Anysphere: Private Communication in Practice

Security Whitepaper

Arvid Lunnemark Shengtong Zhang Sualeh Asif

{arvid,stzh1555,sualeh}@anysphere.co

July 6, 2022

Last updated: August 29, 2025

Abstract

We describe Anysphere, a metadata-private communication system deployed in the real world. Using private information retrieval based on homomorphic encryption, our system guarantees metadata privacy even if all of our servers are compromised and any number of the users and network observers are malicious. In this whitepaper, we precisely define our threat model, and show how we achieve security against it both in theory and in practice.

1 Introduction

Electronic communication runs the world. Yet, it is not as secure or private as the in-person communication it replaced. Advances such as end-to-end encryption are great for protecting *what* is being said, but information about who is talking to whom, how often and when — *the metadata* — is still being leaked at scale. Hackers, social media companies, and malicious nation states have open access to the metadata, even for individuals, organizations and governments that use the most secure communication platforms on the market.

What is needed is a new approach: every part of every conversation must be protected, including all metadata. The best way to protect something is to not give anyone access to it in the first place. Therefore, Anysphere operates on the principle of *no needless trust*. Even if all of our servers are compromised, everyone's communication history and pattern would still be secure.

This whitepaper explains how Anysphere works. Our code is open source and available at `github.com/anysphere/client`.

2 Security Context

In the early days of the internet, everything sent over it was public. If A sent a message to B, anyone on their path through the internet could read it. Email still operates in this model of no privacy. Today, there exist end-to-end encrypted alternatives, most notably the messaging app Signal. Unfortunately, if their servers are hacked, one of their employees bribed, or an ISP or a government targets you, the attacker can

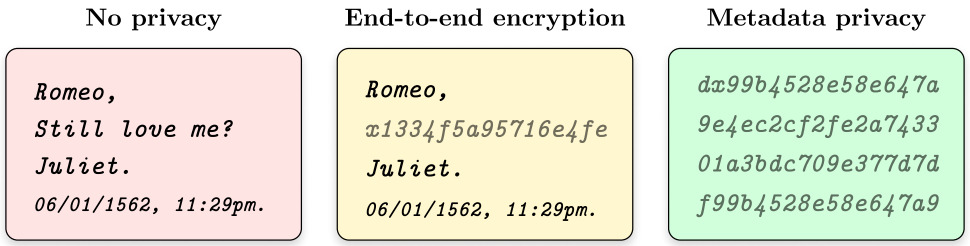

Figure 1: The view of a powerful adversary. With end-to-end encryption, the adversary can still see the metadata.

learn when, where and with whom you are talking. Anysphere guarantees metadata privacy, meaning that all information is hidden from everyone — just like an in-person conversation. Figure 1 illustrates the difference.

This section describes our desired properties and the threat model we want them to hold in. In short, we guarantee metadata privacy against everyone except the conversation partners themselves.

2.1 Goals

1. Metadata privacy. An attacker cannot learn anything about what messages are being sent when, between whom. This implies that the attacker cannot read messages (because it does not even know that the messages exist). We guarantee one of the strongest possible versions of metadata privacy; for more details, see Section 3.2.

2. Integrity. An attacker cannot forge a message from anyone, or edit any of the messages being sent.

3. Resistance to client-side denial-of-service attacks. An attacker that merely controls some number of users cannot block messages from being delivered. We do not guarantee service against an attacker that controls the server or the network.

4. Reasonable load on the client. Users are able to run a client on any commonly sold internet device. The client load should be adjustable.

2.2 Threat Model

No needless trust means that our threat model is as extensive as possible.

1. The attacker may compromise all servers. To achieve privacy, we do not put any trust in the server. Our threat model assumes a global adversary who has full control over all servers, and can observe and manipulate all network traffic. This is similar to other anonymous communication schemes based on private information retrieval (for example [Ahm+21], §2.2), but stronger than most other anonymous

communication schemes (e.g. Tor [DMS04] and Nym [Pio+17], which require partial trust in the servers).

2. The attacker may control the entire internet. See above.

3. The attacker may compromise strangers. We assume that the attacker has control over all clients that are not a contact of a given user, and can send maliciously crafted messages to and from these clients.

4. The attacker cannot compromise contacts. We assume that a user's contacts are trusted, and that the attacker does not have access to their computers. In [ALT18], Angel, Lazar and Tzialla describe an attack on a general metadata-private communication system, which shows how to leak metadata in the presence of a compromised contact. The amount of metadata leaked to contacts is small. In our security definition [LZA], we precisely define the leakage and show how to prevent it.

5. The attacker cannot access the user's computer. We assume that a user's local computer is trusted and is running a correct implementation of our system. In Section 5 we explain how we can relax this assumption slightly.

6. The attacker cannot break standard cryptography. Our threat model assumes the security of the standard cryptographic primitives we use. This includes Libsodium's AEAD implementation (XSalsa20) [Den13], and Microsoft SEAL's homomorphic encryption implementation (BFV) [MR22].

2.3 Non-goals

1. Not a cryptocurrency. Anysphere uses advanced cryptography, but not blockchains or cryptocurrencies.

2. Not a plugin to an existing ecosystem. Anysphere is not compatible with existing messaging systems like Signal or email. This is intentional: interfacing with legacy systems would mean accepting their (much lower) standards of security and privacy.

3. Not steganographic. Anysphere does not make an attempt to hide who is using our service.

3 Core Protocol

Consider two fictional users, Alice and Bob. Alice and Bob want to message each other over an untrusted network without leaking any data or metadata to anyone. They use authenticated end-to-end encryption to hide the message content and ensure integrity. They employ two key ideas to hide metadata: sending data at a constant rate and retrieving homomorphically compressed data.

When signing up, each user gets their own *outbox* on the server, which is a dedicated storage space for their messages. Once every minute, Alice sends exactly 1 KB of data

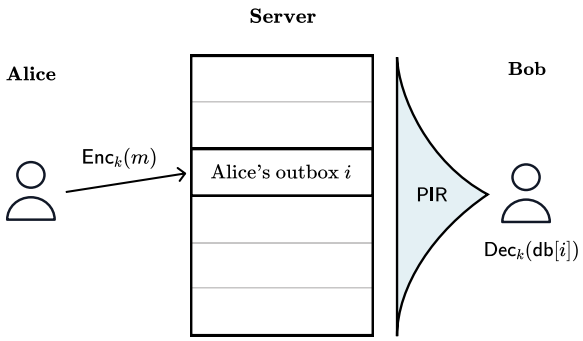

Figure 2: Alice sends the encryption of a message m to her outbox once every minute. Bob retrieves Alice's outbox using private information retrieval (PIR), which appears to everyone else as if he had downloaded any outbox.

to her outbox. If she has a message to send, she sends the padded encryption of that message. Otherwise, she sends a random sequence of bytes encrypted with a random key. This simple idea ensures that neither the server nor any network observers know when Alice sends a real message.

The message needs to be routed to Bob. In a traditional messaging system, Bob would download data from Alice's outbox and decrypt it. However, this leaks metadata: the server learns that Bob reads from Alice's outbox, linking the two together.

One solution is for Bob to, once every minute, download *all* outboxes from the server. He can then check the value of Alice's outbox locally. This way, it is impossible for the server to link Alice to Bob. All metadata is protected.

This is almost how Anysphere works. Obviously, Bob cannot download all outboxes every minute — that would be way too much data! — so instead he uses *private information retrieval*, a well-studied cryptographic primitive, as a way of compressing his download size. Figure 2 illustrates the core protocol. The following subsections describe the system in more detail. An exact description of the core protocol, together with rigorous security proofs, can be found in our security definition [LZA].

3.1 Private information retrieval

We view the outboxes as an array, called db. Bob wants to download Alice's outbox $\mathsf{db}[i]$ without revealing i to anyone. This problem was first introduced as *private information retrieval* (PIR) in 1995 [Cho+95], extended in 1997 to our threat model under the name cPIR [KO97], and has been extensively studied since then [Mel+16; Ang+18; Ahm+21].

All known cPIR schemes use homomorphic encryption [Gen10]. To compute the query q, Bob encrypts i with a homomorphic encryption scheme using a secret key s. That is, $q = \mathsf{HEnc}_s(i)$. The server can then homomorphically evaluate the function $f(i) = \mathsf{db}[i]$, producing the answer $a = \mathsf{HEnc}_s(\mathsf{db}[i])$. Bob can finally decrypt to

PROTOCOL 1: ANYSPHERE CORE PROTOCOL

Registration. Server allocates outbox i to Alice and generates authentication token tk. Alice receives (i, tk).

Trust establishment. Alice and Bob agree on a shared secret key k, using a protocol described in Section 4.

Sending. Exactly once every minute:

- If Alice has a queued message m: she sends $(i, \mathsf{tk}, \mathsf{Enc}_k(m))$ to the server, where k is the key shared with Bob.
- Otherwise: she sends $(i, \mathsf{tk}, \mathsf{Enc}_{k_r}(r))$ to the server, where r is a random sequence of bytes and k_r is a random key.

The server receives (i, tk, c) and stores c in outbox i if tk is correct.

Receiving. Exactly once every minute:

- Bob runs $\mathsf{PIR.Query}(i)$ to obtain a query q and a PIR secret sk. Bob sends q to the server.
- The server responds with $a = \mathsf{PIR.Answer}(\mathsf{db}, q)$ and Bob decodes it into $c = \mathsf{PIR.Decode}(\mathsf{sk}, a)$.
- Bob tries to decrypt $\mathsf{Dec}_k(c)$ using the shared secret key k.

Figure 3: The simplest version of our core protocol.

find $\mathsf{db}[i]$. In practice, $f(i)$ is often defined in terms of a dot product with a unit vector representing i, because BFV, the homomorphic scheme being used [FV12], is particularly good at dot products.

Our implementation currently uses FastPIR, one of the fastest cPIR schemes [Ahm+21]. All cPIR schemes have the same security properties, and we are actively researching faster schemes (see Section 7).

3.2 Security

The simplest version of our core protocol is shown in Figure 3. In this section, we show that Alice and Bob enjoy metadata privacy without having to trust anyone else.

We use the indistinguishability-based notion of metadata privacy defined in the extended version of Pung [AS16]. An adversary cannot distinguish between two different PIR queries, which means that the retrievals give the adversary no information. The sends also do not give the adversary any information, because we use a blockcipher for the symmetric encryption scheme, which is key-private. Hence, the only thing an adversary may learn is: the times when users are online, and the timestamps and contents of messages sent directly to compromised users. Given that we assume that the adversary does not control an honest user's contacts, the last case does not

reveal any information. In conclusion, the adversary learns nothing about the users' messages.

Going offline. Users will not always be connected to the internet. At night, most people put their computers to sleep. This means that users will not be sending and receiving exactly once every minute, which is the reason the adversary learns when a user is online. For security-critical use cases, we urge users to keep a regular transmission schedule by putting their computers to sleep at a regular time each day.

Authentication token. On registration, the server creates a unique authentication token for the new user. This allows the server to restrict access to that user's outbox, preventing denial-of-service attacks from other users. It should still be noted that, in accordance with our threat model, we do not prevent against denial-of-service attacks by ISPs or the server itself — fundamentally, a powerful actor can always shut down your internet access. In Section 7 we discuss plans for distributing the server such that, say, only 1 out of 3 servers need to be trusted to provide service.

3.3 Multiple contacts

The simple protocol in Figure 3 assumes that Alice has a single contact Bob. When Alice has multiple contacts, she picks a message from her queue and encrypts it with the correct key. When Bob has multiple contacts, he randomly picks a contact and retrieves their outbox that round.

This way of using the same outbox for all contacts may unfortunately leak some metadata about Alice and Bob's conversation to Alice's other contacts, as described in [ALT18]. Our threat model assumes that Alice trusts her contacts, which means that this does not compromise security for us. Nevertheless, for users that do not wish to trust their contacts, we show how to eliminate this leakage in [LZA].

In the future, we are also planning to implement probabilistic batch codes [Ang+18], which will make it possible for Alice and Bob to send and retrieve many messages at once.

3.4 Chunks and ACKs

If Alice wants to send a message longer than 1 KB, she splits the message into chunks and delivers the chunks to Bob in separate PIR transmissions. To ensure successful delivery, we took inspiration from TCP/IP. Each message from Alice to Bob is labeled with an integral message identifier m. Each chunk of message m is further labeled with a sequence number starting from 1. When Bob receives chunk c of message m, he sends Alice a short acknowledgement message $\text{ACK}(m, c)$ over a separate PIR database. Alice listens to Bob's ACK message using PIR, and sends chunk $c + 1$ of message m only after reading $\text{ACK}(m, c)$ from Bob. Figure 4 illustrates this.

Given the limited communication capacity, ACKs turn out to be slightly more subtle than this. In particular, if one user goes offline, we need to continue sending ACKs to

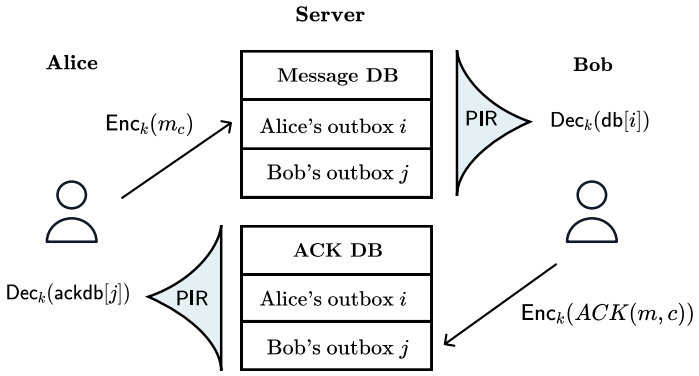

Figure 4: All server interactions for a message from Alice to Bob.

them, which could potentially block other ACKs and slow down message transmission. Because each ACK only needs to encode two integers (m, c), we can encode the ACKs for 20 contacts in one database row, avoiding this problem.

4 Trust Establishment

Our core protocol requires users to share a secret key. Creating this shared secret is generally referred to as the problem of trust establishment [Ung+15], and involves making sure that the person you talk to is who they say they are. We have an additional requirement that has only recently been explored by the research community [LT21]: the trust establishment procedure must not leak metadata.

We provide two different methods of trust establishment: one that completes instantly, but requires the two users to meet in person, and one that allows asynchronous invitations, but can take up to 24 hours. In both methods, we assume that Alice and Bob each have a Curve25519 key exchange key pair $\mathsf{kx} = (\mathsf{kx}^P, \mathsf{kx}^S)$, and aim to inform each other of their kx^P to derive a shared secret k. Alice and Bob additionally need to know each other's outbox indices.

4.1 In-person Trust Establishment

Alice and Bob establish trust in an in-person meeting, and do a simple key exchange where they each share their key-exchange public key kx^P and outbox index i out loud. Figure 5 explains the protocol in detail.

This trust establishment protocol needs no third party, which means that it completes instantly, cheaply, and securely. It does, however, require interaction from both parties at the same time. While it may seem prohibitive to set up a meeting and type manually, we believe that it is worth it for security-critical use cases.

PROTOCOL 2: IN-PERSON TRUST ESTABLISHMENT

Encode. Alice encodes her key exchange public key kx_A^P and her allocation index i_A into a human-readable story s_A. Bob similarly encodes his story s_B.

Share. Alice and Bob meet. They share their stories, and type each other's story into their own client.

Establish contact. Alice decodes Bob's story to obtain kx_B^P and i_B. She computes the shared secret $k = \mathsf{DH}(\mathsf{kx}_B^P, \mathsf{kx}_A^S)$, and adds i_B to her set of listening indices.

Figure 5: Establishing trust when the participants can talk to each other.

4.2 Asynchronous Invitations

Alice and Bob can also establish trust asynchronously. Here, we assume that Alice knows Bob's *public identifier*, id_B. For example, Bob may have posted his public identifier on a public platform like Twitter or his website. Alice can use this id_B to privately send an invitation, which Bob can later accept or reject.

To send the invitation, Alice encrypts her own id_A using Bob's public key and sends the result to the server. Bob then downloads the entire database of invitations, and finds the invitations meant for him by trying to decrypt each one of them. The full protocol is described in Figure 6.

This method achieves metadata privacy by hiding the timing of the invitation and the recipient of Alice's request. The client sends invitations on a fixed schedule, and for the public-key cryptography we use an ElGamal-based algorithm, which satisfies key-privacy [Bel+01].

This method offers trust establishment with convenience on par with existing messaging platforms, at the expense of increased bandwidth costs. A $\mathsf{db}_{\text{async}}$ entry requires ≈ 200 B of storage, which for 10 million users is 2 GB for the entire database. An individual user might wish to download the database only once or twice daily, which saves bandwidth but delays the detection of invitations. On the other hand, a company user could afford to download the database every few seconds, to ensure that incoming asynchronous invitations get detected almost instantly.

With an optimization, we can make detection even more accessible. Instead of downloading the whole database, the server streams a log of changes to $\mathsf{db}_{\text{sync}}$ to each user. If each user writes to $\mathsf{db}_{\text{sync}}$ every 20 minutes, then every second only 1/1200 of the 10 million users will change their entries, which corresponds to a change log of less than 2MB. Thus, any user with download bandwidth at least 2MB / s can detect incoming asynchronous invitations instantly.

The asynchronous method introduces more attack vectors. For example, an attacker can socially engineer their way to make Alice believe that a fake id_B belongs to Bob. Note that the identifier does not include Bob's name, which we believe reduces the

PROTOCOL 3: ASYNCHRONOUS TRUST ESTABLISHMENT

Send invitation. Alice initiates contact with Bob.

1. **Visibility.** Bob computes a "public id" $\mathsf{id}_B = (\mathsf{ki}_B^P, \mathsf{kx}_B^P, i_B)$, where ki_B^P is the invitation public key, kx_B^P is the key exchange public key, and i_B is the outbox index. Bob shares id_B publicly.
2. **Invitation.** Alice drafts a message m_{AB} for Bob and obtains the triple $(\mathsf{ki}_B^P, \mathsf{kx}_B^P, i_B)$ from Bob's profile. She periodically sends the pair $(i_A, c_{AB} = \mathsf{Enc}_{\mathsf{kx}_B^P}(\mathsf{id}_A | m_{AB}))$ to the server, which stores it in a database $\mathsf{db}_{\text{async}}$. When Alice has no invitations, she sends $(i_A, \mathsf{Enc}_{\mathsf{kx}^P}(\mathsf{id}_A | m))$, where kx^P and m are random.
3. **Invitation message.** Alice computes a shared secret $\mathsf{sk} = \mathsf{DH}(\mathsf{kx}_B^P, \mathsf{kx}_A^S)$ with Bob. She then sends a message inv_{AB} via our PIR transport layer, inviting Bob to join a conversation with her. She waits for Bob's ACK.

Retrieve invitation. Bob receives Alice's invitation.

1. **Database download.** Bob periodically downloads the entire $\mathsf{db}_{\text{async}}$.
2. **Find invitations.** Bob computes $\mathsf{Dec}_{\mathsf{kx}_B^S}(c)$ over all key-value pairs (i, c) in $\mathsf{db}_{\text{async}}$. If the decryption fails, Bob ignores this pair. If the decryption succeeds for the pair $(i, c) = (i_A, c_{AB})$, Bob decodes $i = i_A, \mathsf{id}_A, m_{AB}$, and records an invitation for him.
3. **Verify.** Bob needs to verify Alice's identity using id_A and m_{AB}, say by checking Alice's profile. Bob now chooses to accept or reject the invitation.

Accept invitation. If Bob accepts, he responds to Alice.

1. **Decode.** Bob decodes $\mathsf{id}_A = (\mathsf{ki}_A^P, \mathsf{kx}_A^P, i_A)$. He computes the shared secret $k = \mathsf{DH}(\mathsf{kx}_B^P, \mathsf{kx}_A^S)$ with Alice. He adds i_A to the list of indices he listens to.
2. **ACK.** Bob recieves the invitation message from Alice, inv_{AB}. Bob ACKs the message.
3. **ACK received.** Alice sees the ACK, finalizing the trust establishment.

Reject invitation. If Bob rejects, no further action is needed.

Figure 6: Establishing trust remotely. Contact can be initiated only by a single participant and requires acknowledgement from both participants.

potential of this attack, because it makes Alice more likely to verify the identifier before accepting it. An attacker could also monitor Alice's traffic to wherever Bob posts his public identifier. In conclusion, we provide the asynchronous method as a convenient choice, but we recommend that security-critical users use in-person invitations whenever possible. Still, to our knowledge, our asynchronous method is more private than any other asynchronous trust establishment that exists in practice.

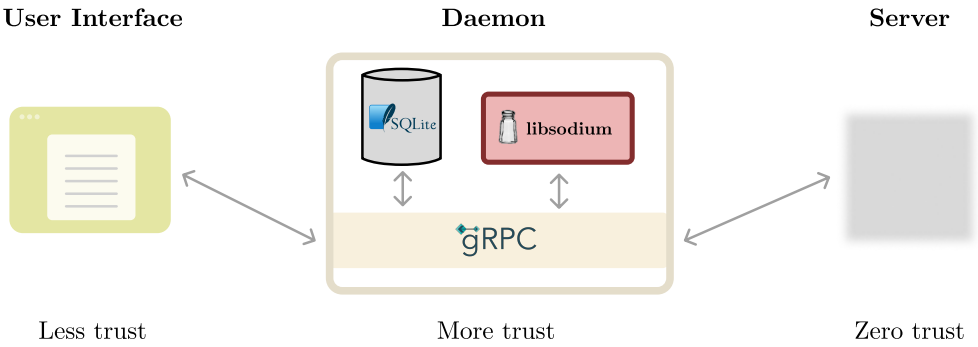

Figure 7: System architecture. The UI and the daemon run on your local computer and require some trust assumptions. The UI is cut off from the internet, and can only communicate via the daemon, meaning that it needs lower trust assumptions. The crypto module in the daemon requires the highest level of trust.

5 Practical Security

Our theoretical guarantees assume that the local computer is completely trusted. Nothing can defend against a preinstalled backdoor, so we can never eliminate client-side risk, but we *can* reduce it. This section outlines the mitigations we undertake: reducing the attack surface through modularization and supply chain protection, open-sourcing our client, securing code distribution and updates, and protecting against non-privileged local malware.

We emphasize that we can never eliminate client-side risk. **PLEASE DO NOT USE ANYSPHERE FOR SECURITY-CRITICAL USE CASES ON A COMPUTER YOU DO NOT TRUST**. This is especially true while we are in beta, and may have bugs.

5.1 Reducing the attack surface

As illustrated in Figure 7, our client consists of two parts: a UI frontend and a daemon backend. We sandbox the UI so that it is not allowed to talk to the internet. Instead, all communication goes through the daemon, which contains all security-critical code. Thus, bugs and malicious code on the frontend have a harder time compromising security.

We also reduce the attack surface of the daemon. In particular, we ensure a small dependency chain to prevent supply chain attacks. We chose to use C++ for all essential daemon code, depending only on a few well-known packages with long-term support and good security practices (Abseil, gRPC, SQLite, Libsodium). We wrote a Rust API for database interactions, being extremely careful with our dependency chain.

5.2 Open source

All code that is required for security is open source. The repositories `github.com/anysphere/client` and `github.com/anysphere/asphr` contain all code for our daemon and UI. Our server is not open source, since we guarantee security no matter what code is run on our server.

5.3 Securing code distribution and updates

Your computer needs to be running an unmodified copy of our code. For this reason, Anysphere is not a web app: serving code on the web can never be secure, because an attacker may at any point decide to serve you malicious code. Instead, Anysphere is a local app. You download it once, and can ensure that the correct code is downloaded by checking the signature and hash.

Local apps need to be updated. Currently, we use Electron's auto-updater to perform the signature checks for us. We plan to build our own update process, where signatures from two members of our team need to be present for the local app to accept the update.

5.4 Protection against non-privileged local malware

If you've granted root privileges to a malicious program on your computer, there is, unfortunately, nothing to be done. We can, nevertheless, reduce the risk of non-privileged malware. Our beta version does not protect against non-privileged malware, but in the future, we are planning to encrypt the local database, require a password to unlock the app, and use OS-level access controls to make sure only certain processes can access the daemon.

6 Related Research

The research community has studied metadata-private communication for decades. In 1981, David Chaum introduced *mix-nets*, which bounce messages between a small number of servers [Cha81]. Combined with onion encryption, mix-nets make it impossible to determine the destination of a given source packet, *assuming that at least one server is honest*. Using mix-nets, Tor was created in 2002 and became one of the most successful privacy-protecting real-world projects [DMS04]. Unfortunately, in addition to the server trust issue, mix-nets leak timing data, making it easy for someone with ISP-level network control to observe who is talking to whom. In today's world, it is getting easier and easier to amass enough data to perform such correlation attacks, making mix-net-based approaches unsuitable [Kar+21].

Systems with stronger metadata-privacy guarantees saw a flurry of interest in the last decade: Dissent and Riposte used so-called DC-nets [CGF10; CGBM15], Vuvuzela, Atom, Talek and many others introduced mix-nets with stronger security guarantees [VDH+15; Che+20; Kwo+17], Clarion, mcMix and Blinder introduced multi-party

computation techniques [Ale+17; EB21; APY20], and NIAR introduced function-hiding functional encryption [SW21; Bün+21]. All but NIAR's approach are less secure than our PIR-based approach, and NIAR is impractical at scale due to computation time. The PIR line of work, started by Angel with Pung [AS16; Ang+18] and continued by Addra [Ahm+21], promises both perfect security and reasonable scalability.

A communication system needs more than just message transmission. There has been much less attention in the literature to other components: establishing trust, managing many-to-many conversations, and defending against denial of service, to name a few. For trust establishment, Liu and Tromer recently initiated work on oblivious message retrieval [LT21], which is unfortunately not scalable enough for us, but a good start to a field we want to develop further.

7 What's Next?

Private communication is a hard problem. We are actively doing research to increase convenience while preserving metadata privacy, and we urge the research community to join us.

7.1 Implementation milestones

Anysphere is under active development. This is a list of some of the things we are working on.

Small files and images. We hope to support the transmission of small files and images through our current protocol.

Deleting messages. We plan to implement support for deleting messages, making sure that the messages are deleted forever with a ratcheting mechanism that guarantees forward secrecy.

Public-key infrastructure. We hope to set up a PKI to help the trust establishment process. That said, setting up a PKI is a notoriously hard problem requiring care.

Calls. We would like to facilitate calls using Anysphere without leaking metadata. However, calls using our current PIR setup require infeasible sub-second latency.

Hardened daemon against local malware. We aim to reduce the amount of trust we place in the client (see Section 5), by hardening the daemon.

7.2 Research problems

A lot of the problems we want to solve are both hard and unsolved. We're actively conducting research on these problems.

Denial-of-service resistance. While we fundamentally cannot prevent an internet provider from blocking access, we can reduce the trust assumptions when it comes to denial of service. For example, it would be great if we could distribute the servers, and provide a guarantee in the form of "the attacker needs to compromise at least k of n servers."

One-to-many and many-to-many conversations. We want to allow groups of people to broadcast messages to each other, without depending on the presence of any specific user. The question is how to do it scalably, while also embodying *no needless trust*.

More efficient trust establishment. We are actively thinking about improving our current methods of trust establishment to be faster, more reliable and less prone to social engineering attacks.

The ACK problem. Currently, if Alice wants to send a 2 KB message to Bob, she needs to wait for Bob to get online and ACK the first chunk before she can send the second chunk. If Alice's and Bob's time zones do not overlap, this can mean that an n KB message takes n days to get delivered.

The multiple contacts problem. In our current protocol, the transmission time for a message scales linearly with the number of contacts a user has. We also want to increase security by handling compromised contacts better.

Large files and images. Transmission of larger files and images is infeasible in our current framework because the number of chunks needed to deliver them is in the thousands.

References

[Ahm+21] Ishtiyaque Ahmad et al. "Addra: Metadata-private voice communication over fully untrusted infrastructure". In: *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 2021.

[Ale+17] Nikolaos Alexopoulos et al. "MCMix: Anonymous messaging via secure multiparty computation". In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1217–1234.

[ALT18] Sebastian Angel, David Lazar, and Ioanna Tzialla. "What's a little leakage between friends?" In: *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. 2018, pp. 104–108.

[Ang+18] Sebastian Angel et al. "PIR with compressed queries and amortized query processing". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 962–979.

[APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. "Blinder: MPC Based Scalable and Robust Anonymous Committed Broadcast." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 248.

[AS16] Sebastian Angel and Srinath Setty. "Unobservable communication over fully untrusted infrastructure". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 551–569.

[Bel+01] Mihir Bellare et al. "Key-privacy in public-key encryption". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2001, pp. 566–582.

[Bün+21] Benedikt Bünz et al. "Non-Interactive Differentially Anonymous Router". In: *Cryptology ePrint Archive* (2021).

[CGBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: An anonymous messaging system handling millions of users". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 321–338.

[CGF10] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: accountable anonymous group messaging". In: *Proceedings of the 17th ACM conference on Computer and communications security*. 2010, pp. 340–350.

[Cha81] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[Che+20] Raymond Cheng et al. "Talek: Private group messaging with hidden access patterns". In: *Annual Computer Security Applications Conference*. 2020, pp. 84–99.

[Cho+95] Benny Chor et al. "Private information retrieval". In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE. 1995, pp. 41–50.

[Den13] Frank Denis. *The Sodium cryptography library*. June 2013. URL: `https://download.libsodium.org/doc/`.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[EB21] Saba Eskandarian and Dan Boneh. "Clarion: Anonymous Communication from Multiparty Shuffling Protocols". In: *Cryptology ePrint Archive* (2021).

[FV12] Junfeng Fan and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption". In: *Cryptology ePrint Archive* (2012).

[Gen10] Craig Gentry. "Computing arbitrary functions of encrypted data". In: *Communications of the ACM* 53.3 (2010), pp. 97–105.

[Kar+21] Ishan Karunanayake et al. "De-anonymisation attacks on Tor: A Survey". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2324–2350.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is not needed: Single database, computationally-private information retrieval". In: *Proceedings 38th annual symposium on foundations of computer science*. IEEE. 1997, pp. 364–373.

[Kwo+17] Albert Kwon et al. “Atom: Horizontally scaling strong anonymity”. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 406–422.

[LT21] Zeyu Liu and Eran Tromer. “Oblivious Message Retrieval”. In: *Cryptology ePrint Archive* (2021).

[LZA] Arvid Lunnemark, Shengtong Zhang, and Sualeh Asif. “Formal Security Definition of Anysphere”. In preparation.

[Mel+16] Carlos Aguilar Melchor et al. “XPIR: Private information retrieval for everyone”. In: *Proceedings on Privacy Enhancing Technologies* (2016), pp. 155–174.

[MR22] WA. Microsoft Research Redmond. *Microsoft SEAL (release 4.0)*. `https://github.com/Microsoft/SEAL`. Mar. 2022.

[Pio+17] Ania M Piotrowska et al. “The loopix anonymity system”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1199–1216.

[SW21] Elaine Shi and Ke Wu. “Non-Interactive Anonymous Router”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pp. 489–520.

[Ung+15] Nik Unger et al. “SoK: secure messaging”. In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 232–249.

[VDH+15] Jelle Van Den Hooff et al. “Vuvuzela: Scalable private messaging resistant to traffic analysis”. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. 2015, pp. 137–152.