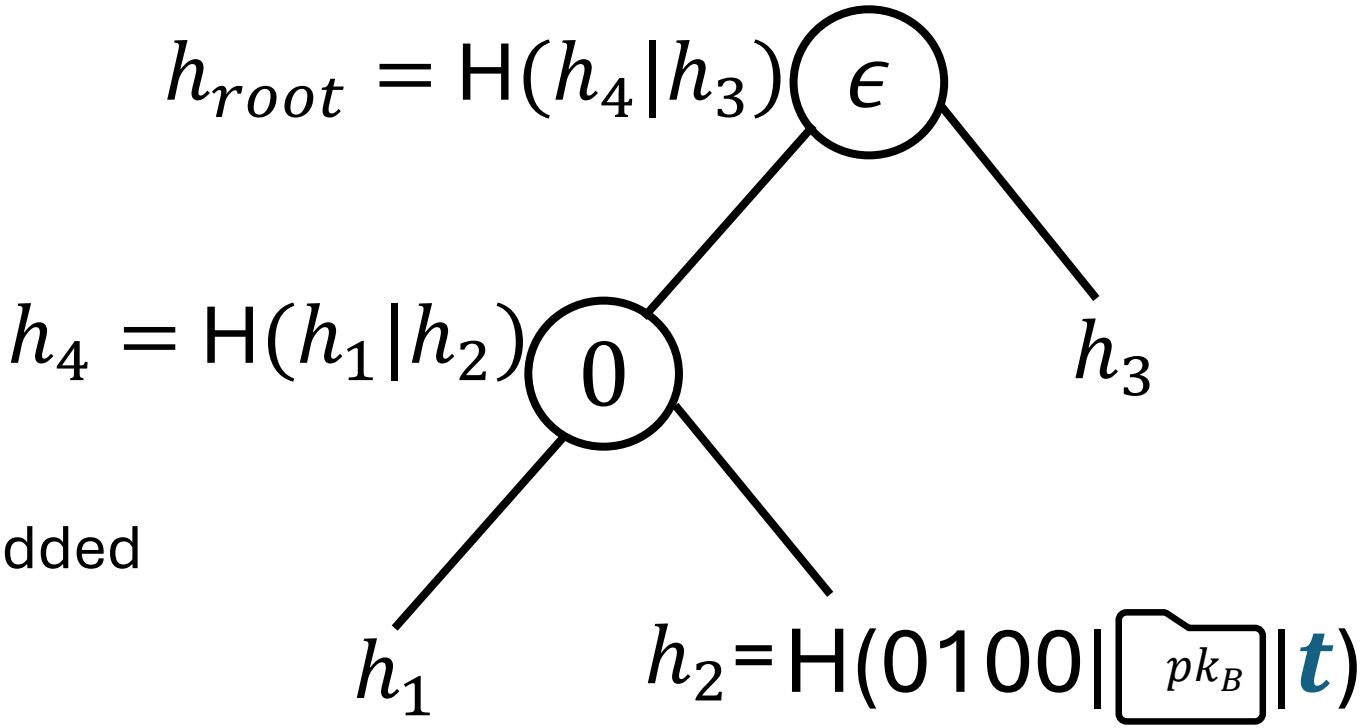
VRF_k(Alice|**v.1**) = 0001

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

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

Tree includes all versions

- Leaf includes epoch it was added



3rd Party Auditor Mode (1/2) [OPTIKS [LCGKM24]]

Auditor checks :

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

3rd Party Auditor Mode (2/2) [OPTIKS [LCGKM24]]

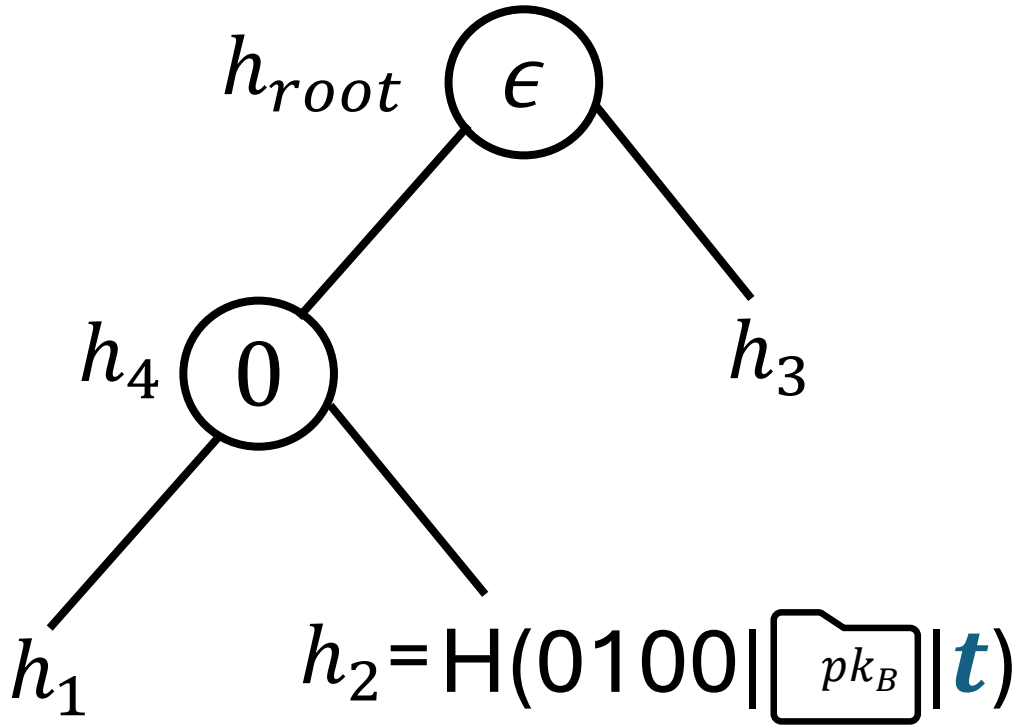
Search(Bob):

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

VRF_k(Bob|**v.2**) ,π_{VRF}, π_{member}

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

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

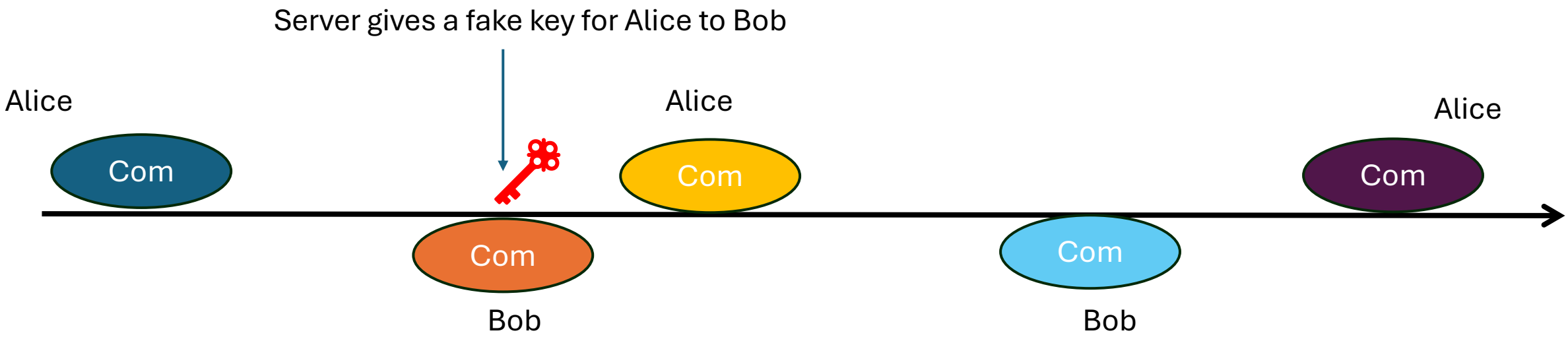


Monitor(Bob):

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

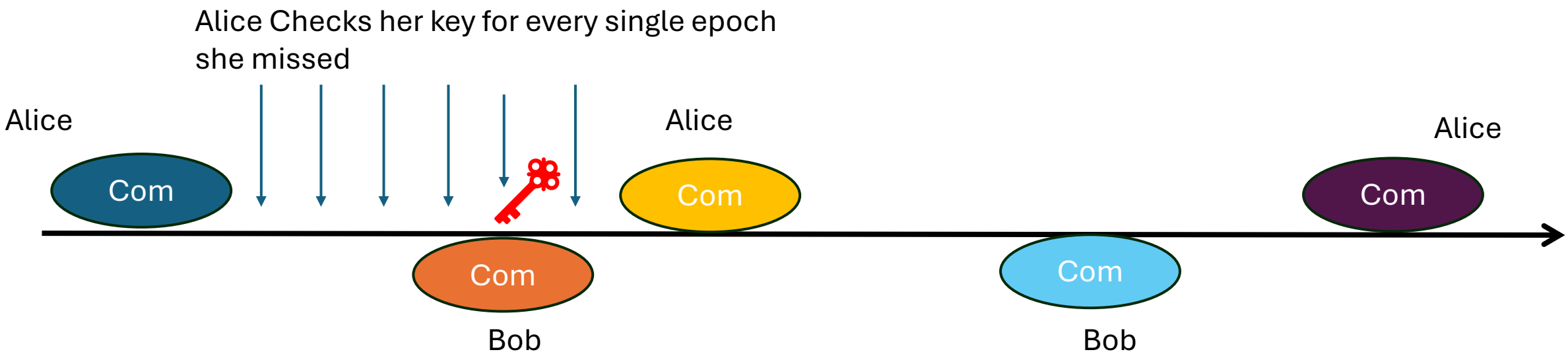
Client Monitoring Mode [wcOPTIKS [GC24]]

- Clients save the state of the key they looked up last
- They perform some additional checks



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

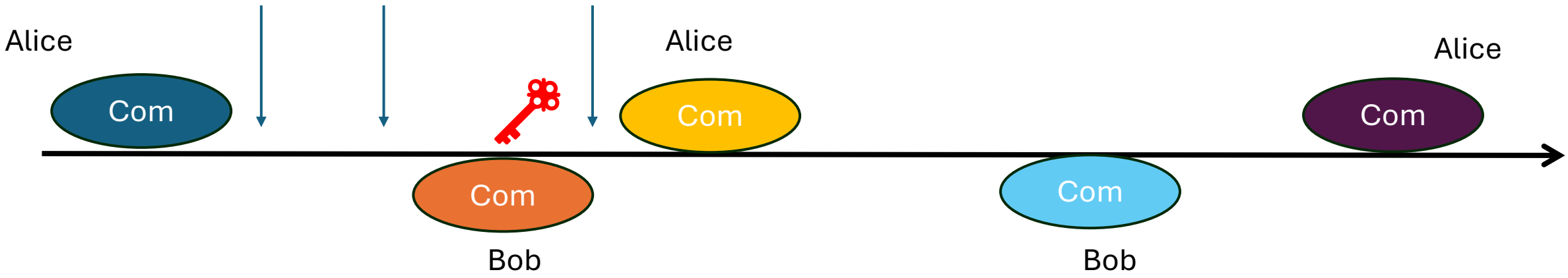
Inefficient Solution: Check every missed epochs



Can we do better?

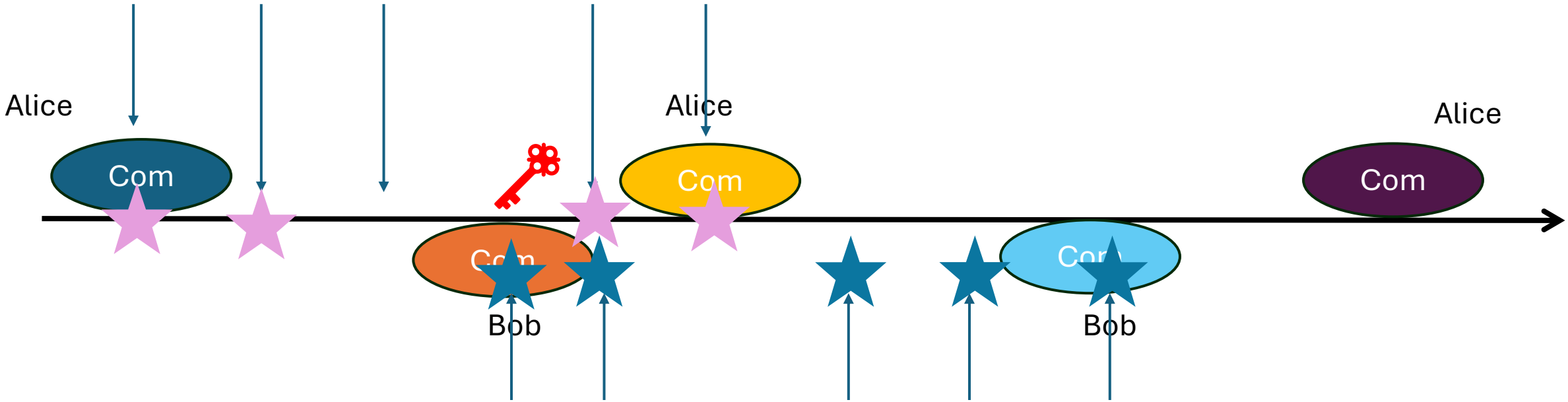
Not trivially, since Alice can easily miss the epoch in which the fake key was given out!

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



Checkpointing [wcOPTIKS [GC24]]

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



Summary

- We have a candidate construction that supports both modes
- It also takes advantage of the auditors to
  - Make the clients more efficient
  - The clients do not need persistent state between queries
- Plenty of room for client algorithm and storage optimization