Myco: Unlocking Polylogarithmic Accesses in Metadata-Private Messaging

Darya Kaviani[1] Deevashwer Rathee[1] Bhargav Annem[2] Raluca Ada Popa[1]

[1]UC Berkeley [2]California Institute of Technology

***Abstract*—As billions of people rely on end-to-end encrypted messaging, the exposure of metadata, such as communication timing and participant relationships, continues to deanonymize users. Asynchronous metadata-hiding solutions with strong cryptographic guarantees have historically been bottlenecked by quadratic $O(N^2)$ server computation in the number of users N due to reliance on private information retrieval (PIR). We present Myco, a metadata-private messaging system that preserves strong cryptographic guarantees while achieving $O(N \log^2 N)$ efficiency. To achieve this, we depart from PIR and instead introduce an oblivious data structure through which senders and receivers privately communicate. To unlink reads and writes, we instantiate Myco in an asymmetric two-server distributed-trust model where clients write messages to one server tasked with obliviously transmitting these messages to another server, from which clients read. Myco achieves throughput improvements of up to 302x over multi-server and 2,219x over single-server state-of-the-art systems based on PIR.**

1. Introduction

While end-to-end encrypted messaging applications such as WhatsApp [1], Signal [2], and Messenger [3] conceal message content, they still expose metadata such as who communicates with whom, when, and how many messages they exchange. This information could enable an attacker to identify a whistleblower through discreet exchanges with a journalist or uncover an activist network by monitoring interactions during protests. Metadata alone has historically deanonymized real-world users [4], [5], [6], [7]. To this day, service providers continue to share sensitive user data with government agencies and third parties [8], [9], [10], [11], [12], [13], exposing users to ongoing threats such as mass surveillance [8], [11], [12].

A broad range of prior systems [14], [15], [16] aim to hide metadata in secure communication. A key challenge in this line of work is achieving strong cryptographic privacy, while maintaining crucial system properties, including near-linear server overhead and asynchrony. Asynchronous metadata-private communication systems allow clients to go temporarily offline without losing their messages. Prior systems typically satisfy only a subset of these properties.

Near-linear overhead & asynchrony. Mix networks [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32] rely on a set of servers to shuffle user messages, resulting in $O(N)$ server overhead in the number of users N. Despite their efficiency, these works usually rely on weaker security guarantees such as heuristic or differential privacy [33], [34], [35], [36], [37]. Recent mixnet advancements have enabled asynchronous conversations with sub-quadratic server overhead, but only with differential privacy [36]. An alternative research direction [38], [39] also achieves near-linear overhead and asynchrony, but relies on trusted hardware.

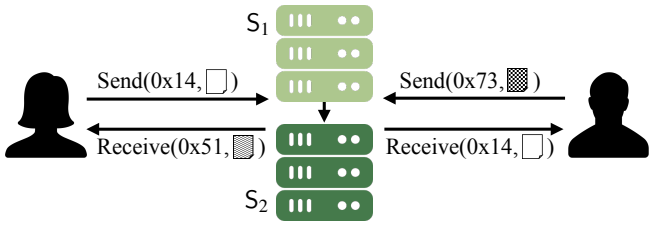


Figure 1: Myco system model: Clients write to S_1, who obliviously writes to S_2, from which clients read.

Cryptographic privacy & near-linear server overhead. The few attempts at mixnets with cryptographic privacy [24], [32] require precise coordination between clients to initiate communication, limiting users to a single synchronous conversation during which they cannot go offline.

Asynchrony & cryptographic privacy. Another line of work provides both cryptographic guarantees and asynchrony by relying on private information retrieval (PIR) [40], a primitive where clients query a database without revealing the queried index [41], [42], [43], [44], [45], [46], [47], [48], [49], [50]. Most PIR-based communication systems rely on a set of non-colluding servers [43], [44], [45], [46], [47], [48], [49], [50], while some only require a single server at the expense of higher computational cost [41], [42]. PIR-based messaging systems are a subset of these works that can support the general-purpose messaging setting [41], [42], [43], [44]. Messages are stored in a database, and receivers use PIR to retrieve their message without revealing which message was read. Messages can be read anytime between when they are written to the database and an expiration time (time-to-live).

However, for N users, PIR-based messaging systems incur $O(N^2)$ server computation for N queries on a database containing $O(N)$ messages. Although preprocessing PIR can reduce per-query computation [51], [52], [53], messag-

ing systems have not benefited from these advancements because the messaging database is dynamic and continually being updated. While batch PIR [42], [54], [55] allows a single client to amortize the cost of performing several queries, it does not amortize costs across users, so these systems nevertheless suffer from quadratic server overheads.

To this end, we present Myco[1], a metadata-private messaging system that surpasses the quadratic bottleneck of PIR-based messaging systems. Myco achieves $O(N \log^2 N)$ server computation by unlocking polylogarithmic accesses as opposed to linear. Like in prior work [43], [44], [45], [46], [47], [48], [49], [50], we leverage the two-server model; however, unlike prior work, we depart from PIR and instead adopt an oblivious RAM (ORAM)-inspired approach, which has typically been discounted in a multi-user setting such as messaging due to its reliance on a trusted client [44]. We overcome this restriction by introducing a novel oblivious data structure that uses the distributed-trust servers *asymmetrically*, as depicted in Figure 1. In doing so, Myco achieves orders of magnitude higher throughput than PIR-based systems. Myco protects both the content and metadata of conversations between honest clients, even if an adversary controls any number of clients and up to one server. The result is the first asynchronous metadata-private messaging system that achieves sub-quadratic server complexity with cryptographic metadata privacy.

1.1. Technical Overview

Asymmetric distributed trust. PIR-based messaging systems [41], [44] are symmetric, which means that all servers perform identical operations. Each pair of communicating clients maintains a shared secret that is used to derive a pseudorandom location ℓ. A sender writes a message to ℓ and the receiver reads by performing a symmetric PIR read at location ℓ across all servers.

In contrast, as depicted in Figure 1, Myco has an asymmetric distributed-trust model: S_1 handles message writes, while S_2 manages reads, decoupling reads from writes. As in the PIR approach, we arrange the database as a series of buckets. We start with a naive attempt where S_1 instantiates a new hash-table of buckets and senders write to S_1 at the pseudorandom bucket ℓ. Next, S_1 copies the hash-table of received messages to S_2. Receivers can then compute the bucket ℓ to read from S_2 and decrypt the message. Since S_1 does not know who reads each message and S_2 does not know who writes them, the communicating parties remain private. This simple setup offers $O(N)$ server work across N clients.

However, simply adopting the bucket structure from PIR-based messaging systems would not be secure, as an adversary controlling S_2 and a fraction of clients could infer the honest write locations, allowing it to link reads and writes. The issue is that the PIR-based works set up the hash-table of buckets such that a message's bucket placement depends on other messages in the hash-table. This interdependence is safe in PIR due to its protection of read access patterns, but PIR incurs a costly linear scan that Myco seeks to avoid.

Independent bucket assignment. Myco remedies this leakage by ensuring that messages are written to locations *independent* of the rest of the messages in the database. To achieve this, buckets are configured to behave as if they are *infinitely sized*: the bucket capacity is set large enough to accommodate all messages in their designated bucket ℓ. Before sending the hash-table to S_2, S_1 fills the empty blocks in each bucket with dummy blocks and shuffles each bucket. This prevents the adversary from inferring information about honest writes based on the position of its own messages within the bucket. Still, a group of malicious senders could attempt to use non-pseudorandom locations to clog up specific buckets and disrupt the protocol.

To address this, Myco prevents malicious senders from biasing write locations by introducing an additional layer of randomness at S_1. This allows us to ensure that a bucket capacity growing only logarithmically in N (Theorem 2) is sufficient to maintain the infinite bucket behavior. Specifically, senders first derive an intermediate bucket index f, which S_1 incorporates with its randomness to compute the final bucket index ℓ from f. After transferring the hash-table to S_2, S_1 broadcasts its randomness, allowing receivers to compute ℓ from f. Clients then read bucket ℓ.

As in prior work, our system operates in discrete time intervals, or epochs, during which each client performs a fixed number of accesses. In each epoch, S_1 writes a hash-table with N buckets to S_2, which stores a matrix of $N \cdot \Delta$ buckets at a time, where Δ is a fixed duration of epochs after which messages expire. We dub this protocol Matrix-Myco.

In the current design, receivers that were offline for δ epochs must read δ buckets to conceal the epoch in which a message was written, which is crucial for hiding user relationships. In addition, if a user is part of Q conversations, it must read from every conversation in every epoch because it can potentially miss messages otherwise. While this achieves the desired asynchrony property, it scales poorly in realistic scenarios involving prolonged offline periods and a large number of conversations. Therefore, we require a more robust primitive to ensure that reads remain oblivious *across epochs*, allowing us to adaptively fetch messages across conversations and epochs as needed.

Read-static messaging tree. To this end, we draw inspiration from the tree-based oblivious RAM (ORAM) literature [56], [57], [58], namely Path ORAM [59], and instead arrange the buckets in a binary tree in S_2. However, as discussed, ORAM has long been dismissed by multi-client messaging systems [44]. The issue is that the ORAM client is trusted as it must store sensitive state like the keys (that can decrypt user data), and the location of data within the ORAM server (such as a position map or stash).

Given this secret state, it is unclear which party should serve as the ORAM client in our system. Clients cannot maintain this sensitive state locally, as this would reveal the locations of other honest clients' messages to an attacker

1. **M**essaging **Y**our **C**ompanions **O**bliviously is inspired by **myco**rrhizal fungi, which plants use for sending anonymous signals through tree roots.

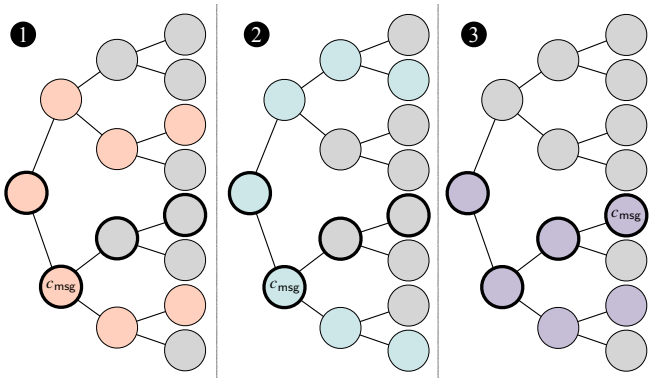


Figure 2: Example of percolation for a message c_{msg} through the tree over several epochs. Color-shaded nodes indicate the path-set, and bolded nodes show the intended message path. In epoch (1), the message is written to the LCA. In epoch (2), it remains stationary as no deeper nodes on the intended path are selected. In epoch (3), more nodes on the path are in the path-set, so c_{msg} moves deeper towards the leaves.

controlling even a single client. If S_1 acts as an ORAM client, and S_2 is the ORAM server, clients would need to route writes and reads through S_1. This would allow an adversarial S_1 to link reads to writes.

To this end, we propose a solution that exploits the inherent access pattern of messaging to eliminate the need for clients to route reads through S_1. Since messages are accessed only once, clients can read directly from S_2 without reassigning them through S_1 after the initial read. This allows us to decouple reads from writes while allowing S_1 to handle writes as an ORAM-like client.

Now, we have enhanced our asymmetric model such that S_1 acts as an ORAM-like client, and S_2 acts as an ORAM-like server, but rather than relying on a client-side position map as in ORAM, both the sender and receiver can independently and dynamically compute their pseudo-random message location ℓ. To read, the receiver can simply download the path $\mathscr{P}(\ell)$ from the root to leaf ℓ from S_2's tree, and trial-decrypt to locate its message.

Oblivious batch evictions. We have yet to discuss how S_1 can write messages along the path $\mathscr{P}(\ell)$ without disclosing ℓ to S_2. If S_1 simply reads, edits, and writes back path $\mathscr{P}(\ell)$, S_2 can trivially link the receiver's read to S_1's write by correlating when particular paths are written with when they are read. S_1 could write all messages to the root of the S_2 tree, but this bucket would quickly overflow. In Path ORAM, evictions happened with reads, but our read-static requirement demands a more creative approach.

To address this, Myco leverages the high volume of messages sent in a messaging system to process writes in batches. In each epoch, S_1 randomly samples N paths from S_2. Next, S_1 writes each message (both new and pre-existing in the path-set) to the deepest node along its intended path that is part of the path-set, or the least common ancestor (LCA). Recall that if we did not always write exactly to the LCA, this would violate independent bucket assignment.

The randomness in sampling both ℓ and the path-set itself ensures that the messages will fit into the buckets with high probability. After each epoch, S_2 will only see that N uniformly random paths were replaced with mutually indistinguishable blocks. Since the eviction path-set is independent of the messages' intended paths, S_2 learns nothing about the intended path of any messages. Over the epochs, messages will percolate effectively towards the leaves of the tree as new path-sets are chosen, as depicted in Figure 2. This frees up space closer to the root for freshly written messages. In addition, messages stored past Δ epochs will expire, ensuring that the tree never fills up. We dub this protocol Tree-Myco. Theorem 3 proves that logarithmically-sized buckets in N and Δ are sufficient to prevent overflow, ensuring polylogarithmic accesses.

Private notifications. While Tree-Myco reads offer cross-epoch obliviousness and polylogarithmic accesses, receivers must still *guess* which path to read each epoch and cannot query the same path again. This is unlike Pung and Talek, where repeated reads are leakage-free and mitigate the risk of missed messages. Indeed, we need a way for a client to know which conversation and epoch to fetch from if it is in Q total conversations.

To eliminate the guesswork, we revisit Matrix-Myco, which has the limitation that it must fetch messages from every epoch and every conversation to hide access patterns across epochs and ensure no message is missed, respectively. Since Tree-Myco needs to know the exact epoch and conversation to fetch from in order to only read $q \ll Q$ real messages, we can use Matrix-Myco to receive this signal. The hybrid solution is still efficient because this signal can be a small λ-bit notification, which is $16\times$ smaller than messages considered in prior work for $\lambda = 128$ [41], [42]. Consequently, our notification cost scales linearly with Q and δ, and logarithmically with Q and N, while fetching actual messages via Tree-Myco scales polylogarithmically with N and Δ. Thus, clients use Matrix-Myco exclusively for λ-bit notifications, enabling receivers to efficiently prioritize their Tree-Myco conversations.

We present protocol details in §3, a simulation-based security definition in §4, a capacity analysis in §5, and a security proof in §A.

1.2. Evaluation Summary

In §6, we evaluate Myco's performance against the PIR-based systems Talek++ and Pung++, which incorporate state-of-the-art PIR techniques into the multi-server Talek [44] and single-server Pung [41] setups, respectively. Myco delivers up to $302\times$ higher throughput than Talek++ and up to $2,219\times$ higher throughput than Pung++, with throughput gains increasing as the number of system users increases. The end-to-end latency for delivering a *single* message in Myco ranges from a few seconds to a minute, which is higher than for PIR-based works. This is due to Myco's batching, which accounts for messages from all system clients, while PIR works report the best-case latency assuming a single message. If we consider the latency of

prior work under typical message volumes, Myco surpasses both Pung++ and Talek++.

2. System Overview

2.1. System Architecture

System roles. A Myco deployment consists of N clients and two servers, S_1 and S_2, as illustrated in Figure 1. S_1 and S_2 operate under distinct trust domains. S_1 handles client writes, then transmits these writes obliviously to S_2. S_2 subsequently manages client reads. Each conversation involves a client pair—a *sender* and a *receiver*. The sender writes a message to S_1, and the receiver retrieves it from S_2 in the next epoch during which it is online.

Epochs & cover traffic. Myco operates in discrete intervals called *epochs*, tracked independently by each server. Clients locally keep track of epochs and query both servers if they lose track, ensuring that both servers agree on the current epoch. Myco resists traffic analysis by generating a consistent rate of fixed-size, random-looking reads and writes for each epoch, using dummy requests when no real activity occurs. This cover traffic effectively hides actual communication patterns.

Contacts. Myco clients can participate in up to Q conversations with Q contacts, but write to and read from at most $q \ll Q$ conversations per epoch. Clients receive notifications through Myco's private notification system, and define their own fetch policies to decide which q conversations to read from out of the Q total conversations. We set $q = 1$ for simplicity, though Myco supports multiple conversations per epoch. This is unlike prior work, which relies on expensive dialing protocols to determine the active conversations [20], [24], [29], [33], [34], [35], [36], [41], [42], [44], [49], [50], [60], [61], [62], [63], [64], [65].

Shared keys. As in prior communication systems [20], [24], [29], [33], [34], [35], [36], [41], [42], [44], [49], [50], [60], [61], [62], [63], [64], [65], users must exchange some information through an out-of-band channel prior to communication (e.g., keys or pseudorandom addresses). In Myco, we assume that users intending to communicate can establish a shared symmetric key. Orthogonal work facilitates this for client pairs through bootstrapping from other applications [66], scanning one another's QR codes in-person, or a metadata-private "add-friend" protocol as in Alpenhorn [37]. Like prior work, we have not added forward secrecy to shared keys, and leave this to future work.

2.2. Threat Model & Security Guarantees

We begin with an informal discussion of Myco's threat model and security guarantees. In §4, we present our formal security definition and theorem. Informally, Myco offers secure two-way communication over the Internet, concealing both message content and metadata—such as timestamps, message count, and participant identities—from everyone except the communicating users. This approach protects both the content of messages and any data that could reveal information about the communication itself. Myco provides privacy guarantees only to conversations between two honest clients; a compromised client can trivially leak the plaintext messages shared with an honest client, though this does not compromise the security of the honest client's other conversations.

Threat model. We consider a threat model where any subset of clients and up to one of the two servers may be maliciously corrupted. Like many PIR-based communication systems [43], [44], [45], [46], [47], [48], [49], [50], we adopt a two-server distributed-trust model. This stands in contrast to their single-server counterparts [41], [42], which achieve a stronger threat model at the cost of reduced performance. Informally, Myco guarantees the following security properties even if the adversary deviates arbitrarily from the protocol, as long as both servers are not simultaneously compromised.

Message privacy and integrity. As long as the communicating clients are honest, messages remain confidential, accessible only to intended recipients, and any tampering or replay attacks are detectable by the receiver to ensure integrity.

Metadata privacy is more difficult to achieve and is the primary challenge of Myco. In Myco, an adversary cannot discern whether any conversation is happening between any pairs of honest clients. This property holds even if the adversary corrupts one of the servers and all of the clients except the two communicating clients. Myco's guarantees are in line with Talek's access sequence indistinguishability [44], where the adversary cannot distinguish between an honest user's access patterns and a random access pattern of the same length.

Although the adversary can infer the anonymity set of the write corresponding to a read based on the number of clients that were online, this does not reveal communication patterns in Myco because clients also issue fake reads indistinguishable from real reads. If all honest clients sent and received a (fake or real) message within an allocated timeframe, the precise metadata-hiding guarantee achieved by Myco is communication unobservability (CUS) [14], where the adversary may infer possible senders and recipients but cannot even detect the existence of a conversation between honest clients.

If honest clients are permitted to go offline and not send cover traffic, then Myco still achieves *at least* relationship unobservability (RUS) [14], where the adversary may infer possible senders and recipients, as well as the number of ongoing conversations, but cannot identify any sender-receiver pair from the pool of active correspondents. In fact, the anonymity is stronger because in the extreme case where only one honest pair of communicating clients was online, the adversary cannot infer whether the clients are communicating or if they are just performing fake accesses. If the adversary learns side-channel information that a message was received by an honest client in epoch t, its anonymity set includes all messages written by online honest clients in epochs $[t - \Delta, t)$, each equally likely to be the one read.

Online-offline behavior. Like other asynchronous metadata-private messaging systems [36], [41], [42], [44], Myco allows users to go offline temporarily as long as the times at which they go offline or online is independent of their communication patterns. As a result, an adversary will not be able to distinguish between any two plausible access sequences of the same length. We model this in our security definition by allowing the adversary to choose which clients are online at any given epoch. We also assume that *within* each epoch, clients query at a time independent of their communication patterns (e.g., at a fixed or random time).

Availability. Myco does not offer protection against denial of service (DoS) attacks. Either server can misbehave to block honest clients from exchanging messages, though this does not compromise their privacy. In particular, any server can block communication between a set of clients without gaining any information about whether those clients were communicating. Thus, we assume that services supporting client communication, such as ISPs, DNS for name resolution, and servers handling requests, do not deny service. Malicious clients alone cannot disrupt service to honest clients. Like Pung and Talek, Myco assumes that clients do not alter their actions in response to server misbehavior (e.g., if the client does not receive an expected message). This ensures that Myco's guarantees remain intact even in the face of selective failure attacks.

3. The Myco Protocol

We now describe the Myco protocol. In §3.1, we provide an overview of the message flow through a simplified version of Matrix-Myco. In §3.2, we introduce independent bucket assignment, resulting in Matrix-Myco. In §3.3, we extend these building blocks to Tree-Myco, which uses a tree-based oblivious data structure to overcome Matrix-Myco's limitations through cross-epoch obliviousness. In §3.4, we describe how we use Matrix-Myco for a private notification system, resulting in the full Myco protocol. Algorithm 1 denotes the full Myco client and server API.

3.1. Overview of Myco

We begin by describing a simplified protocol flow through Myco's asymmetric distributed-trust servers.

3.1.1. Client setup. To exchange messages in Myco, a client must first run $\mathsf{client.Setup}(\vec{k}_Q)$, where $\vec{k}_Q$ maps each of the user's Q contacts' user IDs to the shared key with that contact. This operation securely derives a set of keys from each shared key k, known exclusively to the communicating client pair. In particular, $k_{\mathsf{enc}} \leftarrow \mathsf{KDF}(k, \texttt{"enc"})$ is responsible for message encryption and $k_{\mathsf{rk}} \leftarrow \mathsf{KDF}(k, \texttt{"routing key"})$ derives the message location, where $\mathsf{KDF} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a secure key derivation function [67].

3.1.2. Sending messages to S_1. Writes are processed through S_1, which handles client-written messages and obliviously writes a batch of these messages to S_2 in each epoch. To write a message to a contact, a client invokes $\mathsf{client.Write}(c_r, m)$, where m is the message intended for the receiver c_r. In a write, the derived key k_{enc} encrypts the plaintext message m into ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}_{k_{\mathsf{enc}}}(m; t)$, concealing the message content from S_1. $\mathsf{Enc}_k(m; t)$ is a key-private (§C) AEAD (authenticated encryption with additional data [68], [69], [70]) scheme that takes key k, plaintext m, and integrity tag input t, where the current epoch t is taken as authenticated data to prevent message replays. Due to the key-privacy property, this ciphertext does not reveal anything about the key used to encrypt it. Thus, it does not reveal anything about the intended recipient of the message.

k_{rk} is then used to key a pseudorandom function [71] with input t to generate a bucket index $\ell \leftarrow \mathsf{PRF}_{k_{\mathsf{rk}}}(t)$ that determines the message's location, where $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^{|\ell|}$. Due to the security of the pseudorandom function, ℓ appears random to S_1 during the write, irrespective of which of its conversations' PRF key the sender used. The sender then transmits ct and ℓ to S_1 by calling $\mathsf{S}_1.\mathsf{Write}(\mathsf{ct}, \ell)$, which adds ct to bucket ℓ of S_1's epoch-specific hash-table T_t.

3.1.3. Batch writes from S_1 to S_2. S_1 is tasked with writing messages to S_2 while concealing the sender and therefore the conversation associated with each message. $\mathsf{S}_1.\mathsf{BatchWrite}()$ transfers the table T_t to S_2, from which clients can read. S_1 signs [72], [73] each bucket in the batch write to allow clients to detect tampering by S_2.

Upon receiving a batch write, S_2 increments its local epoch counter. Messages written in the current epoch will be made available to receivers beginning in the next epoch.

3.1.4. Receiving messages from S_2. To read their message, the receiver uses k_{rk} to compute ℓ and fetches the ℓ-indexed bucket from S_2. The receiver then trial-decrypts each entry in the bucket using k_{enc} until it successfully decrypts its message m. Since the message was encrypted by an AEAD, the receiver can detect any tampering from the servers.

Because the servers already know which epochs each client goes offline, clients can read from all missed conversations after coming back online after δ epochs without leaking new information. Let L be the matrix of $Q \cdot \delta$ message locations, one for each potential bucket containing a message for the receiver across the δ unread epochs. The receiver sends L to S_2, who responds with the buckets at the requested indices.

In §3.2, we will discuss how we ensure that the bucket indices do not reveal any information to S_2 about which conversations were written to. In addition, the receiver will not miss any messages since it checks all Q conversations for each system epoch. Similarly, S_1 has no knowledge of when reads occur or which receiver accessed which buckets, so it cannot link the metadata of written messages to a specific receiver, conversation, or read access.

3.1.5. Message expiration. We must support asynchrony while preventing the message database from growing indefinitely. To do so, messages are stored in S_2 for Δ epochs before being garbage-collected. This mirrors the approach of current messaging apps [1], [2], [3], which temporarily store messages before requiring the receiver to re-request the message from the sender once they are online again. In Matrix-Myco, this means the hash-table $T_{t-\Delta}$ is deleted from S_2's memory every epoch t.

3.1.6. Fake accesses. Typically, a client would not want to perform Q reads and writes per epoch. As such, each online client must perform some fake accesses to pad their accesses in each epoch to Q. In particular, if no real message needs to be sent, the client generates dummy values for these elements using $\mathsf{client.FakeWrite}()$, which is indistinguishable from a real write from the perspective of S_1 and therefore S_2. A fake write calls $\mathsf{S}_1.\mathsf{Write}(\mathsf{ct}', \ell')$ where $\mathsf{ct}' \leftarrow \mathsf{Enc}_{k'}(0^{\mathbf{m}})$, $k' \xleftarrow{\$} \{0,1\}^{\lambda}$, and $\ell' \xleftarrow{\$} \{0,1\}^{|\ell|}$. Similarly, $\mathsf{client.FakeRead}()$ downloads bucket $\ell' \xleftarrow{\$} \{0,1\}^{|\ell|}$. Due to the security of the key-private AEAD scheme and the pseudorandomness of the PRF, fake accesses are indistinguishable from real ones, which we ensure in §3.2 and prove in §A.

3.2. Matrix-Myco

In this section, we will complete the Matrix-Myco protocol. As we have detailed, each epoch t has a new hash-table T_t, consisting of $Q \cdot N$ buckets. S_1 places each client's message ct in bucket ℓ of T_t. Myco clients write to a bucket using a pseudorandom location ℓ. At the end of the epoch, S_1 fills all empty blocks of T_t with dummy values, and shuffles each bucket. We now enhance our bucket structure with independent bucket assignment to protect against malicious clients and ensure fake read indistinguishability.

3.2.1. Infinite buckets. Each bucket must inherently have a bucket capacity Z_M. Suppose that Z_M is a small constant, and messages get pushed to another bucket if bucket ℓ is full. This is the case in Talek [44], in which message locations depend on other messages in the database. This does not leak any information for Talek because the PIR reads do not reveal the bucket that was read to the servers.

In our case, an adversary controlling a subset of receivers and S_2 could infer information about the placement of honest clients' messages by observing which bucket it found its message in. For instance, if the receiver finds its message in a backup bucket, it can deduce that the first bucket it checked was full with other honest user messages, leaking information about those messages. Similarly, if a fake read occurs to a bucket which is full of malicious client messages, the attacker can deduce that the read was fake.

By always adhering to the invariant that receivers find their messages in a deterministically computable bucket, a malicious receiver colluding with S_2 cannot gain additional information based on which bucket contains its message. This effectively requires that we configure our buckets to act as *infinitely sized*; in particular, Z_M should be large enough such that messages will always fit in the bucket corresponding to their PRF location ℓ.

3.2.2. Fake access indistinguishability. Infinite buckets ensure that fake and real accesses are indistinguishable.

Fake & dummy writes. Recall that before writing buckets to S_2 at the end of each epoch, S_1 adds dummy blocks to fill each bucket, then randomly shuffles them before sending them to S_2, which prevents malicious receivers from learning anything from their message's position *within* the bucket. To remain indistinguishable from real values, dummy blocks are generated by S_1 as $\mathsf{Enc}_{k'}(0^{\mathbf{m}})$ where $k' \xleftarrow{\$} \{0,1\}^{\lambda}$.

Fake reads. The only difference between a real read and a fake read is that the former has a corresponding real write. As long as each fake read could plausibly correspond to a real write, the adversary cannot distinguish real reads from fake ones. Since real writes are indistinguishable from dummy writes, and we can accommodate any number of valid real writes, all fake reads could also potentially have a corresponding real write, and fake and real reads are indistinguishable.

3.2.3. Layered PRFs for unbiasable writes. A remaining challenge in our bucket structure is preventing a malicious sender from crafting a message location ℓ to distort the distribution of message placements, potentially causing bucket overflows. Naively, we would have to set the bucket capacity very large to ensure that buckets behave as if they are infinitely sized, because an attacker could force all of its messages in the same bucket.

To prevent such an attack, S_1 does not write the sender's ciphertext ct directly to the client-specified location. Instead, we use the sender's submitted PRF output as an intermediate input f. S_1 then derives the final message location ℓ by applying an additional PRF to f, using an epoch-specific, client-agnostic server routing key $k_{\mathsf{srk},t}$. In particular, $\ell \leftarrow \mathsf{PRF}_{k_{\mathsf{srk},t}}(f, c_s)$. After the epoch, S_1 sends $k_{\mathsf{srk},t}$ to S_2. To compute ℓ, the receiver then downloads $k_{\mathsf{srk},t}$ from S_2 and uses it to key the external PRF layer, yielding the message location ℓ. Since each message is placed at the output of a PRF on distinct inputs, this distributes each message uniformly over the buckets and thwarts any adversarial strategy to manipulate bucket distribution. In Theorem 2 of §5, we prove that the asymptotic bucket size of Matrix-Myco is logarithmic in $Q \cdot N$, resulting in a corresponding asymptotic access linear in $Q \cdot \delta$ and logarithmic in $Q \cdot N$.

3.3. Tree-Myco

Matrix-Myco has a shortcoming: The number of buckets downloaded by the client scales with $Q \cdot \delta$, which can be problematic since clients typically have many contacts Q, and can go offline for an extended number of epochs δ. We now introduce Tree-Myco, which solves this problem

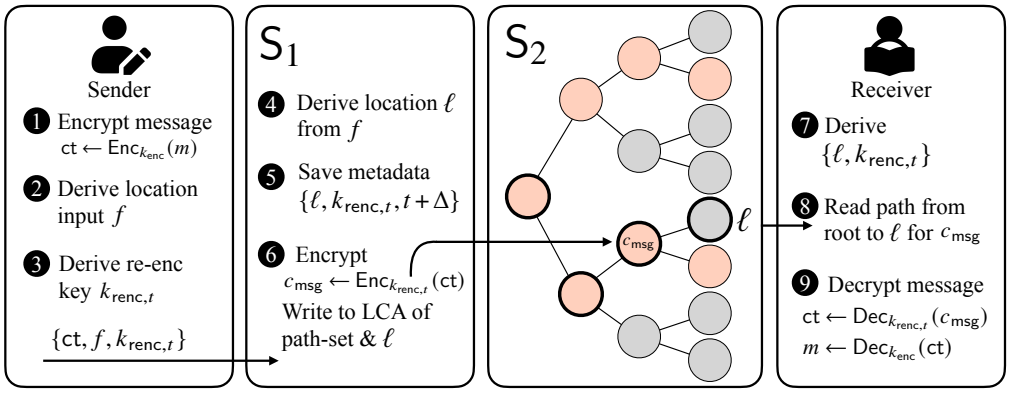


Figure 3: Simplified message flow of a sender sending a single message m through Tree-Myco to message location ℓ, and a receiver reading this message. Shaded orange indicates the epoch's path-set and tree nodes with bold outline indicate the path read by the receiver, $\mathscr{P}(\ell)$.

with cross-epoch obliviousness. The Tree-Myco message database is instead arranged as a binary-tree in S_2, taking inspiration from the tree-based ORAM literature [58], [59]. S_1 obliviously writes to a bucket on the root-to-leaf path $\mathscr{P}(\ell)$, and receivers download the path $\mathscr{P}(\ell)$, trial decrypting the path to locate their message. Figure 3 shows the message flow through Tree-Myco.

3.3.1. Setup. Since the entire message database is stored in a single data structure unlike Matrix-Myco's epoch-specific hash-tables, S_1 must re-encrypt any buckets that are edited to prevent S_2 from observing how the messages are moving across the tree. To do so, however, S_1 needs to re-encrypt message ciphertexts with a key that the receiver can derive. Thus, in addition to k_{enc} and k_{rk}, client.Setup derives a conversation-specific re-encryption key k_{renc}. However, it is essential to prevent S_1 from deducing the receiver's identity by correlating re-encryption keys used by the sender across different epochs. To address this, we can create message-specific values by making them unique to the current epoch t. To this end, senders derive a message-specific re-encryption key $k_{\mathsf{renc},t} \leftarrow \mathsf{PRF}_{k_{\mathsf{renc}}}(t)$. S_1 encrypts the ciphertext ct with $k_{\mathsf{renc},t}$ to create a double-encrypted ciphertext c_{msg}. This also means we now compute dummy blocks as $\mathsf{Enc}_{k'}(\mathsf{Enc}_{k''}(0^{\mathbf{m}}))$ where $k', k'' \xleftarrow{\$} \{0,1\}^{\lambda}$.

Both S_1 and S_2 store copies of the tree. S_2's copy is where clients read from. S_1's copy is void of dummy blocks and is simply $\perp$ for blocks with no real message. Additionally, S_1 also stores a metadata tree T_{md}, which is a mirror of the message tree storing the $k_{\mathsf{renc},t}$ value, intended path ℓ, and expiration date t_{exp} of each message, which will be used in future batch evictions.

3.3.2. Batch evictions. S_1 must hide the message's intended path $\mathscr{P}(\ell)$ when writing a c_{msg} value to S_2. At the start of each epoch t, we call S_1.BatchInit(), in which S_1 randomly samples N leaves with replacement, where N denotes the number of clients in the system. S_1 creates a path-set P from the union of these root-to-leaf paths in the tree. P contains the only buckets where messages for the current epoch will be written. Each c_{msg} is added to the bucket at the least common ancestor (LCA) of P and $\mathscr{P}(\ell)$. Recall from §3.2 that Myco requires independent bucket assignment. Thus, a critical difference between our eviction protocol and Path ORAM's is that in Path ORAM, messages are simply evicted as deep as they go within the freshly sampled path. This would lead to leakage in Myco. For instance, if the receiver finds its message at a node closer to the root than the LCA, it can deduce that the LCA was full with other honest user messages, leaking information about those messages' intended paths.

S_1 follows the same process as fresh message writes for handling preexisting messages found within P. First, it retrieves the corresponding metadata from the relevant bucket and block in T_{md}. If t_{exp} has passed the current epoch, the message is expired and is deleted from the path-set P and T_{md}. If the message is unexpired, S_1 reads the message's intended path ℓ from the metadata tree and inserts the data c_{msg} into the bucket that is the LCA of the current epoch's path-set and ℓ. Finally, $k_{\mathsf{renc},t}$ from the metadata is used to re-encrypt the underlying ciphertext ct, providing a fresh encryption for the message. The corresponding block in T_{md} is updated accordingly. In order to avoid timing attacks, we ensure that each bucket involves Z_T decryptions and Z_T encryptions in total, where Z_T is the Tree-Myco bucket capacity. This avoids timing attacks by ensuring that each epoch's duration is independent of the number of dummies vs. real messages in the path-set.

At the end of the epoch, S_1 sends the path-set P to S_2, which overwrites the corresponding paths with S_1's edited path-set. As in Matrix-Myco, all data in the buckets is re-encrypted, while empty blocks are filled with dummy blocks. Consequently, S_2 can only observe that a set of random paths were written by S_1 with data indistinguishable from dummy data.

Figure 2 depicts an example of a message's movement through the tree over time. Messages gradually move deeper towards the leaves through successive epochs. In Theorem 3 of §5, we prove that the asymptotic bucket size of Tree-Myco is logarithmic in N and Δ, resulting in polylogarithmic accesses in N and Δ per user.

3.3.3. Receiving messages. Tree-Myco enables receivers to download a single root-to-leaf path in which they are guaranteed to find their message. As such, receivers only need to process a polylogarithmic number of data blocks by

deriving ℓ and downloading the path $\mathscr{P}(\ell)$ from S_2. Since the honest receiver does not know which path-sets were selected in previous epochs, it trial-decrypts each block in the path using $k_{\mathsf{renc},t}$ until it finds the correct one. Finally, it decrypts the inner layer with k_{enc} to retrieve the message m. Because the sender used AEAD, the receiver can detect any replay attacks or tampering to the internal ciphertext by S_1. A malicious S_2 cannot link the receiver's read to any prior write since only the path-sets were written to and re-encrypted per epoch, and S_1 used independent bucket assignment.

3.4. Full Myco

Tree-Myco requires clients to read Q paths every epoch to avoid missing messages, similar to Matrix-Myco. Alternatively, clients could *guess* which path to read each epoch, but could therefore risk missing a message they never checked before expiry.[2] To fully utilize Tree-Myco's cross-epoch obliviousness, a private notification system is needed to identify the paths containing new messages.

Although Matrix-Myco is inefficient for messaging due to the need to check all Q inboxes across δ epochs, its asymptotically quasilinear server workload makes it well-suited for notifications, which only need to be λ bits long (e.g., $\lambda = 128$ bits), $16\times$ smaller than message sizes of prior work [41], [44].

In this design, S_1 and S_2 run a lightweight version of Matrix-Myco, replacing messages with notifications. Clients derive notification keys $\{k^{\mathsf{ntf}}, k^{\mathsf{ntf}}_{\mathsf{rk}}\}$ from their shared key k during client.Setup. When a sender writes a message to S_1 in Tree-Myco, they also write a notification $\mathsf{ct}^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{\lambda,k^{\mathsf{ntf}}}(t)$ to Matrix-Myco, where $\mathsf{PRF}_\lambda : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$. Notification locations are derived using $k^{\mathsf{ntf}}_{\mathsf{rk}}$ to obtain f^{ntf} as the intermediate location, and ℓ^{ntf} as the final bucket index. Receivers compute $\mathsf{ct}^{\mathsf{ntf}}$ and f^{ntf} locally, and if they find $\mathsf{ct}^{\mathsf{ntf}}$ in the bucket with index ℓ^{ntf}, this indicates that the Tree-Myco path derived from t contains a new message from the sender associated with the shared key k. This prompts the receiver to fetch the corresponding path in Tree-Myco. The small size of notifications significantly reduces Matrix-Myco's client bandwidth consumption when downloading missed notifications instead of missed messages after an offline period.

Notifications are unforgeable because to add a fake notification which gets accepted by a receiver, the server would have to guess the PRF output in each epoch t without knowing the PRF key, undermining the security of a PRF. An adversary who deletes or tampers with a notification would simply result in the receiver not finding their message, and has no impact on security.

Push-based system. The private notification system enables a push-based model where each receiver checks their inboxes depending on which notifications it receives. We define a deterministic, client-specific pushPolicy that dictates inbox selection, e.g., prioritizing smaller user IDs, random selection, or client-specified priority lists. Unlike Pung and Talek, where clients must repeatedly check inboxes until a message arrives, Myco only requires downloading a message when one is received.

2. By contrast, systems like Pung and Talek allow repeated queries without revealing information, enabling clients to re-check mailboxes to avoid message loss.

Key Exchange Ideal Functionality $\mathcal{F}_{\mathsf{KE}}$

Initialize keyCount and keyRequests to $\{\perp\}$.

On input $\langle \mathsf{GenerateKey}, c' \rangle$ from client c:

1) If keyRequests contains (c', c), send $k \xleftarrow{\$} \{0,1\}^\lambda$ to c and c'.
2) Else, add (c, c') to keyRequests.

Figure 4: Ideal functionality of symmetric key exchange.

4. Security

4.1. Ideal Functionality $\mathcal{F}$

In this section, we define the ideal functionality $\mathcal{F}$ securely realized by Myco and explain how it encapsulates the properties outlined in §2.2. Specifically, we present a simulation-based definition that captures Myco's security guarantees. Unlike Pung and Talek, which use game-based definitions [41], [44], our definition captures denial of service attacks, which are inherent to prior work but not explicitly captured by their game-based definitions.

4.1.1. Key exchange ideal functionality $\mathcal{F}_{\mathsf{KE}}$. Recall from §2.1 that Myco clients are assumed to have a shared key agreed upon out-of-band. We describe this functionality using $\mathcal{F}_{\mathsf{KE}}$ defined in Figure 4 and prove Myco's security in the $\mathcal{F}_{\mathsf{KE}}$-hybrid model. $\mathcal{F}_{\mathsf{KE}}$ receives $\langle \mathsf{GenerateKey}, c' \rangle$ from client c seeking to communicate with another client c', and returns the same symmetric key k to both if c and c' queried each other. If both clients did not query for each other, they do not receive any signal from $\mathcal{F}_{\mathsf{KE}}$. As a result, a malicious client does not learn whether the honest user is interacting with others in the system. We assume each user's list of Q contacts remains fixed during the protocol for simplicity, though this can be extended to allow dynamic contact additions.

4.1.2. Myco ideal functionality $\mathcal{F}$. Myco's ideal functionality, defined in Figure 5, mediates communication between clients. Clients first specify the list of contacts from which they will accept messages. In each epoch, a client can send one message using Write and can receive one message using Read. If the receiver has more than one unread message, $\mathcal{F}$ applies a deterministic client-specific pushPolicy to prioritize the returned message. Messages are accessible until they expire after Δ epochs. Incrementing the epoch requires a request from the simulator to capture delays introduced by

a corrupted server or network, and $\mathcal{F}$ obliges provided the minimum epoch time has passed.

Message privacy, integrity, and metadata privacy. For honest communicating clients, the adversary learns nothing about message contents or metadata because $\mathcal{F}$ does not send any information to our simulator $\mathcal{S}$ regarding these accesses. All either server or $\mathcal{S}$ should see is when a client makes a write or read request within an epoch, not the content of any message or any knowledge about which parties communicated. In fact, the requests can be fake: For writes, the receiver c_r could be $\perp$, and for reads, there could be no unread messages at all. A message that is received by c_r is marked as read ($\top$) and cannot be marked as unread again. $\mathcal{F}$ also does not allow sending messages to non-contacts by confirming that any message request's receiver contains the message sender in their contact list. To prevent replay attacks, observe that a malicious server can only delete items from the database DB to deny service, but has no ability to replay an already-read message.

Since we cannot ensure security for malicious clients and we do not hide communication patterns in this case, $\mathcal{F}$ notifies the simulator $\mathcal{S}$ of the (c_s, c_r) sender-receiver IDs if at least one of the clients is malicious. For a malicious c_r, $\mathcal{F}$ sends the IDs, message content, and current epoch to $\mathcal{S}$. For a malicious c_s, $\mathcal{F}$ sends the IDs and epoch in which the malicious message was read to $\mathcal{S}$. We allow malicious clients to change their previously written messages as long as the message is not already marked as read.

$\mathcal{S}$ could learn if an honest c_s is talking to other clients simply based on when it fetches the adversary's message. To avoid this, clients can use a pushPolicy that fetches messages in each conversation independently of other conversations [74].

Availability. Either server can deny service in Myco. We prove that malicious server actions can be modeled as denial of service attacks and do not compromise the security of honest clients. S_1 can deny service by tampering with the data or storing it in a location other than where the client expects to find it. Similarly, S_2 can deny service by failing to return the intended data to clients or by tampering with it in any manner.

We model DoS attacks from the servers in two ways: (1) simply rejecting read and write requests, and (2) the Avail API. $\mathcal{F}$ allows the servers to block communication between two clients without knowing if they are interacting. S_1 can reject any write request and S_2 can reject any read request, both without any knowledge of the message or the other participant in the conversation. However, S_1 has an additional, more nuanced way that it can deny service. It can deny a write when the sender requests it, but also process it later and reintroduce the write into the system sometime before it is read. To model this ability to toggle a message's availability, we allow $\mathcal{S}$ to access our more expressive Avail API, which allows $\mathcal{S}$ to select if the message sent by c_s in epoch t is currently retrievable or not. If pushPolicy selects a message which is not available, this message will still be marked as read because honest clients will not attempt to read it again.

Ideal Functionality $\mathcal{F}$

$\mathcal{F}$ maintains a message database DB for all clients, where $\mathsf{DB}[c_s, c_r, t]$ stores the message at epoch t between sender c_s and receiver c_r, and $\mathsf{Avail}[c_s, t]$ stores whether the message sent by c_s in epoch t is currently available for reading. DB entries can also have the following special symbols: uninitialized (ϕ) and marked as read ($\top$). The message written in epoch t can only be read after the end of epoch t. Initially, $\mathcal{F}$ sets $t_{\mathsf{curr}} \leftarrow 0$. For $t \in \mathbb{Z}_{\geq 0}$ and $c_s, c_r \in \mathcal{C}$, $\mathcal{F}$ sets $\mathsf{DB}[c_s, c_r, t] = \phi$ and $\mathsf{Avail}[c_s, t] = \mathsf{true}$. $\mathcal{F}$ receives a contact list $\mathsf{contacts}[c]$ and a push policy $\mathsf{pushPolicy}[c]$ (determines which unread message is fetched first) from each client c. $\mathcal{F}$ sends all honest-malicious client pairs (c, c') to $\mathcal{S}$ s.t. $c \in \mathcal{C}_H$, $c' \in \mathcal{C}_M$, and $c' \in \mathsf{contacts}[c]$. Δ is the number of epochs a message is accessible before expiration.

On input $\langle \mathsf{Write}, c_r, m, t \rangle$ from sender client c_s:

1) If $\mathsf{DB}[c_s, c_r, t] = \top$, $c_r = \perp$, or $m \in \{\phi, \top\}$, $\mathcal{F}$ ignores this request.
2) Send $\langle \mathsf{Write}, c_s, t \rangle$ to $\mathcal{S}$ and wait for its approval. If $\mathcal{S}$ disapproves, ignore this request.
3) If $c_r \in \mathcal{C}_M$, send $\langle \mathsf{Write}, c_s, c_r, m, t \rangle$ to $\mathcal{S}$.
4) Update $\mathsf{DB}[c_s, c_r, t] \leftarrow m$.

On input $\langle \mathsf{Read} \rangle$ from receiver client c_r:

1) For all $c_s \in \mathsf{contacts}[c_r]$ and $t \in [t_{\mathsf{curr}} - \Delta, t_{\mathsf{curr}} - 1]$, if $m = \mathsf{DB}[c_s, c_r, t] \notin \{\phi, \top\}$, add (c_s, c_r, t) to a list $\mathcal{M}_{\mathsf{all}}$.
2) If $\mathcal{M}_{\mathsf{all}}$ is empty, return $\perp$.
3) Get $(c_s, c_r, t) \leftarrow \mathsf{pushPolicy}[c_r](\mathcal{M}_{\mathsf{all}})$, set $m \leftarrow \mathsf{DB}[c_s, c_r, t]$ and $\mathsf{DB}[c_s, c_r, t] \leftarrow \top$.
4) If $c_s \in \mathcal{C}_M$, send $\langle \mathsf{Read}, c_s, c_r, t_{\mathsf{curr}} \rangle$ to $\mathcal{S}$, else send $\langle \mathsf{Read}, c_r, t_{\mathsf{curr}} \rangle$.
5) Wait for approval from $\mathcal{S}$ and return $\perp$ if it disapproves.
6) Return m to c_r if $\mathsf{Avail}[c_s, t] = \mathsf{true}$, else $\perp$.

On input $\langle \mathsf{Avail}, c_s, t, b \rangle$ from $\mathcal{S}$:

1) Set $\mathsf{Avail}[c_s, t] \leftarrow b$, where $b \in \{\mathsf{true}, \mathsf{false}\}$.

On input $\langle \mathsf{NextEpoch} \rangle$ from $\mathcal{S}$:

1) Increment t_{curr} if minimum epoch time has passed.

Figure 5: Ideal functionality of Myco.

Recall from §2.2 that clients will not alter their actions based on server misbehavior, which ensures security in the face of selective failure attacks. Indeed, if a client's read fails, then they will simply continue as usual and move onto their next message without altering their actions.

4.2. Definition & Security Theorem

We now define the ideal and real-world experiments, which are parameterized by the security parameter λ. Communication patterns are depicted in Figure 11. In the beginning of both the real and ideal world experiments, the environment $\mathcal{Z}$ establishes the list of all malicious clients $\mathcal{C}_M \subseteq \mathcal{C}$. We use a static adversary model. For honest clients, the environment $\mathcal{Z}$ decides the push policy, the contact list, which clients are online or offline, the exact time of each request, and who each client messages in every epoch. This is subject to the following constraints: the contact list can be of length at most Q, clients may message at most one contact per epoch, and clients can only message when they are online. In addition, $\mathcal{Z}$ also sees the honest client outputs in each epoch, as soon as a client receives the output from $\mathcal{F}$. $\mathcal{Z}$ ensures that S_1 and S_2 follow their respective APIs if they are honest. $\mathcal{Z}$ gets the view of the adversary $\mathcal{A}$ after every operation. Note, however, that $\mathcal{Z}$ and $\mathcal{A}$ can communicate throughout the protocol, not only at operation boundaries.

The ideal experiment. In the ideal world, honest parties interact only with $\mathcal{F}$, and $\mathcal{S}$ interacts with $\mathcal{F}$ on behalf of the corrupted parties. $\mathcal{S}$ simulates the honest parties in the real protocol for $\mathcal{A}$. Each client executes the setup protocol by submitting their contacts list before epoch 0. In each epoch $t \in \mathbb{Z}_{\geq 0}$ and for each honest client $c \in \mathcal{C}_H$, $\mathcal{Z}$ specifies a set of at most one write access $\mathsf{W}_c = \{c_r, m\}$. If $\mathsf{W}_c \neq \perp$, c invokes $\mathcal{F}$ with input $\langle \mathsf{Write}, c_r, m, t \rangle$. Otherwise, c invokes $\mathcal{F}$ with input $\langle \mathsf{Write}, \perp, \perp, t \rangle$. For reads, c invokes $\mathcal{F}$ with input $\langle \mathsf{Read} \rangle$, returning the result m to the environment $\mathcal{Z}$. At the end of the experiment ($\mathcal{Z}$ decides when), $\mathcal{Z}$ outputs a bit, which we denote by $\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda)$.

The real experiment. In the real world, honest parties follow the real Myco protocol described in §3 and Algorithm 1. Each client executes the setup protocol $\mathsf{client.Setup}(\vec{k}_Q)$. In each epoch $t \in \mathbb{Z}_{\geq 0}$ and for each honest client $c \in \mathcal{C}_H$, $\mathcal{Z}$ specifies a set of at most one write access $\mathsf{W}_c = \{c_r, m\}$. If $\mathsf{W}_c \neq \perp$, c executes $\mathsf{client.Write}(c_r, m)$. Otherwise, c executes $\mathsf{client.FakeWrite}()$. For reads, c executes $\mathsf{client.Read}()$, returning the result m to the environment $\mathcal{Z}$. S_1 performs $\mathsf{S}_1.\mathsf{BatchInit}()$ at the beginning of each epoch, and $\mathsf{S}_1.\mathsf{BatchWrite}()$ at the end of each epoch. At the end of the experiment, $\mathcal{Z}$ outputs a bit $\textsc{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$.

Definition 1. *Let Π be a protocol in the $\mathcal{F}_{\mathsf{KE}}$-hybrid model. Then Π securely realizes $\mathcal{F}$ (Figure 5) in the $\mathcal{F}_{\mathsf{KE}}$-hybrid model if for all PPT adversaries $\mathcal{A}$ controlling a malicious subset of clients $\mathcal{C}_M \subseteq \mathcal{C}$ (where $\mathcal{C}$ is the set of all clients and $\mathcal{C}_M$ is chosen adversarially) and at most one of S_1 and S_2, there exists a PPT simulator $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$, $\lambda \in \mathbb{N}$, we have:*

$$\left| \begin{array}{l} \Pr[\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda) = 1] \\ \quad - \Pr[\textsc{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda) = 1] \end{array} \right| \leq \mathit{negl}(\lambda)$$

Theorem 1. *As per Definition 1, the Myco protocol defined in §3 and Algorithm 1 securely realizes the ideal functionality $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{KE}}$-hybrid model when instantiated with a secure pseudorandom function, a key-derivation function, a key-private authenticated encryption with authenticated data scheme, and a digital signature scheme.*

We prove Theorem 1 in §A.

5. Capacity Analysis

We now provide theoretical bounds on the bucket capacities Z_T (Tree-Myco) and Z_M (Matrix-Myco) that ensure negligible overflow probability. We then conduct an empirical simulation to offer a smaller heuristic Z_T and Z_M that do not overflow in practice.

5.1. Theoretical Bounds

5.1.1. Bucket capacity. The following theorems show that setting the bucket capacity to be logarithmic in Q and N for Matrix-Myco, and logarithmic in N and Δ for Tree-Myco, ensures negligible overflow probability across all buckets over Δ epochs. We prove both theorems in §B.

Theorem 2 (Matrix-Myco Capacity)**.** *Choosing the bucket capacity $Z_M = \Theta(\kappa + \log(QN))$ ensures that the total overflow probability in each epoch of Matrix-Myco (i.e., the probability that some node receives more than Z_M messages) is at most $2^{-\kappa}$ for security parameter κ.*

Theorem 3 (Tree-Myco Capacity)**.** *Choosing the bucket capacity $Z_T = \Theta(\kappa + \log(N\Delta^3))$ ensures that the total overflow probability across Δ epochs of Tree-Myco (i.e., the probability that some node receives more than Z_T messages) is at most $2^{-\kappa}$ for security parameter κ.*

Beyond Δ epochs. Note that since the capacity analysis holds for Δ epochs, it will hold for infinite epochs. This is because after Δ epochs, the system's overflow probability reaches a steady state. Since messages expire after Δ epochs, any expired message encountered again through future path-set selection will be deleted. Consequently, expired messages do not occupy space in the system beyond their lifetime. As such, the same argument as above can apply to the range $t \in [1, \Delta + 1)$, $t \in [2, \Delta + 2)$, $t \in [3, \Delta + 3)$, etc. This effect is exemplified in Figure 6. Since there will only be polynomially many epochs, the union bound on the probability that no overflow happens in any of these epochs is still negligible in κ.

5.1.2. Total server work per epoch. We now compute the total work per epoch.

Matrix-Myco. In each epoch, $Q \cdot N$ buckets of size Z_M are processed. Thus, the total server work in each epoch is:

$$O\Big(Q\,N\,(\kappa + \log(Q\,N))\Big)$$

Tree-Myco. When an eviction batch is triggered, N root-to-leaf paths are evicted. Each path has $D = \log(N\Delta)$

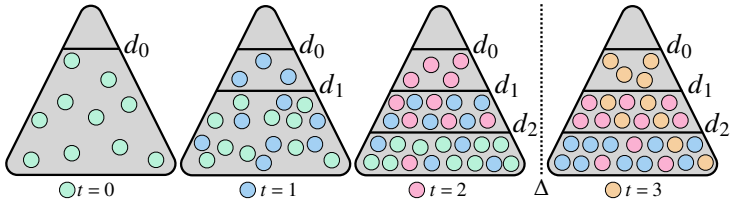


Figure 6: Tree-Myco percolation example with $\Delta = 3$, $N = 10$. In epoch 0, green messages settle below depth d_0. In epoch 1, blue messages settle below d_0, and green shift below d_1. In epoch 2, pink settle below d_0, blue shift below d_1, and green shift below d_2. In epoch 3, green expire (after Δ epochs); orange settle below d_0, pink shift below d_1, and blue shift below d_2. For clarity, we show a separation between d_i and d_{i+1}, though they may be equal.

buckets, each with capacity Z_T. Therefore, the total number of blocks processed in an epoch is:

$$O(N\,D\,Z_T) = O\Big(N\,\log(N\Delta)\,\left(\kappa + \log(N\Delta^3)\right)\Big)$$

Thus, the cost per client is polylogarithmic in N and Δ.

5.2. Empirical Simulation

In this section, we perform a simulation to derive heuristic upper bounds on the bucket capacity that outperform the conservative bounds we obtain theoretically in the previous section. We note that this is in line with many foundational and practically efficient ORAM schemes [59], [75], [76].

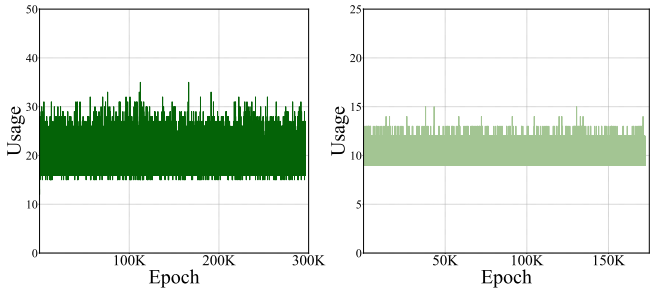


Figure 7: Usage of maximally filled buckets for $N = 335.5$K. Tree-Myco (left) and Matrix-Myco (right).

We perform the simulation for Matrix-Myco and Tree-Myco for the largest parameter setting we consider in the evaluation (§6): $N = 335.5$K and $\Delta = 25$, as the theoretical bucket capacity grows with these parameters. Figure 7 reports these results. We set the bucket capacity to $Z_M = 25$ and $Z_T = 50$, and it is evident from the figure that this is well above the maximum bucket usage of 15 and 35 in Matrix-Myco and Tree-Myco, respectively.

6. Evaluation

We now answer: *How does Myco's performance compare to PIR-based metadata-private messaging systems?*

6.1. Implementation

We implement Myco using $5,500$ lines of pure Rust. Our codebase can be found at:

https://github.com/myco-org/myco/

We use well-reputed libraries for the cryptographic building blocks: `aes-gcm` of RustCrypto[3] for authenticated encryption with AES-128-GCM, the `ring`[4] crate's HKDF-SHA256 for our key derivation function and HMAC-SHA256 for our pseudorandom functions, `rand_chacha`'s ChaCha20[5] for pseudorandom number generation, and `ed25519_dalek`[6] for digital signatures. We build a networking stack with the `tonic` RPC framework[7]. All network communication occurs over TLS. We remark that while our protocol and design are secure against timing attacks, our implementation is not guaranteed to be void of timing side channel attacks.

6.2. Experiment Setup

Baselines. As discussed in §1 and §2.2, Myco's strong cryptographic guarantees are most comparable to the PIR-based approaches, such as Pung [41], which is single-server, and Talek [44], which is multi-server. Thus, we plug the state-of-the-art non-preprocessing single-server and multi-server PIR protocols into the Pung and Talek settings, respectively. For two-server PIR, we use Google's implementation[8] of incremental distributed point functions [77]. For single-server PIR, we use Microsoft's implementation[9] of SealPIR [42] via the `sealpir-rust` library[10]. We emphasize that single-server PIR schemes do not require the non-collusion assumption, which remains a limitation of both Myco and the multi-server PIR approach. Inspired by Talek's naming convention [44], we refer to these configurations as Pung++ and Talek++.

Deployment. We deploy the baseline and Myco servers on 64 vCPU, 640 GB `n2-custom-64-655360-ext` machines in `us-west1` with extended memory to accommodate the larger database configurations and high-memory throughput experiments. Clients use 16 vCPU, 64 GB memory `n2-standard-16` machines in `us-east4` to simulate realistic WAN latency between clients and servers. We

3. https://github.com/RustCrypto
4. https://github.com/briansmith/ring
5. https://github.com/rust-random/rand/tree/master/rand_chacha
6. https://github.com/dalek-cryptography/curve25519-dalek
7. https://github.com/hyperium/tonic
8. https://github.com/google/distributed_point_functions
9. https://github.com/microsoft/SealPIR
10. https://github.com/sga001/sealpir-rust

Clients	System	Throughput	Local Latency		End-to-End Latency (s)			Bandwidth	
		Clients/Min.	*Client (ms)*	*Server (s)*	*Write*	*Read*	*Total*	*Client (KB)*	S_1-S_2 *(GB)*
10.5K	*Myco*	255.7K	1	0.83	1.58	0.38	1.96	297.3	1.13
$= 2^{18}/25$	*Talek++*	24.1K	2	0.09	0.11	0.15	0.26	10.6	-
	Pung++	2.4K	27	0.74	0.11	1.09	1.20	394.2	-
21.0K	*Myco*	243.8K	1	1.70	3.06	0.45	3.51	311.5	2.27
$= 2^{19}/25$	*Talek++*	12.0K	2	0.18	0.11	0.24	0.35	10.7	-
	Pung++	1.3K	27	1.34	0.11	1.75	1.86	394.2	-
41.9K	*Myco*	236.1K	1	3.65	6.25	0.52	6.77	325.8	4.53
$= 2^{20}/25$	*Talek++*	5.5K	2	0.37	0.11	0.43	0.54	10.8	-
	Pung++	0.7K	27	2.52	0.11	2.92	3.03	459.9	-
83.9K	*Myco*	223.9K	1	7.55	12.63	0.64	13.27	340.1	9.06
$= 2^{21}/25$	*Talek++*	2.7K	2	0.66	0.11	0.72	0.83	11.0	-
	Pung++	0.4K	27	4.74	0.11	5.13	5.24	459.9	-
167.8K	*Myco*	215.6K	1	15.25	25.56	0.88	26.44	354.3	18.12
$= 2^{22}/25$	*Talek++*	1.4K	2	1.37	0.11	1.44	1.55	11.1	-
	Pung++	0.2K	27	10.09	0.11	10.98	11.09	459.9	-
335.5K	*Myco*	203.8K	1	33.18	54.30	1.12	55.42	368.6	36.25
$= 2^{23}/25$	*Talek++*	0.7K	2	2.78	0.11	3.05	3.16	11.3	-
	Pung++	0.1K	27	20.45	0.11	20.98	21.09	459.9	-

TABLE 1: Throughput, single-client latency, & bandwidth of Myco, Talek++, and Pung++.

measured a roundtrip time of 0.41 ms between S_1 and S_2 and 56 ms between the client and the servers. The network bandwidth between S_1 and S_2 is 32 Gbps. The client has 410 Mbps bandwidth with the servers.

Parameters. We set $N = 2^D/\Delta$, where D is the tree depth and N represents system clients, following Talek's [44] approach to message time-to-live. We use an expiration period of $\Delta = 25$, ensuring messages persist for 25 epochs before deletion by S_1. We set the number of conversations each client is participating in to $Q = 64$, similar to prior work [36], [41], [42], [44].

Given the empirical analysis of §5.2, we set $Z_T = 50$ for Tree-Myco and $Z_M = 25$ for Matrix-Myco. Message ciphertexts are 256 bytes, as evaluated in prior work [41], [42]. Our experiments vary the number of clients from $2^{18}/25 \approx 10.5$K clients to $2^{23}/25 \approx 335.5$K clients. We average results over 10 consecutive protocol epochs.

6.3. Throughput

Myco achieves higher throughput than the baselines due to polylogarithmic reads and writes, as shown in Figure 8 and Table 1. For each N that the system is configured to handle, all N clients write, followed by all clients reading a message before proceeding to the next epoch. We report the number of clients processed per minute.

For the throughput experiment, we simulate all the N clients on an additional server in the same region as S_1 and S_2 to capture the effect of bandwidth on throughput. For the PIR baselines, we take a conservative approach and do not consider the communication costs.

Varying N from 10.5K to 335.5K, Myco achieves 11-302× the throughput of Talek++ and 106-2,219× the

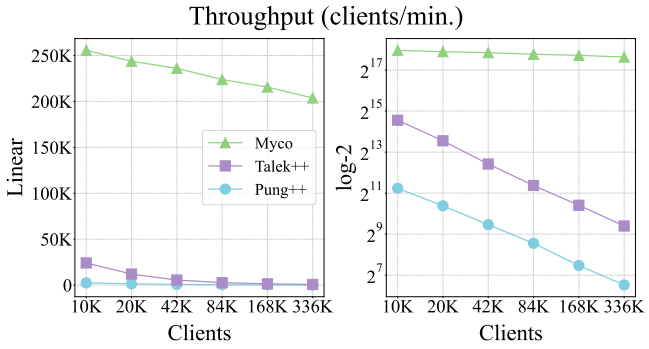


Figure 8: Server throughput in clients processed per minute in linear (left) and $\log_2$-scale (right). Each client sends and receives one message per epoch.

throughput of Pung++.[11] In all three systems, throughput declines when the system must accommodate more clients (Table 1). However, as demonstrated in Figure 8, Myco experiences significantly less throughput degradation as the system scales. This is because of Myco's asymptotic improvement, which means that the servers perform $O(N \log^2 N)$ work in N per epoch. This is in contrast with the PIR-based systems, which degrade quadratically in N.

6.4. Single-Client Latency

We now measure the end-to-end latency for sending a single message through our system and compare it to our baselines, as shown in Table 1 and Figure 9. When a single client is writing and reading a message, Myco's single-client end-to-end latency ranges from a few seconds to under a

11. Ratios use full-precision data; table values are rounded.

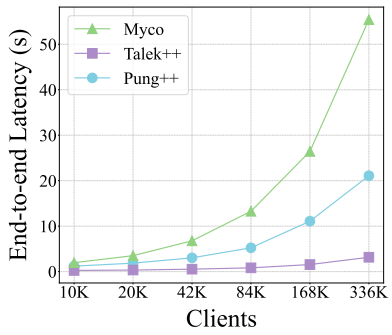


Figure 9: End-to-end latency of a single client sending a message across total number of supported clients.

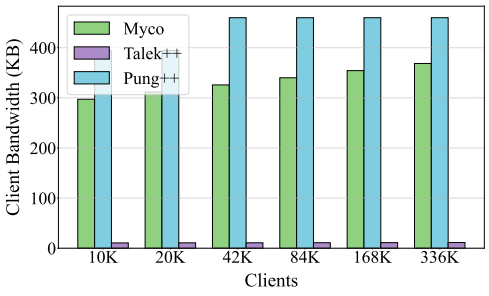


Figure 10: Client-server bandwidth of sending one message.

minute, while Talek++'s remains under a few seconds and Pung++'s reaches up to 21 seconds.

A limitation of Myco is that even if only one user is online, S_1 must perform a batch write on N paths to ensure our capacity analysis in §5 holds. Thus, the end-to-end latency includes: (1) S_1 invoking BatchInit, (2) a single client executing a S_1.Write with its message, (3) S_1 calling BatchWrite to write the processed path-set, including the message, back to S_2, and (4) the client reading from S_2. The true value of Myco emerges in realistic scenarios with simultaneous participation of many clients, as shown in our throughput experiments (§6.3). In these situations, the average latency of Talek++ and Pung++ will be much higher than Myco. Assuming all clients write in an epoch, the worst-case latency of Myco for $N = 335.5$K is under 2 minutes, while the worst-case latency for Talek++ and Pung++ is around 8 and 60 hours, respectively. We note that the worst-case latency of the PIR works could be reduced linearly by introducing more servers, but it would require hundreds of servers to achieve a latency similar to Myco.

6.5. Bandwidth

Table 1 and Figure 10 report the communication comparison of Myco versus the PIR-based works. A tradeoff of Myco is that it has larger communication costs than Talek++ in terms of both server-server communication and client-server communication. While Talek++ has no server-server communication, Myco sends 1.13-36.25 GB of data per epoch between the servers. The client-server communication of Myco grows up to 369 KB, which is also much higher than Talek++'s 11 KB. Like Talek++, Myco's client-server communication grows sub-linearly with N.

7. Related Work

Myco is the first asynchronous metadata-private messaging system offering strong cryptographic guarantees and high throughput. We now discuss how Myco is situated in the related work. Prior SoKs [14], [15], [16] provide more details on secure and anonymous communication systems.

PIR-based messaging systems [41], [42], [43], [44], [45], [46], [78] closely align with Myco's asynchronous mailbox design [79], but incur $O(N^2)$ total server work for handling N queries by N users. Receivers use private information retrieval (PIR) to read messages obliviously. Oblivious message retrieval [80], [81] enables clients to retrieve messages obliviously but incurs high computational costs due to its reliance on fully-homomorphic encryption, and requires a linear scan over the message database, as in PIR-based works. Distributed PIR [82] shifts PIR server computation to the clients, but this significantly increases client overhead.

DC networks [62], [63], [64], [65], [83] offer cryptographic guarantees like PIR-based systems. However, they require an all-to-all broadcast of messages among all users, resulting in prohibitively high communication costs. Consequently, these systems are typically limited to very small client groups.

Mix networks (mixnets) [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32] rely on servers to shuffle messages before delivery. Mixnets are efficient with $O(N)$ server overhead, but are typically only synchronous, limiting users to one conversation for which they must be perpetually online. Originally, these systems either depended on trusted servers [30], [31] or faced potential manipulation by mixnet servers [84], [85], [86], [87], [88], [89], [90], [91], [92]. Peer-to-peer renditions of mixnets and routing protocols [93], [94], [95], [96], [97], [98], [99], [100], [101] similarly can only withstand a limited number of malicious clients and are vulnerable to strong Sybil adversaries [102], [103], [104], [105], [106], [107], [108].

Differential privacy [109] has more recently been applied to enhance mixnets by introducing noise into the network [33], [34], [35], [36], [37]. While these systems retain the high performance of mixnets, their security depends on differential privacy. XRD [24] and Yodel [32] are mixnets that offer cryptographic guarantees, but these works limit clients to a single conversation for which they must constantly be online to avoid missing messages. Groove [36] is the first work to add asynchronous flexibility to mixnets, but also relies on differential privacy.

MPC-based systems [60], [61], [110], [111], [112] leverage multi-party computation at the servers to enhance mixnets. These synchronous systems also limit users to one conversation for which they must constantly be online.

Reverse-PIR-based systems [47], [48], [49], [50] use reverse PIR to target a different setting of whistleblowing, where the writer is hidden, but reads go to trusted entities. Our goal is instead the messaging setting.

TEE-based metadata-hiding approaches [38], [39], [113], [114], [115] have been proposed, but enclaves are susceptible to side-channel attacks that undermine remote attestation [116], [117], [118], [119]. With root access to servers, application providers can exploit these vulnerabilities to access secrets. Consequently, while enclaves serve as a supplementary defense, real-world applications often prioritize cryptography as the primary security measure [120].

Onion routing [121], [122], [123], [124] is widely adopted for anonymous communication due to its scalability. However, it remains highly vulnerable to traffic analysis attacks that undermine its anonymity guarantees [89], [125], [126], [127], [128], [129], [130], [131], [132].

Oblivious data structures. ORAM [56], [57], particularly tree-based ORAMs [58], [59] inspire many of our techniques. Multi-client ORAMs [133], [134], [135] mostly assume semi-honest clients. PANDA [136] allows for a limited set of malicious clients, while solutions that tolerate arbitrarily many malicious clients remain theoretical [137].

8. Conclusion

This work introduces Myco, a metadata-private messaging system that achieves polylogarithmic read and write efficiency. Myco uses an asymmetric distributed-trust model and introduces a novel tree-based oblivious data structure that allows clients to write to S_1, who obliviously transfers messages to S_2, from which clients read. In doing so, Myco overcomes the linear-access barrier of prior works with strong cryptographic guarantees, and takes a significant step towards cryptographic metadata-private messaging at scale.

Acknowledgments. We thank the anonymous reviewers, our shepherd, and the students in the Sky security group for their feedback. We thank Ratan Kaliani, Ethan J. Jackson, and Alex Krentsel for discussions about network stack design, Syomantak Chaudhuri, Julien Piet, and Kshitij Kulkarni for discussions about the capacity analysis, Rolfe Schmidt for discussions about applicability to Signal, Yiping Ma and Sebastian Angel for writing feedback, and Vivian Fang, Sajin Sasy, and Giulio Malavolta for early discussions. This work is supported by gifts from Accenture, AMD, Anyscale, Cisco, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Mibura, Microsoft, NVIDIA, Samsung SDS, SAP, and VMware.

References

[1] "WhatsApp," https://www.whatsapp.com/.

[2] "Signal," https://signal.org/.

[3] "End-to-end encryption on Messenger explained," https://about.fb.com/news/2024/03/end-toend-encryption-on-messenger-explained/.

[4] B. Schneier, *Data and goliath: The hidden battles to collect your data and control your world*, 2015.

[5] J. R. Mayer, P. Mutchler, and J. C. Mitchell, "Evaluating the privacy properties of telephone metadata," *Proc. Natl. Acad. Sci. USA*, vol. 113, no. 20, pp. 5536–5541, 2016.

[6] A. Rusbridger, "The Snowden Leaks and the Public," https://www.nybooks.com/articles/2013/11/21/snowden-leaks-and-public/, 2013.

[7] D. Cole, "'We Kill People Based on Metadata'," https://www.nybooks.com/online/2014/05/10/we-kill-people-based-metadata/, 2013.

[8] D. B. Johnson, "FTC details how streaming services, social media have become 'mass surveillance' machines," https://cyberscoop.com/ftc-report-streaming-social-media-surveillance-privacy/, 2024.

[9] M. Liedtke, "Google will purge billions of files containing personal data in settlement of Chrome privacy case," https://apnews.com/article/google-chrome-privacy-lawsuit-settlement-203cc5063f1a1d4013de1900d9376814, 2024.

[10] D. Cameron and D. Mehrotra, "Secretive White House Surveillance Program Gives Cops Access to Trillions of US Phone Records," https://www.wired.com/story/hemisphere-das-white-house-surveillance-trillions-us-call-records/, 2023.

[11] B. Fung, "The NSA buys Americans' internet data, newly released documents show," https://www.cnn.com/2024/01/26/tech/the-nsa-buys-americans-internet-data-newly-released-documents-show/index.html, 2024.

[12] J. Keegan, "Each Facebook User is Monitored by Thousands of Companies," https://themarkup.org/privacy/2024/01/17/each-facebook-user-is-monitored-by-thousands-of-companies-study-indicates, 2024.

[13] N. Lomas, "Meta ordered to suspend Facebook EU data flows as it's hit with record €1.2BN privacy fine under GDPR," https://techcrunch.com/2023/05/22/facebook-eu-us-data-flows-decision/, 2023.

[14] S. Sasy and I. Goldberg, "SoK: Metadata-Protecting Communication Systems," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 1, pp. 509–524, 2024.

[15] G. Danezis, C. Díaz, and P. F. Syverson, "Anonymous Communication," in *Handbook of Financial Cryptography and Security*. Chapman and Hall/CRC, 2010, pp. 341–389.

[16] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2015, pp. 232–249.

[17] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[18] O. Berthold, H. Federrath, and S. Köpsell, "Web MIXes: A system for anonymous and unobservable Internet access," in *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer, 2001, pp. 115–129.

[19] O. Berthold and H. Langos, "Dummy Traffic against Long Term Intersection Attacks," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 2482. Springer, 2002, pp. 110–128.

[20] N. Gelernter, A. Herzberg, and H. Leibowitz, "Two Cents for Strong Anonymity: The Anonymous Post-office Protocol," in *CANS*, ser. Lecture Notes in Computer Science, vol. 11261. Springer, 2017, pp. 390–412.

[21] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol," in *S&P*. IEEE Computer Society, 2003, pp. 2–15.

[22] C. Gülcü and G. Tsudik, "Mixing Email with Babel," in *NDSS*. IEEE Computer Society, 1996, pp. 2–16.

[23] D. Kesdogan, J. Egner, and R. Büschkes, "Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System," in *Information Hiding*, ser. Lecture Notes in Computer Science, vol. 1525. Springer, 1998, pp. 83–98.

[24] A. Kwon, D. Lu, and S. Devadas, "XRD: scalable messaging system with cryptographic privacy," in *NSDI*. USENIX Association, 2020, pp. 759–776.

[25] A. Kwon, D. Lazar, S. Devadas, and B. Ford, "Riffle: An Efficient Communication System With Strong Anonymity," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 2, pp. 115–134, 2016.

[26] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, and A. T. Sherman, "cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations," in *ACNS*, ser. Lecture Notes in Computer Science, vol. 10355. Springer, 2017, pp. 557–578.

[27] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally Scaling Strong Anonymity," in *SOSP*. ACM, 2017, pp. 406–422.

[28] S. Langowski, S. Servan-Schreiber, and S. Devadas, "Trellis: Robust and Scalable Metadata-private Anonymous Broadcast," in *NDSS*. The Internet Society, 2023.

[29] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix Anonymity System," in *USENIX Security Symposium*. USENIX Association, 2017, pp. 1199–1216.

[30] S. L. Blond, D. R. Choffnes, W. Caldwell, P. Druschel, and N. Merritt, "Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for voip systems," in *SIGCOMM*. ACM, 2015, pp. 639–652.

[31] S. L. Blond, D. R. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis, "Towards efficient traffic-analysis resistant anonymity networks," in *SIGCOMM*. ACM, 2013, pp. 303–314.

[32] D. Lazar, Y. Gilad, and N. Zeldovich, "Yodel: strong metadata security for voice calls," in *SOSP*. ACM, 2019, pp. 211–224.

[33] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: scalable private messaging resistant to traffic analysis," in *SOSP*. ACM, 2015, pp. 137–152.

[34] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A Distributed Metadata-Private Messaging System," in *SOSP*. ACM, 2017, pp. 423–440.

[35] D. Lazar, Y. Gilad, and N. Zeldovich, "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis," in *OSDI*. USENIX Association, 2018, pp. 711–725.

[36] L. Barman, M. Kol, D. Lazar, Y. Gilad, and N. Zeldovich, "Groove: Flexible Metadata-Private Messaging," in *OSDI*. USENIX Association, 2022, pp. 735–750.

[37] D. Lazar and N. Zeldovich, "Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata," in *OSDI*. USENIX Association, 2016, pp. 571–586.

[38] K. Fredrickson, I. Demertzis, J. Hughes, and D. Long, "Sparta: Practical Anonymity with Long-Term Resistance to Traffic Analysis," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2025, pp. 65–65.

[39] P. Jiang, Q. Wang, Y. Wu, and C. Wang, "Poster: Metadata-private messaging without coordination," in *CCS*. ACM, 2023, pp. 3615–3617.

[40] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private Information Retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[41] S. Angel and S. T. V. Setty, "Unobservable Communication over Fully Untrusted Infrastructure," in *OSDI*. USENIX Association, 2016, pp. 551–569.

[42] S. Angel, H. Chen, K. Laine, and S. T. V. Setty, "PIR with compressed queries and amortized query processing," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 962–979.

[43] L. Sassaman, B. Cohen, and N. Mathewson, "The pynchon gate: a secure method of pseudonymous mail retrieval," in *WPES*. ACM, 2005, pp. 1–9.

[44] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. E. Anderson, A. Krishnamurthy, and B. Parno, "Talek: Private Group Messaging with Hidden Access Patterns," in *ACSAC*. ACM, 2020, pp. 84–99.

[45] N. Borisov, G. Danezis, and I. Goldberg, "DP5: A private presence service," *Proc. Priv. Enhancing Technol.*, vol. 2015, no. 2, pp. 4–24, 2015.

[46] L. Kissner, A. Oprea, M. K. Reiter, D. X. Song, and K. Yang, "Private Keyword-Based Push and Pull with Applications to Anonymous communication," in *ACNS*, ser. Lecture Notes in Computer Science, vol. 3089. Springer, 2004, pp. 16–30.

[47] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An Anonymous Messaging System Handling Millions of Users," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2015, pp. 321–338.

[48] Z. Newman, S. Servan-Schreiber, and S. Devadas, "Spectrum: High-bandwidth Anonymous Broadcast," in *NSDI*. USENIX Association, 2022, pp. 229–248.

[49] A. Vadapalli, K. Storrier, and R. Henry, "Sabre: Sender-Anonymous Messaging with Fast Audits," in *SP*. IEEE, 2022, pp. 1953–1970.

[50] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic privacy," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 1775–1792.

[51] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One Server for the Price of Two: Simple and Fast Single-Server Private information retrieval," in *USENIX Security Symposium*. USENIX Association, 2023, pp. 3889–3905.

[52] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the Servers Computation in Private Information Retrieval: PIR with preprocessing," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 1880. Springer, 2000, pp. 55–73.

[53] H. Corrigan-Gibbs and D. Kogan, "Private Information Retrieval with Sublinear Online Time," in *EUROCRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 12105. Springer, 2020, pp. 44–75.

[54] M. H. Mughees, H. Chen, and L. Ren, "OnionPIR: Response Efficient Single-Server PIR," in *CCS*. ACM, 2021, pp. 2292–2306.

[55] M. H. Mughees and L. Ren, "Vectorized Batch Private Information Retrieval," in *SP*. IEEE, 2023, pp. 437–452.

[56] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[57] R. Ostrovsky, "Efficient Computation on Oblivious RAMs," in *STOC*. ACM, 1990, pp. 514–523.

[58] E. Shi, T. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with $O((\log N)^3)$ Worst-Case Cost," *IACR Cryptol. ePrint Arch.*, p. 407, 2011.

[59] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *CCS*. ACM, 2013, pp. 299–310.

[60] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "MCMix: Anonymous Messaging via Secure Multiparty Computation," in *USENIX Security Symposium*. USENIX Association, 2017, pp. 1217–1234.

[61] S. Eskandarian and D. Boneh, "Clarion: Anonymous Communication from Multiparty Shuffling Protocols," in *NDSS*. The Internet Society, 2022.

[62] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *J. Cryptol.*, vol. 1, no. 1, pp. 65–75, 1988.

[63] H. Corrigan-Gibbs and B. Ford, "Dissent: accountable anonymous group messaging," in *CCS*. ACM, 2010, pp. 340–350.

[64] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in Numbers: Making Strong Anonymity Scale," in *OSDI*. USENIX Association, 2012, pp. 179–182.

[65] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, "Proactively Accountable Anonymous Messaging in Verdict," in *USENIX Security Symposium*. USENIX Association, 2013, pp. 147–162.

[66] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: scaling private contact discovery," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 4, pp. 159–178, 2018.

[67] F. F. Yao and Y. L. Yin, "Design and analysis of password-based key derivation functions," *IEEE Trans. Inf. Theory*, vol. 51, no. 9, pp. 3292–3297, 2005.

[68] M. J. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," 2007.

[69] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 531–545.

[70] F. Banfi and U. Maurer, "Anonymous symmetric-key communication," in *SCN*, ser. Lecture Notes in Computer Science, vol. 12238. Springer, 2020, pp. 471–491.

[71] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2001, vol. 2.

[72] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems (reprint)," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, 1983.

[73] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography*, ser. ACM Books. ACM, 2022, vol. 42, pp. 365–390.

[74] S. Angel, S. Kannan, and Z. B. Ratliff, "Private resource allocators and their applications," in *SP*. IEEE, 2020, pp. 372–391.

[75] L. Ren, C. W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas, "Constants count: Practical improvements to oblivious RAM," in *USENIX Security Symposium*. USENIX Association, 2015, pp. 415–430.

[76] X. Wang, T. H. Chan, and E. Shi, "Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound," in *CCS*. ACM, 2015, pp. 850–861.

[77] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight Techniques for Private Heavy Hitters," in *SP*. IEEE, 2021, pp. 762–776.

[78] I. Ahmad, Y. Yang, D. Agrawal, A. E. Abbadi, and T. Gupta, "Addra: Metadata-private voice communication over fully untrusted infrastructure," in *OSDI*. USENIX Association, 2021, pp. 313–329.

[79] D. A. Cooper and K. P. Birman, "Preserving privacy in a network of mobile computers," in *S&P*. IEEE Computer Society, 1995, pp. 26–38.

[80] Z. Liu and E. Tromer, "Oblivious Message Retrieval," in *CRYPTO (1)*, ser. Lecture Notes in Computer Science, vol. 13507. Springer, 2022, pp. 753–783.

[81] Z. Liu, E. Tromer, and Y. Wang, "PerfOMR: Oblivious message retrieval with reduced communication and computation," in *USENIX Security Symposium*. USENIX Association, 2024.

[82] E. Tovey, J. Weiss, and Y. Gilad, "Distributed pir: Scaling private messaging via the users' machines," in *CCS*. ACM, 2024, pp. 1967–1981.

[83] E. G. Sirer, S. Goel, M. Robson, and D. Engin, "Eluding carnivores: file sharing with strong anonymity," in *ACM SIGOPS European Workshop*. ACM, 2004, p. 19.

[84] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright, "Timing Attacks in Low-Latency Mix Systems (Extended Abstract)," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 3110. Springer, 2004, pp. 251–265.

[85] N. Mathewson and R. Dingledine, "Practical Traffic Analysis: Extending and Resisting Statistical Disclosure," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 3424. Springer, 2004, pp. 17–34.

[86] L. Nguyen and R. Safavi-Naini, "Breaking and Mending Resilient Mix-Nets," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 2760. Springer, 2003, pp. 66–80.

[87] B. Pfitzmann, "Breaking Efficient Anonymous Channel," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 950. Springer, 1994, pp. 332–340.

[88] B. Pfitzmann and A. Pfitzmann, "How to Break the Direct RSA-Implementation of Mixes," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 434. Springer, 1989, pp. 373–381.

[89] J. Raymond, "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems," in *Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, vol. 2009. Springer, 2000, pp. 10–29.

[90] V. Shmatikov and M. Wang, "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses," in *ESORICS*, ser. Lecture Notes in Computer Science, vol. 4189. Springer, 2006, pp. 18–33.

[91] D. Wikström, "Five Practical Attacks for "Optimistic Mixing for Exit-Polls"," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, vol. 3006. Springer, 2003, pp. 160–175.

[92] D. Kesdogan, D. Agrawal, D. V. Pham, and D. Rautenbach, "Fundamental Limits on the Anonymity Provided by the MIX Technique," in *S&P*. IEEE Computer Society, 2006, pp. 86–99.

[93] M. Rennhard and B. Plattner, "Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection," in *WPES*. ACM, 2002, pp. 91–102.

[94] B. Zantout, R. Haraty *et al.*, "I2P data communication system," in *Proceedings of ICN*. Citeseer, 2011, pp. 401–409.

[95] A. Beimel and S. Dolev, "Buses for Anonymous Message Delivery," *J. Cryptol.*, vol. 16, no. 1, pp. 25–39, 2003.

[96] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval system," in *Workshop on Design Issues in Anonymity and Unobservability*, ser. Lecture Notes in Computer Science, vol. 2009. Springer, 2000, pp. 46–66.

[97] G. Danezis, C. Díaz, C. Troncoso, and B. Laurie, "Drac: An Architecture for Anonymous Low-Volume Communications," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 6205. Springer, 2010, pp. 202–219.

[98] M. J. Freedman and R. T. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *CCS*. ACM, 2002, pp. 193–206.

[99] A. Nambiar and M. K. Wright, "Salsa: a structured approach to large-scale anonymity," in *CCS*. ACM, 2006, pp. 17–26.

[100] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.

[101] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P^5: A protocol for scalable anonymous communication," *J. Comput. Secur.*, vol. 13, no. 6, pp. 839–876, 2005.

[102] C. Egger, J. Schlumberger, C. Kruegel, and G. Vigna, "Practical Attacks against the I2P Network," in *RAID*, ser. Lecture Notes in Computer Science, vol. 8145. Springer, 2013, pp. 432–451.

[103] P. Mittal and N. Borisov, "Information Leaks in Structured Peer-to-Peer Anonymous Communication systems," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 5:1–5:28, 2012.

[104] A. Singh, T. Ngan, P. Druschel, and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," in *INFOCOM*. IEEE, 2006.

[105] M. Schuchard, A. W. Dean, V. Heorhiadi, N. Hopper, and Y. Kim, "Balancing the shadows," in *WPES*. ACM, 2010, pp. 1–10.

[106] P. Tabriz and N. Borisov, "Breaking the Collusion Detection Mechanism of MorphMix," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 4258. Springer, 2006, pp. 368–383.

[107] Q. Wang, P. Mittal, and N. Borisov, "In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems," in *CCS*. ACM, 2010, pp. 308–318.

[108] J. R. Douceur, "The Sybil Attack," in *IPTPS*, ser. Lecture Notes in Computer Science, vol. 2429. Springer, 2002, pp. 251–260.

[109] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," in *TCC*, ser. Lecture Notes in Computer Science, vol. 3876. Springer, 2006, pp. 265–284.

[110] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its application to anonymous communication," in *CCS*. ACM, 2019, pp. 887–903.

[111] I. Abraham, B. Pinkas, and A. Yanai, "Blinder: MPC Based Scalable and Robust Anonymous Committed Broadcast," *IACR Cryptol. ePrint Arch.*, p. 248, 2020.

[112] D. Lu and A. Kate, "RPM: Robust Anonymity at Scale," *Proc. Priv. Enhancing Technol.*, vol. 2023, no. 2, pp. 347–360, 2023.

[113] D. V. Le, L. T. Hurtado, A. Ahmad, M. Minaei, B. Lee, and A. Kate, "A tale of two trees: One writes, and other reads," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 2, pp. 519–536, 2020.

[114] K. Wüst, S. Matetic, M. Schneider, I. Miers, K. Kostiainen, and S. Capkun, "Zlite: Lightweight clients for shielded zcash transactions using trusted execution," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 11598. Springer, 2019, pp. 179–198.

[115] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "BITE: bitcoin lightweight client privacy using trusted execution," in *USENIX Security Symposium*. USENIX Association, 2019, pp. 783–800.

[116] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," https://sgaxeattack.com/, 2020.

[117] K. Murdock, D. F. Oswald, F. D. Garcia, J. V. Bulck, F. Piessens, and D. Gruss, "Plundervolt: How a little bit of undervolting can create a lot of trouble," *IEEE Secur. Priv.*, vol. 18, no. 5, pp. 28–37, 2020.

[118] P. Borrello, A. Kogler, M. Schwarzl, M. Lipp, D. Gruss, and M. Schwarz, "Æpic leak: Architecturally leaking uninitialized data from the microarchitecture," in *USENIX Security Symposium*. USENIX Association, 2022, pp. 3917–3934.

[119] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, "CIPHERLEAKS: breaking constant-time cryptography on AMD SEV via the ciphertext side channel," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 717–732.

[120] Y. Lindell, D. Cook, T. Geoghegan, S. Gran, R. Schmidt, E. Kret, D. Kaviani, and R. A. Popa, "The Deployment Dilemma: Merits and Challenges of Deploying MPC," https://mpc.cs.berkeley.edu/blog/deployment-dilemma, Sep 2023. [Online]. Available: https://mpc.cs.berkeley.edu/blog/deployment-dilemma

[121] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*. USENIX, 2004, pp. 303–320.

[122] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "PIR-Tor: Scalable Anonymous Communication Using Private Information retrieval," in *USENIX Security Symposium*. USENIX Association, 2011.

[123] P. Mittal, M. K. Wright, and N. Borisov, "Pisces: Anonymous Communication Using Social Networks," in *NDSS*. The Internet Society, 2013, pp. 1–18.

[124] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous Connections and Onion Routing," in *S&P*. IEEE Computer Society, 1997, pp. 44–54.

[125] N. Hopper, E. Y. Vasserman, and E. Chan-TIN, "How much anonymity does network latency leak?" *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, Mar. 2010.

[126] S. J. Murdoch and G. Danezis, "Low-Cost Traffic Analysis of Tor," in *S&P*. IEEE Computer Society, 2005, pp. 183–195.

[127] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: website fingerprinting attacks and defenses," in *CCS*. ACM, 2012, pp. 605–616.

[128] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden services," in *USENIX Security Symposium*. USENIX Association, 2015, pp. 287–302.

[129] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *WPES*. ACM, 2011, pp. 103–114.

[130] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," in *WPES*. ACM, 2013, pp. 201–212.

[131] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website Fingerprinting at Internet Scale," in *NDSS*. The Internet Society, 2016.

[132] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 143–157.

[133] E. Blass, T. Mayberry, and G. Noubir, "Multi-client Oblivious RAM Secure Against Malicious Servers," in *ACNS*, ser. Lecture Notes in Computer Science, vol. 10355. Springer, 2017, pp. 686–707.

[134] C. Sahin, V. Zakhary, A. E. Abbadi, H. Lin, and S. Tessaro, "TaoStore: Overcoming Asynchronicity in Oblivious Data Storage," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2016, pp. 198–217.

[135] A. Chakraborti and R. Sion, "ConcurORAM: High-Throughput Stateless Parallel Multi-Client ORAM," in *NDSS*. The Internet Society, 2019.

[136] A. Hamlin, R. Ostrovsky, M. Weiss, and D. Wichs, "Private Anonymous Data Access," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 11477. Springer, 2019, pp. 244–273.

[137] S. S. M. Chow, K. Fech, R. W. F. Lai, and G. Malavolta, "Multi-client Oblivious RAM with Poly-logarithmic Communication," in *ASIACRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 12492. Springer, 2020, pp. 160–190.

[138] D. A. McGrew and J. Viega, "The security and performance of the galois/counter mode (GCM) of operation," in *INDOCRYPT*, ser. Lecture Notes in Computer Science, vol. 3348. Springer, 2004, pp. 343–355.

Algorithm 1: Myco

client.Setup($\vec{k}_Q$)

for $(c, k) \in \vec{k}_Q$ **do**
  $k_{\mathsf{enc}} \leftarrow \mathsf{KDF}(k, \texttt{"enc"})$
  $k_{\mathsf{renc}} \leftarrow \mathsf{KDF}(k, \texttt{"re\_enc"})$
  $k_{\mathsf{rk}} \leftarrow \mathsf{KDF}(k, \texttt{"routing\_key"})$
  $k^{\mathsf{ntf}} \leftarrow \mathsf{KDF}(k, \texttt{"notif"})$
  $k^{\mathsf{ntf}}_{\mathsf{rk}} \leftarrow \mathsf{KDF}(k, \texttt{"routing\_notif"})$
  client.keys$[c] = \{k_{\mathsf{enc}}, k_{\mathsf{renc}}, k_{\mathsf{rk}}, k^{\mathsf{ntf}}, k^{\mathsf{ntf}}_{\mathsf{rk}}\}$

client.Write(c_r, m)

$\{k_{\mathsf{enc}}, k_{\mathsf{renc}}, k_{\mathsf{rk}}, k^{\mathsf{ntf}}, k^{\mathsf{ntf}}_{\mathsf{rk}}\} \leftarrow$ client.keys$[c_r]$
$\mathsf{ct} \leftarrow \mathsf{Enc}_{k_{\mathsf{enc}}}(m; \text{client}.t)$
$\mathsf{ct}^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{\lambda, k^{\mathsf{ntf}}}(\text{client}.t)$
$f \leftarrow \mathsf{PRF}_{k_{\mathsf{rk}}}(\text{client}.t)$
$f^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{k^{\mathsf{ntf}}_{\mathsf{rk}}}(\text{client}.t)$
$k_{\mathsf{renc},t} \leftarrow \mathsf{PRF}_{k_{\mathsf{renc}}}(\text{client}.t)$
S_1.Write(ct, $\mathsf{ct}^{\mathsf{ntf}}, f, f^{\mathsf{ntf}}, k_{\mathsf{renc},t}$)

client.Read()

$L, \mathcal{M}_{\mathsf{all}} \leftarrow \{\}, \{\}$
$\vec{k}_{\mathsf{srk}} \leftarrow \mathsf{S}_2$.GetPRFKeys()
$\delta \leftarrow$ epochsOffline()
for $t \in [\text{client}.t - \delta, \text{client}.t)$ **do**
  for $c_s \in$ client.contacts **do**
    $\{\_, \_, \_, \_, k^{\mathsf{ntf}}_{\mathsf{rk}}\} \leftarrow$ client.keys$[c_s]$
    $f^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{k^{\mathsf{ntf}}_{\mathsf{rk}}}(t)$
    $k_{\mathsf{srk},t} \leftarrow k_{\mathsf{srk}}[\Delta - (\text{client}.t - t)]$
    $\ell^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{k_{\mathsf{srk},t}}(f^{\mathsf{ntf}}, c_s)$
    $L[t]$.Add(ℓ^{ntf})
$p_{\mathsf{ntf}} \leftarrow \mathsf{S}_2$.ReadNotifs($L$)
for bkt $\in p_{\mathsf{ntf}}$ **do**
  for $b \in [0, Z_M)$, $t \in L$.keys(), $c_s \in$ client.contacts **do**
    $\{\_, \_, \_, k^{\mathsf{ntf}}, \_\} \leftarrow$ client.keys$[c_s]$
    if $\mathsf{PRF}_{\lambda, k^{\mathsf{ntf}}}(t) =$ bkt$[b]$ **then**
      $\mathcal{M}_{\mathsf{all}}$.Add($(c_s, t)$)
if $(c_s, t) \leftarrow$ pushPolicy($\mathcal{M}_{\mathsf{all}}$) $\neq \perp$ **then**
  $\{k_{\mathsf{enc}}, k_{\mathsf{renc}}, k_{\mathsf{rk}}, \_, \_\} \leftarrow$ client.keys$[c_s]$
  $k_{\mathsf{renc},t} \leftarrow \mathsf{PRF}_{k_{\mathsf{renc}}}(t)$
  $f \leftarrow \mathsf{PRF}_{k_{\mathsf{rk}}}(t)$
  $k_{\mathsf{srk},t} \leftarrow k_{\mathsf{srk}}[\Delta - (\text{client}.t - t)]$
  $\ell \leftarrow \mathsf{PRF}_{k_{\mathsf{srk},t}}(f, c_s)$
  $p \leftarrow \mathsf{S}_2$.Read($\ell$)
  for each bkt $\in p$ **do**
    if Verify(k_{svk}, bkt.sig, bkt) **then**
      for $b \in [0, Z_T)$ **do**
        if ct $\leftarrow \mathsf{Dec}_{k_{\mathsf{renc},t}}(\text{bkt}[b])$ succeeds **then**
          return $m \leftarrow \mathsf{Dec}_{k_{\mathsf{enc}}}(\text{ct})$
else
  client.FakeRead()

client.FakeWrite()

$k' \xleftarrow{\$} \{0,1\}^\lambda$
$\mathsf{ct}' \leftarrow \mathsf{Enc}_{k'}(0^{\mathbf{m}})$
$\mathsf{ct}^{\mathsf{ntf}\prime} \xleftarrow{\$} \{0,1\}^\lambda$
$f' \xleftarrow{\$} \{0,1\}^D$
$f^{\mathsf{ntf}\prime} \xleftarrow{\$} \{0,1\}^{Q \cdot N}$
$k'_{\mathsf{renc},t} \xleftarrow{\$} \{0,1\}^\lambda$
S_1.Write($\mathsf{ct}', \mathsf{ct}^{\mathsf{ntf}\prime}, f', f^{\mathsf{ntf}\prime}, k'_{\mathsf{renc},t}$)

client.FakeRead()

$\ell' \xleftarrow{\$} \{0,1\}^D$
S_2.Read(ℓ')

S_1.BatchInit()

$\mathsf{S}_1.k_{\mathsf{srk},t} \xleftarrow{\$} \{0,1\}^\lambda$
$\mathsf{S}_1.P \leftarrow \bigcup_{i \in [N]} \mathsf{S}_1.\mathscr{P}(\ell_i), \quad \ell_i \xleftarrow{\$} \{0,1\}^D$
$\mathsf{S}_1.T_{\mathsf{ntf},t}, \mathsf{S}_1.P_t, \mathsf{S}_1.T_{\mathsf{md},t} = [\,], \text{emptyTreeFrom}(P), \text{emptyTreeFrom}(P)$

S_1.Write(ct, $\mathsf{ct}^{\mathsf{ntf}}, f, f^{\mathsf{ntf}}, k_{\mathsf{renc},t}$)

$t_{\mathsf{exp}} = \mathsf{S}_1.t + \Delta$
$c_s \leftarrow$ getCallingClient()
$\ell \leftarrow \mathsf{PRF}_{\mathsf{S}_1.k_{\mathsf{srk},t}}(f, c_s)$
$\ell^{\mathsf{ntf}} \leftarrow \mathsf{PRF}_{\mathsf{S}_1.k_{\mathsf{srk},t}}(f^{\mathsf{ntf}}, c_s)$
S_1.insertMessage(ct, $\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}}$)
$\mathsf{S}_1.T_{\mathsf{ntf},t}[\ell^{\mathsf{ntf}}]$.Add($\mathsf{ct}^{\mathsf{ntf}}$)

S_1.BatchWrite()

for each bkt $\in \mathsf{S}_1.P$ **do**
  for each $b \in [0, Z_T)$ **do**
    $\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}} \leftarrow \mathsf{S}_1.T_{\mathsf{md}}[\text{bkt}][b]$
    if $\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}} \neq \perp, \perp$ and $\mathsf{S}_1.t < t_{\mathsf{exp}}$ **then**
      $c_{\mathsf{msg}} \leftarrow \text{bkt}[b]$
      $\text{ct} \leftarrow \mathsf{Dec}_{k_{\mathsf{renc},t}}(c_{\mathsf{msg}})$
      S_1.insertMessage(ct, $\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}}$)
for each bkt $\in \mathsf{S}_1.P_t$ **do**
  $s \xleftarrow{\$} \{0,1\}^\lambda$
  bkt.shuffle(s)
  $\mathsf{S}_1.T_{\mathsf{md},t}[\text{bkt}]$.shuffle($s$)
$\mathsf{S}_1.T_{\mathsf{md}}$.Write($\mathsf{S}_1.T_{\mathsf{md},t}$)
$\mathsf{S}_1.P = \mathsf{S}_1.P_t$
for each bkt $\in \mathsf{S}_1.P_t$ **do**
  for $b \in [0, Z_T)$ **do**
    if bkt$[b] = \perp$ **then**
      bkt$[b] \leftarrow \mathsf{Enc}_{k'}(\mathsf{Enc}_{k''}(0^{\mathbf{m}}))$ where $k', k'' \xleftarrow{\$} \{0,1\}^\lambda$
  bkt.sig $\leftarrow \mathsf{Sign}(\mathsf{S}_1.k_{\mathsf{ssk}}, \text{bkt})$
S_2.Write($\mathsf{S}_1.P_t$)
for each bkt $\in \mathsf{S}_1.T_{\mathsf{ntf},t}$ **do**
  for $b \in [\text{bkt.size}(), Z_M)$ **do**
    bkt$[b] \xleftarrow{\$} \{0,1\}^\lambda$
  bkt.shuffle(s), $s \xleftarrow{\$} \{0,1\}^\lambda$
S_2.WriteNotifs($\mathsf{S}_1.T_{\mathsf{ntf},t}$)
S_2.AddPRFKey($\mathsf{S}_1.k_{\mathsf{srk},t}$)
$\mathsf{S}_1.t \mathrel{+}= 1$

S_2.Read(ℓ)

return $\mathscr{P}(\ell)$

S_2.GetPRFKeys()

return S_2.PRFKeys

S_2.ReadNotifs(L)

$p \leftarrow [\,]$
for each $t \in L$.keys(), $i \in [Q]$ **do**
  $j = \Delta - (\mathsf{S}_2.t - t)$
  p.Add($\mathsf{S}_2.T_{\mathsf{ntf}}[j][L[t][i]]$)
return p

S_1.insertMessage(ct, $\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}}$):

$c_{\mathsf{msg}} = \mathsf{Enc}_{k_{\mathsf{renc},t}}(\text{ct})$
$\text{bkt} \leftarrow \mathsf{LCA}(\mathsf{S}_1.P_t, \ell)$
bkt.Add(c_{msg})
$\mathsf{S}_1.T_{\mathsf{md},t}[\text{bkt}]$.Add($\ell, k_{\mathsf{renc},t}, t_{\mathsf{exp}}$)

S_2.Write(P_t)

$\mathsf{S}_2.P = P_t$

S_2.AddPRFKey($k_{\mathsf{srk},t}$)

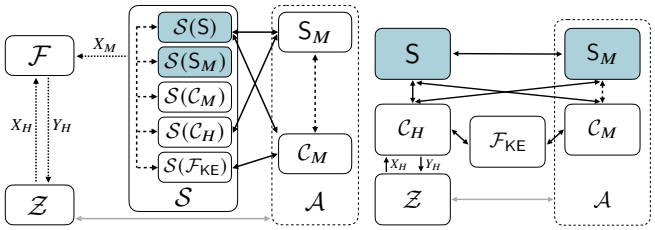
S_2.PRFKeys.Add($k_{\mathsf{srk},t}$)
if $\mathsf{S}_2.t \geq \Delta$ **then**
  S_2.PRFKeys.Remove(0)
$\mathsf{S}_2.t \mathrel{+}= 1$

S_2.WriteNotifs($T_{\mathsf{ntf},t}$)

$\mathsf{S}_2.T_{\mathsf{ntf}}$.Add($T_{\mathsf{ntf},t}$)
if $\mathsf{S}_2.t \geq \Delta$ **then**
  $\mathsf{S}_2.T_{\mathsf{ntf}}$.Remove(0)

Appendix A.
Security Proof

We now define our simulator $\mathcal{S}$ in §A.1 and prove Theorem 1 in §A.2. Although our security proof is independent of ORAM, it relies on Myco's ORAM-style properties, such as padding buckets with indistinguishable dummy blocks and sampling fresh bucket indices per epoch.



(a) Ideal World (b) Real World

Figure 11: S and $\mathcal{C}_H$ denote the honest server and clients; S_M and $\mathcal{C}_M$ denote the malicious server and clients. Dashed outlines mark parties fully controlled by the adversary. The simulator $\mathcal{S}$ maintains internal state per party, denoted $\mathcal{S}(\cdot)$. Shaded nodes run the server protocol. Solid arrows represent protocol interactions; dashed, internal communication within $\mathcal{A}$ and $\mathcal{S}$; dotted, ideal-world interactions; and gray, communication between $\mathcal{A}$ and $\mathcal{Z}$. X_H and X_M are honest and malicious client inputs; Y_H are the honest clients' outputs. X_H is chosen by $\mathcal{Z}$, which also receives Y_H from the reactive functionality $\mathcal{F}$ each epoch.

A.1. Simulator $\mathcal{S}$

$H \to H$, $M \to H$, $H \to M$, and $M \to M$ represent the honesty of the sender and receiver: $H \to H$ indicates that both are honest, $M \to H$ a malicious sender with an honest receiver, $H \to M$ an honest sender with a malicious receiver, and $M \to M$ both being malicious. Let S and S_M denote the honest and malicious server, respectively.

At a high level, when $H \to H$, $\mathcal{S}$ replaces each write performed by a simulated honest client with a fake write and each read with a fake read. For $H \to M$, $M \to H$, or $M \to M$, $\mathcal{S}$ adheres to the real protocol when simulating honest clients. $\mathcal{S}$ can simulate the honest server S by following the real protocol.

We now provide the detailed simulator descriptions for the two cases: (i) S_2 is malicious, and (ii) S_1 is malicious, respectively. In both cases, at system instantiation, upon receiving $\{(c, c') \mid c \in \mathcal{C}_H,\ c' \in \mathcal{C}_M,\ c' \in \mathsf{contacts}[c]\}$ from $\mathcal{F}$, $\mathcal{S}$ uses $\mathcal{F}_{\mathsf{KE}}$ to establish shared symmetric keys between honest and malicious clients. By default, the writes and reads performed by simulated clients (in case $\mathcal{Z}$ decided to keep them online) are fake writes and fake reads. Based on information received from $\mathcal{A}$ and $\mathcal{F}$, $\mathcal{S}$ modifies the behavior of these simulated clients.

A.1.1. Simulator $\mathcal{S}$ for Case 1. $\mathcal{S}$ simulates S_1 and honest clients $\mathcal{C}_H$ for the adversary, while interacting with $\mathcal{F}$ on behalf of malicious clients $\mathcal{C}_M$. In this case, $\mathcal{S}$ approves all write requests from honest clients. $\mathcal{S}$ faithfully simulates S_1 according to the protocol. When S_2 sends the ACK to the simulated S_1 after S_1's batch write, $\mathcal{S}$ sends $\langle \mathsf{NextEpoch} \rangle$ to $\mathcal{F}$. Per conversation:

- $H \to H$: Honest clients perform fake writes and fake reads. In particular, they perform one fake write for both the message and notification, Q fake notification reads, and one fake message read. When $\mathcal{F}$ informs $\mathcal{S}$ of a read request $\langle \mathsf{Read}, c_r, t \rangle$, $\mathcal{S}$ makes the fake read request from c_r. If the simulated c_r does not abort, $\mathcal{S}$ approves the read request of the real c_r; otherwise, it disapproves.
- $M \to H$: $\mathcal{S}$ monitors its simulation of S_1 for messages from a $c_s \in \mathcal{C}_M$ (with $c_s \in \mathsf{contacts}[c_r]$ for an honest client $c_r \in \mathcal{C}_H$) in epoch t. If c_s sends a well-formed message m (i.e., one that the simulated c_r can retrieve successfully), then $\mathcal{S}$ sends $\langle \mathsf{Write}, c_r, m, t \rangle$ to $\mathcal{F}$ on behalf of c_s at the end of epoch t. If c_r is able to retrieve more than one message from c_s in the same epoch, $\mathcal{S}$ does not send any write request from c_s in epoch t to $\mathcal{F}$. When honest c_r eventually reads this message, $\mathcal{F}$ informs $\mathcal{S}$ by sending $\langle \mathsf{Read}, c_s, c_r, t' \rangle$ for some epoch $t' > t$. When this happens, $\mathcal{S}$ makes a real read request from simulated c_r. Rejecting reads in case simulated c_r aborts is the same as in $H \to H$.
- $H \to M$: $\mathcal{S}$ receives $\langle \mathsf{Write}, c_s, c_r, m, t \rangle$ from $\mathcal{F}$ and uses its shared keys to perform a real write for simulated client c_s to send m to c_r in epoch t.
- $M \to M$: $\mathcal{S}$ follows the protocol and does not need to communicate anything to $\mathcal{F}$.

A.1.2. Simulator $\mathcal{S}$ for Case 2. $\mathcal{S}$ simulates S_2 and honest clients $\mathcal{C}_H$ for the adversary, while interacting with $\mathcal{F}$ on behalf of the malicious clients $\mathcal{C}_M$. In this case, $\mathcal{S}$ approves all read requests from honest clients. $\mathcal{S}$ faithfully simulates S_2 according to the protocol. If the batch write from S_1 is well-formed, $\mathcal{S}$ sends $\langle \mathsf{NextEpoch} \rangle$ to $\mathcal{F}$ provided the minimum epoch time has elapsed. Per-conversation:

- $H \to H$: Honest clients perform fake reads and fake writes. $\mathcal{S}$ monitors its simulation of S_2 and does the following based on how the corrupted S_1 processes the write:
  1) If the notification corresponding to the fake write that a simulated c_s performs in epoch t is not in the right bucket, $\mathcal{S}$ disapproves the write request from c_s on receiving $\langle \mathsf{Write}, c_s, t \rangle$.
  2) If the notification is in the right bucket, $\mathcal{S}$ approves the write request from c_s and tracks whether the message is retrievable in epochs $t' \geq t$. At the end of each epoch t', $\mathcal{S}$ sends $\langle \mathsf{Avail}, c_s, t, b \rangle$ to $\mathcal{F}$ where $b = \mathsf{true}$ if the message is in the right bucket during the batch write from S_1 in epoch t'.
- $M \to H$: $\mathcal{S}$ monitors its simulation of S_2 for messages from a $c_s \in \mathcal{C}_M$ (with $c_s \in \mathsf{contacts}[c_r]$ for $c_r \in \mathcal{C}_H$) in epoch t. Suppose $\mathcal{S}$ sees a well-formed notification from c_s in epoch t. First, $\mathcal{S}$ sends $\langle \mathsf{Write}, c_r, \perp, t \rangle$ to $\mathcal{F}$ and approves it. Then, it observes the current retrievable

message corresponding for this notification in epoch $t' \geq t$, and does the following in epoch t':

1) If there is a single retrievable message m, $\mathcal{S}$ sends $\langle \mathsf{Write}, c_r, m, t \rangle$, approves it, and then sends $\langle \mathsf{Avail}, c_s, t, \mathsf{true} \rangle$ to $\mathcal{F}$.
2) Else (there are zero or multiple retrievable messages), $\mathcal{S}$ sends $\langle \mathsf{Avail}, c_s, t, \mathsf{false} \rangle$ to $\mathcal{F}$.

- $H \to M$ & $M \to M$: Same as Case 1.

A.2. Security Proof

Proof. (Theorem 1) There are two cases.

Case 1: $\mathcal{A}$ controls $\mathsf{S}_M = \mathsf{S}_2$ and $\mathcal{C}_M$.

Hybrid 0 $\mathcal{H}_0$: Real world. The real Myco protocol.

Hybrid 1 $\mathcal{H}_1$: Simulator controls $\mathcal{F}_{\mathsf{KE}}$, honest clients $\mathcal{C}_H$, and S_1. Protocol responsibilities are delegated to $\mathcal{S}$: it controls $\mathcal{F}_{\mathsf{KE}}$, the honest clients $\mathcal{C}_H$, and S_1 according to the protocol. Consequently, it generates the shared symmetric keys and it knows honest party inputs (contacts, messages to send, push policies). Since $\mathcal{S}$ faithfully simulates $\mathcal{F}_{\mathsf{KE}}$, the honest clients, and S_1, $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable.

Hybrid 2 $\mathcal{H}_2$: Replace client's KDF-derived keys with uniformly random keys. In this hybrid, the honest clients freshly and independently sample all KDF-derived keys (i.e., k_{enc}, k_{renc}, k_{rk}, k^{ntf}, and $k^{\mathsf{ntf}}_{\mathsf{rk}}$) for their $H \to H$ conversations, instead of deriving them from the corresponding shared secret k. The simulator ensures that communicating honest clients use the same keys so that the receivers can successfully retrieve the messages as in the previous hybrid. Since the shared secret k (i.e., source key material to KDF) has at least λ bits of entropy, this hybrid is indistinguishable due to the pseudorandomness of the KDF.

Hybrid 3 $\mathcal{H}_3$: Replace client's PRF outputs with uniformly random values. In this hybrid, if an honest client c_s is sending a message to another honest client c_r, it replaces the PRF outputs (i.e., f and f^{ntf}, key $k_{\mathsf{renc},t}$, and notification $\mathsf{ct}^{\mathsf{ntf}}$) with uniformly random values, instead of deriving them from $\mathsf{client.keys}[c_r]$ (i.e., k_{rk}, $k^{\mathsf{ntf}}_{\mathsf{rk}}$, k_{renc}, k^{ntf}, respectively). Again, the simulator ensures that the honest client c_r uses the same random values to replace PRF outputs to ensure the same output behavior as before. This hybrid is indistinguishable due to the pseudorandomness of the PRF since all PRF keys are uniformly random in this hybrid.

Hybrid 4 $\mathcal{H}_4$: Authenticity of notifications. In this hybrid, the simulator aborts if an honest client c_r accepts a notification $\mathsf{ct}^{\mathsf{ntf}}$ from honest client c_s with respect to epoch t even though c_s did not send this notification in epoch t. This hybrid is indistinguishable because the notifications for each epoch are uniformly random values of λ bits, and there is a negligible probability that the adversary will guess the correct value.

Hybrid 5 $\mathcal{H}_5$: Authenticity of messages. In this hybrid, the simulator aborts if an honest client c_r accepts a message m from honest client c_s with respect to epoch t, even though c_s did not send m during epoch t. This hybrid is indistinguishable because the simulator aborts with negligible probability due to the INT-CTXT property of the AEAD scheme which provides integrity for both the ciphertext and the associated data. The integrity of the ciphertext prevents the adversary from presenting a valid ciphertext that c_s did not generate, and the integrity of the additional data prevents it from replaying an old valid ciphertext because the associated data embeds the write epoch.

Hybrid 6 $\mathcal{H}_6$: Confidentiality of client messages. In this hybrid, if any honest client c_s wants to send some message m to another honest client c_r, it sends 0 instead: $\mathsf{ct} \leftarrow \mathsf{Enc}_{k_{\mathsf{enc}}}(0; \mathsf{client}.t)$. To keep the outputs of honest parties consistent with the previous hybrid, if c_r receives a valid ciphertext from c_s in epoch t, the simulator has c_r output m in epoch t instead of 0. This hybrid is indistinguishable due to the IND-CPA property of the AEAD scheme. Note that the IND-CPA property is enough in this hybrid because we already have the guarantee that honest clients can identify adversarially generated ciphertexts.

Hybrid 7 $\mathcal{H}_7$: Real $\to$ fake client writes. In this hybrid, when an honest client c_s sends a message to another honest client c_r in epoch t, instead of using k_{enc}, it samples a uniformly random key $k'_{\mathsf{enc}} \xleftarrow{\$} \{0,1\}^\lambda$, that is freshly sampled each epoch t. Accordingly, c_r trial decrypts with k'_{enc} for that epoch instead of using k_{enc}. This hybrid is indistinguishable due to the key-privacy property of the AEAD scheme. Note that in this hybrid, honest clients have replaced real writes to honest clients with fake writes.

Hybrid 8 $\mathcal{H}_8$: $H \to H$ client writes $\to$ dummy writes. In this hybrid, S_1 replaces $k_{\mathsf{renc},t}$ in Tree-Myco for $H \to H$ conversations with the random key $k'_{\mathsf{renc},t} \xleftarrow{\$} \{0,1\}^\lambda$, that is freshly sampled during each epoch t. Accordingly, the honest receivers also use $k'_{\mathsf{renc},t}$ for that epoch to trial-decrypt $c_{\mathsf{msg}} \leftarrow \mathsf{Enc}_{k'_{\mathsf{renc},t}}(\mathsf{ct})$. It also replaces the dummy writes as follows: sample random keys $k', k'' \xleftarrow{\$} \{0,1\}^\lambda$ and output $\mathsf{Enc}_{k'}(\mathsf{Enc}_{k''}(0^{\mathbf{m}}))$. Due to the key-privacy and the IND-CPA property of the AEAD scheme, this hybrid is indistinguishable. Note that the $H \to H$ client writes are now identically distributed to the dummy writes introduced by S_1.

Hybrid 9 $\mathcal{H}_9$: Corrupted S_2 response leads to abort. In this hybrid, the simulator aborts if an honest client does not output abort in an epoch even though S_2's response in that epoch was not according to the protocol specification. S_2 can deviate from the protocol in one of the following three ways:

1) It can refuse service by not responding to the client's read request.
2) It can provide the wrong epoch counter to the client.
3) It can tamper with the buckets sent by S_1.

The honest client can detect all three deviations. For (2), it asks for the epoch counter from both servers and confirms they are the same before accepting it, and for (3), it verifies a signature from S_1 on each bucket it accepts. Thus, $\mathcal{S}$ will abort with negligible probability due to the EUF-CMA property of signatures, and this hybrid is indistinguishable.

Hybrid 10 $\mathcal{H}_{10}$: Correct S_2 response guarantees $H \to H$ message delivery. In this hybrid, $\mathcal{S}$ aborts if S_1

aborts because it cannot fit the client writes (both honest and malicious) in the Myco buckets. If S_1 does not abort and S_2 provides the right response to a client, the client is guaranteed to receive its $H \to H$ messages. In §5 and §B, we prove that we choose the Myco parameters such that S_1 can always accommodate the writes in the buckets assuming the bucket indices for each write are chosen uniformly at random. Recall that bucket indices ℓ and ℓ^{ntf} are chosen by a PRF output. Since the PRF seed $k_{\mathsf{srk},t}$ is chosen after the inputs to the PRF are decided, it is straightforward to prove that an adversary that can make a bucket overflow with PRF outputs can also distinguish the PRF from a random function. Thus, the simulator never aborts in this hybrid and it is indistinguishable from the previous one.

Hybrid 11 $\mathcal{H}_{11}$: Real $\to$ fake client reads. In this hybrid, any honest client receiver c_r makes the following changes: the buckets to read f^{ntf} and f for conversations with any honest sender c_s are chosen at random (independently of the corresponding sender c_s). As a result, the real client reads for $H \to H$ conversations are now identical to the fake client reads. Client c_r preserves the output from the previous hybrid if it does not detect protocol deviation from S_2 because the correct output is guaranteed if S_2 does not deviate.

Since we have just replaced real $H \to H$ client reads with fake reads, this hybrid is indistinguishable from the previous hybrid if real $H \to H$ and fake client reads are indistinguishable for the adversary which sees reads through corrupted S_2. To prove that this is indeed the case, consider the following facts about this hybrid:

- The real $H \to H$ writes are identical to the dummy writes introduced by S_1 from adversary's perspective. So we can think of real $H \to H$ writes also as dummy writes.
- The only difference between a real $H \to H$ read and a fake read in the previous hybrid is that the former has a corresponding real write.

Thus, as long as each fake read in this hybrid could potentially have a corresponding real write, any of these could potentially be a real $H \to H$ read for the adversary. This is the case if the total number of fake reads by honest receivers (includes real $H \to H$ reads) to a bucket do not exceed the number of dummy writes (includes real $H \to H$ writes) in that bucket. All the malicious client writes and the fake reads from honest receivers can be seen as $Q \cdot N$ and N random accesses to the Matrix-Myco and Tree-Myco buckets, and our parameter selection in §5 ensures the accesses never overflow the bucket.

Hybrid 12 $\mathcal{H}_{12}$: Ideal world. In this hybrid, we realize the complete simulator described in §A.1.1. The following changes are made from the previous hybrid:

1) The honest clients and S_1 are no longer controlled by the simulator, and the honest clients only interact with the ideal functionality $\mathcal{F}$. Thus, $\mathcal{S}$ no longer has access to honest client inputs like their messages, contacts, and push policies.
2) $\mathcal{S}$ receives $\{(c, c') \mid c \in \mathcal{C}_H \ \wedge \ c' \in \mathcal{C}_M \ \wedge \ c' \in \mathsf{contacts}[c]\}$ from $\mathcal{F}$, and accordingly makes request from the simulated honest clients to $\mathcal{F}_{\mathsf{KE}}$ to setup shared symmetric keys with malicious clients just like before.
3) When S_2 sends the ACK to the simulated S_1 after S_1's batch write, $\mathcal{S}$ sends $\langle \mathsf{NextEpoch} \rangle$ to $\mathcal{F}$. Note that simulated S_1 will only perform a batch write after minimum epoch time has elapsed.
4) For $H \to H$ conversations, the honest clients in the previous hybrid are only performing fake writes and fake reads, and thus, the simulator does not need to know their inputs to simulate them. From the previous hybrid, we have the following guarantees for $H \to H$ conversations: messages from honest senders cannot be spoofed or replayed, they are either delivered according to push policy if S_2 follows the protocol or the honest client outputs $\perp$ otherwise. Whenever $\mathcal{F}$ sends $\langle \mathsf{Read}, c_r, t \rangle$ to $\mathcal{S}$, it performs a fake read from simulated c_r, and if simulated c_r does not abort, it approves the read request of real c_r. Once the read is approved, $\mathcal{F}$ delivers reads according to the push policy and the message delivery from honest senders is guaranteed without spoofing or replaying. In case c_r aborts, $\mathcal{S}$ disapproves the read request and $\mathcal{F}$ informs c_r to output $\perp$. Importantly, in the previous hybrids, if a read is disapproved in epoch t that was supposed to output some message m_t, then the next epoch $t' > t$ when a successful read happens, c_r outputs the message $m_{t'}$ according to the push policy, and skips over $\{m_t, \ldots, m_{t'-1}\}$ (as is prescribed in the real protocol). $\mathcal{F}$ also enforces the same output behavior by setting $\mathsf{DB}[c_s, c_r, t] = \top$ (marking it as read) even if c_r's read was disapproved.
5) For $M \to H$ conversations, since $\mathcal{S}$ has set up shared keys between honest and malicious clients and it simulates S_1, it can detect when a malicious client c_s in the contact list[12] of an honest client c_r is sending a message m to c_r in epoch t. Accordingly, at the end of epoch t, $\mathcal{S}$ sends $\langle \mathsf{Write}, c_r, m, t \rangle$ to $\mathcal{F}$ on behalf of c_s if the message is well-formed such that the simulated honest client can actually retrieve it without outputting $\perp$. Sending these requests at the end of epoch t is okay because honest clients can only receive them in epoch $t+1$ onwards. Note that a malicious client c_s can potentially send multiple messages to an honest c_r in a single epoch (potentially through multiple malicious senders). In case this happens, the protocol prescribes c_r to output $\perp$ (since that proves the sender is malicious). Accordingly, if $\mathcal{S}$ detects that there are multiple retrievable messages to a c_r from the same c_s, it does not send any message to $\mathcal{F}$ from c_s. Finally, when honest c_r eventually reads the message sent by a malicious c_s, $\mathcal{F}$ informs $\mathcal{S}$ by sending $\langle \mathsf{Read}, c_s, c_r, t' \rangle$ for some epoch $t' > t$. When this happens, $\mathcal{S}$ makes a real read request from simulated c_r to ensure that the adversary's view of $M \to H$ conversations is identical to the previous hybrid. Rejecting

12. The simulator does not care about malicious clients outside the contact list of an honest client because the honest clients in the previous hybrid do not assign messages to clients outside their contact list and the ones interacting with $\mathcal{F}$ do not get messages from clients outside their contact list.

reads in case simulated c_r aborts is the same as in the $H \to H$ conversations.

6) For $H \to M$ conversations, whenever an honest client c_s writes a message m to malicious client c_r during epoch t, $\mathcal{F}$ sends $\langle \mathsf{Write}, c_s, c_r, m, t \rangle$ to $\mathcal{S}$. Since $\mathcal{S}$ already has a shared key setup with c_r and an honest client only performs a single write per epoch, $\mathcal{S}$ performs a real write (instead of a fake write) for simulated client c_s to send m to c_r in epoch t.
7) For $M \to M$ conversations, $\mathcal{S}$ does not change anything from the previous hybrid as it does not need to simulate these clients.

This hybrid is the same as the ideal world, and ensures that the output of honest parties and adversary's view are identical to the previous hybrid.

Case 2: $\mathcal{A}$ controls $\mathsf{S}_M = \mathsf{S}_1$ and $\mathcal{C}_M$.

$\mathcal{H}_0$**: Real world.** The real Myco protocol.

$\mathcal{H}_1$**: Simulate $\mathcal{F}_{\mathsf{KE}}$, honest clients $\mathcal{C}_H$, and S_2.** Same as Case 1 except $\mathcal{S}$ controls S_2 instead of S_1.

$\mathcal{H}_2, \ldots, \mathcal{H}_7$**:** Same as Case 1. As in Case 1, in this hybrid, real $H \to H$ writes have been replaced with fake writes, we have authenticity of notifications and messages for $H \to H$ conversations, and the bucket indices for $H \to H$ messages are chosen uniformly at random (with the constraint that a communicating honest pair of clients write and read to the same bucket) instead of being derived from a PRF.

Hybrid 8 $\mathcal{H}_8$**: Real $\to$ fake client reads.** The same change as $\mathcal{H}_{11}$ in Case 1. This hybrid is indistinguishable because the adversary does not control S_2, and as such, does not get to see the honest client reads.

Hybrid 9 $\mathcal{H}_9$**: Ideal world.** In this hybrid, we realize the complete simulator described in §A.1.2. The following changes are made from the previous hybrid:

1) The honest clients and S_2 are no longer controlled by the simulator, and the honest clients only interact with the ideal functionality $\mathcal{F}$, as described in the description of ideal world (§4.2). Thus, $\mathcal{S}$ no longer has access to honest client inputs like their messages, contacts, and push policies.
2) $\mathcal{S}$ sets up the shared symmetric keys between honest and malicious clients as in Case 1.
3) If the batch write sent by S_1 to simulated S_2 is well-formed (has valid signatures and has the right structure), $\mathcal{S}$ sends $\langle \mathsf{NextEpoch} \rangle$ to $\mathcal{F}$ on behalf of S_1 provided the minimum epoch time has elapsed.
4) For $H \to H$ conversations, the honest clients in the previous hybrid were performing fake reads and writes, and thus, the simulator does not need to know their inputs to simulate them. For any epoch t and any honest sender client c_s, the simulated S_2 can track whether the message sent by the simulated c_s in epoch t is retrievable in some epoch $t' \geq t$. A message is retrievable if and only if the corrupted S_1 put it in the right bucket according to the protocol. The notification must be put in the right bucket during epoch t for the message to be retrievable because the notifications have separate buckets for each epoch. Thus, if the notification was not in the right bucket, $\mathcal{S}$ disapproves the write request from c_s on receiving $\langle \mathsf{Write}, c_s, t \rangle$. Otherwise, $\mathcal{S}$ approves the write request and tracks whether the message is retrievable in epochs $t' \geq t$. At the end of each epoch t', $\mathcal{S}$ sends $\langle \mathsf{Avail}, c_s, t, b \rangle$ to $\mathcal{F}$ where $b = \mathsf{true}$ if the message is in the right bucket during the batch write from S_1 in epoch t'. Note that the $H \to H$ messages are delivered according to the push policy (modulo their availability) in this hybrid and the previous one. Every successful notification in the previous hybrid is considered by an honest receiver while applying the push policy, and $\mathcal{S}$ tells $\mathcal{F}$ to approve all $H \to H$ write requests which had a successful notification, which are then ultimately considered by $\mathcal{F}$ while applying the push policy. Note that it is okay to delay setting the availability of writes requests until the end of epoch (when corrupted S_1 performs the batch write) because messages written in epoch t will only be read epoch $t+1$ onwards.
5) For $M \to H$ conversations, like in Case 1, $\mathcal{S}$ can detect when a malicious client c_s in the contact list of an honest client c_r is sending a message m to c_r in epoch t (since $\mathcal{S}$ simulates S_2). In case S_1 follows the protocol, $\mathcal{S}$ simulates the same way as Case 1. If not, corrupted S_1 and c_s can deviate in the following ways:
   a) The notification is not in the right bucket during epoch t.
   b) The notification is in the right bucket but the corresponding message changes every epoch, and for some epochs, it could even be unavailable (in the wrong bucket or absent).

   For the first behavior, $\mathcal{S}$ does not perform a write on behalf of c_s because the honest client's push policy will not consider this message without a valid notification, and notification for epoch t cannot be introduced in a later epoch. For the second behavior, it first sends $\langle \mathsf{Write}, c_r, \perp, t \rangle$ to $\mathcal{F}$ and approves it to ensure the notification is considered by the push policy of c_r. Then, it observes the current retrievable message corresponding for this notification in epoch $t' \geq t$, and does the following in epoch t': if there is a single retrievable message m, $\mathcal{S}$ sends $\langle \mathsf{Write}, c_r, m, t \rangle$, approves it, and then sends $\langle \mathsf{Avail}, c_s, t, \mathsf{true} \rangle$ to $\mathcal{F}$. Else (there are zero or multiple retrievable messages), $\mathcal{S}$ sends $\langle \mathsf{Avail}, c_s, t, \mathsf{false} \rangle$ to $\mathcal{F}$. With this simulation strategy, we ensure that malicious client messages availability is the same in ideal world and the previous hybrid. Since the push policy only depends on the valid notifications, these messages are also delivered to an honest client according to their push policy.
6) $\mathcal{S}$ handles $H \to M$ and $M \to M$ conversations the same way as in the ideal world of Case 1.

This hybrid is the same as the ideal world, and ensures that the outputs of honest parties and the adversary's view are identical to the previous hybrid.

□

Appendix B.
Capacity Analysis Proof

B.1. Matrix-Myco

Proof. (Theorem 2) Let QN be the number of buckets in Matrix-Myco, and let X_v denote the number of messages (out of these QN) that land in bucket v. Because each message chooses its bucket uniformly at random, the placement of any single message in v is a Bernoulli trial with probability $p = 1/(QN)$. Hence

$$\Pr[X_v > Z_M] \;=\; \Pr\big[\mathrm{Binomial}(QN, 1/(QN)) > Z_M\big].$$

Set $Z_M = 1 + \delta$ with $\delta = 2\big(\kappa + \log(QN)\big) \geq 2$. Applying the Chernoff bound gives

$$\begin{aligned}
&\Pr\big[\mathrm{Binomial}(QN, 1/(QN)) > Z_M\big] \\
&\quad\leq \exp\Big(-\tfrac{\delta^2}{2+\delta}\Big) \qquad\qquad \text{(Chernoff)} \\
&\quad\leq \exp\Big(-\tfrac{\delta}{2}\Big) \\
&\quad\leq \exp\Big(-\kappa - \log(QN)\Big) \\
&\quad\leq 2^{-\kappa - \log(QN)}.
\end{aligned}$$

Taking a union bound over all QN buckets,

$$\begin{aligned}
\Pr\Big[\bigcup_{v\in[QN]} \{X_v > Z_M\}\Big] &\leq QN \cdot 2^{-\kappa-\log(QN)} \\
&\leq 2^{-\kappa}.
\end{aligned}$$

□

B.2. Tree-Myco

B.2.1. Overflow probability for a single node.

Theorem 4. *Let $X_{v,d}$ be a random variable representing the total number of messages (out of N) that are assigned to a node v at depth d. Then the probability that $X_{v,d}$ exceeds capacity Z_T satisfies*

$$\begin{aligned}
\Pr[X_{v,d} > Z_T] \;=\; 2\Big[\big(1-2^{-(d+1)}\big)^n \;-\; \big(1-2^{-d}\big)^n\Big] \\
\cdot\; \Pr\Big[\mathrm{Binomial}(N, 2^{-(d+1)}) > Z_T\Big].
\end{aligned}$$

Proof. (Theorem 4) Let $P = (V_P, E_P)$ denote the sparse binary tree representing the path-set of n paths, where V_P is the set of nodes in the tree that are part of the path-set, and E_P is the set of edges between them. Let $L_{d,n}$ be an event that denotes that exactly one child of the node v is one of the path-set nodes in V_P. Note that this is a pre-requisite for v to be an LCA. If both children are in the path-set, v can not be the deepest node for any path that intersects with the path-set; this is because v's child or an even deeper node would be considered the LCA. Conversely, if none of the children are in the path-set, then v is also not in the path-set.

We can define the probability of $L_{d,n}$ as follows: we want at least one path in the path-set to go through v at depth-d, but we do not want any path in the path-set to go through a child w of v.

To calculate the probability that the path-set goes through v but not w, we first analyze the probability that the path-set goes through a node v. Note that:

$$\Pr[v \in V_P] = 1 - \Pr[v \notin V_P] = 1 - (1 - 2^{-d})^n$$

Likewise, the probability that the path-set goes through the child w of this particular node v is given by:

$$\Pr[w \in V_P] = 1 - \Pr[w \notin V_P] = 1 - (1 - 2^{-(d+1)})^n$$

Note that every path that passes through a child node w must also pass through its parent v. In other words, the event $\{w \in V_P\}$ is a subset of the event $\{v \in V_P\}$. Therefore, subtracting the probability $\Pr[w \in V_P]$ from $\Pr[v \in V_P]$ directly yields the probability that the path-set goes through v but not through w. Since $\Pr[L_{d,n}]$ is the probability that path-set goes through v (depth d) but not w (depth $d+1$) for two choices of w, we can write:

$$\begin{aligned}
\Pr[L_{d,n}] &= 2 \cdot [\Pr[v \in V_P] - \Pr[w \in V_P]] \\
&= 2 \cdot [(1 - (1 - 2^{-d})^n) - (1 - (1 - 2^{-(d+1)})^n)] \\
&= 2 \cdot [(1 - 2^{-(d+1)})^n - (1 - 2^{-d})^n]
\end{aligned}$$

Now that we have established the probability a node v could be an LCA, we look at the probability that a message with a random path ℓ will be put in v. Let A be this event. Then, we have ($\Pr[A|\neg L_{d,n}] = 0$):

$$\begin{aligned}
\Pr[A] &= \Pr[L_{d,n}] \cdot \Pr[A|L_{d,n}] + \Pr[\neg L_{d,n}] \cdot \Pr[A|\neg L_{d,n}] \\
&= \Pr[L_{d,n}] \cdot \Pr[A|L_{d,n}] \\
&= \Pr[L_{d,n}] \cdot 2^{-(d+1)}
\end{aligned}$$

In the above expression, $\Pr[A|L_{d,n}] = 2^{-(d+1)}$ because the path ℓ needs to go exactly through the child w (at depth $d+1$) for v to be the LCA.

We can extend this analysis to N messages, and since all of these messages are independently chosen w.r.t. a given path-set. Once the path-set is fixed and node v is an LCA with probability $\Pr[L_{d,n}]$, each message sampling can be seen as an independent Bernoulli trial with probability $p = 2^{-(d+1)}$. Thus, we can write the probability $\Pr[X_{v,d} > Z_T]$ as follows:

$$\begin{aligned}
\Pr[X_{v,d} > Z_T] &= \Pr[L_{d,n}] \cdot \Pr[X_{v,d} > Z_T \mid L_{d,n}] \\
&\quad + \Pr[\neg L_{d,n}] \cdot \Pr[X_{v,d} > Z_T \mid \neg L_{d,n}] \\
&= \Pr[L_{d,n}] \cdot \Pr[X_{v,d} > Z_T \mid L_{d,n}] \\
&= \Pr[L_{d,n}] \cdot \Pr[\mathrm{Binomial}(N, 2^{-(d+1)}) > Z_T]
\end{aligned}$$

□

B.2.2. Overflow probability for a single epoch.

Theorem 5. *Choosing the bucket capacity $Z_T = \Theta(\kappa + \log N\Delta)$ ensures that the total overflow probability in a single epoch in the tree (i.e., the probability that some node receives more than Z_T messages) is at most $2^{-\kappa}$ for security parameter κ.*

Proof. (Theorem 5) We split this analysis into two parts. Let $d_0 = \lfloor \log(\frac{N}{2\ln 2(\kappa+\log N\Delta+2)}) \rfloor$ be the breakeven tree level where we split the analysis. First, we analyze the overflow probability of nodes at shallow levels $0 \le d \le d_0$ and then later we look at the overflow probability of nodes at deeper levels $d_0 < d \le D$.

Shallow levels $0 \le d \le d_0$. Recall from Theorem 4 that $\Pr[X_{v,d} > Z_T] = \Pr[L_{d,n}] \cdot \Pr[\text{Binomial}(N, 2^{-(d+1)}) > Z_T] \le \Pr[L_{d,n}]$. We first argue that $\Pr[L_{d,n}]$ is $O(2^{-(\kappa+\log N)})$ for these shallow levels.

Given $n = N$, we can re-write $\Pr[L_{d,n}]$ as follows for levels $0 \le d \le d_0$:

$$\begin{aligned}
&\Pr[L_{d,n}] \\
&= 2 \cdot [(1-2^{-(d+1)})^N - (1-2^{-d})^N] \\
&= 2 \cdot [2^{N\log(1-2^{-(d+1)})} - 2^{N\log(1-2^{-d})}] \\
&\le 2 \cdot 2^{N\log(1-2^{-(d+1)})} \\
&\le 2 \cdot 2^{-N\cdot\frac{2^{-d}}{2\ln 2}} \\
&\qquad \left(\log(1-x) \le \frac{-x}{\ln(2)} \text{ for } 0 < x < 1\right) \\
&\le 2 \cdot 2^{-\kappa-\log N\Delta-2} \\
&\qquad \left(\frac{2\ln 2(\kappa+\log N\Delta+2)}{N} \le 2^{-d_0} \le 2^{-d}\right) \\
&\le \frac{2^{-\kappa}}{2N\Delta} \le \frac{2^{-\kappa}}{2N}
\end{aligned}$$

Given that there are at most $2^{d_0+1} \le 2\cdot\frac{N}{2\ln 2(\kappa+\log N\Delta)} \le N$ nodes in the shallow levels, by the union bound it follows that:

$$\Pr\Big[\bigcup_{0\le d\le d_0}\bigcup_{v\in V_d}\{X_{v,d} > Z_T\}\Big] \le N\cdot\frac{2^{-\kappa}}{2N} \le 2^{-\kappa-1}.$$

Deep levels $d_0 < d \le D$. Recall from Theorem 4 that $\Pr[X_{v,d} > Z_T] = \Pr[L_{d,n}] \cdot \Pr[\text{Binomial}(N, 2^{-(d+1)}) > Z_T] \le \Pr[\text{Binomial}(N, 2^{-(d+1)}) > Z_T]$. We first argue that $\Pr[\text{Binomial}(N, 2^{-(d+1)}) > Z_T] \le 2^{-(\kappa+\log N\Delta)}$ for the deep levels $d > d_0$, and then take a union bound over all $N\Delta$ buckets to show that all of them overflow with at most $2^{-\kappa-1}$ probability.

For $d > d_0$, the mean of binomial is $\mu \le N\cdot 2^{-(d+1)} \le N\cdot 2^{-d_0} \le N\cdot\frac{4\ln 2(\kappa+\log N\Delta+2)}{N} = 4\ln 2\cdot(\kappa+\log N\Delta+2)$. Let $\mu_0 = 4\ln 2(\kappa+\log N\Delta+2)$ and $Z_T = (1+\delta)\cdot\mu_0$, where $\delta = 2$ is a constant. Now, we can rewrite the binomial for depth d as follows:

$$\begin{aligned}
&\Pr[\text{Binomial}(N, 2^{-(d+1)}) > Z_T] \\
&\le \Pr[\text{Binomial}(N, \frac{\mu_0}{N}) > Z_T] \qquad (\frac{1}{2^{d+1}} < \frac{\mu_0}{N}) \\
&\le \Pr[\text{Binomial}(N, \frac{\mu_0}{N}) > (1+\delta)\cdot\mu_0] \\
&\le \exp(-\frac{\delta^2}{2+\delta}\cdot\mu_0) \qquad \text{(Chernoff Bound)} \\
&\le \exp\Big(-\frac{\delta^2}{2+\delta}\cdot(\kappa+\log N\Delta+2)\Big) \\
&\le \exp\Big(-(\kappa+\log N\Delta+2)\Big) \qquad (\delta = 2) \\
&\le \frac{2^{-\kappa}}{4N\Delta}
\end{aligned}$$

Given that there are at most $2^D \le 2N\Delta$ nodes in the deep levels, by the union bound it follows that:

$$\Pr\Big[\bigcup_{d_0<d\le D}\bigcup_{v\in V_d}\{X_{v,d} > Z_T\}\Big] \le 2N\Delta\cdot\frac{2^{-\kappa}}{4N\Delta} \le 2^{-\kappa-1}.$$

Combining the results. If we take a union bound over the shallow as well as deep layers, we get the following result for $Z_T = \Theta(\kappa+\log N\Delta)$:

$$\Pr\Big[\bigcup_{0\le d\le D}\bigcup_{v\in V_d}\{X_{v,d} > Z_T\}\Big] \le 2^{-\kappa-1} + 2^{-\kappa-1} \le 2^{-\kappa}.$$

□

B.2.3. Overflow probability across Δ epochs.

Proof. (Theorem 3) Building on the analysis of shallow and deep layers from Theorem 5, we prove that across epochs $t \in [0, \Delta-1]$, the combined overflow probability of any bucket is negligible.

Let $d_i = \lfloor \log(\frac{(i+1)\cdot N}{2\ln 2(\kappa+\log(N\Delta^3)+2)}) \rfloor$. Intuitively, the d_i value helps us define the level above which messages from certain epochs are guaranteed to not be stored. In particular, in epoch t, we have the guarantee that messages from epochs $\le t-i$ will always occupy levels $> d_i$. This is shown pictorially in Figure 6.

To bound the combined overflow probability, we define some bad events that we do not want to happen.

Bad Events. $\forall t \in [0,\Delta), i \in [0,t]$, let $A_{t,i}$ denote the bad event that any epoch i message falls within layers $0 \le d \le d_{t-i}$ in epoch t. $\forall t \in [0,\Delta), i \in [1,t]$, let $B_{t,i}$ denote the bad event that at least one bucket in layers $d \in (d_{i-1}, d_i]$ overflows when $i\cdot N$ messages (corresponding to epochs $(t-i, t]$) are inserted in epoch t. Note that if events $\{A_{t,j}\}_{j\in[0,t-i]}$ do not happen, these buckets are only supposed to get messages from epochs $(t-i, t]$. $B_{t,t+1}$ is a special case that considers layers $(d_t, D]$ and the bad event is that one of the buckets in these layers overflows when $(t+1)\cdot N$ messages (corresponding to all $t+1$ epochs) are inserted.

Success Condition. If for all epochs $t \in [0, \Delta-1]$ and $i \in [0, t]$, all the bad events $A_{t,i}$ and $B_{t,i+1}$ do not happen, note that no overflows happen across Δ epochs. Now, we bound the probability that these bad events happen.

Analyzing $\Pr[A_{t,i}]$. This is similar to the shallow-level argument from Theorem 5. Note that epoch i messages have access to a $(t+1-i)\cdot N$-sized pathset in epoch t.

This is the case because the following are equivalent in terms of the placement of a message: (1) the path-set being sampled incrementally, with N paths sampled at a time in each epoch and a message moving from its current location to its LCA with the path-set; (2) the path-set being sampled for all t' epochs at once and a message is placed at its LCA with this path-set. To see why, consider a message with a fixed, random leaf. In epoch 0, it is assigned to the deepest node along its path that appears in the initial set of N random paths. In each subsequent epoch, N new independent random paths are added, and the message is reassigned to the deepest node common to its leaf path and the new path-set. After t' epochs, the union of all path-sets comprises $t'N$ independent random paths. Since the LCA operator is monotonic, adding paths can only push the LCA deeper or leave it unchanged. Thus, the final assignment is equivalent to computing the LCA over a single batch of $t'N$ paths.

Recall from Theorem 4 that $\Pr[X_{v,d} > 0] \leq \Pr[L_{d,n}]$. Given $n = (t+1-i)\cdot N$, we can re-write $\Pr[L_{d,n}]$ as follows for levels $0 \leq d \leq d_{t-i}$:

$$\begin{aligned}
\Pr[L_{d,n}] &= 2\cdot[(1-2^{-(d+1)})^n - (1-2^{-d})^n] \\
&\leq 2\cdot 2^{n\log(1-2^{-(d+1)})} \\
&\leq 2\cdot 2^{-n\cdot\frac{2^{-d}}{2\ln 2}} \\
&\qquad \left(\log(1-x) \leq \frac{-x}{\ln(2)} \text{ for } 0<x<1\right) \\
&\leq 2\cdot 2^{-\kappa-\log N\Delta^3-2} \\
&\qquad \left(\tfrac{2\ln 2(\kappa+\log N\Delta^3+2)}{(t+1-i)\cdot N} \leq 2^{-d_{t-i}} \leq 2^{-d}\right) \\
&\leq \frac{2^{-\kappa}}{2N\Delta^3}
\end{aligned}$$

Given that there are at most $2^{d_{t-i}+1} \leq 2\cdot\frac{(t+1-i)\cdot N}{2\ln 2(\kappa+\log N\Delta^3+2)} \leq (t+1-i)\cdot N$ nodes in the shallow levels, by the union bound it follows that:

$$\begin{aligned}
\Pr[A_{t,i}] &\leq \Pr\Big[\bigcup_{0\leq d\leq d_{t-i}}\bigcup_{v\in V_d}\{X_{v,d}>0\}\Big] \\
&\leq (t+1-i)N\cdot\frac{2^{-\kappa}}{2N\Delta^3} \leq \frac{t+1-i}{2\Delta^3}2^{-\kappa}.
\end{aligned}$$

Analyzing $\Pr[B_{t,i}]$. This is similar to the deep-level argument from Theorem 5. Recall from Theorem 4 that $\Pr[X_{v,d} > Z_T] = \Pr[L_{d,n}]\cdot\Pr[\text{Binomial}(i\cdot N, 2^{-(d+1)}) > Z_T] \leq \Pr[\text{Binomial}(i\cdot N, 2^{-(d+1)}) > Z_T]$. For $d_{i-1} < d \leq d_i$, the mean of binomial is $\mu \leq i\cdot N\cdot 2^{-(d+1)} \leq i\cdot N\cdot 2^{-d_i} \leq i\cdot N\cdot\frac{4\ln 2(\kappa+\log N\Delta^3+2)}{(i+1)\cdot N} = 4\ln 2\cdot(\kappa+\log N\Delta^3+2)$. Let $\mu_0 = 4\ln 2(\kappa+\log N\Delta^3+2)$ and $Z_T = (1+\delta)\cdot\mu_0$, where $\delta = 2$ is a constant. Note that μ_0 is independent of i and t, and thus, Z_T stays consistent for all $B_{t,i}$. Now, we can rewrite the binomial for depth d as follows:

$$\begin{aligned}
&\Pr[\text{Binomial}(i\cdot N, 2^{-(d+1)}) > Z_T] \\
&\quad\leq \Pr[\text{Binomial}(i\cdot N, \tfrac{\mu_0}{i\cdot N}) > Z_T] \qquad \left(\tfrac{1}{2^{d+1}} < \tfrac{\mu_0}{i\cdot N}\right) \\
&\quad\leq \Pr[\text{Binomial}(i\cdot N, \tfrac{\mu_0}{i\cdot N}) \\
&\qquad\qquad > (1+\delta)\mu_0] \\
&\quad\leq \exp(-\tfrac{\delta^2}{2+\delta}\cdot\mu_0) \qquad \text{(Chernoff Bound)} \\
&\quad\leq \exp\big(-\tfrac{\delta^2}{2+\delta} \\
&\qquad\qquad \cdot 4\ln 2(\kappa+\log N\Delta^3+2)\big) \\
&\quad\leq \exp(-(\kappa+\log N\Delta^3+2)) \qquad (\delta=2) \\
&\quad\leq \frac{2^{-\kappa}}{4N\Delta^3}
\end{aligned}$$

Given that there are at most $2^{d_i+1} - 2^{d_{i-1}+1}$ nodes in the deep levels, by the union bound it follows that:

$$\begin{aligned}
\Pr[B_{t,i}] &\leq \Pr\Big[\bigcup_{d_{i-1}<d\leq d_i}\bigcup_{v\in V_d}\{X_{v,d}>Z_T\}\Big] \\
&\leq 2\cdot(2^{d_i}-2^{d_{i-1}})\cdot\frac{2^{-\kappa}}{4N\Delta^3} \leq (2^{d_i}-2^{d_{i-1}})\frac{2^{-\kappa}}{2N\Delta^3}.
\end{aligned}$$

For $B_{t,t+1}$, the levels $d_t < d \leq D$ are considered that could have at most $2^{D+1} - 2^{d_t+1}$ nodes. Thus, we have the following by the union bound:

$$\begin{aligned}
\Pr[B_{t,t+1}] &\leq \Pr\Big[\bigcup_{d_t<d\leq D}\bigcup_{v\in V_d}\{X_{v,d}>Z_T\}\Big] \\
&\leq 2\cdot(2^{D}-2^{d_t})\cdot\frac{2^{-\kappa}}{4N\Delta^3} \leq (2^{D}-2^{d_t})\frac{2^{-\kappa}}{2N\Delta^3}.
\end{aligned}$$

Union bound over all bad events. Now, we take a union bound over all bad events to upper bound the failure probability.

$$\begin{aligned}
&\cup_{t\in[0,\Delta-1]}\cup_{i\in[0,t]}(\Pr[A_{t,i}]\cup\Pr[B_{t,i+1}]) \\
&\quad\leq \sum_{t\in[0,\Delta-1]}\sum_{i\in[0,t]}(\Pr[A_{t,i}]+\Pr[B_{t,i+1}]) \\
&\quad\leq \sum_t\sum_i\frac{t+1-i}{2\Delta^3}2^{-\kappa} + \sum_t\Big((2^D-2^{d_t})\frac{2^{-\kappa}}{2N\Delta^3} \\
&\qquad + \sum_{i\in[1,t]}(2^{d_i}-2^{d_{i-1}})\frac{2^{-\kappa}}{2N\Delta^3}\Big) \\
&\quad\leq 2^{-\kappa}\Big(\frac{1}{2}+\sum_{t=0}^{\Delta-1}\frac{2^D}{2N\Delta^3}\Big) \\
&\quad\leq 2^{-\kappa}\Big(\frac{1}{2}+\frac{1}{2\Delta}\Big) \\
&\quad\leq 2^{-\kappa}
\end{aligned}$$

Success Probability. Note that the success probability (no bucket overflows over Δ epochs) is as follows:

$$\Pr[\text{success}] \geq \cap_{t\in[0,\Delta-1]}\cap_{i\in[0,t]}(\Pr[\neg A_{t,i}]\cap\Pr[\neg B_{t,i+1}])$$

In other words, the success probability is $\geq$ none of the bad events happening. Using DeMorgan's principle, we can

alternatively write it as:

$$\Pr[\text{success}] \geq 1 - \bigcup_{t \in [0,\, \Delta-1]} \bigcup_{i \in [0,\, t]} (\Pr[A_{t,i}] \cup \Pr[B_{t,i+1}])$$
$$\geq 1 - 2^{-\kappa}$$

□

Appendix C. Key Privacy

Prior work shows that if a probabilistic (or nonce-based with randomized nonces) authenticated encryption scheme produces ciphertexts indistinguishable from random, this implies key-privacy [70].

Definition 2 (Key Privacy)**.** *For any PPT adversary $\mathcal{A}$, given access to two encryption oracles $\mathsf{Enc}(k_0, \cdot)$ and $\mathsf{Enc}(k_1, \cdot)$ with randomly chosen keys $k_0, k_1 \in \{0,1\}^l$, the advantage of distinguishing which key is used is:*

$$\mathbf{Adv}^{\mathsf{KP}}_{\text{Æ}}(\lambda) := \left| \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Enc}(k_b, \cdot)}(1^\lambda) : b \xleftarrow{\$} \{0,1\}\right] - \frac{1}{2} \right|$$

We say the scheme is key-private if this advantage is negligible in λ.

As we note in §6.1, we use AES-GCM for authenticated encryption with random nonces, which indeed has indistinguishability from random ciphertexts property [138]. Thus, we consider AES-GCM to be a key-private AE.

Appendix D. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

The paper designs a metadata-private communication system, Myco, in the two-server model. They use techniques based on ORAM to achieve polylogarithmic server computation and polylogarithmic communication complexity for sending and receiving messages.

D.2. Scientific Contributions

- Addresses a Long-Known Issue
- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

D.3. Reasons for Acceptance

1) Provides interesting technique to utilize ORAM towards designing metadata-private messaging system.
2) Consequently, makes significant improvement in asymptotic complexity for server computation, compared to existing PIR-based systems.

D.4. Noteworthy Concerns

1) The paper does not solve the scenario when the long-term secret key material is compromised. They discuss the issue and consider it as a future work.
2) Similar to many other systems in the literature, Myco is also not resilient against DoS attacks and other potentially "undetectable" malicious behaviors by adversarial servers. However, they discuss the issue in the paper and discussed how such attacks do not impact the privacy of the honest users.