Anysphere: Private Communication in Practice

Security Whitepaper

Arvid Lunnemark Shengtong Zhang Sualeh Asif
{arvid,stzh1555,sualeh}@anysphere.co

June 30, 2022

Last updated: July 1, 2022

ABSTRACT

We describe Anysphere, a metadata-private communication system deployed in the real world. By using private information retrieval based on homomorphic encryption, our protocol guarantees security 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.

CONTENTS

hi

1 MOTIVATION

When the internet was first established, everything sent over it was public. If A sent a message to B, anyone on their path through the internet could see that such a message was sent, as well as read the actual message. As of today, many messaging services are end-to-end encrypted, meaning that no one can read the contents of messages. However, for sufficiently powerful adversaries — hackers, ISPs, government agencies — it is still possible to find out who is talking to whom, as well as when and how often messages are sent. Our goal is to hide this metadata: we want to create a system where A and B can send messages to each other over an untrusted network, without anyone knowing that they talk to each other. Such a private communication system would be critically important to protect and expand freedom in the world [Lun21].

2 THREAT MODEL

[**TODO**: add table comparing our threat model with e.g. signal and email] [**TODO**: add an illustration of a walled garden, with the walls containing only your computer and your friends' computers]

[**TODO**: Figure out a better format here. See e.g. Skiff's whitepaper.]

Similar to most PIR schemes(for example [Ahm+21], §2.2), our threat model assumes a global adversary who can compromise the entire communication infrastructure except for the user's and their friends' client end. In particular, we assume the adversary has control over all the servers, and can observe and manipulate all network traffic.

End-user trust is more subtle matter. In [ALT18], Angle, Lazar and Tzialla describes the compromised friend(CF) attack on a general meta-data private messaging system, which shows that perfectly hiding metadata while not trusting the user's friends is computationally prohibitive. In our security model, a user trusts that the devices of themselves and all their friends are uncompromised and running an unmodified copy of anysphere's client-side code. The user assumes that any other end-user device might be compromised.

[**TODO**: Can we assume that only a small number of friends are compromised?]

Finally, we assume the security of the standard cryptography primitives we use, including microsoft SEAL's BFV cryptosystem and libsodium's AEAD cryptosystem.

Denial of Service(DoS) attacks are unavoidable if the adversary controls all our servers. In the case of such attacks, we do not guarantee

liveliness of our service, but continue to guarantee metadata security. We also defend against DoS attacks launched by an end-user with no access to the servers.

2.1 Desired Properties

1. **Perfect Metadata Protection**: All contents and metadatas associated with a conversation are only visible to the users involved with the conversation. Even with a compromised server, an adversary should not even be able to find out whether two users are engaging in any conversation.

2. **Forward Secrecy**: [**TODO**: do we provide this?]

3. **Resistant to man-in-the-middle attacks**.

4. **Resistant to social engineering attacks**.

[**TODO**: add more.]

3 CORE PROTOCOL

Alice wants to message Bob over an untrusted network, without leaking any data or metadata to anyone. To hide the message content they just use end-to-end encryption. To hide metadata they employ two key ideas: sending data at a constant rate, and retrieving homomorphically compressed data.

When signing up, each user gets their own *outbox* on the server. This outbox is a dedicated storage space that the user sends messages to. Once every minute, Alice will send exactly 1 KB of data to her outbox on the server. If she has a message to send, she sends the encryption of that message, and if she has no message to send, she sends a random sequence of bytes. With this simple first idea, no one, including the server and any network observers, will know when Alice actually sends a message.

The message needs to be routed to Bob. Now, a traditional messaging system would have Bob download data from Alice's outbox, and then try to decrypt it to see if it was meant for him. But this leaks metadata: the server would know that Alice wrote to outbox x and that Bob read from outbox x, which links the two of them together!

Our solution is for Bob to, once every minute, download *all* outboxes from the server. On his own computer, he can then check Alice's outbox. This way, no one, not even the server, has any way of linking Alice to Bob. All metadata is protected.

This is 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. The following subsections will describe the system in detail.

[**TODO**: Figure: Alice, the server outboxes, and bob on the other side downloading the entire database.]

3.1 Private information retrieval

Bob wants to download outbox i without revealing i to anyone. Viewing the collection of outboxes as a database array db, he wants to retrieve $\text{db}[i]$ privately. 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].

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

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: $q = \text{HEnc}_s(i)$. The server can then homomorphically evaluate the function $f(i) = \text{db}[i]$, producing the answer $a = \text{HEnc}_s(\text{db}[i])$. Bob can finally decrypt to find $\text{db}[i]$. In practice, $f(i)$ is often defined in terms of a dot product with a unit vector representing i, because the homomorphic scheme being used, BFV [FV12], is particularly good at dot products.

3.2 Security proof

The simplest version of our core protocol is shown in Figure 1. In this section, we prove: (1) that Alice and Bob enjoy complete metadata-privacy without having to trust anyone else, and (2) that our protocol is resistant to denial of service attacks from users.

In this section, we formally state and prove metadata security.

[**TODO**: simulation security definition, going offline, friend attack, key privacy because prf.]

3.2.1 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. [**TODO**: how much information does this leak?]

3.2.2 Authentication token. On registration, the server creates a unique authentication token for a 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 from before assumed Alice had a single contact Bob. Our system allows many more contacts.

[**TODO**: link the compromised friend attack]

3.4 Chunks and acknowledgements

If Alice wants to send a message longer than 1 KB, she needs to chunk the message up. Here, we have taken inspiration from TCP/IP. We require a message to receive an acknowledgement, shortened ACK, before we send the next message in the sequence.

[**TODO**: describe the separate PIR table for ACKs.]

[**TODO**: Figure: pseudocode for Register, Send, Retrieve (with everything)]

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

Trust establishment. Alice and Bob agree on a shared secret key k. Details in Section 4.

Sending. Exactly once every minute:

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

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

Receiving. Exactly once every minute:

- Bob sends $q = \text{PIR.Query}(i)$ to the server.
- The server responds with $a = \text{PIR.Answer}(\text{DB}, q)$ and Bob decodes it into $c = \text{PIR.Decode}(a)$.
- Try to decrypt $\text{Dec}_k(c)$ where k is the key shared with Alice.

Figure 1: The simplest version of our core protocol.

4 TRUST ESTABLISHMENT

Most existing metadata private messaging systems, such as Pung or Addra, assumes a prior key exchange between users. In our messaging application, we need a metadata-secure mechanism for the key exchange itself. In other words, if Alice knows the public key pk_B of Bob, then Alice should be able to send an "invitation" to Bob. Bob must be able to retrieve this invitation from the server, and complete a key exchange with Alice. We call this process "trust establishment".

This problem is known as Oblivious Message Detection(OMD) in [LT21]. The scheme proposed in [LT21] aims to minimize user download size, but it costs each user $1 per million messages scanned, which is prohibitive for our messaging application. We provide two alternate methods of trust establishment with better computational cost and security.

In the following two methods, we assume that Alice and Bob's clients have generated a Curve25519 key exchange keypair $kx = (kx^P, kx^S)$.

4.1 Face-to-face Invitations

Our first method assumes that Alice and Bob are able to set up a face-to-face meeting with each other, either in person or over zoom. The trust establishment process is a simple key exchange implemented as follows.

1. 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.

2. Alice and Bob meets and types the other's story into their own client.

3. Alice decodes Bob's story to obtain kx_B^P and i_B. She computes the shared secret $sk = DH(kx_B^P, kx_A^S)$, and adds i_B to her set of listening indices.[**TODO**: Better name?] Bob does the same. They can now communicate to each other using our PIR transport layer.

Using this method, all interactions between the users do not require the internet. Thus, trust establishment can be completed instantly, cheaply, and securely. Our system do require Alice and Bob to be able to set up a meeting and type each other's story manually. To justify this approach, we believe a privacy-conscious Alice would be willing to set up a face-to-face meeting with Bob before sending him sensitive information.

4.2 Asynchronous Invitation

[**TODO**: mention key privacy]

Our second method targets the case when Bob does not know about Alice's invitation beforehand. For example, Alice can be a sensitive client who wishes to privately reach out to Bob's company.

This method proceeds in three steps. First, Alice sends an encrypted invitation to the server. Second, Bob retrieves this invitation via a full database download. Third, Bob informs Alice of his acceptance via an ACK message.

To send an invitation

1. Upon registration, each user's daemon generates an additional invitation keypair $ki = (ki^P, ki^S)$. Bob's daemon computes a "public id" id_B, which contains his invitation public key ki_B^P, key exchange public key kx_B^P, and allocation i_B encoded in plaintext. It then displays the id on Bob's GUI. Bob posts id_B on his public profile, such as on twitter or on his company's website.

3. When Alice wishes to send an invitation to Bob, she obtains id_B from Bob's public profile, and decodes it to obtain (i_B, kx_B^P, ki_B^P). She drafts an initial message m_{AB} to accompany her invitation.

4. Alice's daemon periodically sends the key-value pair $(i_A, c_{AB} = \text{Enc}(kx_B^P, id_A | m_{AB}))$ to the server[1], which stores it in a separate **AsyncInvitationDatabase**. When Alice has no invitations, her daemon sends $(i_A, \text{Enc}(kx^P, id_A))$ for a random public key kx^P.

4. As Alice's daemon sends the invitation, it also compute the shared secret with Bob $sk = DH(kx_B^P, kx_A^S)$. It sends Bob a "control message" ctm_{AB} via our PIR transport layer, and listens for Bob's ACK.

To retrieve an invitation

1. Bob's daemon periodically downloads the entire **AsyncInvitationDatabase**. It computes $\text{Dec}(kx_B^S, c)$ over all key-value pairs (i, c). If the decryption fails, Bob's daemon ignores this pair. Now suppose $(i, c) = (i_A, c_{AB})$, and the decryption succeeds. Bob's daemon decodes $i = i_A, id_A, m_{AB}$ from the decrypted data, and displays on Bob's GUI that he received an incoming invitation from id_A with message m_{AB}.

2. Bob verifies Alice's identity using id_A and m_{AB}, for example by checking Alice's public profile. Bob then chooses to either accept or reject the invitation. If Bob rejects the invitation, no further action is performed.

To accept an invitation

[1] It is important to redo this encryption each round, otherwise adversaries will observe the same message repeatedly

1. If Bob accepts Alice's invitation, Bob's daemon decodes id_A to obtain kx_A^P, and computes the shared secret $sk = DH(kx_A^P, kx_B^S)$. It adds i_A to the set of listening indices.

2. Since Alice is sending the control message ctm_{AB} to Bob using the same shared secret sk, Bob's daemon will read ctm_{AB} from the PIR database. It sends an ACK to Alice's control message.

3. When Alice's daemon reads Bob's ACK to ctm_{AB} from the PIR ACK database, it displays on Alice's GUI that Bob has accepted Alice's invitation. Alice and Bob can now communicate to each other using our PIR transport layer.

This method achieves metadata security. We hide the timing of the invitation by making the daemon send to and download **AsyncInvitationDatabase** on a fixed schedule. We hide the recipient of Alice's request since our encryption scheme is key private([**TODO**: Cite arvid's section]).

This method offers convenient trust establishment on par with most existing messaging platforms. Its disadvantage is that downloading the entire database is expensive and time-consuming for user B. We estimate that a key-value pair in the database take approximately 200B. If we have 1 million users, then the whole database will have a size of approximately 200MB. Thus, a private user might wish to only download the database once or twice each day, which saves bandwidth but delays the detection of invitations. On the other hand, a company might be able to afford downloading the database every few seconds, which costs download bandwidth but can ensure incoming asynchronous invitations get detected almost instantly.

5 CLIENT-SIDE SECURITY IN PRACTICE

Our theoretical threat model assumes that the user's local computer is completely trusted, and that it is running a correct implementation of our protocol. If your computer is compromised, or you are running a buggy or intentionally incorrect version of our code, none of our previously outlined theoretical guarantees will apply. This is inherent and unavoidable — no matter the fancy encryption schemes you come up with, nothing will help you if your computer comes with a preinstalled backdoor. While **we fundamentally cannot eliminate the client-side risk**, we can *reduce* it, which we will do in this section.

5.1 Reducing the attack surface

The first step in mitigating security is to reduce the attack surface. To do so, we architected our client to consist of two parts: a UI frontend and a daemon backend, where the daemon backend contains all security-critical code. We sandbox the UI frontend in such a way that it is not allowed to talk to the internet, and let all message sending go through the daemon, which handles the cryptography. That way, even if there are bugs in the UI frontend, or potentially malicious code, there is not much it can do.

[**TODO**: Figure of client architecture with UI and daemon, showing how we cut off internet access.]

We also reduce the attack surface of the daemon itself. In particular, we use C++ instead of other popular languages (Rust, Go, Python), because all other practical languages are significantly more susceptible to supply chain attacks. Our daemon has 4 direct dependencies (Abseil, gRPC, SQLite, Libsodium) and 0 transitive dependencies. A comparable implementation in a language with a package manager would easily use 100s of transitive dependencies. We elaborate more on our choice of C++ in this blog post [**TODO**: Link.].

5.2 Code distribution

We sign everything.

Maybe we sign everything twice?

Maybe we have a cold-storage and a hot-storage signing key?

Do we store a backup key in cold storage that we can use to revoke a version? And people can disi

[**TODO**: Either understand whether standard OS signing is good enough, or whether we should sign things ourselves.]

5.3 Updates

Every update needs to be signed. In fact, it needs to be signed twice: Arvid holds one key, and Sualeh holds one key. Both signatures must be present for the local client to accept the update.

We implement our own signature check. Many popular frameworks have built-in signature checks, such as AutoUpdater for Electron and Updater for Tauri, but to ensure that we are really certain that updates work the way we want them to, we do it ourselves.

This means that if either of us loses our private key, you would not get any updates. This is by design.

5.4 Protecting against non-privileged local malware

If you've granted administrator access to a malicious program on your computer, there is, unfortunately, nothing to be done. We can, nevertheless, reduce the risk of non-privileged malware.

[**TODO**: Actually implement: allow to encrypt the database, in which case the both the GUI and the CLI need to require passwords (and the GUI may cache the password for some amount of time).]

Again, we do not aim to eliminate the risk here. Non-privileged malware may still gather information from side-channel attacks, and potentially other avenues. Once an attacker has access to your computer, it is very, very hard to shield yourself from them.

6 RELATED RESEARCH

Metadata-private communication has been studied for decades. In 1981, David Chaum introduced so-called mix-nets, which bounce messages between a small number of servers. Using onion encryption, if at least one of the servers is honest, it is impossible to determine the destination of a given source packet. Tor, created in 2002, is one of the most successful privacy-protecting real-world projects, and uses mix-nets [DMS04]. Unfortunately, even aside from the server trust issue, mix-nets leak timing data which makes it easy for someone with ISP-level control of the network 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 for real security [Kar+21].

The 2010s included a flurry of research papers trying out a few different methods of achieving scalable metadata-privacy: so-called DC-nets were tried by Dissent and Riposte [CGF10; CGBM15], mix-nets with stronger security guarantees were tried by Vuvuzela, Atom, Talek and many others [VDH+15; Che+20; Kwo+17], multi-party computation techniques were tried by Clarion, mcMix and Blinder [Ale+17; EB21; APY20], and function-hiding functional encryption was tried by NIAR [SW21; Bün+21]. These approaches are either less secure than the PIR-based approach we are using (all but NIAR), or are impractical at scale due to computation time (NIAR). The PIR line of work, started by Angel with Pung [AS16; Ang+18] and continued by Addra [Ahm+21], is the only one that promises both perfect security and reasonable scalability.

While there has been a lot of research focusing on the theoretical problem of message transmission, there has been less attention on everything else that needs to exist for a communication system to be useful in practice: initiating connections, handling arbitrary failures, and distributing code securely, to name a few. We draw on the knowledge of the past for message transmission and invent novel protocols for the rest, some of which may be of interest to the research community.

7 FUTURE WORK

8 OTHER COMMUNICATION PLATFORMS

Privacy and security are important, and there are many previous solutions.

Signal is great. It is end-to-end encrypted, open source, and run by a trustworthy group. Unfortunately, if their servers are hacked, one of their employees bribed, or you are simply attacked by a network-level powerful actor, there is nothing Signal can guarantee.

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.

[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.

[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.

[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).

[Lun21] Arvid Lunnemark. *It Is Time to Move Beyond End-to-End Encryption*. 2021. URL: https://anysphere.co/it-is-time-to-move-beyond-end-to-end-encryption/ (visited on 06/16/2022).

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

[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.

[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.