Privacy for Key Transparency

Kevin Lewi

IETF 118 - November 10, 2023

Outline

- Recap on key transparency operations
- Different ways in which a design can leak privacy
- Privacy tools
- Handling data deletion and retention
- Takeaways

Key Transparency Operations

Service provider keeps track of a DB of user identifiers to public keys.

Identifiers	Public Keys
Alice	pk_Alice
Bob	pk_Bob
…	…

Users can:

1. Add/Update their entry
2. Search / Lookup: User checks an entry in the database + inclusion proof
3. Monitor / Audit: User (or auditor) checks consistency proofs of append-onlyness

Service providers need to be careful to not ***inadvertently leak user data*** through these operations!

Privacy Leakage for KT proofs

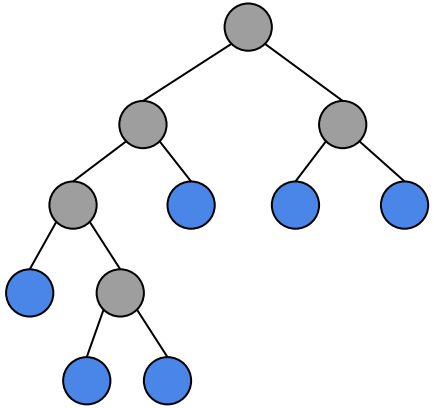
Example #1: Merkle tree inclusion proofs

Problem #1: Traditional Merkle tree inclusion proofs reveal the approximate height of the tree.

- But maybe this isn't too bad…

Problem #2: They also reveal the values of neighboring nodes along the path to the root

- Can be partially addressed with VRFs, will discuss this later



Privacy Leakage for KT proofs

Example #2: Monitoring when users update their entries

Design question: If a user updates their public key 5 times, do these 5 updates get stored together, or is it indistinguishable (to an adversary) from 5 different users updating their own keys?

If it's the former, then an adversary could potentially learn statistics such as: the frequency distribution of how often people are updating keys (some people seem to be updating once per day, but then suddenly stopped)

Privacy Leakage for KT proofs

Example #3: Update history in the context of identifier re-use

Should a user be able to look arbitrarily far into the past for the update history for their phone numbers?

Phone number recycling: What if your phone number was owned by a famous person, and now you have control over it… does this mean that the service provider will show you information regarding the behavior patterns of the previous owner?

Privacy Leakage for client queries

Example #4: Service provider learns information based on client queries

For instance, if the service provider sees that one entry is queried a lot, they learn some information about this user (they are popular).

To some extent, this is a privacy problem with a regular database (no key transparency) anyway, but with key transparency we are asking clients to also query for their key update history.

Privacy-Preserving Tools: VRFs

Verifiable Random Functions (VRFs)

- VRF(server key, phone number) -> (random id, proof)
- VRFVerify(public key, phone number, random id, proof) -> true/false

Typically, the “position” of a leaf node in a Merkle prefix tree is uniquely determined by the user identifier.

To preserve privacy, we can pass the user identifier through a VRF and use the random output to determine the position instead. (CONIKS [MBBFF’15])

Privacy-Preserving Tools: VRFs

However, doesn’t solve all problems

- What happens when a user wants to update their value? Should it modify the existing entry?
- One option: Pass (user identifier, version #) through the VRF, so that each key update is placed in a random location in the tree (SEEMless [CDGM’19])

Issue: What if the server VRF private key gets leaked?

- Rotatable Zero Knowledge Sets (https://eprint.iacr.org/2022/1264.pdf) addresses this issue
- Post Quantum-secure VRFs: https://eprint.iacr.org/2022/141

Privacy-Preserving Tools: zkSNARKs

Another option, highly theoretical though, is to use zero knowledge proofs to hide information.

In theory, zkSNARKs can be used to achieve ideal privacy.

However, zkSNARKs don’t appear to be (I think?) practical enough for key transparency deployments today.

… But this is an open research question and advancements in zk technology could change this in the future!

Handling data deletion

Merkle tree nodes correspond to hashes of user data

Can we just delete the raw user data and leave the hashes? Does this suffice for “deleting the user data”?

Should service providers be required to keep a complete history of all data updates since the beginning of time?

Solutions:

- “Reset the tree”
- Temporary consistency requirements:
    - Key transparency for a certain time period (of, say, the past 2 years), and all older data gets automatically cleaned up

Takeaways

- Managing privacy leakage can be tricky
  - Especially because more private usually means less performant / scalable
- We should have extreme clarity about the privacy leakage for any designs we propose
  - Ideally: **Provable guarantees**, i.e. “This proof will leak _____, and nothing else”
- Different service providers may make different choices about what is ok / not ok to leak
  - And we could potentially have some flexibility in the design we settle on for allowing “more private” vs. “less private” solutions