Literature Review:
Key Transparency

Crosby, Scott A., and Dan S. Wallach. "Efficient data structures for tamper-evident logging." *USENIX security symposium*. 2009.

Introduces the model of having an untrusted logger be the sole author and signer of a tamper-evident log – as opposed to previous systems where the log may be stored somewhere untrusted but still has a trusted author. Clients insert events into the log and get incremental proofs that their insertions were done correctly; auditors check that the log provides a consistent view of data.

Extra data can be stored in the intermediate nodes of the Merkle tree that aggregate information about the leaf nodes below that intermediate. This extra information can enable securely searching through the log. It can also enable the log to delete events that match some policy, while still being able to prove that only events that match the policy were deleted.

Ryan, Mark D. "Enhanced certificate transparency and end-to-end encrypted mail." *Cryptology ePrint Archive* (2013).

Aims to improve Certificate Transparency by adding support for efficient certificate revocation, though the same approach could be used managing certificates for email encryption.

Certificates are stored in two different types of Merkle trees: a log tree where certificates are added chronologically as they're issued, and a prefix tree that maps subject name to a certificate. Auditors read the entire chronological tree to ensure that the prefix tree is constructed correctly. Users periodically verify the most recent version of their binding doesn't contain a certificate they didn't authorize, relying on auditors to ensure that nothing was added and then omitted in previous versions.

Note: Only one certificate is allowed per domain, but in practice this isn't a reasonable constraint.

Melara, Marcela S., et al. "CONIKS: Bringing Key Transparency to End Users." *24th USENIX Security Symposium (USENIX Security 15)*. 2015.

First instance of a verifiable key directory occurring in the literature. Stands for CONsistent Identity and Key Service. Goals: non-equivocation, no spurious keys, privacy-preserving consistency proofs, concealed number of users, human-readable names.

Data structure is a Merkle prefix tree with a signed root that chains back to the previous epoch. Auditors ensure only a linear STR (Signed Tree Root) history; users must verify their key binding is correct in every epoch. STRs include VRF (Verifiable Random Function) public keys, and leaf nodes include a commitment to the username the leaf is for, since VRFs may not be collision-resistant. Providers can produce signed, temporary bindings to a user's key to allow them to start using their key right away while also letting the time between epochs be larger; temporary bindings are validated in the next epoch.

Proposes that CONIKS providers act as auditors of other CONIKS providers. Has a distinction between "normal" and "strict" users. Strict users can encrypt their binding with a symmetric key, and sign all account changes with their private key.

Note: It's not clear how relying users would enforce that all "strict" account changes are signed.

Bonneau, Joseph. "EthIKS: Using Ethereum to audit a CONIKS key transparency log." ***International Conference on Financial Cryptography and Data Security*****. Springer, Berlin, Heidelberg, 2016.**

Shows that it's possible to implement CONIKS as an Ethereum contract. This enables frequent STRs (as frequently as Ethereum can generate new blocks), while relying on the Ethereum network to audit that the tree structure of CONIKS is maintained correctly and free from forks.

Note: I have difficulty figuring out the conversion, but I believe the prices quoted in the paper would be 1000-1500x more expensive in today's dollars:

Operation	Jan 2016 Cost (USD)	Jul 2022 Cost (USD)
Create Tree	$0.0036	$5.04
Insert New User	$0.0004	$0.56
Update Mapping	$0.0001	$0.14
Delete User Data	$0.0001	$0.14
Change Ownership	$0.0002	$0.28
Tombstone User	$0.0002	$0.28

Yu, Jiangshan, Mark Ryan, and Cas Cremers. "How to detect unauthorised usage of a key." ***IACR Cryptol. ePrint Arch.*** **2015 (2015): 486.**

Extends the idea of using a Certificate Transparency-like system to an Instant Messaging context, or generally any context where a high-value signature key pair exists. Receivers in a messaging system authenticate short-lived certificates with their long-term signing key and publish them in a transparency log. The short-lived certificate public keys are

what are used for encryption. Senders require proof that a certificate is in the transparency log, and receivers audit the transparency log to ensure that all the certificates issued for their identity are legitimate.

Lerner, Ada, Eric Zeng, and Franziska Roesner. "Confidante: Usable encrypted email: A case study with lawyers and journalists." ***2017 IEEE European Symposium on Security and Privacy (EuroS&P)*****. IEEE, 2017.**

They built a standalone email client that only supports sending encrypted email, as opposed to a browser extension that modifies an existing email client. Having a client which is only for encrypted mail helps prevent users from accidentally sending sensitive information over plaintext, though many users express a desire to have an integrated client anyway. Email encryption keys are attached to a user's Keybase profile. The social aspect of Keybase made finding the right encryption keys much easier for people, though it would've been easier still if Keybase could've handled the association between encryption keys and email addresses.

Tomescu, Alin, et al. "Transparency logs via append-only authenticated dictionaries." ***Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*****. 2019.**

Goal of the paper is to design an authenticated data structure that has both inclusion and consistency proofs that are only logarithmic in size. Normally, Merkle trees can only provide one of these types of proof in logarithmic size – the other proof type would be linear. Their construction is based on bilinear pairings, though not believed to be deployable.

Chu, Dawei, et al. "Ticket transparency: Accountable single sign-on with privacy-preserving public logs." ***International Conference on Security and Privacy in Communication Systems*****. Springer, Cham, 2019.**

Proposes using transparency logs to improve the security of SSO by recording in a log every time an Identity Provider (IdP) issues a ticket that authorizes the user to access a Relying Party (RP) website. This way, if a user's account is compromised they can review the log to discover which RP websites are affected. Goals: RPs should only accept tickets that have been stored correctly in a transparency log, and the data available in/to the log shouldn't infringe user privacy.

Normal SSO tickets consist of a signature over the user's name and the RP's name, possibly with some other policy data. When a transparent IdP issues a ticket, it requests a blinded signature on the ticket from the log provider. The log provider adds the blinded signature, the blinding factor (encrypted with identity-based encryption), and the user pseudonym to the log. Identity-based encryption was chosen to allow the ticket to be recovered later by the user with minimal key management. The still-blinded signature and the blinding factor are sent from the IdP to the RP: the RP verifies that the blind signature was included in

the log, the user pseudonym included in the log is correct, and that the unblinded signature is valid.

Chase, Melissa, et al. "Seemless: Secure end-to-end encrypted messaging with less trust." *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019.

Describes a verifiable key directory that, unlike CONIKS, requires at least one trusted external auditor to ensure the consistency of each update but allows users to audit their own key less frequently. The operational model therefore ends up being the same as Enhanced CT. Goals: more efficient auditing, preventing “tracing” attacks (monitoring a user’s key updates without permission).

The data structure is a pair of prefix trees, called All and Old. Every time an account is updated, an entry is added to the All tree with the key “username||version” while the previous most-recent entry is added to the Old tree. This way, the Old tree contains everything the All tree does except the most recent version of each key, and the fact that keys aren’t reused prevents tracing. Providing a combination of proofs from both All and Old convinces users they’re getting the most recent version of a key.

Meiklejohn, Sarah, et al. "Think Global, Act Local: Gossip and Client Audits in Verifiable Data Structures." *arXiv preprint arXiv:2011.04551* (2020).

Goal is to provide improved gossiping mechanisms for transparency logs, and improve the performance of users auditing their own keys. The gossip protocol proposed consists of having auditors sign and publish the most recent N STHs that they’ve verified; all STHs a user observes from at least Q auditors are accepted. The verifiable key directory construction is the same as Enhanced CT, though there’s no auditors so users monitor each epoch themselves (with a focus on only their own key).

Hu, Yuncong, et al. "Merkle 2: A Low-Latency Transparency Log System." *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

Goal is to improve the latency of updates and also support efficient per-user monitoring without external auditors. The core data structure is a Merkle log tree where each intermediate node contains the root of a prefix tree. The prefix tree in each intermediate contains the keys and values of only the leaf nodes of that subtree. Inclusion/non-inclusion proofs from the prefix tree guide binary search for either all or the most recent instance of a key in the tree.

This tree structure still allows some spurious values to be inserted by a malicious server, so subsequent updates to a key are signed by a public key stored in the leaf node where the key was first inserted in the log.

Note: Doing a binary search with prefix trees embedded in internal nodes (where the prefix tree aggregates over the subtree) seems to be equivalent to doing binary search in the log with prefix trees only in the leaves (which aggregate over all previous leaves).