Amigo: Secure Group Mesh Messaging in Realistic Protest Settings

David Inyangson[1*], Sarah Radway[2*],
Tushar M. Jois[3], Nelly Fazio[3] and James Mickens[2]

[1] Johns Hopkins University
dinyang1@jhu.edu
[2] Harvard University
sradway@g.harvard.edu, mickens@g.harvard.edu
[3] City College of New York
tjois@ccny.cuny.edu, nfazio@ccny.cuny.edu

Abstract. During large-scale protests, a repressive government will often disable the Internet to thwart communication between protesters. Smartphone mesh networks, which route messages over short-range, possibly ephemeral, radio connections between nearby phones, allow protesters to communicate without relying on centralized Internet infrastructure. Unfortunately, prior work on providing secure communication in Internet shutdown settings fails to adequately consider protester needs. Previous attempts fail to support efficient *private group* communication (a crucial requirement for protests), and evaluate their solutions in network environments which fail to accurately capture link churn, physical spectrum contention, and the mobility models found in realistic protest settings. In this paper, we introduce Amigo, a novel mesh messaging system which supports group communication through a decentralized approach to continuous key agreement, and forwards messages using a novel routing protocol. Amigo is uniquely designed to handle the challenges of ad-hoc routing scenarios, where dynamic network topologies and node mobility make achieving key agreement nontrivial. Our extensive simulations reveal the poor scalability of prior approaches, the benefits of Amigo's protest-specific optimizations, and the challenges that still must be solved to scale secure mesh networks to protests with thousands of participants.

Keywords: mesh messaging · censorship circumvention · continuous group key agreement · mesh routing protocols · mobility modeling

1 Introduction

Protests are an important tool in the fight against authoritarian governments. By occupying a physical space, ordinary citizens can express their grievances and bring visibility to their concerns. In countries where political freedoms are curtailed, protests may be the *only* mechanism for the public to express their grievances.

When protests become movements comprising thousands of people, communication becomes vital for organizing protest actions and keeping participants safe. In a modern protest, the demonstrators often rely on smartphone-based messaging platforms like WhatsApp or Telegram to communicate; using these apps, protesters learn about rally points and inform each other about the location of the police or the military [ABJM21a]. Authoritarian governments are thus highly motivated to intercept or alter the messages

[*] These authors contributed equally to this work.

that protesters exchange. Using intercepted messages, the government can identify protest leaders and shift police deployments to disrupt nascent rally points; using altered messages, the government can send spoofed messages, ostensibly from protest leaders, that direct protesters towards harm. These concerns are not theoretical. For example, law enforcement – even in relatively liberal governments – possess *cell-site simulators* which create fake cellphone towers [Cel15]. By inducing protester phones to pair with a simulator, law enforcement can launch a variety of attacks, e.g., downgrading the phone's cellular connection to the unencrypted 2G protocol, allowing the government to intercept messages to and from that phone [Ele23].

A repressive government may also try to degrade or completely disrupt app-based communication during a protest. For example, during the 2019 Hong Kong anti-ELAB protests, the government (or its partisans) launched a denial-of-service attack against Telegram servers [BeW19]. In 2019, Iran responded to protests involving economic discontent by enacting a large-scale Internet shutdown [Amn20]. Similarly, in Myanmar, Internet shutdowns are frequently imposed to prevent civilians from documenting human rights violations or accessing information about military operations [Acc24a]. Overall, there were as many as 283 Internet shutdowns across 39 counties in 2023 alone – the worst year for Internet shutdowns on record [Acc24b].

Thus, protesters need a robust way to exchange authenticated, encrypted messages, even if the government has disabled Internet connectivity or tampered with cellular routing infrastructure. A promising approach is to exchange encrypted messages via an *ad-hoc mesh network* built from point-to-point radio links (e.g., Bluetooth between smartphones). In a mesh network, there is no centralized routing infrastructure; instead, user devices exchange messages over (potentially ephemeral) short-range links created via physical proximity as users move in space. Multi-hop message forwarding over these point-to-point links enable information exchange between users too far away to establish direct connections.

Because mesh networks do not rely on centralized infrastructure, there is no single point of failure for a regime to attack. Unfortunately, classic mesh protocols [AW09, AWW05, ZLH06] do not safeguard user privacy, revealing message senders and recipients to even passive observers. These protocols are ill-suited for protest scenarios where users wish to hide their identities from the government [ABJM21a].

The research community has introduced mesh networks that attempt to hide user activity [PJW+22, PSEB22, BRT23, RL08, RYLZ09, LQK09, Sen12, SS14, PAC14, HBDF+13]. However, previous systems cannot practically handle large-scale protests for two major reasons. First, prior systems lack efficient (or any) support for private groups, despite the widespread use of group-based messaging in protest settings [ABJM21a]. Second, prior systems use routing protocols which are scalable in benign network conditions, but can experience persistent congestion collapse for the mobility patterns and smartphone radio technologies found in realistic protest settings; the result of the persistent network congestion is a very low rate for application-level message delivery.

To address these challenges, we introduce *Amigo*, a new system for efficient, secure mesh communication in protest settings. Amigo's high-level novelty is that it is designed *full stack*, accounting for both security challenges at the cryptographic layer and performance/reliability challenges at Layers 1 and 2 of the network stack. To protect messages exchanged by protesters, Amigo leverages a new protocol for continuous group key agreement (CGKA), efficiently providing security properties specifically required by protesters (e.g., post-compromise security, and fast removal of a member from a group). Additionally, Amigo introduces a new mesh routing protocol which, compared to earlier protocols, enjoys higher delivery rates for mobility patterns often seen in real protests. Our comprehensive simulations, which leverage those realistic mobility models and high-fidelity representations of low-level network phenomena like radio collisions between nearby phones, demonstrate that Amigo's protest-specific optimizations provide secure, protester-desired functional-

ity with better routing performance than earlier work. However, our evaluation results also reveal fundamental scalability challenges for all known approaches to secure mesh networking (including Amigo); a fundamental contribution of the paper is illuminating those practical challenges (previously unknown due to the lower-fidelity simulations of prior work), and suggesting several directions for future work.

Contributions. In summary, we contribute the following:

- We introduce a new protocol for decentralized group key agreement in mesh networks. The protocol tolerates an unreliable mesh that may drop or reorder messages, providing important security properties missing from prior schemes. It is also efficient with respect to network consumption and CPU usage – critical for a protocol that runs atop resource-constrained phones.
- We introduce a new routing approach, based on clique-level forwarding, tailored to protest-based mobility patterns.
- To evaluate Amigo, we introduce new mobility models which capture how real-life protesters organize in physical space, both during stable times and when external shocks arise (e.g., the unexpected appearance of the police). Our evaluation experiments also capture important Layer 1 and Layer 2 network phenomena that were ignored in prior work on secure mesh networking.
- We test Amigo's performance when Amigo uses our clique-based routing approach and routing approaches from prior work, demonstrating (1) the benefits of Amigo's protest-specific optimizations at the cryptographic layer and the routing layer, and (2) the scalability and robustness challenges (previously unknown) that still remain unsolved, given the challenging network environment created by protest-based mobility patterns and the nuances of real-life short-range communication technologies. Our results suggest mesh routing protocols previously thought to be practical will often function poorly in realistic protest environments due to a self-reinforcing cycle of network congestion. We discuss the implications of our results on future research, and will open-source all of our code upon publication as artifacts.

2 Motivation and Background

A large-scale protest involves hundreds or thousands of people gathered in (and moving throughout) a shared physical space. The success of a such a protest increasingly depends on the ability of protesters to organize via smartphone-based communication [Shi11, LA10, ABJM21a, AG15]. Without the ability to effectively communicate, critical information regarding group movement, law enforcement activity, and collective decision-making propagates slowly among protesters. The consequences for the protesters are dire: physical assault, imprisonment, or even death [CG18].

As a case study, consider the widespread 2019–2020 protests in Hong Kong. The protests erupted in response to the government's attempt to pass the Extradition Law Amendment Bill (ELAB). The bill would have allowed Hong Kong citizens to be extradited to China; many Hong Kong citizens feared that the bill would further erode Hong Kong's legal independence from China [Lee20, LYTC19]. Using smartphone apps like WhatsApp, Telegram, and Signal, protesters decided where and when to hold demonstrations. Once protesters arrived on site, these apps allowed protesters to respond in real time to events like the arrival of police [ABJM21a].

The ability of citizens to efficiently organize *during* the protest itself was a marked departure from static, pre-smartphone-era demonstration strategies. In the anti-ELAB protests, a column of marchers might quickly decide to split into groups, each taking a path down a different street and abruptly changing directions in response to newly-placed barricades; using apps like Telegram, the independent groups could synchronize their long-term movement trajectories and later reform into a single human chain [BeW19].

Prior work has explored smartphone-based *mesh networks* as a way for protesters to communicate without centralized Internet access. Mesh networks leverage short-range radio technologies like Bluetooth or Wi-Fi Direct, routing messages between a sender and a receiver across potentially ephemeral radio connections between mobile nodes. In the protest context, *end-to-end encryption* of signed/authenticated messages ensures that network eavesdroppers cannot inspect cleartext or alter messages without detection. An ideal mesh network would also limit the impact of an attacker who discovers the keys used by protesters at time t. *Forward secrecy* ensures the attacker cannot break the confidentiality of messages sent before t. *Post-compromise security* means that, after the key leakage at t, the attacker cannot break the confidentiality of messages sent *after* t. Broadly, forward and post-compromise security are implemented by having participants rotate their keys periodically.

Protester needs. Albrecht et al. performed wide-ranging interviews with 11 anti-ELAB protestors from Hong Kong [ABJM21a]. Below, we highlight the relevant findings from that study with respect to how communication apps for protesters should be designed:

1. **Protest communication occurs primarily in group chats.** Groups can range from a handful of members to thousands.
2. **Groups must be able to securely add members as a protest unfolds.** Some protesters may hear about a demonstration offline, e.g., word-of-mouth. Upon arrival, such a person may wish to join a group for protest coordination. So, protest-focused messaging requires the ability to add new members on-the-fly.
3. **Groups must be able to remove members.** Protesters are resigned to the fact that law enforcement may be able to infiltrate groups (especially larger ones). Protesters also understand that law enforcement may coerce a loyal group member to nonetheless relinquish their device and their login credentials. Thus, revoking a user's group membership is critical functionality.
4. **Communications should be possible during an Internet shutdown.** Protest-focused communication apps should still function if centralized Internet infrastructure has been degraded. During the Telegram blackout in the anti-ELAB protests, some protesters attempted to use the mesh networking app Bridgefy [Bri22] to exchange messages, but it failed to handle the load presented by the large numbers of protesters [ABJM21a].
5. **Communication should be anonymous.** The threat of retaliation from law enforcement requires that participation in an online protest group should not reveal the real-life identity of the sender or the receiver. The peer-to-peer nature of mesh networking does not automatically provide anonymity. For example, security researchers have found that Bridgefy [Bri22] protects neither message confidentiality nor message authenticity, and allows outsiders to deanonymize users [ABJM21b].

3 System Design

At a high level, Amigo's goal is to provide a mesh network that meets the protester needs described above while being reliable, low-latency, secure, and resource-efficient. *Reliability* means that Amigo successfully forwards the vast majority of messages to their intended recipients, despite the stochastic nature of pairwise node connectivity. *Low-latency* forwarding means that Amigo delivers messages as quickly as possible, given the prevailing network conditions. *Secure* forwarding ensures that messages are end-to-end encrypted, integrity-protected, and enjoy both forward security and post-compromise security; additionally, key agreement among group members should not leak cryptographic information to participants outside of the group. *Resource efficiency* is important because Amigo targets smartphone deployments in which network bandwidth, CPU cycles, memory space, and available power are limited compared to traditional server-based environments.

To the best of our knowledge, Amigo meets or outperforms all prior work (§8) along these evaluation metrics. However and importantly, our evaluation experiments (§7) illuminate fundamental performance and robustness challenges for secure mesh networks—challenges that must be solved for these systems to reliably scale to protests involving thousands of participants (§9).

Architecture. Figure 1 depicts Amigo's high-level architecture. Users interact with Amigo via the **application layer**, creating and destroying groups, adding or removing members to groups, and exchanging messages with group members. Users directly generate and read text messages or picture messages; behind the scenes, those messages are protected with keys negotiated by Amigo's **session layer**. The session layer implements an efficient CGKA scheme which we describe in detail in Section 4. Amigo focuses on small to medium-sized groups with <200 members; such close-knit groups are perceived by protesters as more trustworthy and less susceptible to infiltration by the government [ABJM21a].

The session layer sends and receives user-generated messages and CGKA-generated messages via the **routing layer**. As we describe in Section 5, Amigo is compatible with three routing schemes from prior work on mesh networking. Amigo also introduces a new approach, dynamic clique routing, which is tailored for the unique connectivity challenges of protest scenarios. As shown in Figure 1, the basic idea behind all of the routing schemes is that, when two nodes wander into communication range, they exchange locally-buffered messages that were generated locally or received from other peers in the past; the routing schemes differ in which messages are exchanged with which peers.

At the bottom of the Amigo stack is the **link layer**. Similar to prior work on secure meshes [LFBD+16, PJW+22, PSEB22], Amigo leverages off-the-shelf point-to-point radio technologies like Bluetooth and Wi-Fi Direct. Using off-the-shelf hardware for Layers 1 and 2 of the network stack makes Amigo easily deployable, but traditional short-range radio protocols lack understanding of end-to-end congestion metrics, and therefore may unwittingly overload the network and trigger precipitous declines in application-level delivery rates.

Threat model. As explained by [ABJM21a], the core goals of an authoritarian regime during a protest are to:

1. identify message senders and group members,
2. read the cleartext content of in-transit messages,
3. tamper with in-transit messages, and
4. compromise protester devices after a protest to decrypt messages that protesters have already received, or inspect/alter messages that protesters will send in the future.

We assume that the adversary has multiple vantage points in the network, but cannot view all traffic emanating from all nodes. We also assume that the adversary, through malicious or lawful means, can physically access a group member's device [ZJG21]; after doing so, the adversary gains access to the messages and the key material associated with all groups on that device. Knowledge of the keys in particular allows the attacker to monitor group communication indefinitely until the attacker is removed from the group.

Given such an adversary, Amigo has three primary security goals. First, Amigo wants to provide messages with confidentiality and integrity (including forward security and post-compromise security). Second, Amigo wants to provide *outsider anonymity*, such that if an eavesdropper receives a message intended for a group that does not include the eavesdropper, the eavesdropper cannot learn the identities of the group members. We discuss these two first security goals in Section 4, evaluate Amigo's ability to provide these goals in Section 7, and provide a proof in Appendix I.

Amigo's third security goal is to defend against denial of service attacks in which malicious nodes intentionally drop messages. Amigo's defenses leverage prior work in which nodes prefer to route messages via trusted peers [LFBD+16, PJW+22]. However, Amigo's core novelty lies in the way that Amigo protects group messaging, so we do not

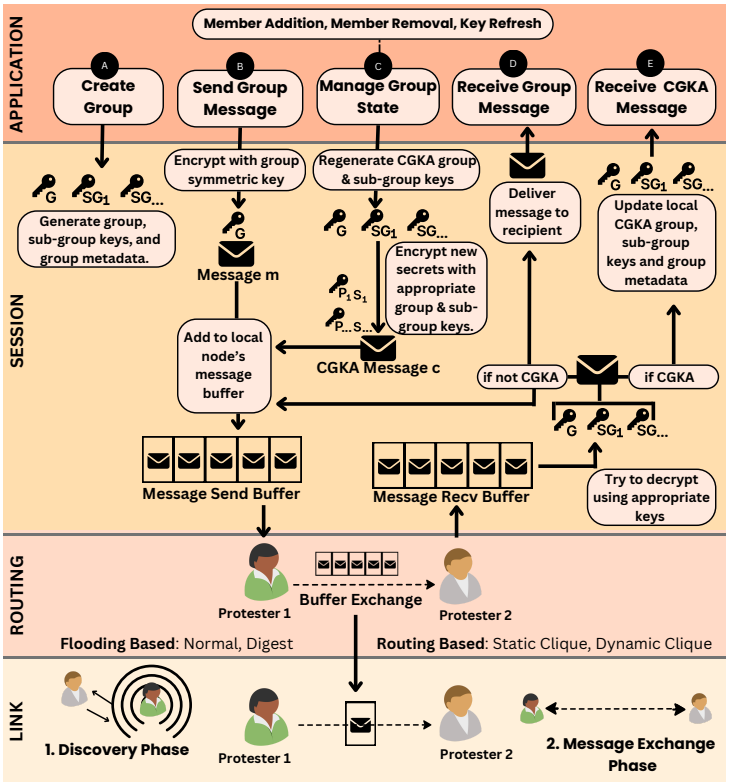

Figure 1: An overview of Amigo and its layers.

focus on DoS protection, e.g., circumventing signal jamming attacks, in the remainder of this paper.

Non-goals. In designing Amigo, we explicitly sought to avoid network behavior that would trigger pathologies in Layer 1 or Layer 2 dynamics. Indeed, a primary contribution of this paper is the demonstration that secure ad-hoc meshes can appear performant when Layer 1 and Layer 2 dynamics are ignored, but may actually experience debilitating congestion collapse when these dynamics are considered (§7). This decision left some attacker behaviors outside of our threat model. For instance, Amigo does not prevent a multi-vantage-point attacker from determining when a user sends an encrypted message because, to the best of our knowledge, state-of-the-art defenses against such attacks require the insertion of dummy messages to thwart correlation analyses [PSEB22]; our simulations (which incorporate Layer 1 and 2 effects) show that secure meshes can experience network collapse *even if those meshes do not inject dummy messages*. In contrast, DoS protections that leverage trusted routing peers [LFBD+16, PJW+22] add minimal network load.

4 A CGKA Protocol for Mesh Messaging

In this section we discuss Amigo's CGKA which is designed to provide the properties protesters need for group communication.

Intuition. Protesters need secure group communication, which intuitively calls for a key agreement protocol that addresses the nuances of mesh messaging during a large-scale protest. For example, the unreliable nature of smartphone mesh links requires our protocol to minimize the number of messages required for consistent key agreement. Another nuance is that in a protest setting, parties do not require *full* anonymity. Protesters vet other

members before inviting them into group communication and do not need to be anonymous within their groups [ABJM21a]. But, they do require *outsider anonymity* [FP12]; group messages sent within the mesh should not include any sender or recipient identifiers to those who are outside the intended group – a natural and appropriate relaxation. Notions of *forward secrecy* and *post-compromise security* are also necessary to limit the impact of a compromise (e.g., due to arrest), and to return the group to a safe state afterwards.

CGKA protocols are promising here [BBR18, KPPW+21, CGCG+18]. These protocols can provide forward secrecy and post-compromise security while also efficiently structuring group members (i.e., through a binary tree) to enable updates of group key material and membership with *sublinear communication complexity*, unlike pairwise approaches (e.g., Signal [Sig14]).

Adapting existing CGKA protocols to a large-scale protest is not trivial, however. Traditional CGKA protocols [BBR18, CGCG+18, IET23] are oriented towards centralized infrastructure, requiring all members to apply protocol messages in the same sequence to maintain consistent state. But, we cannot assume this constraint holds in a mesh during an Internet shutdown.

Another issue is that existing CGKAs protocols do not consider the unique threats of the protest setting. For instance, while CGKAs refresh cryptographic state, the time intervals between these states, or *epochs*, vary. Epochs are not tied to messages but rather group state changes, e.g., a new member. The frequency of these key rotations may be inadequate given the dynamics of the protest environment; forward secrecy and post-compromise security *per message* may be more desirable.

Due to the aforementioned challenges, CGKA protocols remain largely unexplored in the protest use case. In this paper, we leverage the advantages of CGKA protocols by appropriately adapting them to the Internet-shutdown protest setting. We propose a CGKA that is bandwidth-efficient, dynamic, outsider anonymous, and enables consistent state for group members regardless of protocol message order. We also orient our CGKA to specifically handle compromise during a protest scenario (e.g., through efficient removals) and provide stronger (per message) and standard (per epoch) notions of forward secrecy and post-compromise security.

Comparison to MLS. As a concrete example of the challenges in porting existing CGKAs to the setting of Amigo, let us consider Messaging Layer Security (MLS) [IET23], perhaps the most prominent prior CGKA protocol. MLS is an IETF-standardized asynchronous group key agreement protocol that provides forward secrecy and post-compromise security for a wide range of group sizes, ranging from two to thousands. In a typical MLS configuration, a centralized provider is responsible for two services: authentication and delivery. The authentication service binds public key material to application-specific identifiers and enables clients to receive and verify these bindings. The delivery service is responsible for distributing messages and key material, but can also help reconcile group states and enforce a global ordering on messages [IET23].

While sufficient for traditional group messaging scenarios, MLS fails to adequately meet the protester specific needs we highlight in Section 2. One reason for this is that MLS is primarily intended for end-to-end encrypted messaging *over the Internet*, taking advantage of centralized infrastructure (i.e., servers) to realize the authentication and delivery services. The mesh setting we consider is fundamentally different, involving (potentially ephemeral) short-range wireless links, dynamic network topology, and resource-constrained devices. Thus, the reliable centralized services required by MLS for authentication or delivery are not feasible. Without these services, the MLS specification is unclear about important details, such as group synchronization, authentication, and secure message dissemination; these are left to implementations to determine [IET23].

Additionally, MLS does not sufficiently defend against the mesh adversary (§3). MLS considers the Internet Threat Model [IET03], which does not consider an adversary who can

shut down Internet connectivity entirely. Furthermore, its threat model does not sufficiently consider threats arising from metadata analysis, which an adversary can use to break the outsider anonymity of groups. In fact, the core MLS specification does not attempt to protect certain group metadata like unique group IDs and epoch numbers [IET23]. As a result, a compromised delivery service or eavesdropper may use this metadata to enumerate unique groups, message frequencies, and message types, leading to inferences on group patterns, membership churn, and other protest dynamics.

Furthermore, MLS does not enforce secure configurations for the issuance of key packages, transport security, ratchet tree distribution, and other design decisions [IET23], which is intended to allow for customization. In our setting, however, relaxed security and privacy configurations can easily lead to identifying group members, behaviors, and dynamics, putting protesters at risk.

4.1 CGKA Definition

We first introduce the definition and syntax of a CGKA, adapted from [KPPW+21], which we will use throughout this section.

Definition 1 (Asynchronous Continuous Group Key Agreement)**.** An asynchronous continuous group key agreement protocol is a tuple of algorithms CGKA = (**Init**, **Add**, **Rem**, **Upd**, **Proc**) with:
$(\gamma) \leftarrow \textbf{Init}(N, ID_0, 1^\lambda)$: A founding member, ID_0, generates their initial group state γ. This initializes the set of members $\mathtt{G} = \{ID_0\}$ and generates a group state γ with a maximum capacity of N participants, where λ is the security parameter.
$(\gamma', S) \leftarrow \textbf{Add}(\gamma, pk_{new}, ID_{new})$: A member of a group can invite a new party, ID_{new}, to join by first sending the current group state in a welcome message, $\mathtt{W}$. After receiving and processing $\mathtt{W}$, the new member inserts themselves into the group $\mathtt{G}$ and sends a protocol message $\mathtt{S}$ to each member in $\mathtt{G}$.
$(\gamma', S) \leftarrow \textbf{Rem}(\gamma, ID_{rem})$: A member removes another, ID_{rem}, by sending a protocol message, $\mathtt{S}$, intended for $\mathtt{G} \setminus \{ID_{rem}\}$.
$(\gamma', S) \leftarrow \textbf{Upd}(\gamma, ID_i)$: A member can refresh key material in γ by sending the corresponding protocol message $\mathtt{S}$ to $\mathtt{G}$.
$(\gamma') \leftarrow \textbf{Proc}(\gamma, S)$: Upon receiving a protocol message $\mathtt{S}$, a member modifies their local group state, γ.

Security game. We use our threat model (§3) to inform our game-based security definition, which can be found in Appendix I.

4.2 Meeting Protester Needs

Enabling group synchronization over a mesh. As noted previously, traditional CGKA protocols require mechanisms for ensuring protocol messages arrive in a consistent total order for all recipients, which is not realistic over a mesh. Amigo applies the idea proposed by [Wei19]; instead of changes to state that require overwriting key material held by nodes in the ratchet tree (e.g., in TreeKEM [BBR18]), state changes *combine* existing and newly generated material, an operation denoted by $\star$. So, members can apply protocol messages regardless of the order. In this way, Amigo can achieve *eventual consistency* of the CGKA state over a mesh as long as all protocol messages are received.

We build this $\star$ primitive in Amigo's CGKA via elliptic curves, where a public key $\mathtt{pk}$ is derived by multiplying a (secret) scalar $\mathtt{sk}$ with the curve's generator point P. Thus, we can instantiate $\star$ as: $(\mathtt{sk}_{x \star y} = \mathtt{sk}_x + \mathtt{sk}_y, \mathtt{pk}_{x \star y} = (\mathtt{sk}_x + \mathtt{sk}_y) \cdot P)$.

Stronger security guarantees. Notions of forward secrecy and post-compromise security for traditional CGKAs [BBR18, CGCG+18] relate to key rotation per epoch rather than

per message. Protester safety may sometimes require stronger guarantees. As a result, we utilize updatable public key encryption (UPKE). Informally, a UPKE scheme is a tuple of algorithms UPKE = (**Gen**, **Enc**, **Dec**) where encrypting a message outputs a new public key alongside a ciphertext, and decryption of a ciphertext outputs the message and associated new secret key. These operations can be used to refresh cryptographic state per message by applying the new key material on the old using our aforementioned $\star$ operation. We define our UPKE and security model formally in Appendix H.

4.3 The Amigo CGKA

With CGKA, $\star$, and UPKE as building blocks, we now show how we can achieve secure group mesh messaging in the large-scale protest setting. Our CGKA implements the Amigo session layer (Figure 1). The core data structure of Amigo's CGKA is a *ratchet tree*, a binary tree where each node holds asymmetric key material [CGCG+18, BBR18]. The leaf nodes are associated with group members, while intermediary nodes up to and including the root allow for (public key) communications to subsets or the entire group respectively. Group members should possess all public keys within the ratchet tree, but only the secret key material of their leaf node and its direct ancestors. As opposed to pairwise approaches where key updates and messages may require linear encryptions, a CGKA protocol utilizing a ratchet tree reduces key updates to logarithmic encryptions and messaging to even a single encryption. Ratchet trees have found practical use as a result (e.g., Messaging Layer Security [IET23]).

Amigo is designed and optimized for group communication. Our CGKA enables parties to form groups on the fly, make changes to group membership, and communicate efficiently within those groups. Members can belong to a single group, or multiple distinct groups each holding a unique group state. This means parties can be members in the same groups without being aware of this fact, a level of privacy useful in a protest setting. Every group $\mathtt{G}$ has its own group state γ, consisting of the following fields:

- $\gamma.\mathtt{members}$: List of members' group-specific identifiers[1]
- $\gamma.\mathtt{pk}$: Vector of all known public keys in the group
- $\gamma.\mathtt{sk}$: Vector of all known secrets in the group
- $\gamma.\mathtt{tree}$: Binary tree mapping between identifiers and keys

State operations transition the group state held by members over periods denoted as epochs, θ. The CGKA operations (Definition 1) enable dynamism, forward secrecy, and post-compromise security by rotating keys held in the ratchet tree via the generation of random seed values Δ_i, corresponding to a unique node in the ratchet tree.[2] A party initiating a state operation generates these seeds and uses them to create and send protocol message $\mathtt{S}$ to the entire group. Upon processing $\mathtt{S}$, members have applied the necessary key updates, evolving their group state from γ to γ', and entering into a new epoch. $\mathtt{S}$ contains the following fields:

- $\mathtt{S}.\mathtt{id}$: Identity of initiator
- $\mathtt{S}.\mathtt{op}$: Type of operation ($op \in \{\mathbf{Add}, \mathbf{Rem}, \mathbf{Upd}\}$)
- $\mathtt{S}.\mathtt{mod}$: Identity of the node to be added or removed[3]
- $\mathtt{S}.\mathtt{seeds}$: Vector of random seeds, where each seed is encrypted under its associated node's public key
- $\mathtt{S}.\mathtt{keys}$: Vector of public keys derived from $\mathtt{S}.\mathtt{seeds}$

[1] These identifiers, e.g., a pseudorandom string, are used in processing changes to group cryptographic state. They are group-specific, so an individual in multiple groups has a different identifier per group. An adversary cannot use one identifier to track members across groups.

[2] For **Add** and **Upd**, these nodes include the leaf and its direct path up to and including the root. For **Rem**, these are the minimum subset of nodes such that every member in the group, excluding the removed party, can decrypt an element in $\mathtt{S}.\mathtt{seeds}$.

[3] Only used when $\mathtt{S}.\mathtt{op} = \mathbf{Add}$ or $\mathtt{S}.\mathtt{op} = \mathbf{Rem}$.

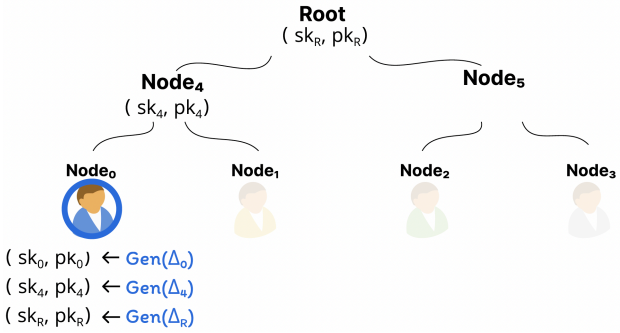

Figure 2: Initializing a group in Amigo. Only member ID_0 is in the group, occupying $\texttt{Node}_0$.

We now describe how Amigo instantiates the CGKA algorithms specified in Definition 1:
Init: As shown in Figure 2, the founding group member, ID_0, generates a ratchet tree of a specified size N. This ratchet tree is initially empty. ID_0 inserts themselves into the left-most leaf in the ratchet tree by deriving separate, randomly sampled Δ_i for themselves and each ancestor node along its direct path to the root. From each Δ_i, a keypair is generated and stored at the appropriate node: $(\texttt{sk}_i, \texttt{pk}_i) \leftarrow \textbf{Gen}(\Delta_i)$, where **Gen** is the generation algorithm of the underlying UPKE scheme (available in Appendix H). Thus, initial group state γ_0 for ID_0 contains the key material they generated, the ratchet tree mapping, and their group specific identifier.
Add: If ID_i wants to add ID_j to the group, ID_j needs the necessary group state to become a member. Thus, ID_i reserves the left-most open leaf node in the ratchet tree for ID_j and sends a *welcome message*, $\texttt{W}$, encrypted to ID_j's public key $\texttt{pk}_i$. This communication can be bootstrapped via an in-person interaction (e.g., via a QR code), in line with how groups are formed on the front lines during large-scale protests [ABJM21a]. $\texttt{W}$ consists of group state γ_i, where $\gamma_{i.\texttt{secret\_keys}}$ only includes the secret keys associated to common ancestors in the path from ID_j to the root node.

Upon receiving $\texttt{W}$, ID_j randomly samples Δ_i for themselves and each ancestor node along their direct path to the root, generating the corresponding key pairs $(\texttt{sk}_i, \texttt{pk}_i) \leftarrow \textbf{Gen}(\Delta_i)$, and using this to update each node's key material on their path to the root. If key material for an ancestor node already exists, it is combined using our $\star$ operator. This is shown in Figure 3. Finally, ID_j sends $\texttt{S}$ to the rest of the group, where: $\texttt{S}_{.\texttt{id}} = ID_i$, $\texttt{S}_{.\texttt{op}}$ denotes **Add**, $\texttt{S}_{.\texttt{mod}} = ID_j$, $\texttt{S}_{.\texttt{seeds}}$ contains the relevant encrypted seeds, and $\texttt{S}_{.\texttt{keys}}$ contains all known public keys. Group members can process $\texttt{S}$ to update their local γ and enter a new epoch.[4]
Rem: Essential to the protest setting is efficient removal from a group, as mass arrests or police infiltration could compromise the keys of group members. A member, ID_i, removes ID_j by generating Δ, deriving a keypair $(\texttt{sk}', \texttt{pk}') \leftarrow \textbf{Gen}(\Delta)$ and using this to update every node in the ratchet tree, rather than just the nodes on their direct path to the root. The remove message is generated by recursing the tree to find the minimum number of subgroups to encrypt to such that everyone except ID_j can decrypt an element in $\texttt{S}_{.\texttt{seeds}}$. For this message, each encrypted Δ in $\texttt{S}_{.\texttt{seeds}}$ may be the same; all remaining members will use this seed to update all nodes within the ratchet tree. Finally, remaining members clear the state of the removed member, $\texttt{S}_{.\texttt{mod}} = ID_j$, by deleting the material held by that

[4]In Amigo, addition is most effective when members join left-to-right, i.e., the most recently joined member adds the new user; upon joining they have all the needed secrets (if they exist) for their ancestor nodes. If this order cannot be enforced, at a minimum a newly joined member will have the group secret, which they can use to communicate with the group and query nodes for the relevant secrets they are missing. We sketch a mechanism for learning missing keys in Appendix E.

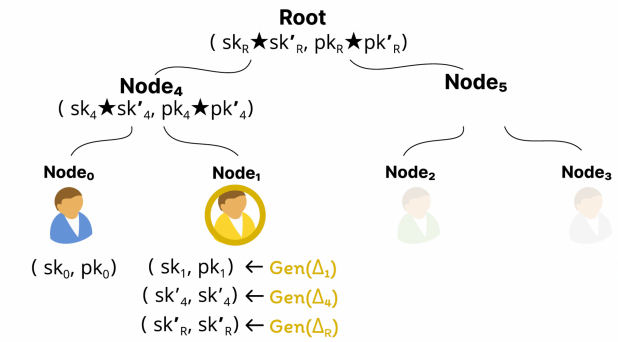

Figure 3: A new member, now occupying `Node`$_1$, joins the group thus updating the ratchet tree.

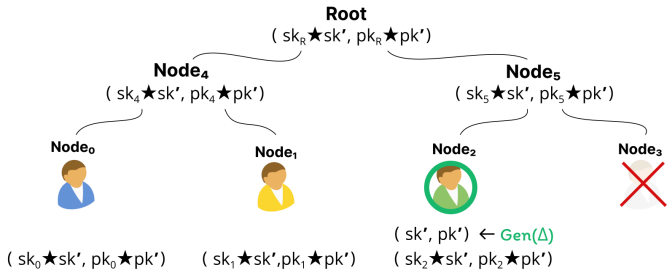

Figure 4: The member occupying `Node`$_2$ removes the member occupying `Node`$_3$ from the group.

leaf, illustrated in Figure 4.

Upd: Consistent and frequent updates lead to fresh key material for the updating nodes and their path to the root, ensuring stronger notions of forward secrecy and post compromise security for themselves and the group. A member ID_i initiates **Upd** by first randomly sampling Δ_i for themselves and their direct ancestors. The key material derived from each Δ_i is used to update the relevant node's key material and generate an update message intended for the group, similar to the process for addition. After processing this update, the updating member and the entire group enter a new epoch with a fresh group state.

Proc: When a member initiates a state operation, other members of the group need to process the corresponding protocol message `S`. When a member receives this protocol message, they first determine $\texttt{S}_{\texttt{.op}}$ and apply the operation to their local state γ, effectively moving it forward to γ' and entering a new epoch θ'. This entails updating (either combining or doing an initial write) the appropriate nodes in the ratchet tree using $\texttt{S}_{\texttt{.seeds}}$, and $\texttt{S}_{\texttt{.keys}}$. Depending on $\texttt{S}_{\texttt{.op}}$, this may also entail changes to $\gamma_{\texttt{.members}}$ and $\gamma_{\texttt{.tree}}$.

Security proof. We adapt our proof from that of [KPPW+21] within the framework introduced by [JKK+17]. We first prove security in the *selective* setting, in which the adversary is required to corrupt all the users at the beginning of the security experiment, even before receiving the public keys of the system. More realistically, however, an adversary could *adaptively* choose to corrupt nodes over a period of time, based on some strategy, so we also consider this setting. Details are available in Appendix I.

4.4 Sending and Receiving Messages

Amigo's CGKA effectively manages group cryptographic state via state operations, so adding secure messaging on top is straightforward and done at the application layer.

Members can chat with each other via the operations **send-message** and **read-message**. A member runs **send-message** to generate $\mathtt{A}$, where $\mathtt{A}_{.\mathtt{payload}}$ contains the actual message content; upon reception, intended recipients run **read-message** on $\mathtt{A}$.

To provide *outsider anonymity*, every message sent within the group, whether from the CGKA ($\mathtt{S}$) or otherwise ($\mathtt{A}$) is encrypted under the group symmetric key (held by the root) and does not contain any identifying metadata. A party who is not an intended recipient has no information about the originator or destination of the message. As a result, parties utilize *trial decryption* to determine whether a message is for them; members will attempt decryption, first with the group symmetric key, before further processing.

To encrypt a message to the entire group, a group member derives a symmetric key from the root node's key material. The member concatenates these keys, hashes them, and passes this as input to a KDF where: $key_i \leftarrow KDF(H(\mathtt{pk}_i||\mathtt{sk}_i))$. The underlying authenticated encryption algorithm we use is AES-GCM.

Better security through UPKE. Encrypting solely under the group symmetric key provides forward secrecy and post-compromise security at an epoch granularity, i.e., until a state operation is processed. Protesters, however, may require more frequent key rotation, e.g., in protests where the threat of compromise is higher. We provide the option for this increased level of protection by incorporating UPKE in our CGKA. At the expense of efficiency, group members can encrypt messages via the UPKE keypairs in the CGKA ratchet tree. Thus, forward secrecy and post-compromise security can be achieved per message within the current epoch.

When using UPKE for messaging, $\mathtt{A}_{.\mathtt{payload}}$ is a UPKE ciphertext, which contains the new public key $\mathtt{pk}'$ resulting from UPKE encryption. Intended recipients know where to apply $\mathtt{pk}'$ and the corresponding secret: the node whose key material successfully decrypted $\mathtt{A}_{.\mathtt{payload}}$. Our key combination operator, $\star$, can be used to combine the newly generated material with the old, effectively rotating keys per message. But, excluded recipients will not know where to apply $\mathtt{pk}'$ in this way. We discuss different strategies to address synchronizing public key state in Appendix H.

Handling mesh conditions. Although our CGKA allows us to provide secure, concurrent group communication and state management in a decentralized mesh setting, there are subtle but important deployment challenges due to the lack of a centralized delivery service. For instance, handling concurrent group additions may require a specific point of control for group admission, e.g., a group administrator, as used by protesters [ABJM21a]. We discuss how Amigo handles these and other challenges in Appendix G.

5 Routing in Mesh Messaging

Any message must be *routed* over the mesh. In Amigo routing, a *message* represents (1) user-generated content or (2) cryptographic material for the CGKA protocol. Each message is tagged with a time-to-live (TTL). A message's destination is either a group or an individual user (a size-1 group).

Each Amigo node has a *message buffer* which stores a single copy of each unique message the node has recently received. When a node wants to send a message, the node inserts the message into the buffer, evicting the oldest message in the buffer if necessary to make space; given the choice between a CGKA message and a non-CGKA message, Amigo prefers to evict the latter, as CGKA messages are important in synchronizing group states. A node periodically decrements the TTL field of all messages in the buffer, evicting messages when their TTLs reach 0.

When two nodes discover each other through physical proximity, the nodes use a *routing protocol* to determine whether and how to share their local message buffers. Amigo can use four such routing protocols – three from prior work in mesh networking, and a

fourth, dynamic clique routing, which Amigo introduces, tailored to the mobility patterns of protests. We now describe these four in detail.

Epidemic flooding. In this approach, each node divides time into 60-second intervals, transmitting a *beacon* at a random time within each interval to avoid Layer 2 collisions. If a peer receives such a beacon, the peer establishes a Layer 2 channel with the beacon originator, and the two nodes exchange their complete message buffers. Prior work has used epidemic flooding as a baseline for evaluating more sophisticated mesh routing protocols [PSEB22, PJW+22, LFBD+16, BRT23]. Epidemic flooding aggressively probes the network for routes between senders and receivers, but risks network congestion due to the indiscriminate nature of message forwarding.

Digest flooding. To prevent two peers from exchanging messages that are already present on both nodes, each peer can first exchange Bloom filters [Blo70] which summarize the locally-buffered messages; a peer X only sends a particular message to peer Y if Y's Bloom filter indicates that Y does not already store the message. Perry et al. [PSEB22] first propose this optimization to epidemic flooding, referring to a Bloom filter as a *message digest*. Details about Amigo's Bloom filter implementation can be found in Appendix C.

Static clique routing. Perry et al. [PSEB22] sketched an optimized form of digest routing based on *cliques*. The basic idea is that each node belongs to a clique, with one node in each clique serving as the *clique leader*.[5] When node X wants to send a message to node Y, X sends the message to X's clique leader; the clique leader then exchanges messages with other clique leaders via digest routing, with the hope that the message will eventually reach Y's leader, who will then transmit that message to its clique members (including Y). Users form a static clique, and choose a clique leader, prior to the protest. A node's leader is static and does not change in response to the spatial arrangement of nodes or other properties of the mesh.

Intuitively, static clique routing should decrease overall network pressure compared to flooding, since only clique leaders perform bulk message exchanges. This approach might result in worse end-to-end delivery rates if clique leaders do not encounter each other often, or if clique members are not near their leaders.

Dynamic clique routing. To mitigate the problems mentioned above, Amigo introduces *dynamic clique routing*, in which both a node's clique and clique leader can change over time.

To determine a node's dynamic clique, Amigo divides the 2D plane into adjacent grid squares, of 3m width. A node uses GPS to identify its current grid square; all members within the same grid reside in the same dynamic clique. Amigo divides time into *rounds*, where each round requires the selection of a clique leader. The round length balances maximizing channel utilization for message delivery and ensuring swift recovery when bootstrapping is unsuccessful. For most mobility models, cliques remained largely stable over a five-minute period; therefore, Amigo uses a five-minute round length. The first minute of a round is reserved for clique leader election; the remaining time is used to exchange messages.

Amigo's leader elections are based on Android's implementation [And24]. Nodes choose a "willingness" value (0-100), indicating their readiness to serve as a clique leader. The node in each given region with the highest "willingness" value is elected as leader. Each node picks a random offset within the election period of the round, and generates a beacon containing the node's approximate location, the node's Layer 2 identifier, and the leader "willingness" value; the random offset decreases the likelihood of Layer 2 collisions.

A node also listens for beacons from other peers. At the end of the election period, a node selects the beacon-generating peer with the highest leader willingness in their region

[5]We note that routing-layer *cliques* are separate from the application-layer *groups* used for messaging. Due to the protections from our CGKA (§4.1), even if an adversary learn cliques, they learns about clique members' groups or communications.

to be its clique leader. If a node hears no other beacons, it becomes its own clique leader.

Under perfect conditions, all nodes in a clique will appoint the same leader—the node with the highest “willingness" value; however, if beacons are not consistently received, nodes may not elect the correct node as a leader. If so, a node might believe itself to be the leader based on its received beacons, even though another node was actually elected leader by the others. Thus, this “false leader” node would not receive any messages for exchange, and essentially sits out the rest of the round. This node can then try again by participating in the leader election for the next round.

The “willingness" value is vulnerable to attack; malicious nodes could send high “willingness" values to be elected clique leaders, and subsequently disrupt traffic. While we do not address this problem in Amigo, we note that for added robustness, nodes could instead include random values in beacons to determine the leader, using methods such as distributed randomness beacons [CMB23].

Empirically, a protest may alternate between moments of tight, highly-ordered spatial density (e.g., when protesters form a human chain), interspersed with less dense spatial arrangements and/or more chaotic mobility patterns (e.g., when protesters randomly scatter due to the sudden arrival of law enforcement). We expect dynamic clique routing will better adapt to these patterns than static clique approaches.

Our dynamic routing protocol self-organizes using location data, but this use does not leak additional information to a global eavesdropper. Such an eavesdropper already knows the location of each node because eavesdropping implies physical proximity to a victim node, and an eavesdropper knows its own location.

Routing privacy. While our CGKA provides the privacy properties required for mesh messaging, our routing protocols also have certain privacy implications. We discuss these implications further in Appendix J.

6 Modeling Protester Dynamics

To rigorously evaluate the performance of a secure mesh network, we need realistic models of user and network dynamics.

6.1 Mobility Models

A mobility model describes how protesters move through a physical space. The model determines how the spatial density of protesters varies over time, and how long protesters linger within the communication radii of each other's short-range radios. Accurate mobility models are crucial for evaluating mesh networks because different mobility models have a dramatic impact on when (and whether) full or partial spanning trees for routing will emerge.

Unfortunately, mobility models from prior work are not well-suited for analyzing protest dynamics. For example, Perry et al. [PSEB22] considered spatial densities of one person every 2.5-15 ft^2, with each person located at the centroid of a grid square, and with a communication radius of 10 m. This communication radius would mean that each protester could directly contact 400 peers. A protest's high density area might have as little as $0.2\,m^2$ per protester, however; if protesters were arranged in a grid, then 1,600 protesters would be within the communication range of any particular individual [Sti19]. Furthermore, in a large-scale protest, multiple kinds of mobility patterns are likely to occur through space and time, as different protesters responds to different stimuli (e.g., the arrival of police or more protesters) in different parts of the protest region.

To build more realistic mobility models, we draw insights from a crowd-sourced guide to protesting, written in 2019 by Hong Kong anti-ELAB demonstrators [Ano23]. The document provides practical advice about the planning and operational execution of a

protest, describing various spatial arrangements that protesters can use. Based on this information, we create several mobility models rooted in protest scenarios, which we describe below.

Marches. A march involves a group walking together in the same direction. In our march mobility model, we assign a starting area for the march, randomly distribute nodes within the starting area, and have all nodes move in the same direction at approximately 1.3 m/s [MMA+21]. The spatial density is about 2 m^2 per protester.

Human chains. A human chain entails a line of stationary people who pass materials from one end of the line to the other. Typically, the line begins in a storage area, and ends in front of the protest [Ano23]. Unlike a march, there is no mobility in a human chain, and the spatial density is sparser, with nodes in a line about 2 m apart.

In our human chain mobility model, we pick two points within the simulated space, and generate a pathway between them. This pathway is not a direct line – instead, we add randomness to account for obstructions between the two points and general stochastisity in the locations of people along the line. We implement this randomness by selecting 0-10 additional random points between the two endpoints, and applying subtle curves to the paths originally generated with linear interpolation. Finally, we assign each protester to a location on the chain, spacing the protesters as above.

Gatherings. In the Hong Kong protest guide, the authors state that protests can also take the form of "gatherings with a specific focus." Examples of such gatherings are musical performances and communal art projects [Ano23]. These kinds of protests involve crowds in which individual participants generally stay within the congregation area, but can migrate within that area at walking speeds.

In our gather mobility model, we assign an area (roughly a city block in size) where an event will take place. We randomly assign nodes to occupy the space, and select a subset of nodes to move randomly within the space (as random waypoints [Joh96]).

Blockades. A blockade occurs when protesters prevent entrance to a given space, e.g., anti-ELAB protesters blockaded the Cross-Harbor Tunnel to stop police from entering a university [PL19].

We implement two blockade-based mobility models. The first represents an "object blockade" in which demonstrators place objects along the blockade frontier. For example, Hong Kong protesters scattered road blocks and then set them ablaze. In our object blockade mobility model, protesters move at roughly 1.4 m/s between areas where materials are stored and the protest frontier. At each location, protesters briefly pause to pick up or drop off material.

Our "human blockade" model captures a scenario in which protesters have gathered near a blockade frontier, and are being monitored by law enforcement. In this model, clumps of police officers move towards the frontier, as do the protesters. If a police officer and a protester get within half a meter of each other for 30 seconds, one of them will retreat to the nearest clump of like-minded individuals; this behavior represents a participant being injured or physically overwhelmed by a member of the opposing group.

Additional models. For comparison, we also implement the static and random waypoint mobility models, which are standard in the literature. For more details on these models, see Appendix B.

6.2 Physical Layer Simulations

Prior work [PSEB22, PJW+22, LFBD+16] uses custom network simulators to evaluate mesh routing protocols. Unfortunately, these works do not model physical-layer events like message collisions between two peers within communication range who transmit data simultaneously. Because prior simulations did not capture these Layer 1 effects, they overestimated the available bandwidth at Layers 2+.

To avoid this problem, we evaluate Amigo using ns-3, a discrete-event network simulator which *does* model important low-level characteristics. We extended ns-3 to support Wi-Fi Direct, a Layer 1+2 technology that, like Bluetooth, allows devices to communicate in a peer-to-peer fashion. We built our Wi-Fi Direct implementation atop ns-3's preexisting support for ad-hoc Wi-Fi; implementation details are in Appendix A. Our implementation uses 802.11ax/Wi-Fi 6, with a maximum data rate of 143.4 Mbps for advertising (MCS Index 11), and 263.3 Mbps for data exchange (MCS Index 6). This network setup is feasible on modern smartphones and representative of Wi-Fi Direct, which can provide up to 250 Mbps [Wi-21]. We limit signal propagation to 10 m, as in previous work [PSEB22].

We found ns-3 hit scaling limits as we increased the simulated protest size. For example, to complete a single run of a 250 node simulation using a static mobility model and digest routing, we required ∼10 GB of memory and one week of wall clock time, even on a high-end cloud instance (GCP `c3d-highmem-180`). Nonetheless, as we demonstrate in Section 7, even modestly-sized protests can trigger crippling network congestion in standard approaches for secure mesh routing (and sometimes in the Amigo-specific dynamic clique approach), suggesting important areas for future study (§9).

6.3 Traffic Model

Our traffic model expresses the rate at which application-level behavior generates application messages and CGKA messages (§4).

For a given simulation lasting an hour of simulated time, we generate 20 CGKA messages, each sent from a random group member and at a random time in the first half of the simulation; the latter constraint ensures the message has sufficient time to propagate to the other group members before the simulation ends. This is roughly equivalent to each group adding a member, removing a member, and performing a key refresh during a protest.

Non-CGKA traffic corresponds to standard text messages sent to a group. We implement a basic traffic model for non-CGKA messages: each node in the simulation sends a message every minute. We choose these values based on the protest manual, which states chats see an excess of 100 messages/min during an event's peak [Ano23].

To generate the senders and receivers for messages, we create group chats of 25 members, based on nodes' locations: we assume 0.8 of a clique is nearby the node, and select the other 0.2 randomly. We imagine many members of a close-knit group chat may enter the protest together, and a fraction may attend separately, or remain home or nearby to perform wellness checks. Mobility models may cause group members to move physically apart from one another during the simulation; for example, to carry supplies, talk to someone new, etc. We assume people within a group chat send messages to each other. Each message is 250 bytes and has a TTL of 5,000 seconds—slightly less than the length of an average action [Ano23].

7 Results

We now discuss our extensive end-to-end benchmarks evaluating Amigo's feasability in realistic protest scenarios.

7.1 CGKA Microbenchmarks

Amigo is intended for deployment atop resource-constrained devices. To evaluate the resource consumption of Amigo's CGKA, we ran microbenchmarks on both a 2020 Moto E phone (1.8 GHz octa-core, 2GB RAM) and a Raspberry Pi 4 with a 1.8 GHz quad-core ARMv8 SoC and 8 GB RAM. The smartphone emulates low-end performance while the

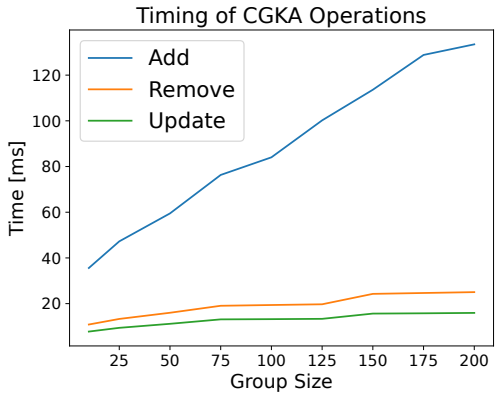

Figure 5: Timing of Amigo CGKA operations.

Table 1: Average packets sent of various type across various routing mechanisms; results averaged across 3 runs of static mobility models for 5,000 KB buffer size. Packet delivery rate in parentheses.

	Epidemic Flooding	Digest Flooding	Static Clique Routing	Dynamic Clique Routing
Messages	422,623,852 (0.21)	411,590,675 (0.31)	2,030,682 (1.0)	13,522,625 (0.98)
Digests	-	24,087,630 (0.95)	472,836 (1.0)	776,972 (1.0)
Willingness Beacons	-	-	-	113,859
Message Beacons	-	-	-	4,800
Link Layer Beacons	6,000	6,000	6,000	1,200
Probe Requests	575,236 (1.0)	574,816 (1.0)	11,258 (1.0)	114,359 (1.0)
Probe Responses	574,064 (1.0)	573,609 (1.0)	11,258 (1.0)	113,875 (1.0)
Total	423,779,152	436,832,727	2,532,033	14,647,688

Raspberry Pi offers similar specifications to medium/high-end modern smartphones. We used the Criterion [cri21] framework for the benchmarks, running each for a minimum of 1000 iterations.

Are CGKA operations fast? As discussed in Section 2, updates and removals have to be fast; when a group member is compromised, e.g., because their phone is confiscated by authorities, all other group members are at risk. Figure 5 shows, in a 200 person group, Amigo on a low end budget phone can perform removal and updates in under 26 ms. Addition is much slower, but adding a new member to a group is not as critical as removing a compromised one; furthermore, we expect additions will be infrequent. At a group size of 200, on our Android phone we could execute roughly 39 member removals, 62 key refreshes, or 7 member additions in one minute. Performance was significantly better on the Raspberry Pi. See Appendix D for detailed results.

What is the CGKA-induced network load? For non-CGKA messages, Amigo is more efficient than pairwise group approaches [Sig14] where the same message is sent encrypted multiple times, once per each symmetric key negotiated with an individual group member. For state operations, Amigo generates one message for each removal or update. Member addition requires two messages: a welcome message to bootstrap in the new member, and an addition message that brings the group into a new epoch with updated group state. See Appendix D for more detailed results.

Is Amigo battery efficient? Appendix D contains a complete discussion. For now, we note Amigo's energy consumption is reasonable and will not unduly drain a phone's battery.

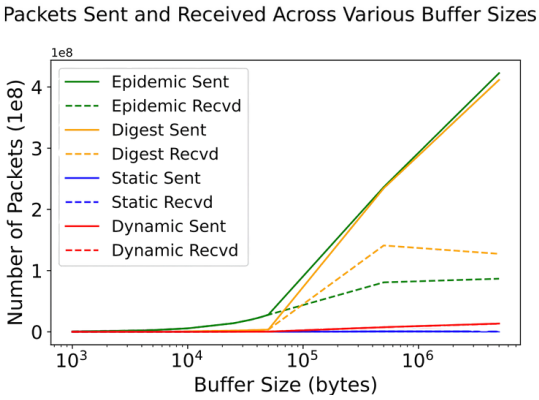

Figure 6: Packets sent and received by routing protocol, for 100 nodes in a static mobility model; average of 3 runs.

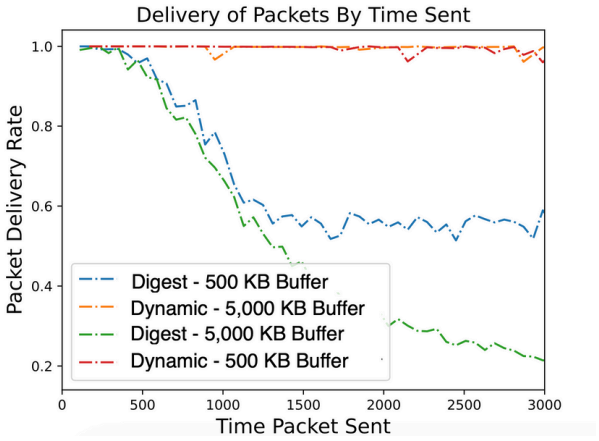

Figure 7: Average delivery rates for packets sent at time t for our digest routing and dynamic routing, for 100 nodes in a static mobility model.

7.2 Network Congestion and Stability

How does buffer size impact network saturation? Figure 6 shows the packet volume generated by various buffer sizes, with all other aspects of the simulation held constant (e.g., traffic model, mobility model, etc.). Table 1 breaks down the various packet types sent, and their end-to-end delivery rates, given a 5,000 KB buffer.

As expected, flooding generates significantly more traffic than other approaches; as buffer size increases, the difference grows (Figure 6). The bulk of the traffic arises from messages and digests, not from Layer 2 management packets (i.e., beacons, probe requests, and probe responses).

Notably, end-to-end delivery rates for epidemic routing and digest routing are very low (e.g., ∼30% for digest routing). The reason is that both schemes gradually congest the network and then make no attempt to reduce their sending rate, even once message buffers have become steady-state full. When buffers become full and sending rates do not back off, new messages are constantly added to buffers, reducing the effectiveness of the digest optimization. As a result, nodes often exchange complete or near-complete copies of their local buffers, resulting in larger Layer 2 messages and more Layer 1 collisions (triggering retransmissions of the associated Layer 2 messages). The net result is, as time progresses,

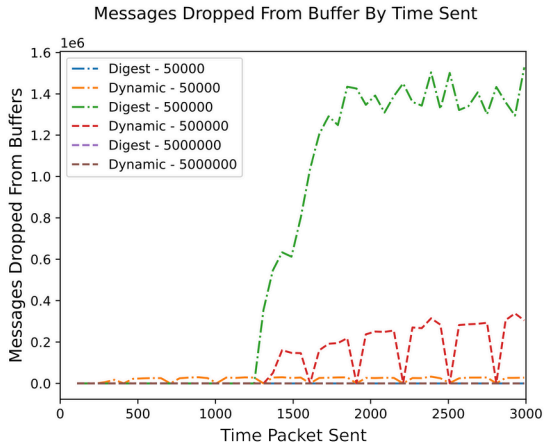

Figure 8: Messages evicted in digest and dynamic mechanisms; 100 nodes in gather mobility model, bucketed into 60s intervals.

the expected delivery rate (Figure 7) for packets sent at time $t+1$ trend monotonically lower than the expected delivery rate for packets sent at time t.

The relatively poor performance of digest routing is surprising, given previous work [PSEB22] introduced digest-based routing as an optimization. However, by modeling realistic Layer 1 and Layer 2 effects, we see with epidemic routing, only message packets collide with themselves, whereas with digest routing, message packets *and digest packets* collide, exacerbating the effects of network saturation. Thus, as shown in Table 1, digest routing's performance is not superlatively better than that of vanilla epidemic routing.

Note that the network saturation in Table 1 represents a scenario in which each Layer 1 collision domain contains 100 protesters. A high-density protest could have as little as $0.2\,\mathrm{m}^2$ per protester [Bre], meaning 1,600 protesters would be in a single collision domain and would suffer from an even more intense congestion collapse.

How does buffer size impact end-to-end delivery rates? All of Amigo's routing schemes use per-node buffers to hold the messages nodes exchange with each other. Given a fixed mobility pattern and traffic matrix, decreasing the buffer sizes will result in fewer messages exchanged when nodes encounter each other, resulting in less network congestion; however, due to contention for buffer space, older messages (i.e., messages with lower TTLs) in a buffer will be evicted more aggressively to make room for new messages, potentially reducing how often messages are shared (and thus potentially reducing end-to-end message delivery rates). Conversely, increasing the buffer size will give messages more opportunities to be transmitted, potentially increasing message delivery rates; however, the resulting increase in traffic volume may induce network congestion that *decreases* end-to-end delivery rates. The tensions between increased congestion and increased delivery rates materialize in different ways for different protest environments.

Figure 8 is a time-series view of the number of messages evicted from node buffers during simulations using the gather mobility model with dynamic routing or digest routing. For 5,000 KB buffers, no evictions occur. For 500 KB, buffer evictions begin to increase at approximately $t = 1200$. Eviction rates remain high, with a brief hiatus for dynamic routing during advertising periods.

This eviction behavior is relevant to Figure 9, which shows the delivery rates for packets sent at various time intervals. We see delivery rates are initially high across all routing mechanisms for buffer sizes of 500 KB and 5,000 KB, but at approximately $t = 1200$ delivery rates become consistently poor. The reason is, once buffer evictions start at $t = 1200$, the effective number of transmission chances a message gets will decrease, hurting the

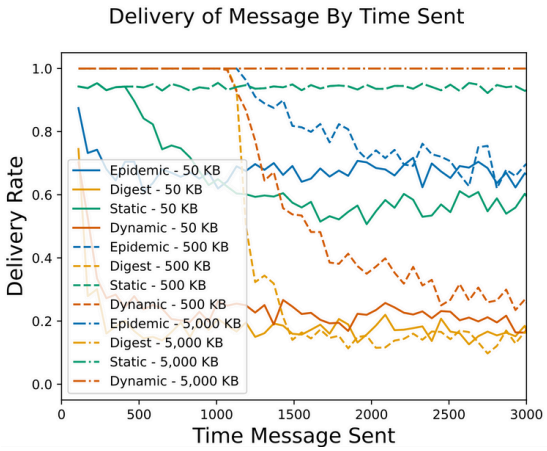

Figure 9: Message delivery for the gather mobility model based on initial send times, bucketed into 60s intervals.

Table 2: Total message packets sent for all routing mechanisms and mobility models; results averaged across 3 runs for 5,000 KB buffer size. Message delivery rate in parentheses. Checkmarks if CGKA group consistent at the end of the simulation.

	Epidemic Flooding	Digest Flooding	Static Clique Routing	Dynamic Clique Routing
static	422,613,855 (1.000) ✓	411,579,195 (1.000) ✓	596,074 (0.705) ✗	13,555,092 (1.000) ✓
random	425,916,153 (1.000) ✓	420,206,611 (1.000) ✓	2,073,922 (0.980) ✗	8,626,361 (0.963) ✗
gather	416,506,046 (1.000) ✓	405,078,143 (1.000) ✓	1,964,185 (0.941) ✗	12,956,503 (1.000) ✓
chain	34,706,437 (1.000) ✓	34,732,147 (1.000) ✓	36,241 (0.017) ✗	14,378,903 (0.999) ✓
march	50,609,963 (0.867) ✗	50,325,119 (0.867) ✗	29,145 (0.026) ✗	6,208,141 (0.726) ✗
blockade1	11,073,410 (0.271) ✗	11,079,296 (0.271) ✗	18,214 (0.008) ✗	87,310 (0.025) ✗
blockade2	435,542 (0.033) ✗	434,633 (0.033) ✗	8,286 (0.005) ✗	12,380 (0.006) ✗

message's likelihood of being end-to-end delivered. For the simulation settings in Figure 9, the network has not become fully utilized before buffer evictions start occurring; this means, for the gather mobility pattern examined in Figure 9, a buffer of 500 KB is unable to fully utilize the available network bandwidth, resulting in unnecessary degradation of end-to-end delivery rates. The overall takeaway is that, even for Amigo's protest-optimized dynamic routing, the lack of on-the-fly adjustment to buffer dynamics and network congestion leads to non-optimal routing performance. We discuss further in Section 9.

7.3 Protester Mobility and Network Partitioning

How does protester mobility impact message delivery? Table 2 explores the number of packets sent and message delivery rates for the routing protocols and various mobility patterns when using a 5,000 KB buffer. Simulations ran for 3,600 seconds, but Table 2 only considers messages sent before $t = 3000$ so as to not mark as dropped a message that was in-transit at the end of the simulation and would have eventually been delivered if the simulation had continued. Table 2 does include the "warm-up" period at the beginning of the simulation, because such a warm-up would also occur at the beginning of a real protest.

As shown in Table 2, static clique routing was the worst performing approach across all mobility patterns. The reasons were that (1) a clique leader often had trouble encountering its clique members or other clique leaders, and (2) peers had no way to attach to a better-suited leader. Ignoring the blockade mobility patterns for a moment, epidemic routing and digest routing outperformed dynamic clique routing because those algorithms

did not trigger congestion collapse for the mobility patterns and node counts explored in the simulations; as a result, the more aggressive forwarding behavior of these algorithms (compared to that of dynamic clique routing) led to better end-to-end delivery rates.

Static clique routing's performance for the chain and march patterns were essentially the same. For the other three routing protocols, the march model was more challenging than the chain model; the reason being, while the chain model is static, the march model is dynamic, making message transmission and clique dynamics harder to maintain.

Table 2 shows that all four routing protocols struggled with blockade mobility patterns. In blockade1, protesters retrieved materials to form a barrier made of physical objects, whereas in blockade2, protesters generally approach a police line, and then spread out due to police intimidation. Both models had low spatial density, with nodes often ten meters apart or more, reducing chances for nodes to hear beacons (or exchange messages more generally).

Overall, we note that our new dynamic clique routing protocol achieves 95% or higher delivery rate on multiple mobility models (including some of the protester-oriented ones) while sending far fewer messages over the course of the simulation. This differences ranges from ∼2× to ∼50×, depending on the mobility model evaluated. In other words, our simulations show that dynamic clique routing can achieve similar levels of message delivery at a fraction of the messages sent – important for smartphones, which are constrained by battery size and CPU capability, as well as for overall network stability, as discussed in Section 7.2.

Can protester groups maintain key agreement? The strong protections of Amigo's CGKA protocol are moot in practice if Amigo nodes struggle to maintain CGKA key state due to an unreliable routing layer. Amigo's CGKA protocol tolerates out-of-order messages (§4.2), but it does not tolerate a group member *never* receiving a CGKA message—in this scenario, the member will be unable to continue in the group. As discussed in Section 5, Amigo prioritizes the delivery of CGKA messages, but is this policy sufficient to allow the CGKA protocol to succeed in practice?

Table 2 shows that dynamic clique routing achieves comparable message delivery rates with far fewer messages, and achieves CGKA convergence for most of the same cases as the flooding protocols, but not for the random mobility model.

To understand Amigo's support for highly dynamic group states, we increase the number of group updates that occur throughout the simulation, modifying our traffic model to include 5× more group update (CGKA) packets. In our experimental setup, this equates to each group experiencing a change in group state every minute. Under these conditions, total message and group state update delivery rates are consistent with those of less dynamic groups. Our complete results are in Appendix K.

Amigo's CGKA can achieve eventual consistency if all CGKA packets are delivered (§4.2); flooding protocols are highly redundant, so they are more likely to achieve this state (i.e., perfect CGKA message delivery) with enough time. But, dynamic clique routing still has a relatively high delivery rate. In the event a member does miss a CGKA message, a straightforward solution is to be re-added to the group by bootstrapping key material with another group member. We believe decentralized CGKAs that are robust to low message delivery rates is an interesting direction for future work.

8 Related Work

Below we discuss related work in the context of mesh messaging, routing, and key agreement for protest scenarios.

Mesh messaging in protest scenarios. Table 3 summarizes prior work in mesh messaging and compares it to Amigo.

Table 3: Related work in mesh messaging. E2E: end-to-end encryption; FS: forward secrecy; PCS: post-compromise security; Anon: anonymity; Pair: pairwise communication; Groups: Group communication.

Scheme	E2E	FS	PCS	Anon	Pair	Groups
Bridgefy [Bri22]	✗	✗	✗	✗	✓	✗
Moby [PJW+22]	✓	✓	✗	✓	✓	✗
ASMesh [BRT23]	✓	✓	✓	✓	✓	✗
Perry et al. [PSEB22]	✓	✗	✗	✓	✓	✓
Rangzen [LFBD+16]	✗	N/A	N/A	✓	✗	✗
Anix [KB25a]	✗	N/A	N/A	✓	✗	✗
Amigo (this)	✓	✓	✓	✓	✓	✓

Bridgefy [Bri22] is a mesh network supporting pairwise messaging via flooding. Though Bridgefy claims to provide anonymity and end-to-end encryption via the Signal protocol, significant vulnerabilities were discovered which put these claims into question [ABJM21b, AEP22].

Moby [PJW+22] is a flooding-based mesh network that supports pairwise messaging. Moby uses Signal's symmetric ratchet [Sig20] to provide end-to-end encryption with forward secrecy. Anonymity is provided through HMAC validation; each pair of communicating nodes shares a key, and a node determines whether a received message is destined for itself by seeing whether any of the node's pairwise keys successfully recreates the message's HMAC. Moby uses Signal's symmetric ratchet [Sig20] to provide end-to-end encryption with forward secrecy. Anonymity is provided through HMAC validation trials. Using this HMAC validation trick, Moby avoids the need to embed cleartext sender or receiver IDs in messages, enabling anonymity.

ASMesh [BRT23] augments Signal's Double Ratchet algorithm for stronger properties and routes messages using flooding.

Perry et al. [PSEB22] use flooding to provide both end-to-end encryption and anonymity via a CPA-secure signcryption scheme. This scheme supports group communication by having a group creator generate a single secret key and share it with each member of the group, but does not provide additional security. To protect anonymity from traffic analysis attacks, parties constantly send a message at a fixed interval, using "dummy" messages if needed.

Rangzen [LFBD+16] propagates messages via epidemic flooding, giving preference (i.e., more local buffer space) to "trusted" messages. Trust is determined by examining how many mutual friends users share in a social graph maintained out-of-band. Rangzen does not end-to-end encrypt messages because Rangzen is intended to spread "micro-blogs" (akin to tweets) that are destined for all users.

Anix [KB25a] is also a "micro-blog" platform over flooding, introducing notions of trust via ephemeral but linkable pseudonyms. Parties can remotely allow their messages to be connected back to them by "trusted users" while remaining anonymous to everyone else.

Mobile ad-hoc networks. In mobile ad-hoc networks (MANETs), mobile devices communicate with each other in an infrastructure-less setting. MANETs are characterized by peer-to-peer communication where nodes can move freely and the network topology may change frequently. There is rich literature focusing on *multicast* MANET protocols: settings where senders can communicate with multiple nodes using a single destination address (e.g., [RP99, LSG02, OKS01, DNRC11, RASJ05, CNR07]). We refer readers to [BK09] for a full survey. While specific details differ, multicast protocols generally provide functionalities for joining/leaving multicast routing groups and maintaining routing state. Multicast protocols aim to establish *forwarding paths* between a source and receivers that

allow messages to be efficiently propagated through the network.

Most MANET routing protocols do not consider anonymous and private route construction/maintenance. Multicast groups are typically addressed by persistent identifiers [BK09], which our adversary (§3) can collect (passively or actively) to identify groups and infer their behavior. Also, routing information sent in the clear and cached routing tables can leak topological network information to an adversary [KLH+07, BK09]. Previous literature has shown that this topological view of the network can be used to reveal the approximate physical locations of nodes [NN01, SRZF03]. While there has been work on securing multicast MANET protocols from attacks [DNRC11, RASJ05, CNR07], they do not adequately discuss anonymity and privacy in these settings.

There is a promising line of work exploring anonymous route constructions for MANETs [KH03, BEKXK04, ZWK+04, YJW06, SKY05, THH15, KM07, Bao07]. These works typically provide properties like sender/recipient unlinkability, route untraceability, and location privacy. These protocols are most effective when there exist relatively stable forwarding paths that remain valid during multicast sessions. However, our simulations suggest that such topological stability is far from the norm. Our results under protest-specific movement and traffic patterns (§7) suggest that handling congestion and high mobility is a challenge in smartphone mesh messaging approaches, and one that must first be addressed before anonymous MANET protocols can be effective in the environments we consider. [6]

Group key agreement in wireless networks. A line of work [DANR09, ZSXJ04, BRZV05, KLNY03] focuses on providing mechanisms to establish and rotate group keys, achieve dynamic group membership, and provide data confidentiality in wireless mesh and ad-hoc networks. These works, however, leverage assumptions and models that differ significantly from those underlying Amigo. Dong et al. [DANR09] provide secure group communication by taking advantage of stationary wireless routers that form a network backbone, infrastructure Amigo cannot assume is present. Zhu et al. [ZSXJ04] provide group rekeying via a keyserver node, a single point of failure where a compromise could put all members of a group at risk indefinitely. Balachandran et al. [BRZV05] incur issues with scalability in our setting, requiring a number of messages linear in the group size to refresh keys. Kaya et al. [KLNY03] treat group member revocations as rare, requiring reasonably more computation and bandwidth resources. Protester safety, however, requires removals to be fast and efficient (§3).

9 Discussion

Secure communication protocols require participants to create and destroy cryptographic keys. Messages associated with key management are exchanged over the same network that handles regular message traffic. Thus, the reliability of the network impacts the reliability of key management. In traditional (non-ad-hoc) networks, a Layer 4 security protocol like TLS can rely on the Layer 3 TCP protocol to provide dependable, in-order message delivery and avoid network congestion. However, the ephemeral, unpredictable nature of node connectivity in a mesh network makes it hard for Layer 3+ protocols to understand (and react to) network dynamics like congestion in real time. Prior work on secure mesh routing has focused on the details of key management, and made simplifying assumptions about the behavior of the network layer (§6.2). However, our results in Section 7 demonstrate that low-level network behavior has a critical impact on the performance and correctness of a secure mesh network. For example, Layer 1 phenomena like collisions and multi-path interference will frequently be triggered in real-life protest situations involving densely-packed people

[6] We hope our high-fidelity simulations (§6) can guide the design of future anonymous, multicast MANET protocols in smartphone mesh networks during Internet shutdowns.

and physical obstacles like buildings. Furthermore, Layer 2 network partitions in the peer-to-peer spanning tree can also be triggered by realistic movement patterns. The limited channel count of real-life peer-to-peer radio technologies like Wi-Fi Direct also limits the scalability of network bandwidth, reducing the effectiveness of the mesh network. These factors suggest secure mesh networking protocols which appear feasible when deployed atop well-behaved networks may in fact perform badly in realistic scenarios. Optimizing cryptographic protocols is not enough; we must also complement improved cryptographic schemes with network-layer innovations.

Future work. Our results (§7) suggest important future research:

- Congestion detection and avoidance for mesh networks.
- Key management protocols more tolerant to packet loss.
- Adaptive routing protocols which modify forwarding in response to dynamic estimates of global properties.
- User-facing feedback mechanisms that allow users to understand current network dynamics and possibly adapt user-level behavior (e.g., by not sending video messages when high network congestion is detected).

These features will be key to providing sufficient bandwidth to support stronger security and anonymity properties [PSEB22, KB25b].

Properly evaluating these approaches for larger protests will also require fundamental optimization work involving network simulators like ns-3. ns-3 currently uses standard C++ data structures for priority queues [321], but our experience is that stock data structures prevent ns-3 from scaling to simulation protests beyond a few hundred nodes – ns-3 either runs out of memory and crashes, or cannot finish a simulation despite running for days of wall-clock time. Optimizing ns-3's use of memory and compute, e.g., Devastator-style techniques [BYJ+24], is crucial for understanding how secure mesh networks behave in practice.

10 Conclusion

We present Amigo, which combines a tailored CGKA and a new, clique-based routing protocol to provide a novel mesh messaging system for protesters. We evaluate Amigo using detailed simulations that consider realistic protester mobility models and realistic low-level network phenomena. Our results demonstrate substantial improvements over prior work, but suggest that further efforts are needed at lower layers in the protocol stack to support strong privacy and efficient message routing.

Acknowledgments

The authors would like to thank Jonathan Rozen, Mathias Jud, and the anonymous reviewers from Real World Cryptography (RWC) 2025 for their feedback, as well as Rosario Gennaro for contributing cloud computing resources to our evaluation.

This work is supported by National Science Foundation (NSF) awards 1955172 and 2451597, an NSF Graduate Research Fellowship, a Sui Foundation Academic Research Award, a PSC-CUNY award, and awards from Google as a part of the Cyber NYC program. This work also used Jetstream2 computing resources through allocations from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by NSF awards 2138259, 2138286, 2138307, 2137603, and 2138296.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors. Any mention of specific companies or products does not imply any endorsement by the authors, by their employers, or by the sponsors.

References

[321] ns 3. Events and Simulator. https://www.nsnam.org/docs/manual/html/events.html#scheduler, 2021. Online; accessed 1 Sept 2024.

[ABJM21a] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Collective information security in large-scale urban protests: the case of hong kong. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 3363–3380. USENIX Association, August 2021.

[ABJM21b] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Mesh messaging in large-scale protests: Breaking bridgefy. In *Topics in Cryptology – CT-RSA 2021: Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17–20, 2021, Proceedings*, page 375–398, Berlin, Heidelberg, 2021. Springer-Verlag.

[Acc24a] Access Now. Myanmar's iron curtain: internet shutdowns and repression in 2023. https://www.accessnow.org/press-release/myanmar-keepiton-internet-shutdowns-2023-en/, May 2024. Online; accessed 1 Sept 2024.

[Acc24b] Access Now. Shrinking democracy, growing violence: Internet shutdowns in 2023. https://www.accessnow.org/wp-content/uploads/2024/05/2023-KIO-Report.pdf, May 2024. Online; accessed 1 Sept 2024.

[ACDT19] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. Cryptology ePrint Archive, Paper 2019/1189, 2019.

[AEP22] Martin R. Albrecht, Raphael Eikenberg, and Kenneth G. Paterson. Breaking bridgefy, again: Adopting libsignal is not enough. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 269–286, Boston, MA, August 2022. USENIX Association.

[AG15] Nezar AlSayyad and Muna Guvenc. Virtual uprisings: On the interaction of new social media, traditional media coverage and urban space during the 'arab spring'. *Urban Studies*, 52(11):2018–2034, 2015.

[Amn20] Amnesty International. Iran: Internet deliberately shut down during November 2019 killings – new investigation. https://www.amnesty.org/en/latest/press-release/2020/11/iran-internet-deliberately-shut-down-during-november-2019-killings-new-investigation/, November 2020. Online; accessed 1 Dec 2023.

[And24] Android Developers. WifiP2pConfig. https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pConfig, 2024. Online; accessed Oct 2024.

[Ano23] Anonymous Authors. The HK19 Manual - Part 2B: How Tos. https://docs.google.com/document/d/1UROUN37_gUqrDd4FYDFYXQAmYuXtsBAfaDLo47Im-Kk/edit#heading=h.9n9f9eiq3xhs, 2023. Online; accessed 1 Dec 2023.

[AW09] Ian F Akyildiz and Xudong Wang. *Wireless mesh networks*. John Wiley & Sons, 2009.

[AWW05] Ian F Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer networks*, 47(4):445–487, 2005.

[Bao07] Lichun Bao. A new approach to anonymous multicast routing in ad hoc networks. In *2007 Second International Conference on Communications and Networking in China*, pages 1004–1008. IEEE, 2007.

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.

[BEKXK04] A. Boukerche, K. El-Khatib, Li Xu, and L. Korba. Sdar: a secure distributed anonymous routing protocol for wireless and mobile ad hoc networks. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 618–624, 2004.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[BeW19] Bewater:seven tactics that are winning hong kong's democracy revolution. https://www.newstatesman.com/politics/2019/08/be-water-seven-tactics-that-are-winning-hong-kongs-democracy-revolution-2, 2019.

[BFE+20] Lamiaa Basyoni, Noora Fetais, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. Traffic analysis attacks on tor: A survey. In *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 183–188. IEEE, 2020.

[BK09] Osamah S. Badarneh and Michel Kadoch. Multicast routing protocols in mobile ad hoc networks: a comparative survey and taxonomy. *EURASIP J. Wirel. Commun. Netw.*, 2009, January 2009.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[Bre] Robert M. Brecht. Festival and Concert Production: Crowd Safety. https://tseentertainment.com/festival-and-concert-production-crowd-safety/. Online; accessed 1 Sept 2024.

[Bri22] Bridgefy. Bridgefy Messaging App - Offline Messaging. https://bridgefy.me, 2022. Online; accessed 1 Dec 2023.

[BRT23] Alexander Bienstock, Paul Rösler, and Yi Tang. ASMesh: Anonymous and secure messaging in mesh networks using stronger, anonymous double ratchet. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1–15. ACM Press, November 2023.

[BRZV05] R.K. Balachandran, B. Ramamurthy, Xukai Zou, and N.V. Vinodchandran. Crtdh: an efficient key agreement scheme for secure group communications in wireless ad hoc networks. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 2, pages 1123–1127 Vol. 2, 2005.

[BS16] Girish Bekaroo and Aditya Santokhee. Power consumption of the raspberry pi: A comparative analysis. In *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, pages 361–366, 2016.

[BYJ+24] John Bachan, Jianlan Ye, Xuan Jiang, Tan Nguyen, Mahesh Natarajan, Maximilian Bremer, and Cy Chan. Devastator: A scalable parallel discrete event simulation framework for modern c++. In *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 35–46, 2024.

[Cel15] Department of justice policy guidance: Use of cell-site simulator technology. https://www.justice.gov/opa/file/767321/dl, 2015.

[CG18] Darin Christensen and Francisco Garfias. Can you hear me now? how communication technology affects protest and repression. *Quarterly journal of political science*, 13(1):89, 2018.

[CGCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1802–1819, New York, NY, USA, 2018. Association for Computing Machinery.

[CMB23] Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 75–92. IEEE, 2023.

[CNR07] Reza Curtmola and Cristina Nita-Rotaru. Bsmr: Byzantine-resilient secure multicast routing in multi-hop wireless networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 263–272, 2007.

[cri21] Criterion.rs documentation. https://bheisler.github.io/criterion.rs/book/criterion_rs.html, 2021.

[DANR09] Jing Dong, Kurt Ackermann, and Cristina Nita-Rotaru. Secure group communication in wireless mesh networks. *Ad Hoc Networks*, 7(8):1563–1576, 2009. Privacy and Security in Wireless Sensor and Ad Hoc Networks.

[DNRC11] Jing Dong, Cristina Nita-Rotaru, and Reza Curtmola. Secure High-Throughput Multicast Routing in Wireless Mesh Networks . *IEEE Transactions on Mobile Computing*, 10(05):653–668, May 2011.

[Ele23] Electronic Frontier Foundation. CELL-SITE SIMULATORS/ IMSI CATCHERS. https://sls.eff.org/technologies/cell-site-simulators-imsi-catchers, March 2023. Online; accessed 1 Sept 2024.

[FP12] N. Fazio and I.M. Perera. Outsider-anonymous broadcast encryption with sublinear ciphertexts. In *IACR Public Key Cryptography—PKC '12*, pages 225–242, Heidelberg, 2012. Springer. LNCS 7293.

[HBDF+13] Shaddi Hasan, Yahel Ben-David, Giulia Fanti, Eric Brewer, and Scott Shenker. Building dissent networks: Towards effective countermeasures against {Large-Scale} communications blackouts. In *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*, 2013.

[IET03] IETF. Guidelines for writing rfc text on security considerations. Technical report, Internet Engineering Task Force, 2003.

[IET23] IETF. Message layer security (mls). Technical report, Internet Engineering Task Force, 2023.

[JKK+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 133–163, Cham, 2017. Springer International Publishing.

[Joh96] D Johnson. Dynamic source routing in ad hoc wireless networks. *Mobile Computing/Kluwer Academic Publishers*, 1996.

[KB25a] Sina Kamali and Diogo Barradas. Anix: Anonymous Blackout-Resistant Microblogging with Message Endorsing . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 15–15, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[KB25b] Sina Kamali and Diogo Barradas. Anix: Anonymous blackout-resistant microblogging with message endorsing. 2025.

[KH03] Jiejun Kong and Xiaoyan Hong. Anodr: anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '03, page 291–302, New York, NY, USA, 2003. Association for Computing Machinery.

[KLH+07] Jiejun Kong, Jun Liu, Xiaoyan Hong, Dapeng Wu, and Mario Gerla. *On Performance Cost of On-demand Anonymous Routing Protocols in Mobile Ad Hoc Networks*, pages 119–142. Springer US, Boston, MA, 2007.

[KLNY03] T. Kaya, G. Lin, G. Noubir, and A. Yilmaz. Secure multicast groups on ad hoc networks. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '03, page 94–102, New York, NY, USA, 2003. Association for Computing Machinery.

[KM07] Jung-Chun Kao and Radu Marculescu. Energy-efficient anonymous multicast in mobile ad-hoc networks. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8, 2007.

[KPPW+21] Karen Klein, Guillermo Pascual-Perez, Michael Walter, Chethan Kamath, Margarita Capretto, Miguel Cueto, Ilia Markov, Michelle Yeo, Joël Alwen, and Krzysztof Pietrzak. Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 268–284, 2021.

[LA10] Jeroen Van Laer and Peter Van Aelst. Internet and social movement action repertoires. *Information, Communication & Society*, 13(8):1146–1171, 2010.

[Lee20] Francis L. F. Lee. Solidarity in the anti-extradition bill movement in hong kong. *Critical Asian Studies*, 52:18 – 32, 2020.

[LFBD+16] Ada Lerner, Giulia C. Fanti, Yahel Ben-David, Jesus Garcia, Paul Schmitt, and Barath Raghavan. Rangzen: Anonymously getting the word out in a blackout. *ArXiv*, abs/1612.03371, 2016.

[LQK09] Xiangfang Li, Lijun Qian, and Joseph Kamto. Secure anonymous routing in wireless mesh networks. In *2009 International Conference on E-Business and Information System Security*, pages 1–5. IEEE, 2009.

[LSG02] Sung-Ju Lee, William Su, and Mario Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networks and Applications*, 7(6):441–453, 2002.

[LYTC19] Francis Lee, Samson Yuen, Gary Tang, and Edmund Cheng. Hong kong's summer of uprising: From anti-extradition to anti-authoritarian protests. *China Review*, 19:1–32, 11 2019.

[MMA+21] Elaine M Murtagh, Jacqueline L Mair, Elroy Aguiar, Catrine Tudor-Locke, and Marie H Murphy. Outdoor walking speeds of apparently healthy adults: A systematic review and meta-analysis. *Sports Medicine*, 51:125–141, 2021.

[NN01] D. Niculescu and B. Nath. Ad hoc positioning system (aps). In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, volume 5, pages 2926–2931 vol.5, 2001.

[OKS01] T. Ozaki, Jaime Bae Kim, and T. Suda. Bandwidth-efficient multicast routing for multihop, ad-hoc wireless networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 1182–1191 vol.2, 2001.

[PAC14] Abhinav Prakash, Dharma P Agrawa, and Yunli Chen. Network coding combined with onion routing for anonymous and secure communication in a wireless mesh network. *International journal of Computer Networks & Communications (IJCNC)*, 6(6):1–14, 2014.

[PJW+22] Amogh Pradeep, Hira Javaid, Ryan Williams, Antoine Rault, David Choffnes, Stevens Le Blond, and Bryan Alexander Ford. Moby: A blackout-resistant anonymity network for mobile devices. *Proceedings on Privacy Enhancing Technologies*, 2022(3):247–267, 2022.

[PL19] Jessie Pang and Kate Lamb. Highway blockade reveals splits in Hong Kong protest movement. https://www.reuters.com/article/world/highway-blockade-reveals-splits-in-hong-kong-protest-movement-idUSKBN1XP06R/, November 2019. Online; accessed 1 Dec 2023.

[Pon24] Poniie pn2000 plug-in kilowatt electricity usage monitor electrical power consumption watt meter w/ extension cord. https://www.poniie.com/products/6, 2024.

[PSEB22] Neil Perry, Bruce Spang, Saba Eskandarian, and Dan Boneh. Strong anonymity for mesh messaging. *arXiv preprint arXiv:2207.04145*, 2022.

[RASJ05] S. Roy, V.G. Addada, S. Setia, and S. Jajodia. Securing maodv: attacks and countermeasures. In *2005 Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005.*, pages 521–532, 2005.

[RL08] Kui Ren and Wenjing Lou. A sophisticated privacy-enhanced yet accountable security framework for metropolitan wireless mesh networks. In *2008 The 28th International Conference on Distributed Computing Systems*, pages 286–294. IEEE, 2008.

[RP99] Elizabeth M. Royer and Charles E. Perkins. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*,

MobiCom '99, page 207–218, New York, NY, USA, 1999. Association for Computing Machinery.

[RYLZ09] Kui Ren, Shucheng Yu, Wenjing Lou, and Yanchao Zhang. Peace: A novel privacy-enhanced yet accountable security framework for metropolitan wireless mesh networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):203–215, 2009.

[Sen12] Jaydip Sen. Secure and privacy-preserving authentication protocols for wireless mesh networks. In *Applied Cryptography and Network Security*, pages 3–34. IntechOpen, 2012.

[SEV+15] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. {RAPTOR}: Routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, 2015.

[Shi11] Clay Shirky. The political power of social media: Technology, the public sphere, and political change. *Foreign Affairs*, 90:28–41, 2011.

[Sig14] Signal: Private group messaging. https://signal.org/blog/private-groups/, 2014.

[Sig20] The double ratchet algorithm. https://signal.org/docs/specifications/doubleratchet/#symmetric-key-ratchet, 2020.

[SKY05] Ronggong Song, Larry Korba, and George Yee. Anondsr: efficient anonymous dynamic source routing for mobile ad-hoc networks. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '05, page 33–42, New York, NY, USA, 2005. Association for Computing Machinery.

[SRZF03] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus P. J. Fromherz. Localization from mere connectivity. MobiHoc '03, page 201–212, New York, NY, USA, 2003. Association for Computing Machinery.

[SS14] Nazatul Haque Sultan and Nityananda Sarma. Papar: Pairing based authentication protocol with anonymous roaming for wireless mesh networks. In *2014 International Conference on Information Technology*, pages 155–160. IEEE, 2014.

[Sti19] G. Keith Still. Standing Crowd Density. https://www.gkstill.com/Support/crowd-density/CrowdDensity-1.html, February 2019. Online; accessed 1 Oct 2024.

[THH15] Somayeh Taheri, Salke Hartung, and Dieter Hogrefe. Anonymous group-based routing in manets. *Journal of Information Security and Applications*, 22:87–98, 2015. Special Issue on Security of Information and Networks.

[Wei19] Matthew Weidner. Group messaging for secure asynchronous collaboration. Master's thesis, University of Cambridge, 2019.

[Wi-21] Wi-Fi Alliance. *Wi-Fi Direct Specification*, 2021.

[YJW06] Liu Yang, Markus Jakobsson, and Susanne Wetzel. Discount anonymous on demand routing for mobile ad hoc networks. In *2006 Securecomm and Workshops*, pages 1–10, 2006.

[ZJG21] Maximilian Zinkus, Tushar M. Jois, and Matthew Green. Sok: Cryptographic confidentiality of data on mobile devices. *Proceedings on Privacy Enhancing Technologies*, 2022:586 – 607, 2021.

[ZLH06] Yan Zhang, Jijun Luo, and Honglin Hu. *Wireless mesh networking: architectures, protocols and standards.* CRC Press, 2006.

[ZSXJ04] S. Zhu, S. Setia, S. Xu, and S. Jajodia. Gkmpan: an efficient group rekeying scheme for secure multicast in ad-hoc networks. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.*, pages 42–51, 2004.

[ZWK+04] Bo Zhu, Zhiguo Wan, M.S. Kankanhalli, Feng Bao, and R.H. Deng. Anonymous secure routing in mobile ad-hoc networks. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 102–108, 2004.

A Simulation Details

Changes to simulate Wi-Fi Direct: In order to implement more than one channel in ns-3 on each device under Ad Hoc Wi-Fi, we create a network interface for each channel on each given node (representing an individual's device). This means that nodes have more than one identifier (i.e. IP address), as they have an identifier for each interface. However, this means that we need to bind an IP address to the node's MAC address in our simulations. We therefore implement an "address discovery" period prior to simulation start. During this time, each node sends a message on each channel to each other node. This allows for the nodes to exchange ARP packets, and learn each others' MAC addresses. We modify the ARP cache expiration values to last the entire duration of the simulation; in this way, we account for any potential discrepancies arising from our implementation.

Wi-Fi Direct Protocol Structure: At a high level, the Wi-Fi Direct protocol entails two phases: the discovery phase and the data exchange phase.

In the discovery phase, devices exchange information on advertisement channels about how to proceed with data exchange. In line with Wi-Fi Direct [Wi-21], we use the 2.4 GHz band for the advertisement channels, with a channel width of 20 MHz. We implement three advertising channels, the standard number of non-overlapping channels for this scenario (channels 1,6, and 11).

We simulate the discovery phase, through sending advertising-related packets; we select reasonably representative packet sizes through examining the Wi-Fi and Wi-Fi Direct specifications [Wi-21], and determining which conditional parameters would likely be included for a basic case. While we predetermine the parameters associated with communication for our simulations, in true Wi-Fi Direct, the information in these packets allows for communication channels and rates, SSIDs, and network parameters to be negotiated. Nodes advertise by sending beacon packets (of size 155 bytes) on these channels at random offsets every minute. When a node receives a beacon packet, it sends a probe request packet (of size 124 bytes) to the original sender. Then, the original sender sends a probe response packet (of size 199 bytes). In this process, various set up procedures are established, for example, a channel for future data exchange communication is negotiated.

Then, the data exchange phase may begin. In the data exchange phase, devices exchange data on a selected channel (in our case, exchanging messages from their message buffers). Data exchange could occur in either the 2.4 GHz band or the 5GHz band. We choose to implement it in the 5 GHz band, with a channel width of 80 MHz. We implement three data exchange channels; nodes select a channel to exchange data on randomly, determined during the probe request/response process.

B Standard Mobility Models

Though they standard mobility models are in network evaluation space, we provide some context surrounding static and random waypoint mobility models.

The *static model* entails placing nodes a set distance apart in a grid, where they remain for the duration of the simulation. This is the system used in previous work [PSEB22]. We use this model to represent a baseline, to help us understand the impact of movement. In our simulation runs, we place protesters 1m apart.

In the *random waypoint model*, each node first pauses for selected amount of time, then selects a random location, and a random speed at which to move to that location. Once the node has reached the selected location, it again pauses, before repeating the process. We use this model to represent a baseline for movement (i.e., when no protest events are occurring).

There are two parameters we must set: (1) the range of possible node speeds and (2) the node pause lengths. For our simulation, we set the range of node speeds to resemble the general range of human mobility from a very slow walk (1.5 km/h) to a quick run (13.5 km/h). We distribute these speeds skewing towards lower speeds, following log-normal distribution, centered at 1.3m/s (average walking speed). For the node pause length, we set a range of 0-50 seconds.

C Bloom Filters

Our Bloom filters are formatted as bit arrays. The size of our Bloom filters are determined by the size of the buffer it represents; we maintain a consistent hash count (n=17) and false positive rate (0.00001). For example, a 50 KB message buffer has a .4KB Bloom filter digest, a 5,000 KB message buffer has a 479 KB Bloom filter digest.

D Additional CGKA Microbenchmarks

Timing benchmarks: In Table 4, we display each state operation (**Add**, **Rem**, **Upd**), and the corresponding time measurements in milliseconds with ciphertext sizes in kilobytes. The number of messages required be sent over the network for addition, removal, and update are two, one, and one respectively.

Power and energy benchmarks: Smartphones have limited battery life, so Amigo should also be power efficient. On an Android Moto e we ran energy consumption benchmarks of Amigo's state operations, shown in Table 5. While addition was power intensive, our

Table 4: Microbenchmarks of state operations ran on a Raspberry Pi.

Size	Add		Rem		Upd	
	Time (ms)	CS (kB)	Time (ms)	CS (KB)	Time (ms)	CS (KB)
10	13.652	21.721	3.634	3.046	3.060	3.517
25	18.565	47.147	5.193	3.805	3.711	3.988
50	24.173	89.281	6.316	4.565	4.405	4.471
75	31.578	131.612	7.816	5.331	5.174	4.933
100	35.339	173.581	7.971	5.307	5.263	4.933
125	42.494	215.304	8.163	5.331	5.347	5.331
150	48.743	257.453	11.570	6.078	6.475	5.356
175	55.178	299.696	12.146	6.094	6.605	5.344
200	58.334	341.517	12.119	6.042	6.664	5.402

Table 5: Power consumption (measured in Watts) and energy consumption (measured in Joules) of Amigo state operations on an Moto E Android phone.

Measurement	Add	Rem	Upd
Power (W)	4.38	4.39	4.39
Energy (J)	588.68	106.16	81.46

Table 6: Power consumption (measured in Watts) and energy consumption (measured in Joules) of Amigo state operations on a Raspberry Pi.

	Baseline		Add		Rem		Upd	
	PC	EC	PC	EC	PC	EC	PC	EC
Idle	2.39	143.40	4.05	243.00	4.22	253.20	4.33	259.80
Video	3.97	238.20	4.88	292.80	5.00	300.00	4.83	289.80
Browsing	4.56	273.60	5.13	307.80	5.18	310.80	5.40	324.00

removals and updates show very reasonable performance when it comes to preserving battery life. Even addition operations can be run sequentially for up to 3 hours before draining the battery. For our Raspberry Pi benchmarks we run several processes in parallel, measuring the additional energy consumed when running Amigo alongside other tasks. Taking inspiration from prior work [BS16], we choose the following tasks: idling, web browsing (we use Reddit.com), and watching a video (we use a 480p YouTube video).

Our measurement setup consisted of the Raspberry Pi attached to a display via (micro-)HDMI and a mouse and keyboard connected via USB. Power consumption was measured every second using the Poniee PN2000 wattmeter [Pon24]. We note that, since the Raspberry Pi is not optimized for battery usage, our measurements reflect an upper bound.

First, we performed each task for 1 minute to determine baseline power consumption. Afterwards, we used Criterion to run Amigo operations for a group size of 200 members alongside each task. We measured the power consumption every second, and calculated the combined energy consumed in Joules. We then calculated the average energy consumed by Amigo operations by by subtracting the combined measurement from the baseline, and then dividing by the number of Amigo operations executed in the time frame. Our results are in Tables 6 and 9.

Under idle conditions, Amigo expectedly contributes the most to the Raspberry Pi's energy consumption. When idle, the Raspberry Pi's power management mechanisms ensure the CPU and other components are in low power states, resulting in higher energy spikes when woken. When already under load, we can see Amigo's contributions decrease. Amigo's most frequent state operations, removal and update, should be the most energy efficient. As we can see in Table 9, the removal and update operations in each task are at least 4 times more energy efficient than addition. While the device is already under load (during the video and browsing tasks), a single removal or update requires less than 1 additional Joule of energy be consumed by the device. Our results are promising as protesters will be able to use Amigo alongside other applications without significant degradation in battery life.

Network load benchmarks: As groups increase in size, state operation message sizes also see an increase. Highlighted in Figure 10, member addition incurs the largest message sizes with 341.517 kilobytes of data needing to be propagated through the network for a group size of 200. As noted previously, however, member additions are not particularly frequent. Other state operations likely to occur more frequently, member removal and update, incur

Table 7: Operation timings on Raspberry Pi for representative message sizes.

Operation	Size: 250 B	Size: 2 MB	Size: 10 MB
Encryption	78.45 µs	57.517 ms	311.24 ms
Decryption	69.08 µs	23.677 ms	282.38 ms

Table 8: Operation timings on Android Moto e for representative message sizes.

Operation	Size: 250 B	Size: 2 MB	Size: 10 MB
Encryption	143.33 µs	149.57 ms	749.00 ms
Decryption	125.17 µs	55.71 ms	729.13 ms

significantly less message sizes which is beneficial to maximizing the performance of the mesh backbone.

E Learning Missing Internal Secret Keys

In situations where a newly joined group member is missing secret keys associated with intermediate nodes within the ratchet tree, they may query the group for the relevant keys they are missing. We sketch one way this can be done. Using Figure 3, consider a scenario where the member occupying $\texttt{Node}_1$ is requesting the secret key of $\texttt{Node}_4$, an intermediate node. The member occupying $\texttt{Node}_0$ knows this key.

In the request phase, the member occupying $\texttt{Node}_1$ uses their leaf node's secret key to sign a request for the secret key associated with $\texttt{Node}_4$. This request is broadcast to the group. In the response phase, the member occupying $\texttt{Node}_0$ verifies the signature on the request, using their view of the ratchet tree to confirm $\texttt{Node}_1$ is a descendant of $\texttt{Node}_4$. This means the member occupying $\texttt{Node}_1$ is a valid member of the group and allowed to access $\texttt{Node}_4$'s secret key. The member occupying $\texttt{Node}_0$ then encrypts the secret key of $\texttt{Node}_4$ under the public key associated with $\texttt{Node}_1$, and broadcasts this to the group. Only the member occupying $\texttt{Node}_1$ can decrypt the secret key.

We note that this process does not allow for removed members to re-join a group. As an example, consider Figure 4 and that $\texttt{Node}_3$, who has been removed from the group, attempts to contact $\texttt{Node}_1$ for missing keys. If $\texttt{Node}_1$ has not yet processed this removal, then the only key material they could provide $\texttt{Node}_3$ is information it is expected to have, i.e., the root key for the current, pre-removal CGKA state γ. Post-compromise security ensures that this existing key material alone is not enough for the removed member to derive valid key material for future epochs. If $\texttt{Node}_1$ has processed this removal, then it has evolved γ to γ' to exclude $\texttt{Node}_3$, and will not process any more messages from $\texttt{Node}_3$.

F Latency

Figure 11 shows the latency for our mechanisms across our routing protocols. We note end-to-end message latency is mostly consistent over time across all mechanisms. We see that, as expected, flooding protocols tend to incur less end-to-end message latency, as they do not have to adhere to a round-based election system, nor route messages through a leader; both of which limit message delivery. Dynamic clique routing looks 'spikey' because of the structure of the election system; the fact that messages are not exchanged during the election period causes some messages to go unsent for longer periods of time.

Table 9: Additional energy consumed by a single Amigo operation alongside each task.

Task	Add	Rem	Upd
Idle	5.859 J	1.340 J	0.776 J
Video	3.211 J	0.754 J	0.344 J
Browsing	2.012 J	0.454 J	0.336 J

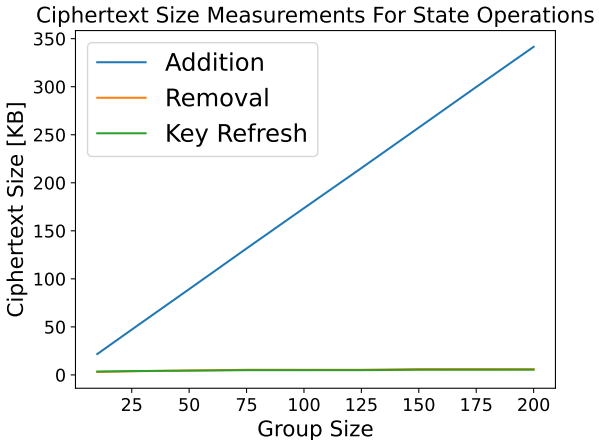

Figure 10: Ciphertext size measurements of Amigo state operations.

G Handling Mesh Conditions

We discuss the challenges of providing group state synchronization within a mesh.

G.1 Handling Concurrent Updates

The decentralized mesh setting can give rise to concurrent updates, in which multiple members initiate CGKA operations (**Add**, **Rem**, **Upd**) within the same epoch. These updates warrant closer analysis of their impact on group state consistency.

In our discussion below, we assume members of a group have consistent state prior to receiving new protocol messages. First, let us consider the **Upd** operation, which can happen concurrently when multiple members refresh their key material within the same epoch. We expect concurrent **Upd** operations to occur often in practice since groups benefit from frequently rotating keys (i.e., forward secrecy and post-compromise security). **Upd** results in a state operation message `S` where $\texttt{S}_{\texttt{.keys}}$ are elliptic curve key material and $\texttt{S}_{\texttt{.seeds}}$ are used to derive the same. As noted in Section 4.2, due to commutative updates this key material can be combined with the existing in any order, so resolving concurrent updates is simple. Removal (**Rem**) is similar, with the most notable difference being that it is possible for concurrent removal messages to target the same member for removal. Again, this poses no problem for our CGKA since removing the same member is akin to clearing that member's leaf node multiple times. If no state for the targeted leaf node exists, no further action needs to be taken.

Now, let us consider the **Add** operation. As suggested by [ABJM21a], group additions are infrequent. Choosing when and whom to invite does not happen spontaneously and is a deliberate process, requiring approval from current group members [ABJM21a], making it relatively unlikely that two members are being added concurrently by different parts of the group. That being said, concurrent **Add**s are still possible, and are more complicated than **Upd**s or **Rem**s; in addition to concurrently modifying the ratchet tree, they can

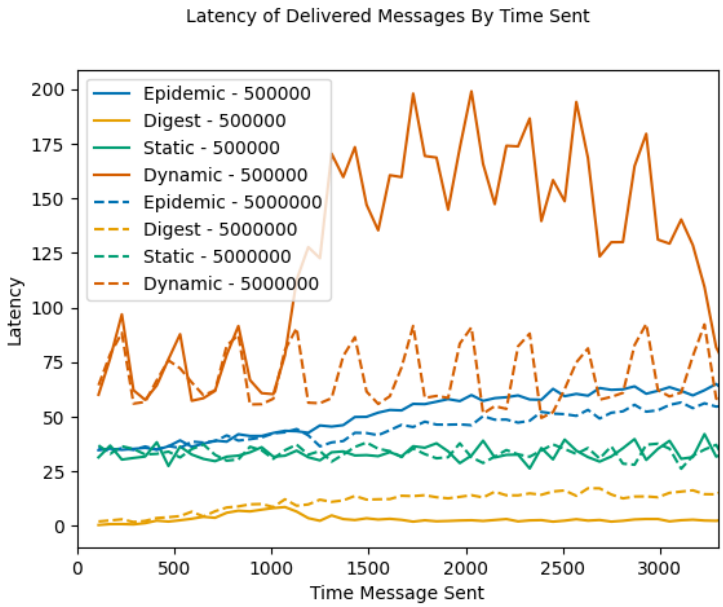

Figure 11: Latency over time for the gather mobility model.

lead to situations where clients need to resolve which leaf node positions newly invited members will occupy when joining.

One way of handling this is incorporating an application-layer mechanism for collaboratively approving changes to group state. In this way, group members would mutually agree on the order and details of proposed state changes before they are processed. Amigo forgoes such a mechanism because its design, informed by past literature on protester dynamics [Shi11, LA10, ABJM21a, AG15, CG18, Lee20, LYTC19, BeW19], prioritizes rapid and efficient state updates and removals. This is essential in protest settings, where an adversary who infiltrates a group needs to be quickly removed, and where frequently updating key material proactively aids in security for all group members. Such a collaborative approval mechanism for additions would require increased network communication, resulting in delays for updates and removals that could inadvertently threaten protester safety.

An alternative solution is to enforce access control policies that restrict additions to a few privileged individuals. This is in line with group dynamics suggested in literature [ABJM21a]: privileged individuals within groups (i.e., organizers) add users their group on behalf of the rest. MLS suggests this approach, noting the delivery service abstraction can be a viable mechanism for enforcing policies on who can add members, dropping messages that do not abide by said policies [IET23]. While Amigo cannot rely on a centralized delivery service, a similar effect can be achieved by having Amigo's client application enforce policies set by group founders – policies restricting the privilege to add new members to a single authorized user. Those without this privilege would have the addition feature disabled. This can be further enforced through cryptographic mechanisms; **Add** protocol messages can include a signature, generated with the private key held by the leaf node of a group member authorized to perform additions. In this way, additions are only processed upon verification of the signature, preventing concurrent updates. In contrast to the aforementioned collaborative approach, this access control approach requires no additional messages for the **Add** operation, only an additional field for a signature in the original message.

G.2 Handling Out-of-Order Messages

In Section 4.2 we discuss how our CGKA allows for protocol messages to be processed in any order. In practice, there is some nuance, as protocol messages are encrypted using keys from the epoch in which they were originated, and which may be different from the epoch in which a group member currently is. In this case, a client may not immediately be able to decrypt and process a valid protocol message. Amigo clients alleviate this issue by placing all received messages in their message buffer. As we discuss in Section 5, messages remain in the buffer before being evicted, with their retention time depending on network conditions and buffer size. Clients, upon receiving new messages, attempt trial decryption of both the newly received message and messages within their buffer. This means messages that have arrived but can not yet be decrypted, i.e., were originated in epochs a client has not yet entered, will sit in a client's buffer until the appropriate CGKA protocol message that allows for its decryption is received. While not identical, this resembles Signal's approach to handling out-of-order messages [Sig20].

Network conditions may cause cause protocol messages to arrive after significant delays; it may be the case that a member no longer retain the key material necessary to decrypt a protocol message once it arrives. We argue that these situations point to failing network conditions that Amigo's CGKA alone is not designed to solve. There do exist application-layer mechanisms, however, that can help alleviate this issue. One mechanism that can help in this situation is for Amigo clients to retain prior key material for a number of epochs agreed upon by the group. At the expense of weakened forward secrecy, this "sliding window" mechanism enables group members waiting for delayed messages to still decrypt them upon arrival using keys from prior epochs. Note that in addition to weakened forward secrecy, this mechanism can allow removed members to contact the group for the specified number of epochs.

Even in the best of network conditions, it is possible a group member may *never* receive a protocol message. While our simulations in Section 7 suggest this is uncommon in reasonable network conditions under appropriate routing protocols, we include a more detailed discussion on this scenario in Section 7.3.

H Updatable Public Key Encryption

In this section, following the results in [ACDT19], we provide the definition and syntax of an updatable public key encryption scheme along with the corresponding security model and game.

Definition 2 (Updatable Public Key Encryption: Syntax)**.** An updatable public key encryption scheme is a tuple of algorithms UPKE = (**Gen**, **Enc**, **Dec**), where:
KEY GENERATION: A party runs **Gen**, with a uniformly random seed Δ_0 as input, to derive a key pair:

$$(\mathtt{sk}, \mathtt{pk}) \leftarrow \mathbf{Gen}(\Delta_0)$$

ENCRYPTION: A party runs **Enc** to encrypt a message $\mathtt{m}$under public key $\mathtt{pk}$, resulting in the corresponding ciphertext $\mathtt{ct}$ and new independent public key $\mathtt{pk}'$:

$$(\mathtt{ct}, \mathtt{pk}') \leftarrow \mathbf{Enc}(\mathtt{pk}, \mathtt{m})$$

DECRYPTION: A party runs **Dec** to decrypt a ciphertext $\mathtt{ct}$ with secret key $\mathtt{sk}$, resulting in the corresponding plaintext $\mathtt{m}$and secret key $\mathtt{sk}'$:

$$(m, \mathtt{sk}') \leftarrow \mathbf{Dec}(\mathtt{sk}, \mathtt{ct})$$

Correctness: A UPKE satisfies the correctness property if for all randomness and message pairs $\{r_i, m_i\}_{i=1}^{q}$:

$$Pr\begin{bmatrix}(\mathtt{sk}_0, \mathtt{pk}_0) \leftarrow \mathbf{Gen}(\Delta_0); \\ \text{For } i \in [q], \quad (c_i, \mathtt{pk}_i) \leftarrow \mathbf{Enc}(\mathtt{pk}_{i-1}, m_i; r_i); \\ (m_i', \mathtt{sk}_i) \leftarrow \mathbf{Dec}(\mathtt{sk}_{i-1}, c_i) : m_i = m_i'\end{bmatrix} = 1.$$

We define an IND-CPA secure UPKE scheme according to [ACDT19]. This notion is similar to CPA-secure public-key encryption (PKE), with one key distinction: for ciphertexts that are honestly generated, the adversary is also given the randomness used in their creation. This models protocol executions where, before the challenge phase, the adversary can observe ciphertexts sent by users (e.g., via network monitoring) and has access to the randomness used by compromised users to encrypt path secrets during that period.

Definition 3 (Updatable Public Key Encryption: Security)**.** For any adversary A with running time t we consider the IND-CPA security game:

- $(\mathtt{sk}_0, \mathtt{pk}_0) \leftarrow \mathbf{Gen}(\Delta_0)$
- $\mathcal{A}$, on input $\mathtt{pk}_0$, outputs (m_0^*, m_1^*), $\{r_i, m_i\}_{i=1}^{q}$
- For $i = 1, \ldots, q$, compute:
  $(\mathtt{ct}_i, \mathtt{pk}_i) \leftarrow \mathsf{Enc}(\mathtt{pk}_{i-1}, \mathtt{m}_i; r_i), (\mathtt{m}_i, \mathtt{sk}_i) \leftarrow \mathsf{Dec}(\mathtt{sk}_{i-1}, \mathtt{ct}_i)$
- Select a random $b \leftarrow \{0, 1\}$, and compute:
  $(\mathtt{ct}^*, \mathtt{pk}^*) \leftarrow \mathsf{Enc}(\mathtt{pk}_q, \mathtt{m}_b^*)$, $(\cdot, \mathtt{sk}^*) \leftarrow \mathsf{Dec}(\mathtt{sk}_q, \mathtt{ct}^*)$
- $b' \leftarrow \mathcal{A}(\mathtt{pk}^*, \mathtt{sk}^*, \mathtt{ct}^*)$

$\mathcal{A}$ wins the game if $b = b'$. A UPKE scheme is (t, ϵ)-CPA-secure if for all t-attackers $\mathcal{A}$,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{cpa}^*} \leq \epsilon.$$

H.1 Synchronizing Public Key State

In our UPKE scheme, **send-message** outputs a message `A` where the payload is a UPKE ciphertext that includes a newly generated public key. To enable synchronicity, we can extract the new public key from this ciphertext and include it as a separate field $\mathtt{A}_{.\mathtt{new\_public}}$ in `A`. The node on which to apply $\mathtt{A}_{.\mathtt{new\_public}}$ is known by the intended recipients of `A` but not by anyone else. This can result in inconsistent group state between intended recipients and other members of the group. However, extracting $\mathtt{A}_{.\mathtt{new\_public}}$ enables a naive solution. An identifier, additional field $\mathtt{A}_{.\mathtt{id}}$, can be included in `A` that denotes where $\mathtt{A}_{.\mathtt{new\_public}}$ should be applied within the ratchet tree. However, this is at odds with outsider-anonymity since unintended recipients within the group can learn, using the ratchet tree structure, who received a message. A different strategy is to synchronize by a local clock, where at a specified time interval members of the group (some of whom are out of sync) will interactively determine what version of a key they should all be using. Thus members can always "catch up" within an epoch, ensuring a maximum time limit for state inconsistencies, at the expense of communication overhead.

I CGKA Security Model and Proof

We design our security game to reflect an authoritarian regime who may attempt to subvert protester communication by arresting protesters and compromising their devices.

We closely follow the security game from [KPPW+21], modifying it for our setting when necessary. This game intuitively captures forward secrecy and post-compromise security guarantees. Like [KPPW+21], we restrict adversaries from sending malformed messages.

Definition 4 (Asynchronous Continuous Group Key Agreement: Security)**.** The security for our CGKA is modeled by a game between a challenger `C` and an adversary `A`. At the beginning of the game, the adversary queries **Create-group** (ID^*, `G`) where ID^* initializes a group including themselves and the specified members `G` = $(ID_0, ..., ID_\ell)$. From here on, the adversary may make a sequence of the below queries in any order. This may involve changing group state, sending and/or reading group messages, and/or choosing to change the corruption status of arbitrary group members.

1. **Add-user**(ID, ID'): A user, ID, adds ID' to the group `G`
2. **Rem-user**(ID, ID'): A user in group `G` removes ID' from `G`
3. **Update**(ID): A user, ID, refreshes their local state
4. **Process**(q, ID'): A user, ID', processes a state operation message originating from query $q \in$ {add, remove, update}
5. **send-message**$([ID_i, ...ID_n], m)$: A user sends a message to participants in the group
6. **read-message**(m): A user reads a message intended for them
7. **start-corrupt**(ID): The internal state and randomness of user ID is leaked to the adversary, A
8. **end-corrupt**(ID): The internal state and randomness leakage of user ID to the adversary, A, is ended
9. **challenge**(q^*): The adversary selects a query $q^* \in$ {**Add-user**, **Rem-user**, **Update** }. The challenger then flips a coin b and returns to the adversary k_b where k_0 is a randomly generated key and k_1 is the group key for the respective epoch.

At the end of the game, the adversary outputs a bit b′ and wins if b′ = b. We call a CGKA scheme (Q, ϵ, t)- CGKA-secure if for any adversary A making at most Q queries of the form **Add-user**$(\cdot,\cdot)$, **Rem-user**$(\cdot,\cdot)$, **Update**$(\cdot,\cdot)$ and running in time t it holds:

$$\text{Adv}_{CGKA}(A) := |\Pr[1 \leftarrow A \mid b = 0] - \Pr[1 \leftarrow A \mid b = 1]| < \epsilon.$$

I.1 Safe Predicate

To prove security of our CGKA scheme, we first have to define a *safe predicate*. We want to rule out situations where the adversary can trivially win the security game. This can occur if, for example, at any point where a query q^* is made on a node ID_i, the adversary has previously corrupted another member in the group and the corruption is still ongoing. Trivially, the attacker would always win the game in this scenario, since they possess the corrupted member's state which can be used to compute the group key via the root node's key material. We consider the group key *safe* when all members in ID_i's (the challenged party) local view of group state γ are uncorrupted.

Definition 5 (Safe Predicate)**.** Let k^* be a group key generated in an action

$$a^* \in \{\textbf{Add-user}(ID^*, \cdot),\ \textbf{Rem-user}(ID^*, \cdot),\\ \textbf{Update}(ID^*),\ \textbf{Create-group}(ID^*, \cdot)\}$$

at time point $q^* \in [Q]$, and let $\mathtt{G}^*$ be the set of users who would end up in the group if query q^* was processed, as viewed by the generating user ID^*. Then the key k^* is considered *safe* if for all users $ID \in \mathtt{G}^*$ (including ID^*), we have that ID is safe at time q^* in the view of ID^*.

I.2 The CGKA and Challenge Graph

In this section, we show that if the safe predicate is satisfied during the CGKA game, this means that the group key k^* corresponding to the challenge cannot be derived from any key material the adversary may possess. To do this, we first define a notion of a CGKA graph, and then a challenge graph.

CGKA Graph. A node i in the CGKA graph is associated with seeds Δ_i and a keypair $(\mathtt{sk}_i, \mathtt{pk}_i) \leftarrow \textbf{Gen}(\Delta_i)$. In our setting, each leaf node generates its seeds independently at random for its ancestors (rather than via hierarchical derivation like in [KPPW+21]). As such, the CGKA graph only contains edges that reflect explicit encryption dependencies. An edge (i, j) in the graph denotes a ciphertext of the form $\textbf{Enc}_{\mathtt{pk}_j}(\Delta_j)$, indicating that node ID_i encrypts the seed Δ_j under node ID_j's public key.[7]

Challenge Graph. The challenge graph is the subgraph of the CGKA graph containing the nodes from which the group key k^* is trivially reachable. This means in the case where all nodes in the CGKA graph have processed the same protocol messages, the challenge graph would include the set of all members (leaf nodes), intermediate nodes, and the root. If the safe predicate is satisfied, then for an adversary challenging via a query q^*, none of the seeds or material necessary to compute k^* will be known to them. We prove this below.

Lemma 1. *For any safe challenge group key in Amigo it holds that none of the seeds and secret keys in the challenge graph is leaked to the adversary via corruption.*

Proof. In our game, an adversary corrupting a user results in obtaining all state and secrets that user holds. This includes the keys of their leaf node, as well as intermediary nodes via the public key encryption dependencies mentioned previously. Let us say the adversary has taken one of the following actions required of their challenge:

$$a^* \in \{\,\textbf{Add-user}(ID^*, \cdot),\ \textbf{Rem-user}(ID^*, \cdot),\\ \textbf{Update}(ID^*),\ \textbf{Create-group}(ID^*, \cdot)\,\}$$

If the safe predicate is satisfied, it guarantees that no nodes in the challenge graph are currently corrupted. Every node has refreshed its cryptographic material via **Update**, and no information that can be used to derive the key k^* (output by the action a^*) is available to the adversary. Such information is only sent to nodes that are refreshed and uncorrupted.

We argue by contradiction: suppose k^* is leaked. This implies that some secret key $\mathtt{sk}$ belonging to a leaf node ID was compromised. This can only happen if ID was already corrupted at the time of the challenge, before processing its update. This means that ID is in a corrupted state during the challenge, violating the safe predicate as viewed by ID^*. Therefore, if k^* is leaked, the safe predicate cannot hold in the view of ID^*. □

[7] These edges reflect seed distribution via encryption dependencies between nodes, as opposed to implicit structure from hierarchical derivation like in [KPPW+21].

I.3 Security Proof of the Amigo CGKA in the Standard Model

We adapt the security proof of [KPPW+21] within the framework introduced by [JKK+17]. The adversary's goal in our security game is to distinguish a *safe* group key, k^*, from a uniformly random and independent seed. We prove *selective* security, where the adversary must commit to all of their queries in advance. The framework of [KPPW+21] shows how our proof can be modified to prove security for an adaptive adversary.

Our proof involves two game executions: the **real** game and the **random** game. We prove their indistinguishability using the *reverse-pebbling* technique from [Ben89]. In reverse-pebbling, there is a directed acyclic graph (DAG) with a unique sink and reversed edges, and the objective is to place pebbles on nodes—according to certain rules—until only the sink remains pebbled. In our case, this DAG is the *challenge graph*, and each pebble represents a challenged node. A pebbling configuration, $\mathcal{P}_\ell$, corresponds to a hybrid game, H_ℓ, where a node v being pebbled means that a randomly sampled seed (independent of the original seed Δ_v) is used during the simulation. The **real game**, H_{real}, corresponds to the initial pebbling configuration $\mathcal{P}_0 = \emptyset$, where no randomness is introduced. The **random game** H_{random} corresponds to the final configuration, $\mathcal{P}_L$, where only the sink node is pebbled, and all challenge-related secrets are replaced with independent randomness.

Definition 6 (Reversible Black Pebbling)**.** A *reversible pebbling* of a directed acyclic graph $G = (V, E)$ with unique sink *sink* is a sequence $\mathcal{P} = (\mathcal{P}_0, \ldots, \mathcal{P}_L)$ with $\mathcal{P}_\ell \subseteq V$ for $\ell \in [0, L]$, such that $\mathcal{P}_0 = \emptyset$, $\mathcal{P}_L = \mathit{sink}$, and for all $\ell \in [L]$, there is a unique $v \in V$ such that: (1) $\mathcal{P}_\ell = \mathcal{P}_{\ell-1} \cup \{v\}$ or $\mathcal{P}_\ell = \mathcal{P}_{\ell-1} \setminus \{v\}$; and (2) for all $u \in \mathrm{parents}(v)$, it holds that $u \in \mathcal{P}_{\ell-1}$.

Due to Lemma 1, we know that none of the keys contained in the challenge graph can be leaked to the adversary. This allows us to prove the indistinguishability of H_{real} and H_{random} from the IND-CPA property of our underlying encryption scheme in Appendix H.

Lemma 2. *Let $\mathcal{P} = (\mathcal{P}_0, \ldots, \mathcal{P}_L)$ be a valid pebbling sequence on the challenge graph. If $\Pi = (\boldsymbol{Gen}, \boldsymbol{Enc}, \boldsymbol{Dec})$ is an (ε, t)-IND-CPA secure encryption scheme, then any two subsequent hybrid games $\mathsf{H}_\ell, \mathsf{H}_{\ell+1}$ are $(2 \cdot \varepsilon, t)$-indistinguishable.*

Proof. Let H_ℓ and $H_{\ell+1}$ be two subsequent hybrid games. $P_{\ell+1}$ and P_ℓ differ by one pebbled node v^* with ingoing encryption edge (u, v^*). We prove indistinguishability by a sequence of hybrids:

$$H_\ell := H_{\ell,0}, H_{\ell,1}, H_{\ell,2} := H_{\ell+1}$$

where $H_{\ell,1}$ is defined similarly to $H_{\ell,0}$ except that the keypair $(\mathtt{pk}_{v^*}, \mathtt{sk}_{v^*})$ is generated from a uniformly random seed instead of Δ_{v^*}.

- Indistinguishability of $H_{\ell,0}$ and $H_{\ell,1}$ follows from the randomness used in seed generation. In game $H_{\ell,1}$, the seed Δ_{v^*} used for key generation is replaced with an independently sampled random seed. Since both seeds are uniformly random and independent, they are computationally indistinguishable. Additionally, all other seeds and edges required to simulate the rest of the hybrid games can be perfectly reconstructed. This implies that any advantage ε an adversary has in distinguishing $H_{\ell,0}$ from $H_{\ell,1}$ translates directly to an advantage ε in distinguishing two random values.

- $H_{\ell,2}$ is defined similarly to $H_{\ell,1}$ except that the encryption keypair $(\mathtt{pk}_{v^*}, \mathtt{sk}_{v^*})$ is used to encrypt a random seed, rather than Δ_{v^*}. Therefore, a reduction can embed an IND-CPA challenge for Δ_{v^*} and a uniformly random seed. An adversary who can distinguish $H_{\ell,1}$ and $H_{\ell,2}$ with advantage ε can therefore break the IND-CPA security of the encryption scheme with the same advantage.

□

The pebbling technique used in the above proof handles selective adversaries, where the list of corrupted users is specified at the beginning of the game. In a realistic setting, however, an adversary could adaptively choose to corrupt nodes over a period of time, based on some strategy. By trivially extending our proof to this adaptive setting (where we would *guess* the adaptive choice of the adversary) would result in an exponential security loss. We notice that, as in [KPPW+21], if we leverage the framework of [JKK+17], we can provide a security reduction with only a quasipolynomial loss of security. We defer the details to Theorem 2 in [KPPW+21].

J Routing Privacy

We describe Amigo with respect to receiver privacy (Can a global network eavesdropper determine which user is the intended receiver for a message?), sender privacy (Can a global network eavesdropper determine which user is the originator of a message?), and sender/receiver privacy (Can a global network eavesdropper determine the sender/receiver pair for a specific message?). We also discuss Amigo's relationship to clique privacy.

Receiver privacy. In our routing protocols, messages bear no explicit metadata about message receivers; nodes use trial decryption (§4.4) to determine if a received message's final destination is the local node. So, a global eavesdropper cannot directly extract receiver identities from a message. Furthermore, a node forwards a message without regard to whether the node is the message's intended final recipient. The result is that our routing mechanisms provide receiver privacy, even in the presence of global eavesdroppers.

Sender privacy. Amigo messages bear no explicit sender metadata due to Amigo's use of trial decryption for implicitly revealing a message's sender. Thus, assuming that Amigo's encryption is correct, a network eavesdropper cannot infer a message's sender through analysis of message contents. However, Amigo does not generate dummy messages when native traffic volume is low (§3); the consequence is that, via traffic analysis, an eavesdropper can violate sender privacy in some scenarios. For example, suppose that a network eavesdropper is following Alice. Further suppose that the eavesdropper notices that, for a time period longer than the maximum TTL time, Alice has not encountered any peers. If the eavesdropper later sees that Alice encounters a peer and transmits a message, the eavesdropper can infer that Alice is sharing a message that Alice herself generated; the message could not have been a buffered message received earlier from an external node, because all such messages would have expired in Alice's buffer.

Digest-based routing optimizations may also violate sender privacy. For example, by logging (1) the hash of each message, (2) the time at which each message hash was first seen, and (3) the Bloom filters that nodes exchange, an eavesdropper could derive a probabilistic estimate of which node first injected each message into the network.

Sender/receiver privacy: Amigo messages contain no explicit, eavesdropper-visible metadata about sender/receiver pairs. To the extent that the attacker can violate sender privacy (see the discussion above), attackers can try to infer which sets of nodes are communicating with each other, based on statistical assumptions about traffic communication patterns (e.g., end-to-end network latency and inter-message user think time). Such attacks have been studied extensively in the Tor literature (e.g., [SEV+15, BFE+20]). Such attacks do not undermine the privacy of message *data*.

Clique privacy. Much like digest-based optimizations, clique-based optimizations improve network utilization at a cost of potentially revealing information about how participants communicate. In the context of clique-based routing, if a global eavesdropper can observe message exchange patterns between nodes, the eavesdropper might learn clique membership and identify clique leaders (e.g., because a clique member only exchanges messages with its

Table 10: Total message packets sent for increased group update messages, for all routing mechanisms and mobility models; results averaged across 3 runs for 5,000 KB buffer size. Message delivery rate in parentheses. Checkmarks if CGKA group consistent at the end of the simulation.

	Epidemic Flooding	Digest Flooding	Static Clique Routing	Dynamic Clique Routing
static	431,852,558 (1.000) ✓	421,084,442 (1.000) ✓	2,079,735 (0.961) ✗	13,860,544 (1.000) ✓
random	434,070,927 (1.000) ✓	430,559,457 (1.000) ✓	2,122,730 (0.980) ✗	8,838,284 (0.969) ✗
gather	425,652,298 (1.000) ✓	414,564,516 (1.000) ✓	2,011,886 (0.940) ✗	13,255,065 (1.000) ✓
chain	35,543,959 (1.000) ✓	35,566,106 (1.000) ✓	36,993 (0.017) ✗	14,735,814 (1.000) ✓
march	51,642,288 (0.866) ✗	51,394,894 (0.866) ✗	29,384 (0.026) ✗	6,384,905 (0.858) ✗
blockade1	11,349,663 (0.271) ✗	11,348,812 (0.271) ✗	18,765 (0.008) ✗	90,545 (0.060) ✗
blockade2	447,616 (0.034) ✗	448,140 (0.034) ✗	8,347 (0.005) ✗	12,901 (0.009) ✗

clique leader). However, we note that cliques are a routing-layer concept; learning cliques does not reveal anything about application-layer group chat communication.

As we note in Section 3, we choose not to introduce dummy traffic as a mechanism to thwart these observation attacks. Our experiments demonstrate that dummy traffic may exacerbate congestion in ad-hoc networks that are already congestion-prone (§7.2). In absence of dummy traffic, MAC address randomization may help improve clique privacy. We discuss further in Appendix L.

K Assessing Group State Consistency

Significantly increasing the number of group updates allows us to understand how Amigo will perform under highly dynamic groups. We update our traffic models to include 100 group state update messages. We send these messages in the first 2,000 seconds of the simulation, so that we are able to account for the time it may take for the packets to traverse the network in delivery evaluations. This represents 5× more group updates occurring compared with our primary simulation runs, with each group updating its state almost each minute, exceeding what we expect in practice.

We run our routing mechanisms with these updated traffic models, for 100 nodes across all mobility models, with a buffer size of 5,000 KB. Our results are shown in Table 10. Comparing this table with Table 2, which presents our results from original simulation runs with a lower number of group updates, we do not observe a significant difference in message packets sent, end-to-end message delivery, or ability to maintain group state. This suggests that Amigo will function consistently even under high group churn.

L Randomizing Link-Layer Device Identifiers

Wi-Fi Direct uses MAC addresses as link-layer device identifiers for routing. MAC addresses are implemented differently across different devices, but generally, modern smartphones rotate MAC addresses periodically to avoid tracking from persistent identifiers. We now describe how Amigo is compatible with this MAC address randomization, thereby achieving better user privacy.

At the routing layer, MAC addresses are exposed to networking code. This fact is most relevant to clique-based routing schemes in which followers must know the MAC address of the leader. Even so, MAC address rotation does not meaningfully impact Amigo's novel dynamic clique routing mechanism, since dynamic clique routing already permits a leader's MAC address to change between epochs. So, as long as this address is stable within an epoch, rotation across epochs will not impact functionality.

We additionally examine the implications of MAC address rotation for prior schemes. For flooding-based approaches, MAC address rotation has no impact, as all nodes exchange

and forward messages opportunistically; as such, identifiers need only be valid for a single broadcast and can be rotated afterward. For static clique routing [PSEB22], however, MAC address rotation *does* have an impact, as identifiers used to initially identify clique members and leaders may no longer be valid later in the protest. Thus, MAC address rotation may cause members and leaders to no longer recognize one another.

Additional mechanisms could enhance the stability of delivery for static cliques with MAC address rotations; for example, the leader may sign each forwarded message using cryptographic state that identifies the leader (e.g., a public/private key pair whose public half is shared with all clique members in the pre-protest setup phase). Even if a leader's MAC address changes, the leader would maintain knowledge of its signature-related keys. So, when a clique follower receives a message from an ostensible static leader with a new MAC address, the follower can verify the message signature to determine the authenticity of the leader's new MAC address.

We note that the discussion of MAC address randomization is only necessary at the routing layer. The application layer is not exposed to MAC addresses, because user-facing messages are sent to application-level groups, not network-level MAC addresses. A node that receives an application-layer message determines if the message is relevant using trial decryption, not by examining the message's Layer 2 destination MAC address. So, at the application level, MAC address rotation has no impact.