WhatsApp Key Transparency

Sean Lawlor
Software Engineer

Kevin Lewi
Software Engineer

Agenda

Overview

Infrastructure

Auditable Dictionaries

Overview

End-to-End Encrypted Messaging

Phone #s	Public Keys
Alice	pk_Alice
Bob	pk_Bob
…	…

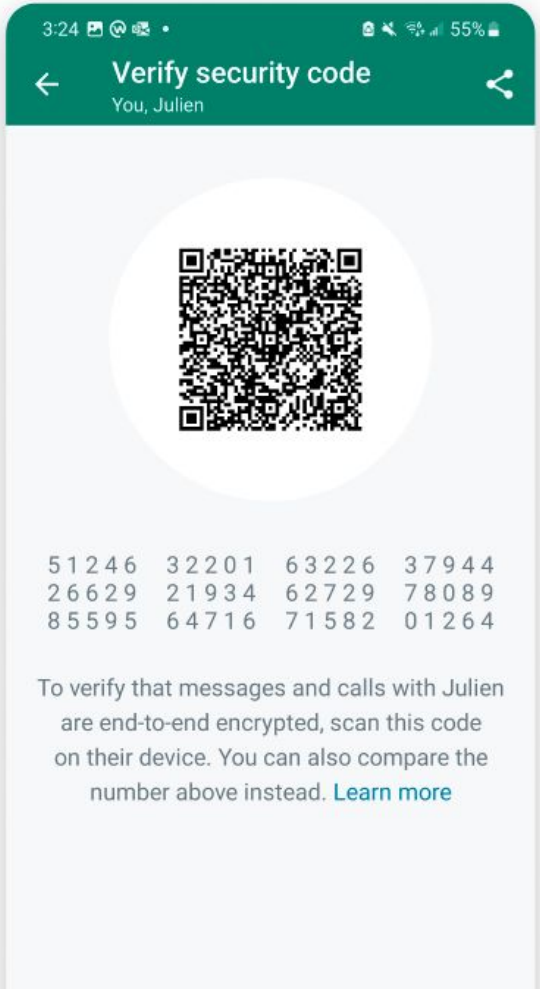
Security Codes

Alice and Bob can verify that they got the correct public key if they have an already-established authenticated channel (e.g. Zoom call, meeting up in person)

Code = Hash(Alice's public keys, Bob's public keys)

Note: this code changes every time Alice or Bob add a new device!

Also: Group chats?

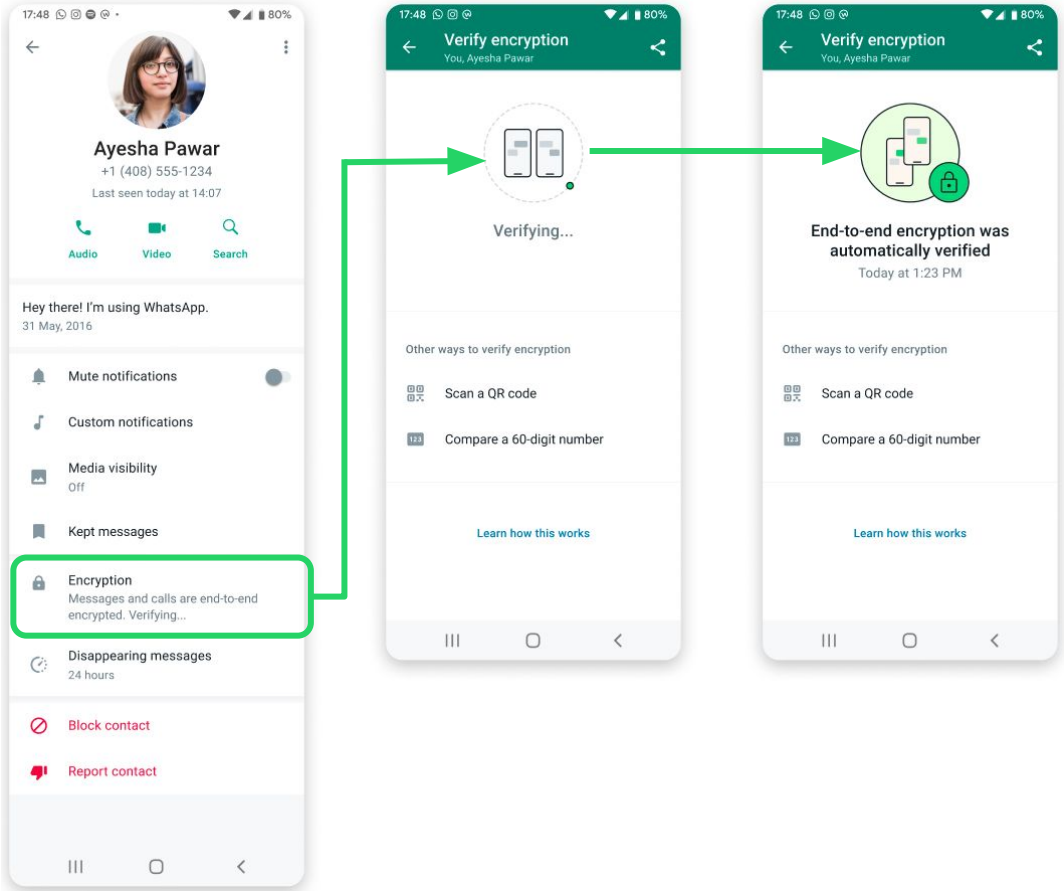


Key Transparency

Automatic validation of public keys

How?

- WhatsApp servers publish a commitment to the database of public keys
- Users check their public keys against this commitment to make they are consistent



Infrastructure

Normal Registration and Lookup

Registration *(Write Path)*:

"Hi, I'm Bob, I want to register a new key: 4c94884df1bc…"

Lookup *(Read Path)*:

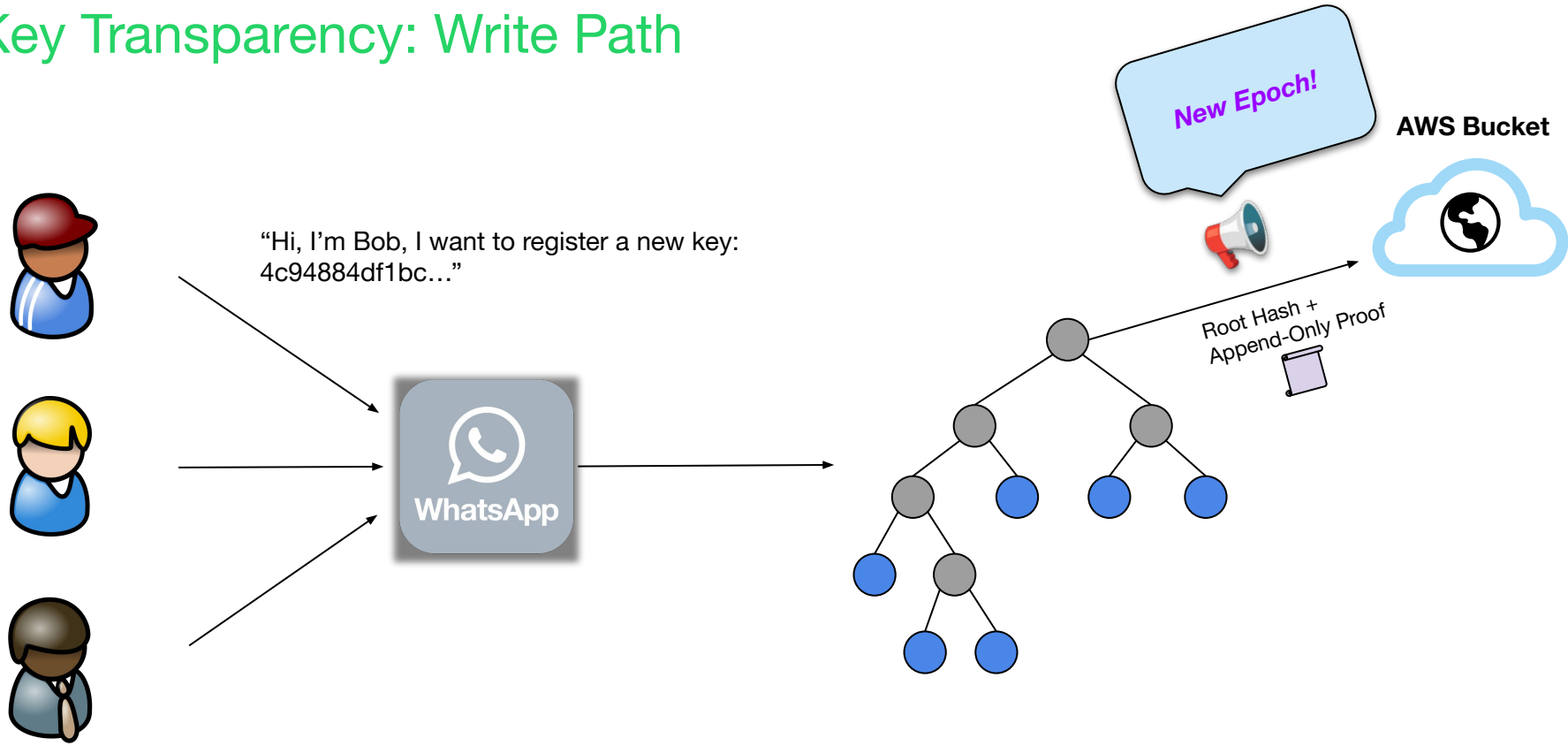
"What is Alice's latest public key?"

ecb6427d8ae8…

Database

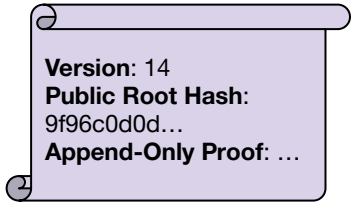
User	Key
Alice	ecb6427d8ae8…
Bob	**4c94884df1bc…**
Charlie	95f64aee5f4d…

Key Transparency: Write Path

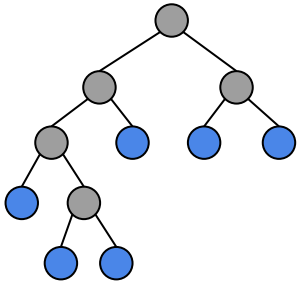


Key Transparency: Publish

Each publish contains:

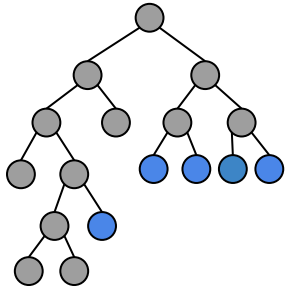


Version: 14
Public Root Hash:
9f96c0d0d583298…



Append-Only Proof

Version: 15
Public Root Hash:
481109384d45…



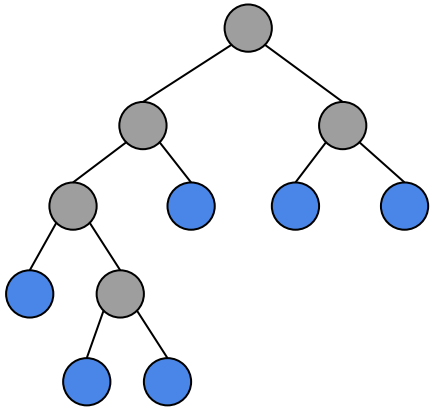
Append-only proofs guarantee that we manage the database consistently

Key Transparency: Read Path

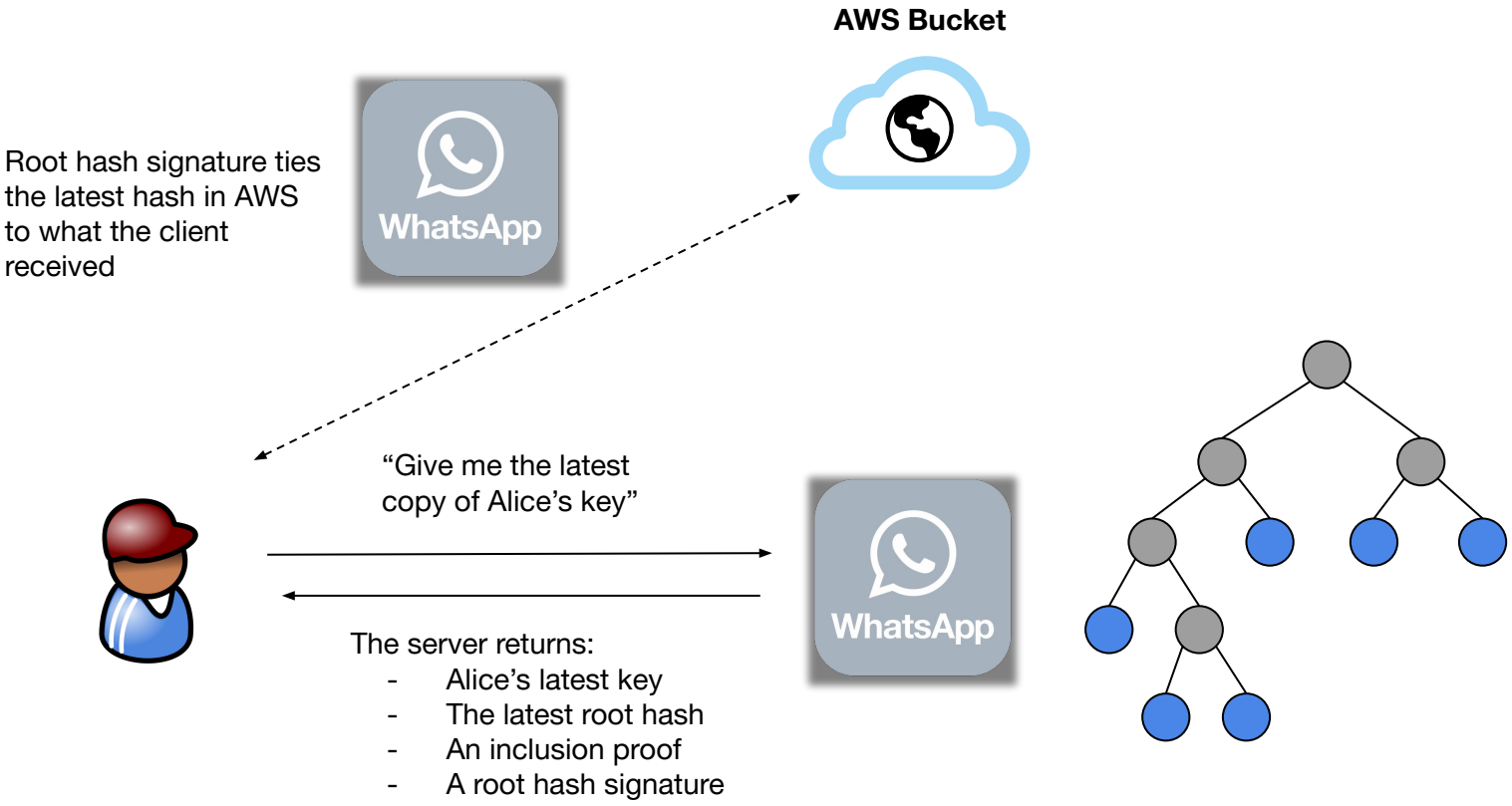
“Give me the latest copy of Alice’s key”

The server returns:

- Alice’s latest key
- The latest root hash
- An inclusion proof
- A root hash signature

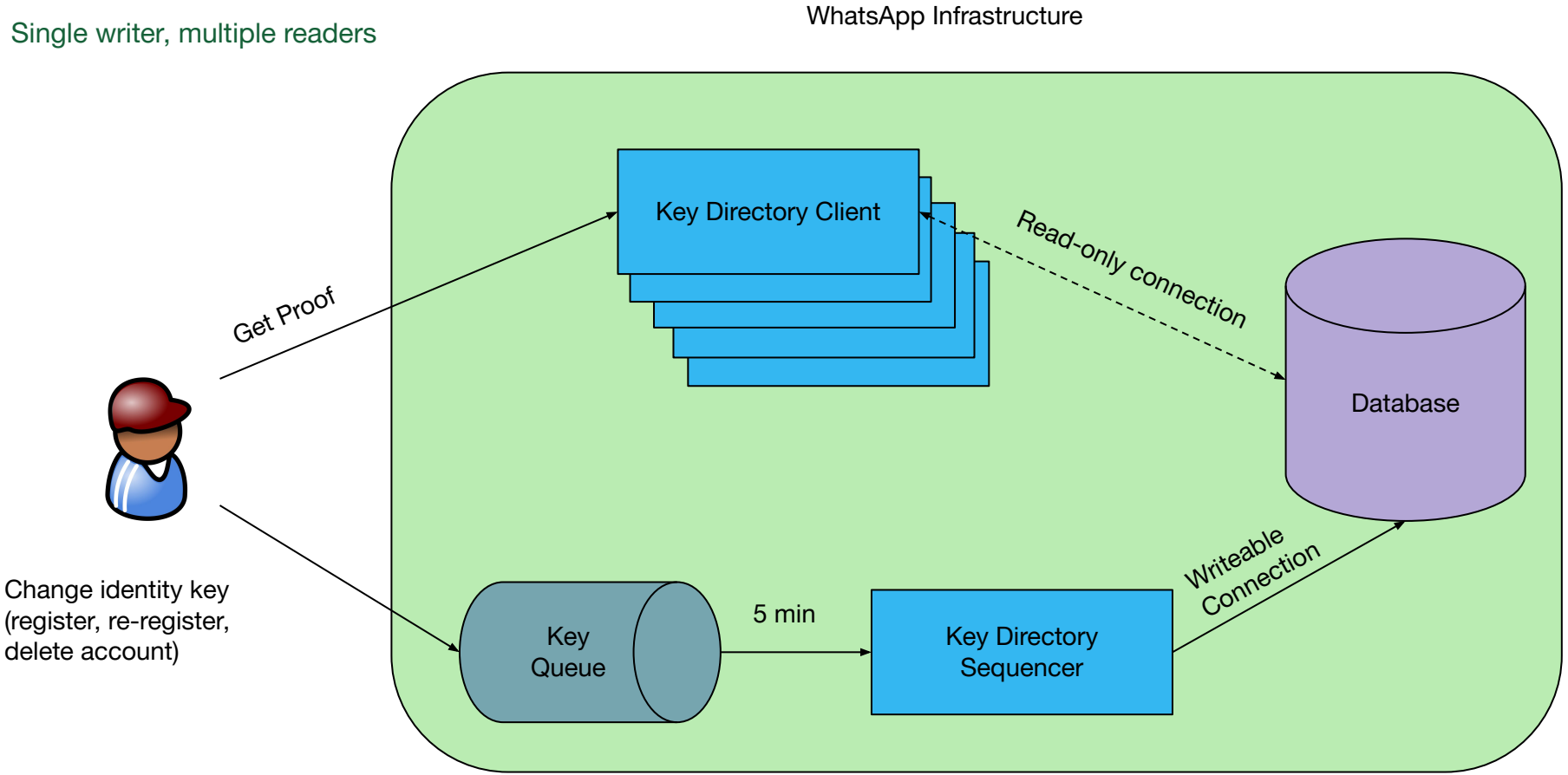


Key Transparency: Read Path



Infrastructure

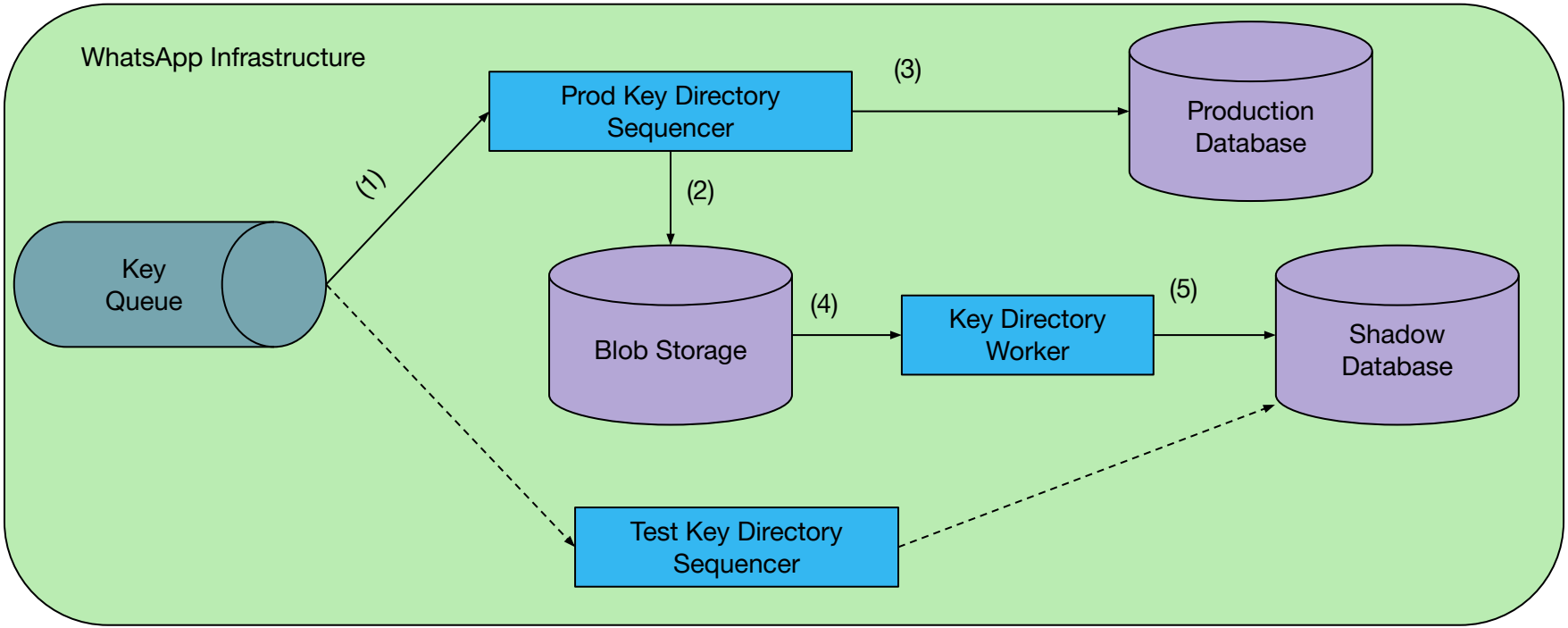
Single writer, multiple readers



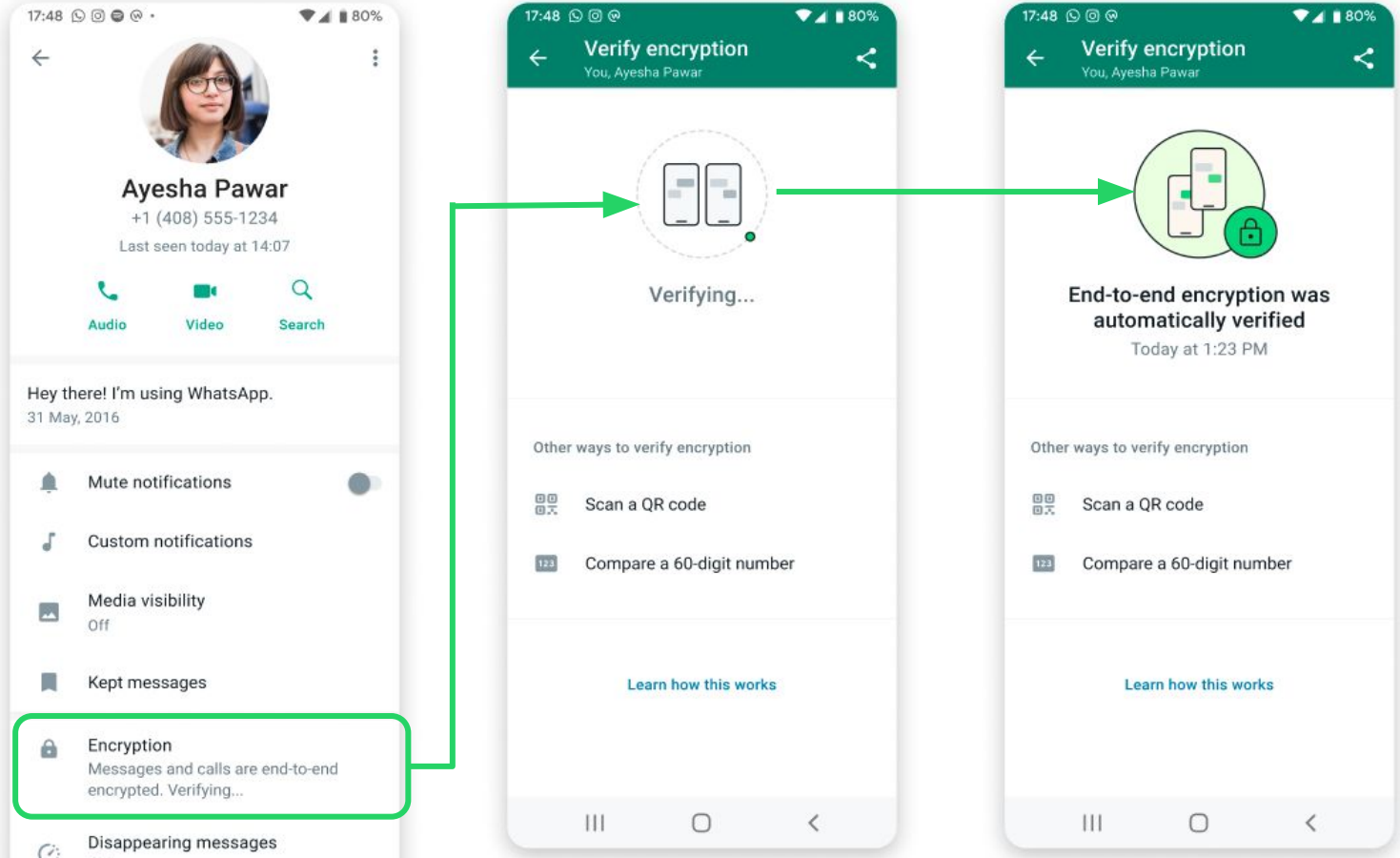
Infrastructure - Some Gotchas

Testing write flows

- Single writer - one binary tree to rule them all
- “Shadow” clone of prod database
- **Pause, resume, replay** all supported from shadow logs



Client Experience



Auditable Dictionaries

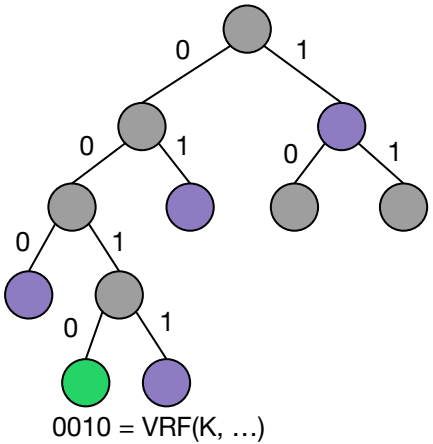
Construction: Sparse Merkle Trees and VRFs [CONIKS 2015]

Sparse Merkle Trees:

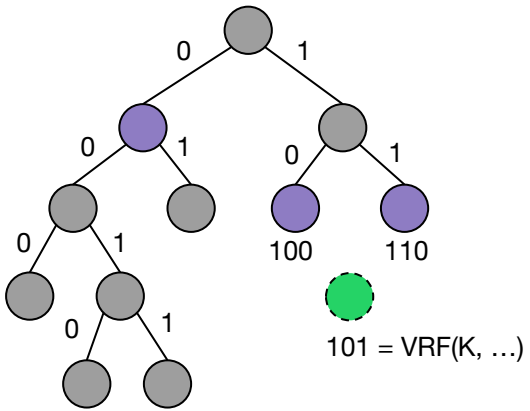
- Unique positions for entries
- Supports inclusion and non-inclusion proofs

We use Verifiable Random Functions (VRFs) to randomize leaf positions in the Merkle tree

Inclusion Proof



Non-Inclusion Proof

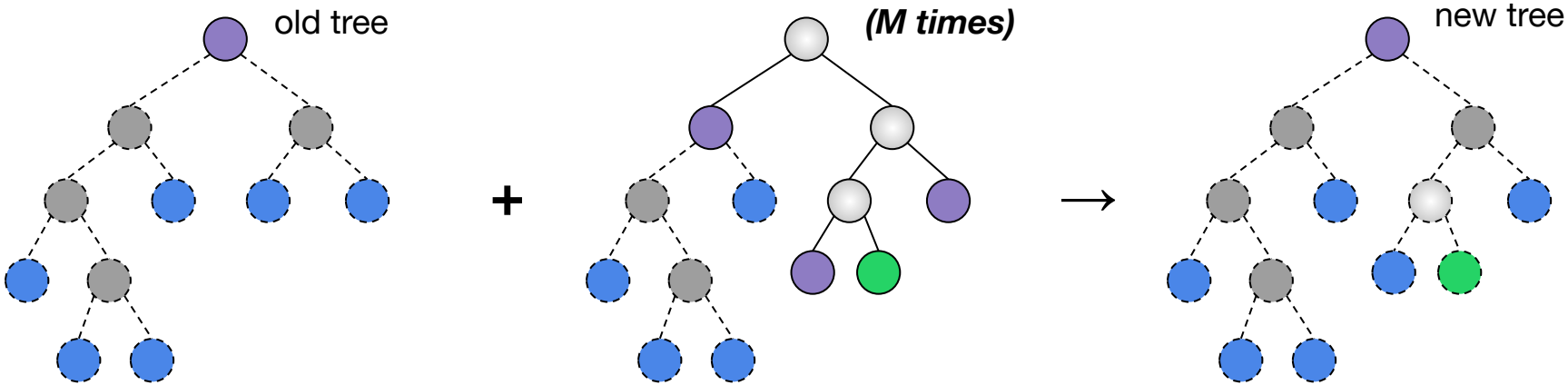


Large Append-Only Proofs

Auditors verify append-only-ness of the tree

Audit proofs only contain leaf values (hashes), not the raw public keys themselves.
However, proofs are O(M log N) in size, where: M = # of updates, N = total # of leaves in the tree

In practice: they are **~200 MB** each!



AKD: Rust Open-Source Library

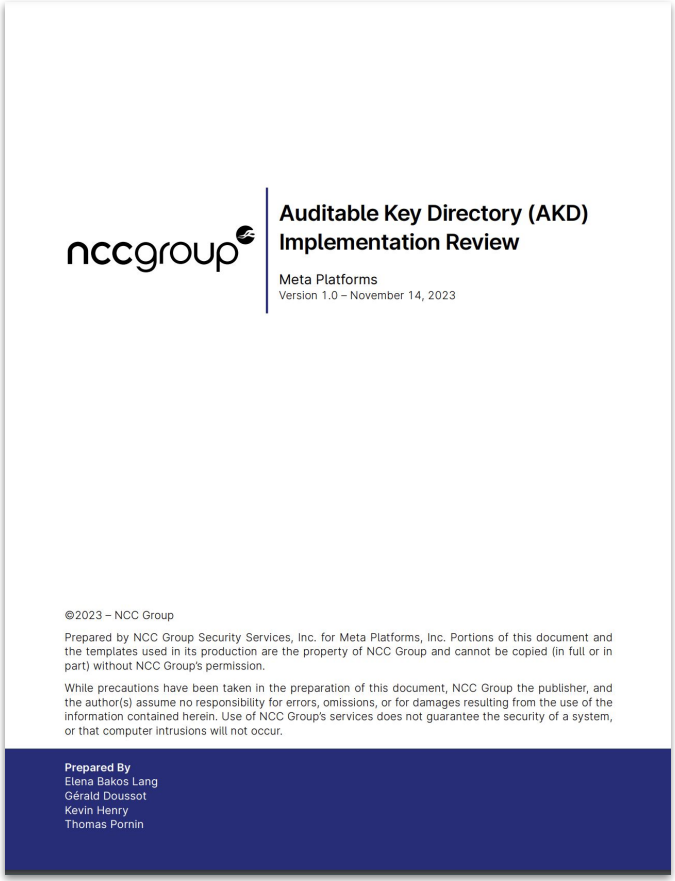
https://github.com/facebook/akd

- An (optimized) implementation of the SEEMless [CDGM'19] protocol
  - Built on top of a Sparse Merkle tree and ECVRF (RFC 9381)
- Same as what we are using in WhatsApp today
  - In fact, you can use it to verify our audit proofs
- Dual-licensed under Apache 2.0 and MIT

AKD: For Industry

https://github.com/facebook/akd

- Composable storage trait for flexibility
- Employs preloading nodes + caching to make operations more efficient
- Audited by NCC Group in Nov 2023



AKD: For Academics

https://github.com/facebook/akd

- We also have benchmarks to test against, future academic works may be able to use this

- Crypto operations are configurable
  - E.g. swapping out VRF, hash function

Related Work

Merkle-tree-based solutions:

- CONIKS [MBBFF'15]
  - SEEMless [CDGM'19]: more efficient history checks + privacy guarantees
    - Parakeet [MKSGOLL'23]: putting SEEMless into practice, handling deletion
- Merkle^2 [HHKYP'21]: Uses Merkle prefix tree + chronological tree together
- Rotatable Zero Knowledge Sets [CDGGKMM'22]
  - Addresses forward secrecy for VRF private key

Algebraic solutions:

- Transparency Logs via Append-Only Authenticated Dictionaries [TBPPTD'19]
- Verdict [TKPS'21], VeRSA [TFZBT'22]

Other implementations:

- Keybase [2015], Google [2017], Zoom [2020], Apple [2023], Proton [2023]