Signal and Ready to MINGLE: In-Band Gossip for Key Transparency Split-View Detection in E2EE Messengers

Edona Fasllija[1,2], Lena Heimberger[2], and Kevin Paul[2]

[1] ASIT
[2] Graz University of Technology, Graz, Styria, Austria
{edona.fasllija, lena.heimberger}@tugraz.at

Abstract. End-to-end encrypted (E2EE) messengers such as Signal, WhatsApp and iMessage increasingly deploy Key Transparency (KT) to make malicious key substitution detectable. Yet KT only delivers its intended protection if users are anchored to the same global append-only KT history. A malicious operator can break this condition by *equivocating*, presenting incompatible views of the KT directory to different clients. Current deployments delegate detection to a small set of third-party auditors, creating a centralized trust bottleneck that can be pressured, compromised, or fail to audit continuously.

We ask whether clients can detect equivocation themselves, without dedicated infrastructure, simply by comparing KT state as they communicate. We introduce *MINGLE*, an opportunistic in-band gossip protocol for end-to-end encrypted messengers. MINGLE piggybacks compact KT commitments on a subset of ordinary messages before encryption, keeping gossip indistinguishable from regular application data while requiring no external services or overlay network. Rather than asking users to manually verify safety numbers or relying on a small set of auditors, MINGLE distributes the consistency check across the entire communication graph: an adversary wishing to sustain a split view must permanently isolate targeted clients from the rest of the network, preventing any cross-partition message from ever being delivered, a requirement that grows increasingly difficult to maintain covertly as the social graph densifies. MINGLE inherits the Trust-on-First-Use (TOFU) assumption standard in E2EE messengers: equivocation that begins at registration evades immediate detection, though the append-only log ensures it remains retroactively exposable once any cross-partition gossip event occurs. Using a temporal communication model, we show that under eventual cross-partition connectivity, conflicting KT views yield publicly verifiable evidence.

We prototype MINGLE in the Signal Android client using Signal's KT Server implementation, incurring a payload overhead of 119 bytes per gossip-carrying message without UI changes. Simulations under realistic messaging patterns show that MINGLE achieves high reliability and fast evidence generation without aggressive gossip flooding. MINGLE yields evidence of a targeted split view in a 12,000-client deployment within about 5 minutes when only 20% of clients participate and gossip is attached to roughly 5% of messages, suggesting that ordinary client communication can serve as a practical audit layer for KT.

1 Introduction

End-to-end encrypted (E2EE) messengers such as Signal and WhatsApp have become the primary communication channel for billions of users. At a high level, their security guarantee is straightforward: only the intended recipient should be able to read a message. In practice, however, this guarantee depends on more than the correctness of the underlying cryptographic protocol. It also depends on whether senders obtain the authentic public keys of their intended recipients. If the service provider can silently replace a client's public key with one it controls, it can mount meddler-in-the-middle (MitM) attacks even if the underlying cryptographic protocols are sound.

This threat is not theoretical; several recent legislative initiatives in the US, UK, and EU have sought to make encrypted services more amenable to scanning or "lawful access", and have prompted sustained concerns about their impact on the deployment of robust E2EE [3].

Key transparency (KT) systems [MBB+15,CDGM19,Bon16,Göb23,McM26] address this problem by committing identity-to-key bindings into an append-only, publicly auditable key directory. Users can obtain cryptographic proofs that their view of a contact's key is consistent with the committed state, and that their own account has not been rebound to an unauthorized key. However, these guarantees are only as strong as the *consistency* of the view presented to each client. Even with KT, a malicious operator may still *equivocate* by showing different users different, individually valid views of the directory.

The consistency problem The central challenge, then, is not merely to verify individual KT responses, but to establish that all clients are being shown a *globally consistent* directory. Existing approaches to this problem fall into the following broad categories: *Manual safety-number verification*, via out-of-band comparison of safety numbers or QR-code scanning [sig,wha26], is the only widely deployed mechanism that gives users direct, end-to-end evidence that they are communicating with the intended key. In practice, however, it is rarely used consistently, because it requires explicit user coordination and introduces well-documented social and cognitive friction [FK23]. *Bulletin-board approaches* publish KT commitments to a globally consistent medium such as a blockchain, making equivocation cryptographically impossible to sustain [Bon16], but at the cost of heavyweight external infrastructure. *Third-party auditors* monitor global log properties and report inconsistencies, as in Parakeet [MKKS+23], but this delegates the core consistency task to external entities and makes the guarantee depend on their honesty and sustained auditing effort. *Anonymous state lookups* [McM26,YGH+22] make targeted equivocation harder by obscuring which client requested which KT state. However, anonymity alone does not establish global consistency across clients.

[3] `https://www.internetsociety.org` (US EARN IT), `https://www.internetsociety.org/issues/encryption/` (UK/CS scanning), `https://edri.org` (EU CSAM / "chat control").

None of these approaches let users verify global consistency using evidence they already hold from regularly querying the KT server.

Gossip for consistency Gossip was first studied extensively in the context of Certificate Transparency (CT). An early IETF draft proposed gossip as a defense against split-view attacks [NGR18], and subsequent work showed that exchanging Signed Tree Heads and consistency proofs can expose inconsistent log views [CSP+15]. This line of work was later refined through formal analysis [OPR20], aggregation-based mechanisms [Dah18], and deployment strategies such as LogPicker [DKM+21]. Despite this progress, CT gossip saw little deployment in practice.

In Key Transparency (KT), the consistency problem has received growing attention, but the dominant approaches have not been in-band gossip in the style of CT. Instead, prior work has emphasized bulletin-board constructions [Bon16], third-party auditing [MKKS+23], and privacy-preserving or anonymous KT lookups [McM26,YGH+22]. These approaches make equivocation harder to sustain or easier to detect, but they do not exploit the existing client communication graph as the audit mechanism itself. The main exception is the client-auditing approach proposed in [YGH+22], which is the closest prior work to in-band gossip in the KT setting. Clients exchange auditing information derived from their KT views, but the mechanism remains tied to a CONIKS-style epoch model and requires explicit auditing exchanges at each epoch.

E2EE messengers impose a distinctive constraint that neither CT gossip nor prior KT auditing mechanisms directly address. The server being audited also mediates all client communication. Any consistency-checking mechanism that becomes externally recognizable therefore creates a new target for selective interference. In this setting, the challenge is not only to compare clients' views, but to do so *in-band*, as part of ordinary encrypted messaging traffic, with the attached KT state cryptographically bound to the delivered payload. Apple's deployment of in-band KT gossip in iMessage [SEA23] suggests piggybacking KT state on encrypted messages is practically viable at scale. However, without a public specification, the security properties of that design remain opaque. We provide the first formally analyzed, openly specified protocol for this approach.

Contributions We present *MINGLE*, a gossip protocol that turns ordinary encrypted conversations into a distributed KT audit mechanism. Rather than relying on new infrastructure or a small set of external auditors, MINGLE uses the fact that clients already accumulate locally verifiable KT state and ensures that this state crosses the communication edges needed to expose equivocation. To the best of our knowledge, this is the first work to formally analyze in-band KT gossip for E2EE messengers, to evaluate it under a measurement-grounded temporal communication model, and to realize it in Signal's actual client and KT stack. Our contributions are threefold:

– **Protocol design and security analysis.** We design MINGLE as an in-band KT consistency-gossip protocol that piggybacks compact KT state

on a subset of ordinary encrypted messages (Section 4). We formalize its operation in terms of client views, gossip payloads, and send/receive rules, and analyze its security in terms of correctness, soundness, detectability, and gossip payload binding (Sections 3.4 and 4.6).
– **Measurement-informed communication model and detection analysis.** We model the communication structure of an E2EE messenger as a temporal interaction graph and characterize detectability in terms of realized cross-partition gossip deliveries in the adopting subgraph (Section 3). To ground this model, we conduct a user study of E2EE messaging behavior and extract distributions for contact degree, clustering, burstiness, and activity windows relevant to gossip-based auditing (Section 5). We then instantiate the model with these data and evaluate detection time, overhead, and latency to equivocation evidence across a range of attack scenarios (Section 6).
– **Real-system prototype and deployment-oriented evaluation.** We integrate MINGLE into Signal's actual client and KT infrastructure by extending the Signal Android client together with Signal's open-source KT server, preserving the existing UI and interoperability with unmodified clients (Section 7). This shows that MINGLE can operate through Signal's real messaging infrastructure rather than a standalone prototype messenger. The resulting gossip payload is small enough to fit naturally within ordinary message delivery, adding only 119 bytes per gossip-carrying message in our prototype. Our simulations further show that, in a 12,000-client deployment with a targeted split view affecting 5% of accounts, MINGLE produces transferable evidence in approximately five minutes at 20% adoption (Section 6).

2 Key Transparency Model and Split-View Detection

We define a Key Transparency (KT) scheme as a tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Lookup}, \mathsf{VerifyLookup}, \mathsf{ProveConsistency}, \mathsf{VerifyConsistency}, \mathsf{VerifyCommit})$. We assume an identity provider $\mathcal{I}$ manages a directory $D_t : \mathcal{U} \to \mathcal{K}$ which evolves over discrete time epochs t, where $\mathcal{U}$ is the universe of client identifiers and $\mathcal{K}$ is the space of public keys.

– $(pk_{\mathcal{I}}, sk_{\mathcal{I}}) \leftarrow \mathsf{Setup}(1^{\lambda})$ generates $\mathcal{I}$'s long-term key pair.
– $H_t \leftarrow \mathsf{Commit}(D_t)$ maps the directory state $D_t : \mathcal{U} \to \mathcal{K}$ to a signed commitment $H_t = (t, C_t, \sigma_t)$ where C_t is a compact commitment to the directory state and $\sigma_t \leftarrow \mathsf{Sign}(sk_{\mathcal{I}}, t \,\|\, C_t)$.
– $(\mathsf{pk}_{\mathsf{id}}, \pi) \leftarrow \mathsf{Lookup}(D_t, \mathsf{id})$ returns the key $\mathsf{pk}_{\mathsf{id}} = D_t(\mathsf{id})$ together with a proof π that this binding is contained in the committed directory state.
– $\{0,1\} \leftarrow \mathsf{VerifyLookup}(H_t, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \pi)$ outputs 1 if π is a valid proof that $D_t(\mathsf{id}) = \mathsf{pk}_{\mathsf{id}}$ under the commitment H_t, and 0 otherwise.
– $\rho \leftarrow \mathsf{ProveConsistency}(H_i, H_j)$ generates a consistency proof ρ that D_j is an append-only extension of D_i.
– $\{0,1\} \leftarrow \mathsf{VerifyConsistency}(H_i, H_j, \rho)$ outputs 1 if ρ is a valid append-only consistency proof and 0 otherwise.

– $\{0,1\} \leftarrow \mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H_t)$ outputs 1 if $H_t = (t, C_t, \sigma_t)$ is a well-formed, operator-signed commitment.

In concrete systems, Commit is realized by a Merkle-tree-based authenticated dictionary managed by $\mathcal{I}$, and H_t is a signed tree root.

Client Views A client u maintains a local *view* of the KT system denoted by $V_u = (\mathcal{H}_u, \mathcal{Q}_u)$, where $\mathcal{H}_u$ is a set of verified commitments and $\mathcal{Q}_u$ is a set of verified query results, consisting of lookups and consistency proofs. We assume that clients populate these sets through regular KT queries: periodic commitment checks via $\mathsf{GetCommitment}()$, consistency queries via $\mathsf{GetConsistencyProof}(H_i, H_j)$ for previously observed commitments, self-monitoring via $\mathsf{Lookup}(D_t, u)$, and contact monitoring via $\mathsf{Lookup}(D_t, v)$ for a contact v. Accepted commitments are added to $\mathcal{H}_u$ and accepted lookup and consistency proofs to $\mathcal{Q}_u$. We define a view as *internally consistent* if for any $H_i, H_j \in \mathcal{H}_u$ with $i \leq j$ there exists a valid proof $\rho \in \mathcal{Q}_u$ such that $\mathsf{VerifyConsistency}(H_i, H_j, \rho) = 1$.

2.1 Adversary Model

We consider a setting in which clients are honest protocol participants and the KT operator $\mathcal{A}$ is malicious but cautious. Honest clients follow the KT lookup, verification, and gossip procedures, maintain local views containing only accepted commitments and query results, and do not collude with $\mathcal{A}$. They may nevertheless be offline, intermittently active, or run legacy software that does not participate in gossip. The adversary $\mathcal{A}$ has the following capabilities:

– **State Control.** $\mathcal{A}$ manages the directory state D_t and can sign arbitrary commitments with $\mathsf{sk}_{\mathcal{I}}$.
– **Network Control.** $\mathcal{A}$ controls all communication between clients and the KT service: it can drop, delay, reorder, and selectively deliver KT queries and responses. For client-to-client message delivery, $\mathcal{A}$ may also suppress or delay individual messages, subject to the *stealthiness requirement*: any pattern of interference that causes a perceptible denial of service to honest clients is considered a failure for $\mathcal{A}$.
– **Equivocation.** $\mathcal{A}$ may present conflicting views $V_u \neq V_v$ to distinct honest clients u, v, as depicted in Figure 3.

Definition 1 (Global consistency). *A KT execution is* globally consistent *if there exists a sequence of directory states $(D_t)_{t\geq 0}$ and signed commitments $(H_t)_{t\geq 0}$ such that for every honest client u with view $V_u = (\mathcal{H}_u, \mathcal{Q}_u)$:*

1. *every $H \in \mathcal{H}_u$ is a valid signed commitment, i.e.,* $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H) = 1$*, and H's epoch index matches its position in the global sequence; and*
2. *every lookup tuple $(\mathsf{id}, t, \mathsf{pk}, \pi) \in \mathcal{Q}_u$ satisfies $D_t(\mathsf{id}) = \mathsf{pk}$ and $\mathsf{VerifyLookup}(H_t, \mathsf{id}, \mathsf{pk}, \pi) = 1$.*

Definition 1 corresponds to the non-equivocation goal in previous work [Göb23,MKKS+23]. Global consistency does not guarantee correct key bindings, but *uniformity*: if the server misbehaves, all clients observe the same behaviour and can collectively detect it. A malicious operator violates global consistency by producing different but individually valid histories for different clients, a behavior called *equivocation* or a *split-view* attack.

Definition 2 (Equivocation). *$\mathcal{A}$ has equivocated if there exists a publicly verifiable* proof of misbehaviour *(PoM) consisting of either of the following forms:*

1. *Conflicting commitments, where two valid commitments $H_t \neq H'_t$ with the same epoch t exist with $C_t \neq C'_t$; or*
2. *Consistency violation: two valid commitments H_i, H_j with $i < j$ both signed by $\mathsf{sk}_{\mathcal{I}}$ such that no proof ρ satisfies $\mathsf{VerifyConsistency}(H_i, H_j, \rho) = 1$.*

Global consistency holds iff no PoM can be constructed from the union of all honest client views.

Remark 1 (PoM and Alarm behaviour). When a client constructs a valid PoM w, it invokes $\mathsf{alarm}(w)$ to signal the detected equivocation. We treat alarm as implementation-defined. Depending on the deployment, it might notify the user, log the evidence, or forward w to a transparency monitor. MINGLE's security guarantees are independent of this choice.

Ratcheted messaging and authenticated encryption Practical implementations of E2EE messengers, like Signal, WhatsApp, and iMessage, use the Double Ratchet algorithm [CCD+17,ACD19] as their message encryption layer. The Double Ratchet derives a fresh AEAD key for every message, providing forward secrecy and post-compromise security. An AEAD scheme simultaneously encrypts a plaintext for confidentiality and computes an authentication tag over the ciphertext, ensuring that any tampering is detectable at decryption time. A key property of ratcheted encryption is that message keys are chained: dropping message i causes the receiver's ratchet state to diverge from the sender's. To handle out-of-order delivery, practical implementations [PMS25] maintain a bounded cache of skipped message keys, allowing a receiver to decrypt messages that arrive late or out of order. However, this cache is finite: an adversary suppressing messages beyond the cache window causes decryption failures that are observable to honest receivers, constituting a denial of service inconsistent with $\mathcal{A}$'s stealthiness requirement. This means $\mathcal{A}$ cannot selectively suppress individual messages for established users without eventually causing observable failures at honest receivers.

3 Social Graph and Temporal Activity Model

Whether MINGLE can expose equivocation depends not only on the KT semantics, but also on the communication structure of the messenger. In an E2EE messenger, gossip does not propagate over a complete network. Instead, it is

constrained by the underlying *social graph* and the intermittent nature of by users' messaging behavior. We formalize this with a layered model consisting of a static social graph and a time-varying interaction graph, which together determine the surface over which split-view evidence can propagate.

3.1 Social and Temporal Structures

We model the system as a set of honest clients U and a malicious KT provider $\mathcal{I}$. The undirected *Static Social Graph*

$$G^{\mathsf{soc}} = (U, E^{\mathsf{soc}})$$

captures long-lived communication opportunities: an edge $\{u, v\} \in E^{\mathsf{soc}}$ denotes an established E2EE channel, such as contact lists or shared groups. Following empirical studies of communication networks [SL20,CBC+25], together with our user study, we assume a *heavy-tailed* social graph with dense clusters and short path lengths. Figure 1 shows an example G^{soc}.

Actual messaging behavior is dynamic and bursty. At each discrete time step $t \in \{0, 1, 2, \ldots\}$, a subset of clients $U_t \subseteq U$ is online. This induces a *Temporal Interaction Graph* consists of a sequence

$$\mathcal{G} = (G_t)_{t \geq 0}$$

where $G_t = (U_t, E_t)$, and E_t is the set of edges (application message events) that have occurred up to time t. Temporal edges arise only along channels already present in E^{soc}.

Group messages We treat group messages from u to a set of recipients $R = \{v_1, \ldots, v_k\}$ as a set of simultaneous directed edges $\{(u, v_i)\}_{v_i \in R} \subseteq E_t$. This high-fanout event allows u to disseminate to all online members of a social clique in a single time step.

3.2 Temporal Paths

Because the interaction graph $\mathcal{G}$ changes over time, gossip does not flow over static graph paths, but over time-respecting paths in the temporal graph.

Definition 3 (Temporal path). *A temporal path (or time-respecting path) from u to v is a sequence of edges*

$$\begin{aligned} &(u = w_0 \to w_1) \in E_{t_1}, \\ (w_1 \to w_2) \in E_{t_2},\ \ldots,\ &(w_{k-1} \to w_k = v) \in E_{t_k} \end{aligned}$$

with $t_1 < t_2 < \cdots < t_k$. v is temporally reachable *from u if there exists such a path.*

3.3 Equivocation as a Graph Partition Attack

We now characterize how $\mathcal{A}$ interferes with $\mathcal{G}$. When $\mathcal{A}$ equivocates (see Definition 2), it induces a partition of the honest clients according to the KT history they observe. Detectability then reduces to whether KT evidence can cross that partition in the temporal gossip graph. We now formalize the resulting structure.

Definition 4 (View Partition). *Let $\mathcal{A}$ equivocate at epoch e, producing the incompatible histories $\left(D_e^{(i)}, H_e^{(i)}\right)_{t\geq e}$ and $\left(D_e^{(j)}, H_e^{(j)}\right)_{t\geq e}$ that agree on epochs $0, \ldots, e-1$. A* view partition *is a pair of two non-empty sets $U^{(i)}, U^{(j)} \subseteq U$ of honest clients such that for each $u \in U^{(i)}$, all commitments and proofs in V_u are consistent with history $\left(D_t^{(i)}, H_t^{(i)}\right)$, and for each $u \in U^{(j)}$, all commitments and proofs in V_u are consistent with history $\left(D_t^{(j)}, H_t^{(j)}\right)$.*

Within a view partition, all honest clients see a mutually consistent sequence of directory states. Across partitions, their views are incompatible as shown in Figure 2.

Remark 2 (Fork Consistency.). Once $\mathcal{A}$ has forked the history, they must maintain the divergence indefinitely to avoid exposure. As soon as any client partition obtains a signed commitment or directory state from the other partition (or vice versa), the two partitions can be compared and turned into a publicly verifiable PoM.

By Remark 2, as soon as any client in $U^{(i)}$, who we will say is on *partition* i, obtains a commitment from Partition $j \neq i$, the two commitments together constitute a PoM as defined in Definition 2, since the two histories are incompatible from epoch e onwards and no valid consistency proof can bridge them. Thus, detectability of equivocation depends on the connectivity property of the temporal interaction graph:

Assumption Eventual Temporal Reachability. *For any two honest clients $u, v \in U$ and any time t_0, there exists some time $t \geq t_0$ such that a temporal path exists from u to v with non-zero probability.*

In terms of the temporal gossip graph $\mathcal{G} = (U, (E_t)_{t\geq 0})$, this means the adversary aims to ensure that no edge $(u, v) \in E_t$ with $u \in U^{(i)}$ and $v \in U^{(j)}$ is ever realised after the fork time t_f. The worst case for detection is that this partition is maintained indefinitely after t_f, the aggregated graph $G[t_f, \infty] = (U, \bigcup_{t\geq t_f} E_t)$ is disconnected along the cut $(U^{(i)}, U^{(j)})$. Avoiding eventual reachability as stated in Assumption 3.3 is an information-theoretically necessary condition for the adversary to evade detection: *no* in-band gossip protocol can detect equivocation if such a permanent partition exists. Our model therefore operates at a fundamental upper bound of detectability.

The assumption of eventual temporal reachability ensures that social boundaries do not induce such a permanent partition of the gossip graph. We treat a permanent blackout as a separate connectivity assumption (Assumption 3.3)

rather than a capability we restrict away, and rely on the empirical observation that real-world social networks are "small-world" and typically lack permanent, non-communicating partitions.

Cross-partition connectivity The converse is equally important: a *single* cross-partition gossip event suffices to trigger detection. In practice, only a fraction of clients may run MINGLE. We therefore state the relevant connectivity condition under which detection is guaranteed on the induced temporal gossip subgraph.

Definition 5 (Cross-partition temporal connectivity under partial adoption). *Let $\alpha \in (0, 1]$ be the adoption fraction and let $U^\alpha \subseteq U$ be the set of clients running MINGLE, selected independently and uniformly. The temporal gossip graph $\mathcal{G}^\alpha = (U^\alpha, (E_t^\alpha)_{t \geq 0})$ is the subgraph of $\mathcal{G}$ induced by U^α. The system is* cross-connected *after the fork if there exists a temporal path in $\mathcal{G}^\alpha$ from some $u \in U^{(i)} \cap U^\alpha$ to some $v \in U^{(j)} \cap U^\alpha$ such that*

$$u = w_0 \to w_1 \to \cdots \to w_k = v.$$

Corollary 1 (Adoption threshold for connectivity). *Under a heavy-tailed social graph with clustering coefficient c and mean degree d, the induced subgraph $\mathcal{G}^\alpha$ is connected with high probability when $\alpha > \alpha^*$, where α^* is the bond percolation threshold of G^{soc}. For small-world graphs empirically α^* is low, which is consistent with our simulation finding that $\alpha = 0.2$ suffices.*

Gossip Events and Group Messages We briefly note the role of group chats in establishing cross-partition connectivity. When client u sends a group message to recipients $\{v_1, \ldots, v_k\}$ at time t, this induces edges $(u, v_i) \in E_t$ for each i, yielding a high-fanout gossip event: u's KT summary can be delivered to all recipients in a single step. If even one recipient v_i lies in a different partition from u, the group message constitutes a cross-partition edge, and by Definition 5 the connectivity condition is satisfied. This makes group chats particularly valuable for detection: an adversary attempting to maintain a permanent partition must ensure not only that no direct 1-to-1 messages cross the cut, but also that no group contains members from both partitions. In practice this is a strong requirement, as it forces the adversary to anticipate and control the entire group membership structure of all targeted clients.

3.4 Security properties

We now state the security guarantees required of a gossip protocol over this communication model. A gossip protocol Π over a KT system should satisfy three properties: it should never raise false alarms when the system is honest (*correctness*); any alarm it raises should constitute a publicly verifiable proof of misbehavior (*soundness*); and, once gossip is attached to a message, it should remain cryptographically bound to the delivered payload so that $\mathcal{A}$ cannot strip or alter it while still delivering the message successfully (*gossip payload binding*).

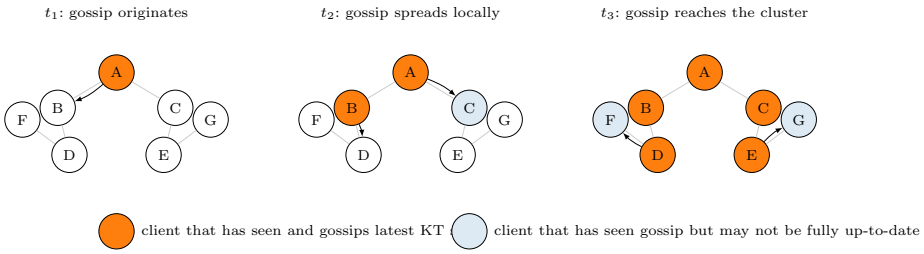


Fig. 1. Gossip about the KT state spreading through a local messaging neighbourhood over successive conversations. The light grey edges show the static social graph G^{soc}. Dark nodes have already received and gossiped the latest committed state; light nodes have partial information; white nodes have not yet been reached.

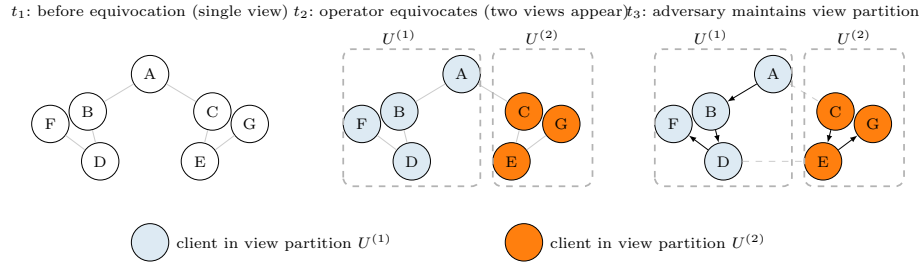


Fig. 2. Evolution of a split-view attack in the gossip graph. Initially (left), all clients share a single consistent KT view. After equivocation (middle), the KT operator assigns two incompatible histories to different subsets of clients, forming view partitions $U^{(1)}$ and $U^{(2)}$ on the same social graph (outlined boxes). If the adversary can prevent cross-partition communications that carry gossip (right), clients only exchange KT state within their own partition and never learn about the other view. Any dashed cross-edge, if realized by a conversation, would expose the split view.

The central liveness property is *equivocation detection*: if $\mathcal{A}$ equivocates, honest clients should eventually produce a PoM, provided the temporal interaction graph satisfies eventual reachability. We now formalize each in turn.

Remark 3 (Robustness against malicious clients). While our model assumes honest clients, the protocol is inherently robust against malicious ones. Since all gossip payload must be signed by $\mathsf{sk}_{\mathcal{I}}$ and pass VerifyConsistency, a malicious client cannot inject false equivocation alarms. At worst, a malicious client can refuse to gossip, which is functionally equivalent to an honest client being offline or running a legacy client.

Definition 6 (Correctness). *Protocol Π is* correct *if, when there exists a single append-only history $(D_t, H_t)_{t\geq 0}$ that explains all local views, meaning the KT system is globally consistent according to Definition 1, no honest client ever outputs* $\mathsf{alarm}(w)$.

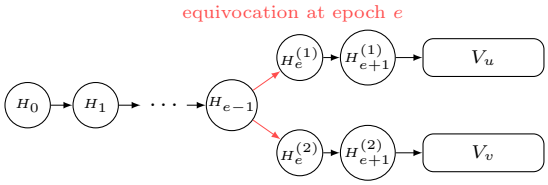


Fig. 3. Equivocation as two incompatible append-only histories. The operator maintains two partitiones that share a common prefix $H_0, \ldots, H_{e-1}$ but diverge at epoch e into distinct sequences $(H_t^{(1)})_{t \geq e}$ and $(H_t^{(2)})_{t \geq e}$.

Definition 7 (Soundness). *Protocol Π is* sound *if, whenever an honest client outputs $\mathsf{alarm}(w)$, w is a valid PoM in the sense of Definition 2, such that any party can verify w independently of Π using only $H^{(1)}$ and $H^{(2)}$.*

Definition 8 (Equivocation Detection). *Protocol Π* detects equivocation *if the following holds: Suppose $\mathcal{A}$ equivocates at epoch t, inducing a view partition $(U^{(i)}, U^{(j)})$ as in Definition 4. If the temporal interaction graph $\mathcal{G}$ satisfies eventual temporal reachability (Assumption 3.3), then with overwhelming probability there exists a time t^* and an honest client $u \in U^{(i)} \cup U^{(j)}$ such that u outputs $\mathsf{alarm}(w)$ at time t^*.*

Definition 9 (Gossip Payload Binding). *Let $\varphi(m, g)$ be the embedding function that attaches gossip g to an application message m before authenticated encryption. Gossip payload binding is the direct consequence of treating $\varphi(m, g)$ as an atomic plaintext under the message-layer AEAD scheme [CCD+17,ACD19]: We say that Π satisfies* gossip payload binding *if for every PPT adversary $\mathcal{A}$, every message m, and every gossip payload g, the following holds: if a receiver accepts a ciphertext derived from an encryption of $\varphi(m, g)$, then the receiver recovers exactly $\varphi(m, g)$. In particular, no adversary can strip or modify g while still causing the receiver to accept delivery of m.*

Remark 4 (Distributional indistinguishability).

MINGLE does not require perfect indistinguishability of gossip-carrying messages: gossip is embedded inside the same encrypted payload as the application data, so its contents are protected by the message-layer authenticated encryption. The remaining objective is traffic-level covertness: attached gossip should not create conspicuous deviations in the surrounding traffic pattern. Implementations may split the payload across messages, pad to a fixed size, or apply any other traffic-shaping strategy. MINGLE's security properties hold regardless, provided each fragment is embedded in the authenticated plaintext before encryption. The only formal suppression-resistance property we require is gossip payload binding.

4 The Protocol

Building on the social and temporal communication model from Section 3, we now describe the MINGLE protocol that runs on top of it. MINGLE is an in-band gossip protocol that runs on top of ordinary application-message events in the temporal interaction graph. For each event $(u, v) \in E_t$, client u attaches a compact summary of its current KT state with probability $p \in (0, 1]$. Client v verifies the received summary against its own local state, updates its view if the two are consistent, and outputs a publicly verifiable proof of misbehavior if they are not.

The protocol therefore specifies (i) protocol setup, (ii) which pieces of KT state are piggybacked on messages, and (iii) how recipients incorporate this state into their local views and, if necessary, detect equivocation. The protocol is *incremental*: rather than auditing every epoch snapshot of the directory, clients maintain a compact summary of their current view and use append-only consistency proofs to check that newer commitments extend older ones. This avoids epoch-by-epoch verification (as in early designs that rebuilt a fresh tree each epoch [MBB+15,YGH+22]) and instead lets clients focus on their own monitoring and lookups and consistency checks.

4.1 Setup

MINGLE operates under *Trust on First Use* (TOFU): a client u registering for the first time has no prior commitment against which to verify the operator's initial response.

Assumption TOFU. *Each honest client u accepts the first signed commitment H_0 it receives from $\mathcal{A}$ as its initial trusted state, setting $H_u^{(\max)} \leftarrow H_0$ without independent verification.*

TOFU is a standard assumption in E2EE messengers, since a first-time client has no out-of-band reference to check against. It allows $\mathcal{A}$ to assign u to a pre-existing fork partition at registration, or open a fresh fork, without immediate detection, including installing a forged key binding to mount a MitM from the outset. MINGLE does not eliminate this risk, which is an inherent bound for any in-band gossip protocol (Section 3.3).

Remark 5 (TOFU and detection). TOFU is a genuine limitation: a patient adversary who forks the history at or before a client's registration can assign that client to a forked partition from the outset, including installing a forged key binding to mount a MitM attack, without triggering any immediate alarm. MINGLE guarantees *eventual retroactive detection*. Equivocation at any epoch produces two operator-signed commitments for the same epoch, which constitute a PoM (Definition 2). Since the append-only log permanently embeds the original fork, $\mathcal{A}$ cannot silently repair a forged key binding after the fact. A cross-partition gossip event therefore allows any client to construct this PoM retroactively, even for forks that predate their registration. Permanent isolation, where $\mathcal{A}$ ensures

the affected client never exchanges gossip with a client from the other partition, is the only scenario in which TOFU-time equivocation evades detection permanently.

4.2 Client State and Gossip Payload

Each client u maintains a local view $V_u = (\mathcal{H}_u, \mathcal{Q}_u)$ from which gossip selects a minimal summary:

- a *distinguished commitment* $H_u^{(\max)} \in \mathcal{H}_u$, the latest KT state u has verified; and
- optionally, a bounded set of consistency proofs from $\mathcal{Q}_u$ certifying how $H_u^{(\max)}$ was reached from earlier states.

From the local view $V_u = (\mathcal{H}_u, \mathcal{Q}_u)$, client u derives a compact gossip payload

$$\gamma_u = \mathsf{EncodeGossip}(H_u^{(\max)}, \rho),$$

where $\rho \in \mathcal{Q}_u$ is an optional consistency proof.

4.3 Protocol variants

We consider two minimal instantiations of our in-band consistency gossip protocol. These variants differ only in how much work they offload from the KT service to the gossip channel.

- **Commitments-only gossip.** Each payload γ_u contains only the sender's distinguished commitment $H_u^{(\max)}$. Whenever a receiver v observes a newer commitment from a peer, it must query the KT operator for a corresponding consistency proof before advancing its own view. This minimizes payload size but maximizes KT queries.
- **Commitments+proof gossip.** Each payload γ_u additionally carries a recently verified consistency proof from the sender's local view. Receivers can often validate and adopt newer commitments using only gossiped proofs, falling back to KT queries only when no suitable proof is available. This reduces server load at the cost of larger payloads.

Appendix B gives pseudocode for both variants (Algorithm 1 and Algorithm 2) using the KT API and view notation of Section 2.

4.4 Send-side Behavior

Whenever the temporal communication model produces an edge $(u \to v) \in E_t$, client u attaches a gossip payload to the outgoing message with probability $p \in (0, 1]$; otherwise, the message is sent unchanged. If gossip is attached, the protocol proceeds as follows:

1. u refreshes its KT state if necessary by obtaining a fresh signed commitment $H_t \leftarrow \mathsf{GetCommitment}()$.
   If $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H_t) = 1$, then:
   - if u has no prior distinguished commitment, it initializes $H_u^{(\max)} \leftarrow H_t$ under the TOFU assumption (Section 4.1);
   - otherwise, it queries $\rho \leftarrow \mathsf{GetConsistencyProof}(H_u^{(\max)}, H_t)$ and, if $\mathsf{VerifyConsistency}(H_u^{(\max)}, H_t, \rho) = 1$, updates $H_u^{(\max)} \leftarrow H_t$ and records ρ in $\mathcal{Q}_u$.
2. u computes the gossip payload

$$\gamma_u = \mathsf{EncodeGossip}(H_u^{(\max)}, \rho),$$

   where ρ is either the latest proof binding the previous distinguished commitment to $H_u^{(\max)}$ or $\perp$ if no such proof is available. (see Section 4.2).
3. u embeds γ_u into the outgoing message m via $\varphi(m, \gamma_u)$, placing it as an opaque field in the authenticated plaintext before encryption. By the gossip payload binding property (Definition 9), any delivered ciphertext preserves this embedded gossip; attempts to remove or alter it cause the ciphertext to be rejected.

4.5 Receive-side behavior and local update

On the receiving side, when v receives a message from u at time t (i.e., the edge $(u \to v) \in E_t$ is realized), the gossip protocol runs the following logic:

1. v decrypts the message as usual and extracts the gossip payload γ_u, if present. If no payload is present, v proceeds with normal message processing and skips the remaining steps.
2. If a payload is present, v parses it as

$$(H, \rho) \leftarrow \mathsf{DecodeGossip}(\gamma_u),$$

   , where H is the sender's distinguished commitment and ρ is an optional consistency proof. The commitment H is accepted only if $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H) = 1$.
3. v compares the received commitment H to its own distinguished commitment $H_v^{(\max)}$:
   - If $H = H_v^{(\max)}$, no further action is needed.
   - If H and $H_v^{(\max)}$ refer to the same epoch or tree size but have different roots, then v has immediate evidence of equivocation. It constructs $w = (H_v^{(\max)}, H, \mathsf{ctx})$ and outputs $\mathsf{alarm}(w)$.
   - If H is more recent than $H_v^{(\max)}$, then v attempts to justify the transition from $H_v^{(\max)}$ to H. In the commitments+proof variant, it first tries the gossiped proof ρ. If the gossiped proof ρ is valid and sufficient to justify

the transition, v records ρ in $\mathcal{Q}_v$ and updates $H_v^{(\max)} \leftarrow H$ without querying the KT service. If no suitable proof is present, it queries

$$\rho' \leftarrow \mathsf{GetConsistencyProof}(H_v^{(\max)}, H).$$

If $\mathsf{VerifyConsistency}(H_v^{(\max)}, H, \rho') = 1$, then v updates $H_v^{(\max)} \leftarrow H$ and records ρ' in $\mathcal{Q}_v$; Otherwise, v records the failed or missing consistency attempt in ctx, together with the queried pair $(H_v^{(\max)}, H)$, and defers alarm generation until it later obtains a constructive PoM.

– If H is older than $H_v^{(\max)}$, v stores H in $\mathcal{H}_v$ and checks consistency in the reverse direction via $\mathsf{GetConsistencyProof}(H, H_v^{(\max)})$, retaining the result in $\mathcal{Q}_v$ for equivocation detection.

Proofs of equivocation A *proof of equivocation* is a finite, publicly verifiable object $w = (H^{(1)}, H^{(2)}, \mathsf{ctx})$, where $H^{(1)}, H^{(2)} \in \mathcal{H}$ are signed commitments and ctx is optional auxiliary data (e.g., failed consistency queries). w is a PoM as defined in Definition 2 if $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H^{(1)}) = \mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H^{(2)}) = 1$ and either:

1. $H^{(1)}$ and $H^{(2)}$ refer to the same epoch or tree size but have different roots, or
2. no valid append-only consistency proof exists between $H^{(1)}$ and $H^{(2)}$.

Remark 6 (Proof withholding is not equivocation). If $\mathcal{A}$ simply refuses to respond to $\mathsf{GetConsistencyProof}(\cdot)$ queries, this constitutes a denial of service but not equivocation under Definition 1. A PoM requires positive evidence of incompatibility, not merely the absence of a proof.

Remark 7 (Auxiliary context). Failed or missing consistency responses are retained only as auxiliary context. They do not by themselves constitute a PoM of equivocation, but they record which commitment pairs could not be reconciled, guide later retries, and may be attached to a subsequently obtained PoM for debugging or forensic purposes. Third-party verification of the PoM still rests only on the signed commitments themselves.

Remark 8 (Incrementality and relation to the temporal model). The protocol in Section 4.5 is intentionally lightweight: clients never need to download or verify the entire directory history. Instead, they maintain a small local state $(\mathcal{H}_u, \mathcal{Q}_u)$, dominated by a single distinguished commitment $H_u^{(\max)}$ and (optionally) a few proofs, and opportunistically exchange this state along the edges of the temporal communication graph $\mathcal{G}$.

4.6 Security Analysis

We now analyse the security properties of MINGLE according to Section 3.4. We first argue that both variants (commitment-only and commitment+proof gossip) are *correct* according to Definition 6 and *sound* according to Definition 7, and then give conditions under which they are *detecting* according to Definition 8.

Theorem 1 (MINGLE is correct). *MINGLE satisfies Definition 6: If the KT operator is globally consistent (Definition 1), then no honest client running either variant of MINGLE (Algorithm 1 and Algorithm 2) ever outputs* $\mathsf{alarm}(w)$.

Proof. Suppose the operator is globally consistent, i.e., there exists a single append-only history $(D_t, H_t)_{t\geq 0}$ such that every GetCommitment and GetConsistencyProof invocation by an honest client returns values consistent with this history.

By induction on protocol execution, every commitment $H \in \mathcal{H}_u$ accepted by an honest client u is a signed H_t from the unique history, and every proof $\rho \in \mathcal{Q}_u$ verifies against it. This holds initially by TOFU (Assumption 4.1) and is maintained inductively because clients re-verify all gossiped items before accepting them (Section 4.5).

It follows that neither PoM form of Definition 2 is possible. For the first form, uniqueness of the global history rules out two valid commitments for the same epoch or tree size with different roots. For the second form, any two valid commitments H_i, H_j with $i < j$ lie on the same append-only history, and therefore there exists a proof ρ such that $\mathsf{VerifyConsistency}(H_i, H_j, \rho) = 1$. Hence no honest client ever outputs $\mathsf{alarm}(w)$, and MINGLE satisfies Definition 6. The argument is identical for both variants: MINGLE with proof gossip (Algorithm 2) only adds additional proofs to the payload, introducing no new alarm conditions.

Theorem 2 (MINGLE is sound). *MINGLE satisfies Definition 7: Whenever an honest client running either variant of MINGLE (Algorithm 1 and Algorithm 2) outputs* $\mathsf{alarm}(w)$*, w is a valid PoM according to Definition 2. Any party can verify w independently of Π.*

Proof. In both Algorithm 1 and Algorithm 2, an honest client outputs $\mathsf{alarm}(w)$ only after obtaining two valid commitments $H^{(1)}$ and $H^{(2)}$ for the same epoch or tree size with different roots. In that case, the protocol constructs

$$w = (H^{(1)}, H^{(2)}, \mathsf{ctx}),$$

where ctx is auxiliary context accumulated during prior reconciliation attempts. This is a valid PoM in the sense of Definition 2. Since both commitments are accepted only after passing VerifyCommit $(\mathsf{pk}_{\mathcal{I}}, \cdot) = 1$, any third party can verify w independently of Π using the two commitments alone; the context ctx is auxiliary and not part of the verifiable claim. Hence every emitted alarm carries a valid, publicly verifiable PoM, and MINGLE is sound in the sense of Definition 7.

Gossip payload binding and suppression resistance Before establishing detectability, we address a prerequisite specific to the E2EE setting: even if equivocation has occurred and a cross-partition temporal path exists, detection can proceed only if gossip-bearing messages are allowed to traverse that path. In a centralized messenger architecture, $\mathcal{A}$ mediates all client-to-server communication and might therefore try to interfere with messages carrying KT state.

The relevant guarantee is not perfect indistinguishability with respect to all observable metadata, but *gossip payload binding*: once gossip is embedded into the authenticated encrypted payload, it is cryptographically bound to the message and cannot be stripped or altered without causing receiver-side authentication failure. This is formalized in Definition 9 and established in Proposition 1. Combined with the ratcheted-messaging argument from Section 2.1, it follows that a stealthy $\mathcal{A}$ cannot suppress gossip on a realized communication edge without suppressing the entire message and eventually causing observable failure unless the client is permanently isolated (Section 4.1).

Accordingly, the detectability analysis can treat realized cross-partition communication events as usable gossip opportunities.

Proposition 1 (MINGLE satisfies gossip payload binding). *If the underlying message layer provides authenticated encryption with ciphertext integrity, then MINGLE satisfies Definition 9. This follows directly from treating $\varphi(m, g)$ as an atomic plaintext under the AEAD scheme: Removing or stripping g causes the ciphertext to fail the receiver-side verification, since the authentication tag covers $\varphi(m, g)$ in its entirety.*

Corollary 2 (Selective gossip suppression requires message suppression). *Under Definition 9, a delivered message carrying gossip delivers that gossip intact. Hence, to prevent gossip from traversing a realized communication edge, $\mathcal{A}$ must suppress the entire message. In a ratcheted messenger, sustained selective message suppression eventually causes observable failures at honest receivers and therefore violates the stealthiness requirement of Section 2.1. Thus, if a cross-partition communication event occurs, a stealthy $\mathcal{A}$ cannot selectively suppress the corresponding gossip without becoming observable.*

Note that this applies to established sessions: at registration, the ratchet is not yet initialized, and $\mathcal{A}$ may suppress the initial gossip exchange without observable consequence. This is precisely the TOFU window identified in Section 4.1 and discussed in Remark 5.

Detectability Correctness and soundness are safety properties: they concern only local client behavior and follow from inspection of the protocol algorithms. Detectability is a *liveness* property: evidence of equivocation must not only exist somewhere in the system, but must eventually reach an honest client in a form that triggers an alarm. Under the current MINGLE variants, alarms are raised constructively from *conflicting commitments* for the same epoch or tree size.

The detectability argument therefore has two ingredients. First, clients perform regular commitment checks as part of their ordinary KT usage, so each partition continues to accumulate fresh commitments from its own forked history. Second, once a cross-partition communication event is realized, Corollary 2 ensures that attached gossip cannot be selectively stripped while preserving stealth. Thus, if a cross-partition edge is realized at a time when its endpoints hold post-fork commitments for the same epoch or tree size, delivery of one partition's commitment to the other immediately yields a constructive PoM.

We therefore first isolate the deterministic core of the argument: a single realized cross-partition gossip delivery suffices to trigger detection, provided the sender and receiver hold post-fork commitments for the same epoch or tree size. We then relate this condition back to the temporal communication model and partial adoption assumptions. This is captured by the following lemma.

Lemma 1 (Exposure on a cross-partition delivery). *Suppose $\mathcal{A}$ equivocates from epoch e onward, inducing view partitions $(U^{(1)}, U^{(2)})$ as in Definition 4. Let $(u \to v) \in E_t$ be a realized cross-partition communication event with $x \in U^{(1)}$ and $y \in U^{(2)}$. Assume that at the time of delivery, v gossips a distinguished commitment $H_r^{(1)}$ with $r \geq e$, while v's local distinguished commitment is $H_r^{(2)}$ for the same epoch $r \geq e$. Then, under either protocol variant (Algorithm 1 and Algorithm 2), client y outputs $\mathsf{alarm}(w)$ for a valid PoM w.*

Proof. See Appendix A.

We can now lift this local argument to the partially deployed system.

Theorem 3 (Detectability under partial adoption). *Suppose $\mathcal{A}$ equivocates from epoch e onward, inducing a view partition $(U^{(1)}, U^{(2)})$. Let $\alpha > \alpha^*$, and assume the temporal gossip graph $\mathcal{G}^\alpha$ is cross-connected after the fork in the sense of Definition 5. If a realized cross-partition gossip event occurs at a time when its sender and receiver hold post-fork distinguished commitments for the same epoch or tree size, then both protocol variants satisfy Definition 8: the receiver outputs $\mathsf{alarm}(w)$ for a valid PoM w.*

Proof. See Appendix A.

5 Simulating User Behaviour

Evaluating MINGLE requires a workload model that reflects the concrete communication opportunities through which KT evidence can cross view partitions. As prior gossip evaluations [CSP+15] in other settings were grounded in observed traffic traces rather than purely synthetic workloads, we similarly seek to anchor our evaluation in messenger-specific empirical data. In this sense, the study is part of the security contribution of the paper: it provides the empirical bridge between the formal detectability condition and a realistic messenger workload. In particular, our simulations need messenger-specific data on interaction degree, group composition, and temporal activity windows. These are the quantities that govern whether and how quickly cross-partition communication occurs. Existing datasets on instant messenger behavior do not simultaneously capture the four dimensions our simulation requires: interaction degree, group composition, temporal activity windows, and multi-device patterns. The most closely related works were conducted for Contact Discovery in 2019 [KRS+19] or focused on social network structure rather than communication timing [CBC+25,SL20]. Since we require fine-grained per-user daily contact counts and activity spans needed to calibrate our temporal gossip model, we conducted a purpose-built survey.

We focused on three key metrics: interaction neighborhood scale (unique peers per day/week), group chat statistics, and temporal activity patterns. As shown in Table 2, we model each client by sampling their contact degree from the weekly distinct 1-1 partner distribution (D_d), activity windows from reported first/last message times (D_{start}, D_{end}, D_{span}), and group sizes (D_g) around a configurable mean. This captures the skewed interaction patterns typical of secure messengers, where users repeatedly message a small contact subset and communicate in dense groups with occasional bridging interactions, which is the precise structure that determines whether cross-partition checks can detect targeted split-view attacks.

6 Evaluation

We simulate communication over social topologies and temporal messaging patterns derived from the user study in Section 5, measuring detection probability, time to evidence, and communication overhead across four scenarios: a baseline static-equivocation regime, a mixing sensitivity study, boundary conditions under weak mixing or suppression, and a protocol-variant cost comparison. All simulations ran in Python on a standard developer laptop (Lenovo ThinkPad T14s Gen 3, i7-1260P).

6.1 Simulation model

We set up the simulation as follows: Each run is parameterized by a client population n, a static social graph G^{soc}, a temporal activity model, and an attack scenario. At each time step t, the simulator samples communication events E_t over online clients $V_t \subseteq U$, executes one round of the gossip protocol per edge $(u, v) \in E_t$, and records any PoM constructed. We model messaging as a mixture of 1–1 and group communication. Each client has a contact set with expected size `avg_contacts`, and, in fork scenarios, cross-partition 1–1 contacts are drawn with probability `p_contact_cross`. This cross-partition traffic determines how quickly gossip can expose inconsistent log views. Group chats of expected size `avg_group_size` are generated with cross-partition membership probability `p_group_cross`, which controls the communication graph together with `p_contact_cross`.

Time advances in discrete steps of length $\Delta =$ `dt_min` minutes over a simulation horizon of `hours`. To reflect the user study's reported diurnal usage and online/offline behavior, each client alternates between *active* and *inactive* periods. Within each step, *active* clients generate 1–1 and group messages according to configurable rates (`msg_rate_1to1`, `msg_rate_group`), while inactive clients do not send.

Gossip protocol parameters A fraction `adoption` of clients participate in KT gossip and attach a gossip payload with probability `p_gossip`. The adversarial server matches the threat model of Section 2.1: it may equivocate across client

subsets and withhold consistency responses, but cannot compromise clients or fabricate valid consistency evidence.

This induces a temporal communication graph $\mathcal{G} = (U, (E_t)_{t=1}^{T})$ that determines opportunities for in-band dissemination and cross-checking of KT log commitments.

The protocol itself is parameterized by two main knobs: the *adoption fraction* and the gossip attachment probability `p_gossip`. The adversary is parameterized by the targeted victim fraction $f =$ `fork_frac`. Finally, censorship is modeled by
$p_{\text{drop}} =$ `p_drop_crosscut`, the probability that a cross-partition delivery is suppressed (while within-partition traffic continues). The full sweep ranges are listed in Appendix C.

Outcome & Cost Metrics We report three outcome and one cost metric. *Detection probability* is the fraction of runs that produce *evidence* of misbehavior (`proof_inconsistent` > 0); this is a *strict* notion of "detection" that does not count server being unresponsive to proof queries. We call a scenario configuration *reliable* if it produces cryptographic evidence in at least 95% of runs (det≥ 0.95). *Evidence latency* is the time (minutes after fork onset) until the first such evidence appears. *Overhead* is mean KT gossip bytes per delivered message, i.e., the bandwidth tax on normal traffic.

6.2 Simulation scenarios and key findings

The simulation suite can be understood as varying three high-level dimensions: server behavior, effective connectivity conditions (normal, weakly mixed, or actively suppressed), and protocol effort (gossip probability, adoption, and payload). We first evaluate MINGLE in a *typical deployment regime* that reflects everyday usage: a large client population, moderate cross-cluster mixing, and partial adoption. We then run three complementary *stress/sensitivity* scenarios that isolate individual bottlenecks for split-view detection: (i) reduced social mixing (*weaker cross-partition connectivity*), (ii) narrower targeting (*smaller victim sets*), and (iii) selective suppression of cross-partition messages (*censorship*).

BASELINE We begin with a representative operating point intended to capture *typical* messenger conditions. produce transferable evidence under moderate mixing and partial deployment. We simulate a deployment with $N = 12{,}000$ clients and a targeted split view affecting a fraction $f = 0.05$ of users. A fixed victim cohort of size f is selected uniformly at random and shown the forked KT view throughout the attack window, while the remaining users see the honest view. Communication follows the moderate-mixing workload derived from the user-study model. We sweep the participating-client fraction $a \in \{0.05, 0.1, 0.2, 0.3, 0.6\}$ and the gossip attachment probability $p_{gossip} \in [0.005, 0.05]$ to quantify how quickly and how reliably clients obtain cryptographic evidence in this typical regime.

This configuration serves as our primary operational baseline for practical parameter choices; the remaining scenarios in Section 6.2 stress-test the mechanism under harsher or more adversarial conditions. Figure 4 summarizes the key trade-offs.

Detection Probability. The top panel plots the probability that equivocation is detected as a function of p_{gossip} for several adoption levels. All curves are monotone: increasing gossip makes detection more likely. The more informative pattern, however, is that broader deployment shifts the curves sharply to the left. In this baseline regime, dependable detection does not require aggressive gossip flooding; wider adoption substantially lowers the gossip probability needed to reach high reliability.

Latency–overhead. The middle panel identifies the latency–overhead frontier. For configurations that reach our reliability target (detection ≥ 0.95), the time to evidence drops quickly once we move away from very low overhead, and then flattens out. This is encouraging for deployment: reliable and fast split-view detection is already achieved at moderate overhead. For example, at $a \approx 0.2$ and $p_{\text{gossip}} \approx 0.05$, the median time to first evidence is about 5 minutes.

Gossip–Adoption As adoption increases, the minimum p_{gossip} needed to reach detection ≥ 0.95 goes down, so wider deployment can substitute for heavier per-message gossip. Even modest adoption already cuts the required gossip rate and thus the per-message overhead for the same reliability. Increasing adoption from 5% to 20% already reduces the required gossip probability by roughly a factor of six, while increasing adoption to 60% reduces it by about a factor of twenty-five relative to the 5% case. Higher adoption enlarges the deployed comparison subgraph, while higher p_{gossip} increases the probability that contradictory summaries are actually carried across its edges.

TRADEOFF We next consider a more adverse operating regime in which both the number of contradictory carriers and the opportunities for cross-partition contact are reduced. Concretely, we combine a small victim cohort ($f = 0.02$) with weak cross-partition mixing ($\texttt{p_contact_cross} = \texttt{p_group_cross} = 0.005$). Relative to the baseline, this makes fork exposure harder in two ways: fewer clients carry the forked KT view, and the communication graph offers fewer realized bridge events across the partition. The goal of this scenario is therefore to characterize the reliability, latency, and overhead trade-off when both the contradictory state and the communication opportunities needed to expose it are scarce.

We sweep the gossip attachment probability p_{gossip} and the adoption fraction α as summarized in Figure 5. The resulting trade-off is clear: increasing p_{gossip}, or increasing adoption, improves detection reliability and typically lowers evidence latency, but at the cost of higher gossip overhead.

Detection probability. The top panel shows that even in this harder regime, detection probability improves steadily with p_{gossip} for all adoption levels. Higher adoption shifts the curves upward and to the left, indicating that wider deployment reduces the gossip rate needed to achieve a given level of reliability. Once the curves approach saturation near 1, additional gossip yields

diminishing returns, since the event of interest is simply that transferable evidence is obtained at least once.

Latency. The middle panel reports time to first evidence, conditional on detection. As expected, increasing p_{gossip} generally reduces latency by increasing the rate at which cross-partition comparisons are attempted. The trend is noisier than in the baseline regime: Since forked carriers and cross-partition bridge opportunities are both scarce, the timing of the first successful exposure is less uniform across runs.

Overhead. The bottom panel shows the expected cost of this improvement. Gossip overhead increases with both adoption and p_{gossip}, approximately in proportion to how many senders participate and how frequently they attach gossip. The figure therefore makes the main trade-off explicit: stronger reliability and shorter time to evidence remain achievable even in a low-mixing, targeted-victim regime, but only by spending more communication budget. For completeness, representative numerical values underlying this trade-off are reported in the appendix (Appendix C.1).

TARGET We next isolate the effect of the targeted victim fraction f, i.e., the size of the forked partition. Smaller victim sets are harder to detect because they create fewer cross-partition communication opportunities and therefore fewer chances for contradictory KT views to meet. Figure 6 varies $f = \texttt{fork_frac}$ at fixed adoption and plots detection probability and evidence latency for several gossip rates p_{gossip}.

Detection probability: Smaller targeted cohorts are harder to catch: fewer victims means fewer cross-boundary conversations involving a victim, so fewer chances to observe inconsistent commitments. Detection probability increases with f, and higher p_{gossip} lifts the curve upward. If the setting already guarantees enough victim-non-victim interactions for most f values, the detection probability saturates quickly and the plot becomes less discriminating.

Latency: As f grows, evidence appears much faster for all p_{gossip}, and the curves compress toward low latency. Higher gossip rates further accelerate detection across the entire range. Overall, Figure 6 shows that narrowly targeted split views are indeed harder to catch, but remain practically detectable in our setting: the main penalty is increased latency at low p_{gossip}, not a collapse in detection probability.

CENSOR We finally consider a cross-partition suppression scenario in which deliveries across the cut are dropped with probability p_{drop}, as shown in Figure 7. We vary the probability p_{drop} that cross-partition deliveries are suppressed as shown in Figure 7. Detection degrades as cross-partition messages are dropped, and latency increases sharply, confirming that split-view detection is fundamentally bottlenecked by rare cross-partition interactions.

Detection Probability: The top panel shows that detection degrades monotonically as p_{drop} increases. This is expected: censorship reduces the number of realized cross-partition deliveries and therefore the number of opportunities for contradictory KT views to meet. For small to moderate censorship rates, larger

p_{gossip} helps sustain high detection probability, since surviving cross-partition messages are more likely to carry gossip. As p_{drop} approaches 1, however, all curves collapse: once cross-partition communication is effectively eliminated, no amount of additional gossip can compensate.

Latency: The middle panel shows that, conditional on detection still occurring, time to evidence increases as p_{drop} grows. The underlying reason is again scarcity of bridge opportunities: when fewer cross-partition messages survive, successful exposure takes longer. The curves become noisier at high censorship rates since many runs never produce evidence and are excluded from the conditional latency statistic.

Overhead. The bottom panel confirms that gossip overhead is essentially unaffected by cross-partition dropping. At fixed adoption, overhead is determined primarily by p_{gossip}, not by whether censored cross-partition messages ultimately survive. This isolates the effect of censorship cleanly: the adversary reduces reliability and increases latency without requiring the protocol itself to become more expensive.

7 Prototype Evaluation: Signal Integration

To assess deployability, we implemented MINGLE in the open-source Signal Android client[4] together with Signal's open-source KT server implementation[5]. Our goal is to show that MINGLE can be embedded into a real E2EE messenger as a natural extension of the existing messaging and KT stack, without any changes to the user experience and while preserving compatibility with legacy clients.

7.1 Prototype implementation

While a public implementation exists, Signal does not yet deploy an official KT server, so our modified Android client uses a local KT server. This section describes the concrete gossip object carried by MINGLE, how it is embedded into Signal messages, and how recipients validate it against their local KT state. Our design goal is to carry just enough state to expose split views while keeping the payload compact and compatible with existing protocols.

Integration We extend Signal's `DataMessage` with a dedicated `key_transparency_field` carrying a compact `StoredTreeHead` and add a Java gRPC layer to retrieve the current distinguished tree head from the KT server, and delegate cryptographic verification to the existing Rust-based `libsignal-keytrans` logic. On send, the client fetches the current distinguished tree head, validates it, converts it to the gossip representation, and embeds it into the encrypted message payload. On receive, the recipient extracts the gossip field, checks it against its locally stored tree head, and updates local state if verification succeeds.

[4] Release Hash: 739f32, version: v7.66.5
[5] Commit hash: 11cf6a, version: 20250808.2.0 .

Table 1. Application-layer and wire-level overhead of MINGLE across Signal message types. Wire overhead varies due to Signal's bucketed ciphertext padding.

Content type	Envelope (base)	Envelope (+gossip)	Ciphertext (base)	Ciphertext (+gossip)	Wire Δ
Smallest message "."	51	169	747	919	172
Short text	90	208	855	939	84
Link with preview	203	320	963	1131	168
Large text block	2004	2121	3415	3543	128

Payload MINGLE gossips a compact `StoredTreeHead` object, the minimum KT state needed for clients to compare directory views, consisting of a SHA-256 root hash, the tree size, a timestamp, and the service signature over the tree head. Summing up these components, the estimated total size of the gossip payload amounts to approximately 114 bytes, and embedding it into a Signal `DataMessage` adds a further 5 bytes of protobuf framing overhead. Hence, at the application layer, each gossip-carrying message grows by 119 bytes. However, the wire-level overhead is not uniform, because Signal pads serialized messages into size buckets before transmission. Figure 8 in Appendix D depicts the structure of the gossip payload. Wire-level overhead varies because Signal pads ciphertext into size buckets before transmission, see Table 1.

8 Conclusion

End-to-end encrypted messaging depends not only on strong cryptographic protocols, but also on the integrity of the key directory that binds identities to public keys. Key transparency makes malicious key substitution auditable, but equivocation remains a practical obstacle: clients may verify their own KT view without knowing whether others are seeing the same directory state. We presented *MINGLE*, an in-band KT consistency-gossip protocol that turns ordinary encrypted conversations into a distributed audit mechanism. MINGLE lets clients cross-check KT state without new global infrastructure, yields publicly verifiable evidence of split views, and works with realistic messaging behavior and partial deployment. By integrating MINGLE into Signal's existing client and KT stack, we showed that this approach is compatible with a real messenger architecture, at only a small per-message overhead. These results suggest that client-driven, in-band consistency checks are a practical complement to KT deployments.

Future work An important direction for future work is interoperable gossip in multi-provider settings. Cross-provider comparison would weaken the single-operator threat model considered here by removing one provider's ability to mediate all cross-partition communication, and makes selective suppression of split-view evidence substantially harder in practice.

Acknowledgment

We thank the anonymous reviewers at USENIX 2026 for their helpful feedback on an earlier draft of this work. This work was supported by the European

Union's Horizon Europe research and innovation programme under grant agreement № 101168311 (LICORICE) and grant agreement № 101225797 (POSEIDON). We are grateful to Melissa Chase for a very helpful discussion on the system design, and Andraž Juvan for the discussions on systems engineering.

References

ACD19. Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. pages 129–158, 2019.

Bon16. Joseph Bonneau. Ethiks: Using ethereum to audit a coniks key transparency log. In *International Conference on Financial Cryptography and Data Security*, pages 95–105. Springer, 2016.

CBC+25. Rense Corten, Laura Boeschoten, Thijs Carrière, Stein Jongerius, Bella Struminskaya, Joris Mulder, Parisa Zahedi, Shiva Nadi Najafabadi, and Adriënne Mendrik. Assessing mobile instant messenger networks with donated data. *Social Network Analysis and Mining*, 2025.

CCD+17. Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. pages 451–466, 2017.

CDGM19. Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. SEEMless: Secure end-to-end encrypted messaging with less trust. pages 1639–1656, 2019.

CSP+15. Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 415–423. IEEE, 2015.

Dah18. Rasmus Dahlberg. Aggregating certificate transparency gossip using programmable packet processors. Master's thesis, jan 2018. `https://www.diva-portal.org/smash/get/diva2:1177741/FULLTEXT01.pdf`.

DKM+21. Alexandra Dirksen, David Klein, Robert Michael, Tilman Stehr, Konrad Rieck, and Martin Johns. Logpicker: Strengthening certificate transparency against covert adversaries. *Proceedings on Privacy Enhancing Technologies*, 2021.

FK23. Matthias Fassl and Katharina Krombholz. Why i can't authenticate—understanding the low adoption of authentication ceremonies with autoethnography. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2023.

Göb23. Thore Göbel. Security analysis of proton key transparency. Master's thesis, ETH Zurich, 2023.

KRS+19. Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. pages 1447–1464, 2019.

MBB+15. Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. {CONIKS}: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015.

McM26. Brendan McMillion. Key Transparency Architecture. Internet-Draft draft-ietf-keytrans-architecture-06, Internet Engineering Task Force, January 2026. Work in Progress.

MKKS+23. Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive*, 2023.

NGR18. Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-Draft draft-ietf-trans-gossip-05, Internet Engineering Task Force, January 2018. Work in Progress.

OPR20. Michael Oxford, David Parker, and Mark Ryan. Quantitative verification of certificate transparency gossip protocols. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2020.

PMS25. Trevor Perrin, Moxie Marlinspike, and Rolfe Schmid. The double ratchet algorithm. `https://signal.org/docs/specifications/doubleratchet/#out-of-order-messages`, 11 2025. Revision 4.

SEA23. Apple SEAR. Advancing imessage security: imessage contact key verification, October 2023. `https://security.apple.com/blog/imessage-contact-key-verification`.

sig. What is a safety number and why do i see that it changed? `https://support.signal.org/hc/en-us/articles/360007060632-What-is-a-safety-number-and-why-do-I-see-that-it-changed`. Retrieved on 2026/04/29.

SL20. Ebrahim Sahafizadeh and Behrouz Tork Ladani. A model for social communication network in mobile instant messaging systems. *IEEE transactions on computational social systems*, 7(1):68–83, 2020.

wha26. Whatsapp encryption overview. `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`, 02 2026. Version 9, page 32, "Verifying Keys".

YGH+22. Tarun Kumar Yadav, Devashish Gosain, Amir Herzberg, Daniel Zappala, and Kent Seamons. Automatic detection of fake key attacks in secure messaging. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3019–3032, 2022.

Ethical Considerations

This work is defensive in nature. Its goal is to strengthen key transparency deployments for end-to-end encrypted messengers by helping clients detect equivocation and silent key substitution, without weakening the underlying encryption protocol.

Our experiments affected three primary stakeholder groups: the service provider, the user base, and the research team. For the service provider, our experiments used Signal's production infrastructure to transport gossip messages and therefore consumed bandwidth and CPU resources.

To minimize this impact, we imposed strict data caps and kept the total experimental footprint below 2 MB, i.e., less than the size of roughly two high-resolution images, so that the resulting load remained negligible for the non-profit service.

For the user base, all messaging experiments used exclusively researcher-controlled accounts; no messages were sent to or received from third-party users. For the research team, we accepted the possibility that our accounts could be

flagged or banned for violating service policies on automated messaging, and treated that risk as limited to the authors themselves.

We further minimized technical harm in three ways. First, our design used a pass-through architecture that preserved Signal's end-to-end encryption and did not modify server-side state. Second, all experimental nodes were researcher-controlled, eliminating risks to human subjects. Third, gossip messages were placed in an unused protocol field and were therefore ignored by unmodified official Signal clients. We believe the benefits of publishing this work outweigh the minimal risks incurred during evaluation. The paper contributes a practical defense against malicious key distribution and illustrates a low-impact method for evaluating such mechanisms in a realistic messenger setting without compromising service stability or user privacy.

Open Science

Our work produced three artifacts that we make publicly available to support reproducibility and future research. All artifacts are available at `https://github.com/meyira/mingle-artifact`.

Signal Gossip Client

We provide a modified Signal client that demonstrates the practical feasibility of implementing MINGLE's gossip protocol in a production messaging application. The implementation uses Signal's `DataMessage` with a dedicated `key_transparency_field` carrying a compact `StoredTreeHead` to propagate gossip messages without breaking compatibility with standard Signal clients. Full build instructions, dependencies, and a test suite are included in signal-gossip-client/README.md.

Network Simulation

We provide a Jupyter notebook (`network-simulation/kt-gossip-simulator.ipynb`) implementing our network simulation framework for evaluating MINGLE's performance under various network topologies and attack scenarios.

Our simulation supports configuring a number of parameters at the top of the notebook: configurable messenger client and group sizes and activity times, multiple server scenarios (honest, isolating in regions, maintaining permanent and transient forks), configurable forking parameters for both size and duration, sticky and non-sticky victims, and a tunable probability of attaching gossip to a message.

The expected runtime is approximately 10 minutes on a standard laptop. The simulation framework is modular and can be extended for future research on gossip-based security protocols.

A Omitted Proofs

Proof (Proof of Lemma 1). Let $(D_t^{(1)}, H_t^{(1)})_{t \geq e}$ and $(D_t^{(2)}, H_t^{(2)})_{t \geq e}$ be the two incompatible post-fork histories inducing the partition, agreeing on epochs $0, \ldots, e-1$ and diverging from epoch e onward.

By hypothesis, $(u \to v) \in E_t$ is a realized cross-partition communication event with $u \in U^{(1)}$ and $v \in U^{(2)}$, and at the time of delivery u gossips a distinguished commitment $H_r^{(1)}$ with $r \geq e$, while v's local distinguished commitment is $H_r^{(2)}$ for the same epoch or tree size r.

Since u and v lie in different view partitions, their post-fork commitments at epoch or tree size r are commitments to different histories. Hence

$$H_r^{(1)} \neq H_r^{(2)}.$$

Both commitments are valid, because they are accepted into local state only after passing $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, \cdot) = 1$.

By Corollary 2, if the message from u is delivered, then its attached gossip payload is delivered intact as well. Thus v receives $H_r^{(1)}$ as part of the delivered gossip payload and verifies it using $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, \cdot)$. Upon receipt, v compares the received commitment against its current distinguished commitment $H_r^{(2)}$. Since the two commitments refer to the same epoch or tree size but have different roots, the receive-side logic in both Algorithm 1 and Algorithm 2 takes the $\mathsf{SameEpochOrSize}$ branch and constructs

$$w = (H_r^{(2)}, H_r^{(1)}, \mathsf{ctx}),$$

then outputs $\mathsf{alarm}(w)$.

This w is a valid PoM in the sense of Definition 2, namely of the first form: two valid commitments for the same epoch or tree size with different roots. Hence the realized cross-partition delivery exposes the fork and triggers detection at v.

Proof (Proof of Theorem 3). By assumption, the adopting gossip graph $\mathcal{G}^{\alpha}$ is cross-connected after the fork, and a realized cross-partition gossip event occurs at a time when its sender and receiver hold post-fork distinguished commitments for the same epoch or tree size. Let $(x \to y)$ denote that realized event. Since u and v lie in different view partitions, the sender's gossiped commitment and the receiver's local distinguished commitment are commitments to different post-fork histories.

By Corollary 2, attached gossip on the delivered message cannot be selectively stripped while preserving stealth. Therefore the receiver v obtains the sender's commitment intact. All conditions of Lemma 1 are satisfied, and the lemma implies that v outputs $\mathsf{alarm}(w)$ for a valid PoM w.

Hence detection follows from any realized cross-partition gossip event of the stated form.

B Protocol Algorithms

The two protocol variants are summarized below. In both variants, alarms are raised only on directly conflicting commitments; failed or missing consistency responses are retained as auxiliary context.

Algorithm 1 Commitments-only MINGLE at client u

State at client u:
  $H_u^{(\max)} \in \mathcal{H}_u \cup \{\perp\}$, $\mathcal{H}_u, \mathcal{Q}_u, \mathcal{C}_u$

function OnSend(u)
    if this message is selected for gossip **then**
        TryRefresh(u)
        return $\mathsf{EncodeGossip}(H_u^{(\max)}, \perp)$
    else
        return $\perp$
    end if
end function

function OnReceive(u, γ_v)
    if $\gamma_v = \perp$ **then**
        return
    end if
    $(H, _) \leftarrow \mathsf{DecodeGossip}(\gamma_v)$
    if $H = \perp$ **or** $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H) = 0$ **then**
        return
    end if
    if $H_u^{(\max)} = \perp$ **then**
        AdoptUnderTOFU(u, H)
        return
    end if
    if $H = H_u^{(\max)}$ **then**
        return
    end if
    if $\mathsf{SameEpochOrSize}(H, H_u^{(\max)})$ **then**
        RaiseAlarm($u, H_u^{(\max)}, H$)
    else if $\mathsf{IsNewer}(H, H_u^{(\max)})$ **then**
        TryAdvance($u, H_u^{(\max)}, H$)
    else
        TryReconcile($u, H, H_u^{(\max)}$)
    end if
end function

Helper routines. We treat the following as abstract helper predicates and routines:

– $\mathsf{IsNewer}(H, H')$: returns true if H is strictly more recent than H'.

Algorithm 2 Commitments+proof MINGLE at client u

State at client u:
  $H_u^{(\max)} \in \mathcal{H}_u \cup \{\perp\}$, $\mathcal{H}_u, \mathcal{Q}_u, \mathcal{C}_u$

function OnSend(u)
    if this message is selected for gossip **then**
        TryRefresh(u)
        $\rho \leftarrow$ LatestProofTo($H_u^{(\max)}, \mathcal{Q}_u$)
        return $\mathsf{EncodeGossip}(H_u^{(\max)}, \rho)$
    else
        return $\perp$
    end if
end function

function OnReceive(u, γ_v)
    if $\gamma_v = \perp$ **then**
        return
    end if
    $(H, \rho) \leftarrow \mathsf{DecodeGossip}(\gamma_v)$
    if $H = \perp$ **or** $\mathsf{VerifyCommit}(\mathsf{pk}_{\mathcal{I}}, H) = 0$ **then**
        return
    end if
    if $\rho \neq \perp$ **then**
        TryStoreGossipedProof(u, ρ)
    end if
    if $H_u^{(\max)} = \perp$ **then**
        AdoptUnderTOFU(u, H)
        return
    end if
    if $H = H_u^{(\max)}$ **then**
        return
    end if
    if $\mathsf{SameEpochOrSize}(H, H_u^{(\max)})$ **then**
        RaiseAlarm($u, H_u^{(\max)}, H$)
    else if $\mathsf{IsNewer}(H, H_u^{(\max)})$ **then**
        TryAdvanceWithGossip($u, H_u^{(\max)}, H, \rho$)
    else
        TryReconcileWithGossip($u, H, H_u^{(\max)}, \rho$)
    end if
end function

- $\mathsf{SameEpochOrSize}(H, H')$: returns true if H and H' refer to the same epoch or tree size.
- $\textsc{AdoptUnderTOFU}(u, H)$: sets $H_u^{(\max)} \leftarrow H$ and records H in $\mathcal{H}_u$.
- $\textsc{TryRefresh}(u)$: refreshes the local distinguished commitment if needed using $\mathsf{GetCommitment}()$ and, when applicable, a valid proof from $\mathsf{GetConsistencyProof}(\cdot, \cdot)$; otherwise it records auxiliary context.
- $\textsc{TryAdvance}(u, H_i, H_j)$: attempts to justify the append-only relation from H_i to H_j using a KT query; if successful, it advances $H_u^{(\max)}$ to H_j, otherwise it records auxiliary context.
- $\textsc{TryReconcile}(u, H_i, H_j)$: attempts to justify the append-only relation from H_i to H_j and stores the result if successful; otherwise it records auxiliary context.
- $\textsc{LatestProofTo}(H, \mathcal{Q})$: returns a recently verified proof in $\mathcal{Q}$ whose upper endpoint is H, or $\perp$ if none exists.
- $\textsc{TryStoreGossipedProof}(u, \rho)$: stores ρ if it verifies for its claimed endpoints; otherwise it records auxiliary context.
- $\textsc{TryAdvanceWithGossip}$ and $\textsc{TryReconcileWithGossip}$: first attempt to use a suitable gossiped proof and fall back to KT queries if needed.
- $\textsc{RaiseAlarm}(u, H_i, H_j)$: outputs $\mathsf{alarm}(w)$ for $w = (H_i, H_j, \mathcal{C}_u)$.

C Simulation Parameters

Our survey (N=82 initial responses, N=31 complete responses [6] optional questions) captured: contact counts, messaging frequency (daily/weekly one-on-one chats), peak activity hours, multi-device usage, group chat sizes, and security awareness. The survey responses were collected anonymously and are retained indefinitely in anonymized form. No personally identifiable information was linked to response data. Complete anonymized data and analysis scripts are provided in the artifact under `user-study-results/`.
Completion of most surveys required ≤ 10 minutes, and participants were aged between 20 to 40.

C.1 Gossip intensity vs. Detection and Cost.

Table 4 summarizes the reliability–latency–overhead trade-off for in-band KT gossip in our hardest setting (small targeted victim partition, sparse cross-partition mixing). Increasing the attachment rate p_{gossip} consistently improves both outcomes: with adoption 0.1, raising p_{gossip} from 0.01 to 0.03 increases detection probability from 0.80 to 0.93 and cuts the median time-to-evidence from 120 to 32.5 minutes, while also tightening the spread (quartiles shift from 40–150 to 11.25–60 minutes). At $p_{gossip} = 0.05$, detection becomes reliable (1.00) and latency drops further to a 30-minute median. Higher adoption shifts this curve

[6] As a participation incentive, we offered a random drawing for free drinks at an affiliated event. Entry required participants to voluntarily disclose their email address.

Table 2. Survey-calibrated parameters used to instantiate the messaging infrastructure (client study N=82; question-level n varies).

Symbol	Meaning	Survey-derived value / distribution
D_d	weekly 1–1 partners	median 10; IQR 6–19.5; mean 12.97; P90 24.7 (n=74).
$D_{d,day}$	daily 1–1 partners	median 5; IQR 3–8; mean 6.07; P90 10 (n=74).
D_{start}	activity start time	median 08:22; IQR 06:52–10:08 (n=52).
D_{end}	activity end time	median 21:45; IQR 18:15–23:15 (n=51).
D_{span}	daily active span (hours)	median 13.75; IQR 7.88–15.25 (n=51).
D_g	group size	median 5; IQR 5–20; mean 18.9; max 225 (n=138 instances).
C_g	group churn (qual.)	low churn; 49.3% "longer" tenure (n=142 instances).

Note: Protocol parameters (e.g., p_{gossip}, adoption, audit rates) and server misbehavior modes are swept independently.

Table 3. Parameter sweep space (baseline values given where applicable).

Parameter	Values swept
Gossip attach prob. p_{gossip}	{0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0}
Adoption fraction a	{0.1, 0.2, 0.4, 0.6, 0.8, 1.0}
Fork fraction f	{0.01, 0.05, 0.1, 0.2, 0.5}
Fork start t_f (hours)	baseline 3 (optional: {1, 3, 6})
Transient duration τ (hours)	baseline 2 (optional: {1, 2, 4})
Equivocation prob. p_{equiv}	{0.05, 0.1, 0.2} (rolling)
Cross-cut drop p_{drop}	{0, 0.1, 0.3, 0.5}

left. With adoption 0.3, even a low $p_{gossip} = 0.01$ achieves reliable detection (1.00) but still requires about 50 minutes median to surface evidence; increasing to $p_{gossip} = 0.05$ reduces the median to 5 minutes. These improvements come at a bandwidth cost. Overhead grows with both p_{gossip} and adoption, from ≈ 2.5 B/msg at (p_{gossip}=0.01, adopt=0.1) to ≈ 34.8 B/msg at $(0.05, 0.3)$, quantifying the practical trade-off between faster, more reliable detection and added payload on normal traffic.

D Signal Integration Experimental Setup

The local key transparency server ran on a Fedora 43 laptop with an i9-13900H, 64GB of RAM and a Nvidia GeForce RTX 4070. We ran two identical virtual Android devices based on a Google Pixel 9 Android 35 device on x86-64 architecture with 2 GB of RAM. The communication runs over the Signal infrastructure.

Despite attempts to measure timing differences, network delivery variance prevented us from detecting any latency introduced by our payload attachment. This indicates that any overhead our approach adds falls below the noise floor of

Table 4. TRADEOFF (hard). Detection probability (det.), evidence latency after fork (lat, minutes; median with quartiles Q_1, Q_3), and overhead (B/msg) versus gossip rate p_{gossip} and adoption.

p_{gossip}	adopt	reps	det.	lat	Q_1	Q_3	B/msg
0.01	0.1	15	0.80	120.0	40.0	150.0	2.5
0.03	0.1	15	0.93	32.5	11.25	60.0	7.4
0.05	0.1	15	1.00	30.0	7.5	42.5	12.3
0.01	0.3	7	1.00	50.0	32.5	95.0	6.9
0.03	0.3	7	1.00	20.0	15.0	47.5	20.3
0.05	0.3	7	1.00	5.0	5.0	17.5	34.8

typical network conditions. All additional logic runs in the existing background threads that already handle message decryption and network I/O; no changes to the user interface were required. In particular, gossip payloads are never rendered, and sending/receiving flows remain indistinguishable from an unmodified Signal client from the user's perspective.

Network overhead To measure concrete message overhead, we compared representative `DataMessage` instances with and without gossip across several message classes, including very short text, formatted text, emoji, link previews, and long text blocks. At the application-payload level, each gossip-carrying message grows by 119 bytes. The resulting wire-level overhead is not uniform, however, because Signal pads serialized messages into size buckets before transmission. As Table 1 shows, MINGLE's payload overhead is constant, while its transport overhead is bucket-dependent.

Although the serialized envelope grows by nearly the same amount across message types, the padded ciphertext overhead varies substantially, reflecting the interaction between the 119-byte gossip payload and Signal's bucketed padding policy.

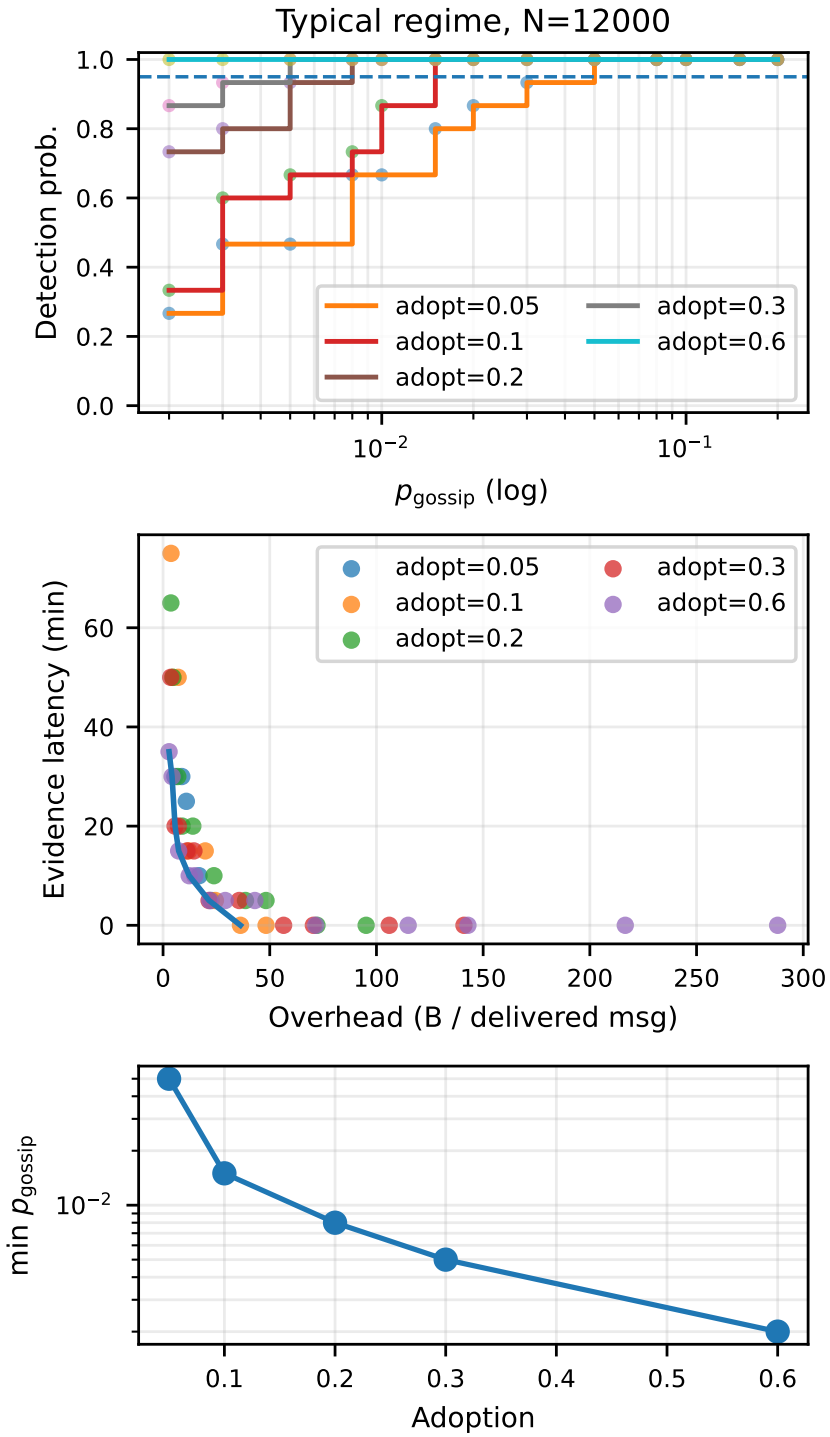


Fig. 4. BASELINE. Typical regime. **Top:** Detection probability vs. p_{gossip} (log). **Middle:** For reliable settings (det≥ 0.95), evidence latency vs. mean gossip overhead. **Bottom:** Minimum p_{gossip} needed for det≥ 0.95 as adoption increases.

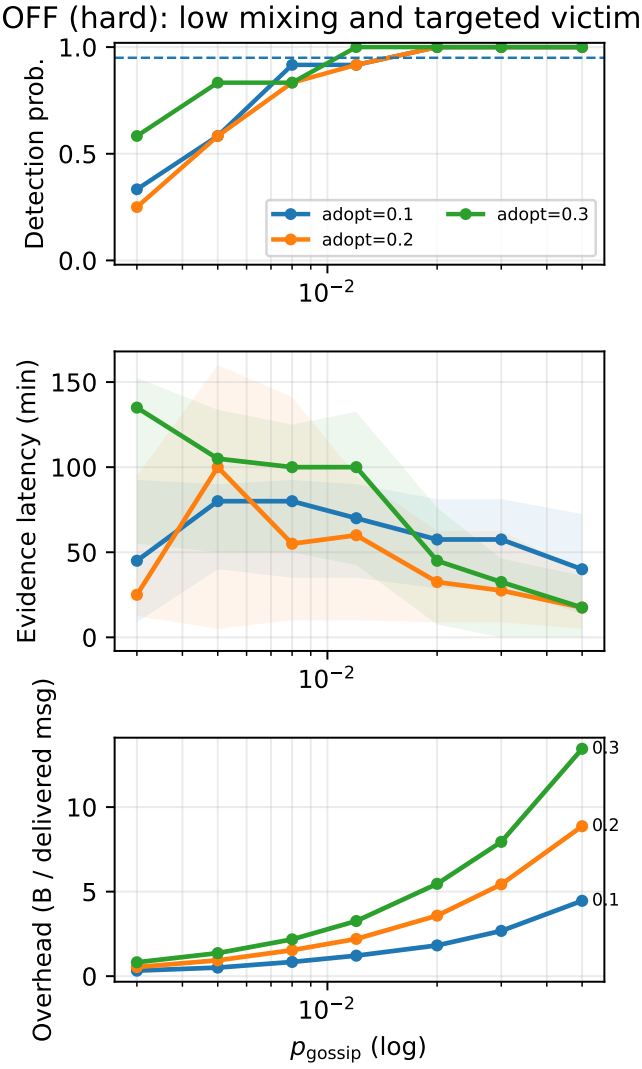


Fig. 5. TRADEOFF (hard). Split-view detection under low mixing and targeted victims ($f = 0.02$, `p_contact_cross` = `p_group_cross` = 0.005). **Top:** detection probability. **Middle:** evidence latency after the fork (median ± IQR). **Bottom:** gossip overhead (bytes per delivered message).

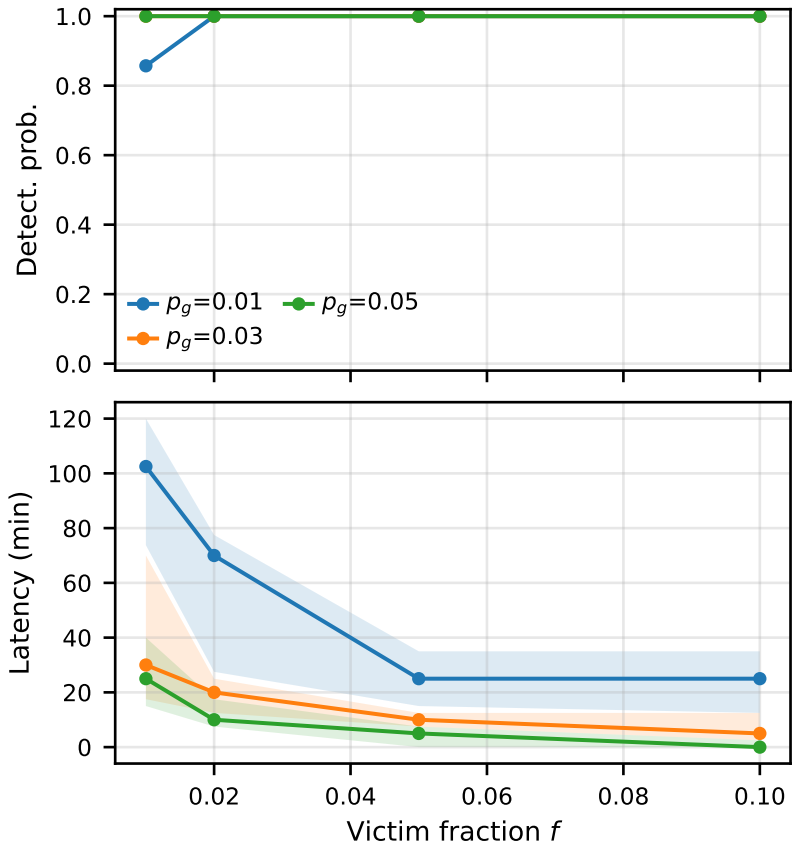


Fig. 6. TARGET Effect of the targeted victim fraction f on split-view detection. We vary f = `fork_frac` and plot detection probability (top) and evidence latency after fork (bottom; median ± IQR, undefined runs excluded), for several gossip rates p_{gossip} at fixed adoption.

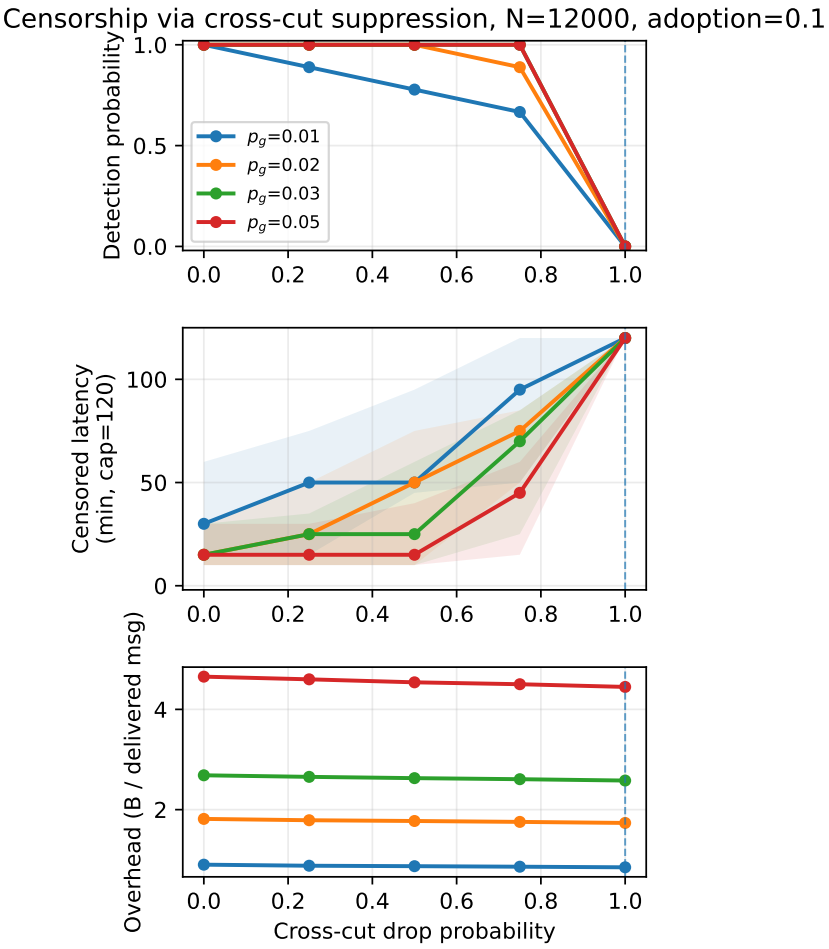


Fig. 7. CENSOR. Effect of suppressing cross-partition communication on split-view detection. We vary the crosscut drop probability p_{drop} and plot detection probability (top) and evidence latency after fork (bottom; median ± IQR), for several gossip rates p_{gossip} at fixed adoption.

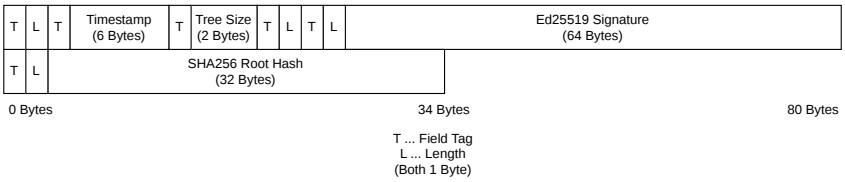


Fig. 8. Byte-level deconstruction of the gossip payload.