Efficient Auditing for Public Key Infrastructure via Server-Aided Gossiping

Presenters*: Shruti Badrish, Andrew Chen, Alexander Tarnavski

Project Mentor: Nirvan Tyagi

*Denotes Equal Contribution

Background

Currently, around 94.1% of Internet users use it for communication [1]. While most messages are encrypted, oftentimes service providers have access to the message in plaintext, posing security and privacy risks. To address this, end-to-end encrypted messaging platforms (such as Signal or Whatsapp) emerged (Figure 1). These platforms enable senders to encrypt a message directly with the recipient's public key, ensuring that the server cannot decrypt the message.

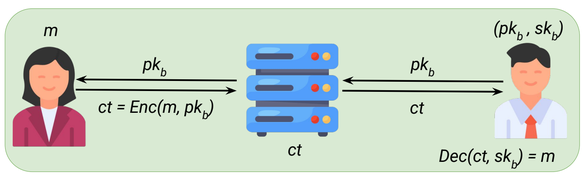

*Figure 1: Server cannot decrypt **ct** without sk_b*

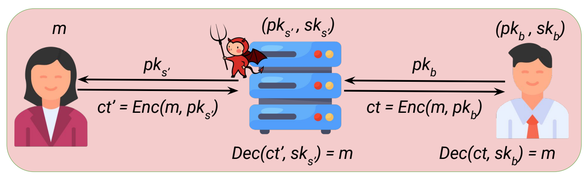

Figure 2: person-in-the-middle attack

Problem

Prior to communicating via end-to-end encrypted messaging services, both parties must have a copy of the recipient's public key. Typically, these keys are publicly hosted on a Public-Key Infrastructure (PKI), available for auditing upon request. However, PKIs may not always be truthful, and a malicious service provider may equivocate and provide compromised keys to perform a person-in-the-middle attack, as to eavesdrop on users (Figure 2).

Equivocation

In this scenario, an "equivocation" (Figure 3) occurs when a server presents different states of the PKI at timestep *i* to users. This allows the malicious server to give compromised keys to one (or both) users. The server "wins" if no users detect an equivocation happened.

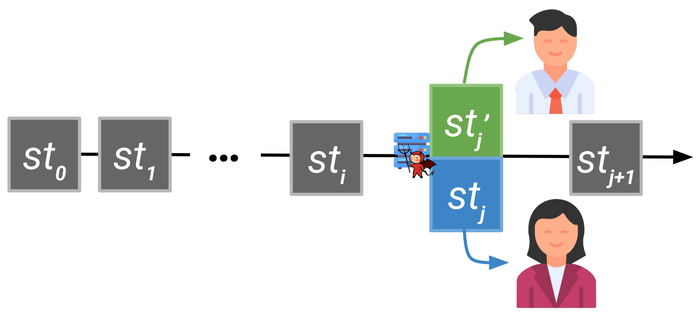

Figure 3: Server equivocation, malicious server provides 2 different states of the PKI for the same time j

Naive Peer-to-Peer Gossiping

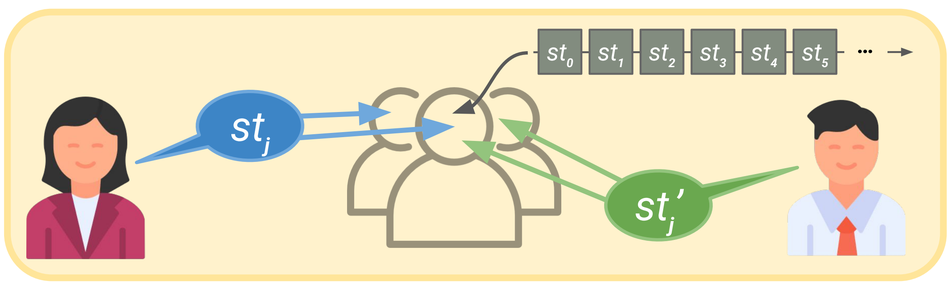

*Figure 4: Naive gossiping, Alice and Bob must gossip / store every state to ensure security. This is **expensive**.*

In the naive approach, users of the messaging platform externally communicate ("gossip") with each other to ensure no equivocation has occurred. In the worst case, users must gossip about every state, requiring maintenance of a linear list of previous epochs. This adds a lot of overhead to system usage due to large space and time complexities, and thus is impractical for real-world deployment.

Current Work

Vector Commitments (VC)

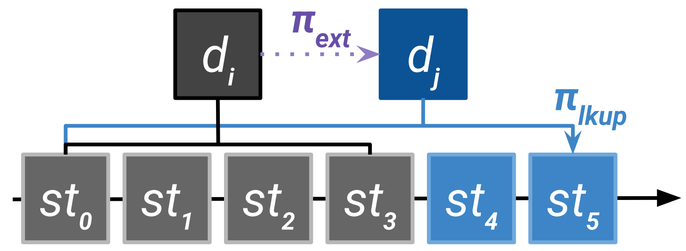

Figure 5: Append-Only Vector Commitment, d_i snapshots $st_{0\text{-}3}$ & d_j snapshots $st_{0\text{-}5}$

An *append-only vector commitment* (VC) is a cryptographic primitive that commits to an ordered list via a short digest *d*, and supports two proofs (Figure 5): a lookup proof π_{lkup} (a value sits at a position) and an extension proof π_{ext} (one digest's list is a prefix of another's).

Server-Aided Gossiping

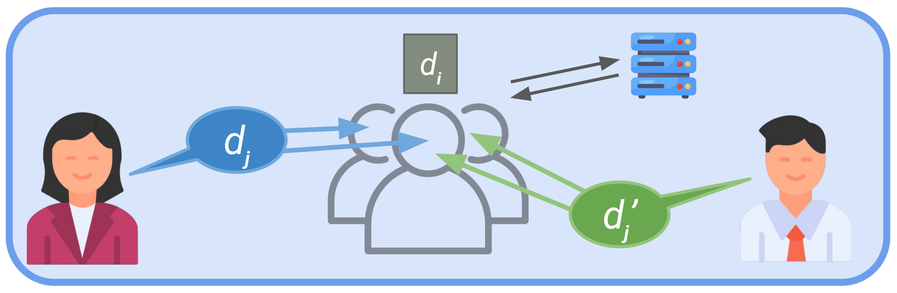

*Figure 6: Bulletin board, Alice and Bob only gossip / store lightweight digests. This is **efficient.***

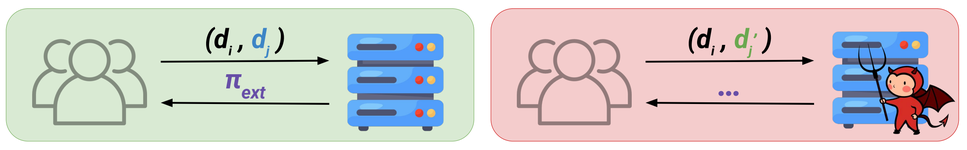

Figure 7: Gossiping protocol, users expect a valid proof for their values

Contributions

Security Definition

Informal security goal: Adversarial system should not be able to serve two different values to two users for the same position *i*.

Protocol: SAGP[VC]

Game $\text{BBPosVecConsistency}^{\mathcal{A}}_{\Pi}(\lambda)$:
$R_A, R_B \leftarrow [\cdot]$
$\ell_A, \ell_b, \ell_G \leftarrow 0$
$st_A \leftarrow \Pi.\text{UserInit}(1^\lambda)$
$st_B \leftarrow \Pi.\text{UserInit}(1^\lambda)$
$i \leftarrow \mathcal{A}^{\text{UpdateView}_X,\ \text{Read}_X,\ \text{Gossip}}(1^\lambda)$
Return $\wedge \begin{cases} R_A[i] \neq R_B[i] \\ \ell_G > i \end{cases}$

Oracle $\text{Gossip}(\pi)$:
If $\ell_A \leq \ell_B$:
  $b \leftarrow \Pi.\text{VerGossip}(\pi, st_A, st_B)$
Else:
  $b \leftarrow \Pi.\text{VerGossip}(\pi, st_B, st_A)$
If b **then** $\ell_G \leftarrow \min(\ell_A, \ell_B)$

Oracle $\text{UpdateView}_X(\ell', d, \pi)$:
Require $\ell' > \ell_X$
$(st_X, b) \leftarrow \Pi.\text{VerUpd}(\ell_X, \ell', d, \pi, st_X))$
If b **then** $\ell_X \leftarrow \ell'$

Oracle $\text{Read}_X(i, v, \pi)$:
Require $i \leq \ell_X$
$(st_X, b) \leftarrow \Pi.\text{VerRead}(i, v, \pi, st_X)$
If b **then** $R_X[i] \leftarrow v$

UserInit():
$(\_, d_{VC,0}) \leftarrow \text{VC.Init}()$
$st_{user} \leftarrow (d_{VC,0}, 0)$
Return st_{user}

VerUpdate$(\ell', d', \pi, st_{user})$:
$(d, \ell) \leftarrow st_{user}$
$b \leftarrow \text{VC.VerUpd}(d, d', \ell, \pi)$
If $b = 1$ **then** $st_{user} \leftarrow (d', \ell')$
Return (st_{user}, b)

VerRead(i, v, π, st_{user}):
$(d, \_) \leftarrow st_{user}$
$b \leftarrow \text{VC.VerLkup}(d, i, v, \pi)$
Return b

VerGossip$(\pi_{12}, st_{user1}, st_{user2})$:
$(d_1, \ell_1) \leftarrow st_{user1}$
$(d_2, \ell_2) \leftarrow st_{user2}$
$b \leftarrow \text{VC.VerUpd}(d_1, d_2, \ell_1, \pi_{12})$
Return b

ServerInit():
$(st_{VC,0}, d_{VC,0}) \leftarrow \text{VC.Init}()$
$st_{bb}, d_0 \leftarrow st_{VC,i}, d_{VC,0}$
Return (st_{bb}, d_0)

ServerPublish$(v, st_{bb,i})$:
$(d_{i+1}, st_{bb,i+1}) \leftarrow \text{VC.Upd}(|v| : st_{bb,i})$
Return $(d_{i+1}, st_{bb,i+1})$

ProveUpdate(ℓ, ℓ', st_{bb}):
$\pi \leftarrow \text{VC.ProveUpd}(\ell : st_{bb,\ell'})$
Return π

ProveRead(i, st_{bb}):
$(v, \pi) \leftarrow \text{VC.Lkup}(i : st_{bb})$
Return (v, π)

ProveGossip$(\ell_1, \ell_2, st_{bb})$:
$\pi \leftarrow \text{VC.ProveUpd}(\ell_1 : st_{bb,\ell_2})$
Return π

Next Steps

SAGP ensures that communicating parties both have a consistent view of the PKI, but provides no guarantee of the correctness of the content. Future steps involve composing SAGP's consistency layer with existing authentication systems such as CONIKS [3] and Merkle² [4], which verifies correctness, towards a single protocol that is provably secure, efficient, and deployable across all systems.

Citations

[1] Thuy D., "Share of internet users accessing online chat and messenger services monthly worldwide from 1st quarter 2022 to 2nd quarter 2025," *Statista*, May 2026. [Online]. Available: https://www.statista.com/statistics/1489440/chat-and-messenger-service-usage/. [Accessed: May 22, 2026]. [2] N. Tyagi, B. Fisch, A. Zitek-Estrada, J. Bonneau, and S. Tessaro, "VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries," Cryptology ePrint Archive, Paper 2021/627, 2021. doi: 10.1145/3548606.3560605. [Online]. Available: https://eprint.iacr.org/2021/627. [3] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: Bringing Key Transparency to End Users," *Cryptology ePrint Archive*, Paper 2014/1004, 2014. [Online]. Available: https://eprint.iacr.org/2014/1004. [4] Y. Hu, K. Hooshmand, H. Kalidhindi, S. J. Yang, and R. A. Popa, "Merkle²: A Low-Latency Transparency Log System," Cryptology ePrint Archive, Paper 2021/453, 2021. [Online]. Available: https://eprint.iacr.org/2021/453