Anysphere: Private Communication in Practice

Security Whitepaper

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

June 30, 2022

Last updated: July 3, 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

1 Introduction

The world runs on electronic communication. Email, instant messaging and video calls form the fundament to our professional and social lives. Yet, all these forms of electronic communication are not as secure or private as the in-person communication they 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 your conversation must be protected, including the metadata. The best way to protect your data is to not give anyone(other than your intended recipient) access to it in the first place — some may even argue it is the only way. Therefore, we operate on the principle of *no needless trust*. We could publish the passwords to our servers online, and your communication history and pattern would still only be visible to you and the people you talked to. This whitepaper accompanies our

source code at github.com/anysphere/client, and aims to prove the security of Anysphere.

2 Security Context

[**TODO**: Explain why Signal is not enough. Good table.]

In this section, we describe what our system achieves beyond end-to-end encrypted messaging applications like Signal and Wickr.

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. **[I don't think so. The point of E2E encryption is to guarantee content security when the server gets compromised.** —STZH] xxx

Furthermore, even with a secure server, it leaks metadata. Using timing attacks, an adversary can figure out when, where and with whom you are talking.

2.1 Desired Properties

1. **Metadata Protection**: All contents and metadata of a conversation are only visible to the users involved in the conversation. Even with all servers compromised, an adversary should not be able to find out whether two users are engaging in conversation.

An especially challenging case is metadata protection during trust establishment. This happens when two users wish to add each other as friends for the first time. This feature is not supported even by Addra or Pung. We outline our solution in Section 4, which offers users a variety of options to establishing trust.

2. **Resistance to MITM attacks**: We use PIR and end-to-end encryption to ensure no man-in-the-middle(MITM) can access metadata associated with any conversation of the user.

3. **Resistance to DoS attacks**: Denial of Service(DoS) attacks are unavoidable if the adversary controls all our servers. In the case of a server-controlling adversary, 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.

4. **Reasonable Load on Client and Server**: Since PIR is expensive, we design our system to minimize load on the client and server without compromising security. We use the most efficient PIR algorithms known on the server. We also give users the option to adjust client side load to suit their upload and download bandwidth.

Non-E2E	**E2E**	**Metadata Hiding**
Bob, Remember 2.1.1978? Alice. 06.01.2022, 10:30 PM.	Bob, xxxxxxxxxxxxxxxxx Alice. 06.01.2022, 10:30 PM.	xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx

Figure 1: What a message Alice sent Bob looks like for a powerful adversary controlling all servers, under non-E2E, E2E, and Metadata-Hiding protocols.

2.2 Threat Model

[add an illustration of a walled garden, with the walls containing only your computer and your friends' computers] xxx

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

1. **Compromised Server**: We do not have any trust in the server. Similar to most PIR schemes(for example [Ahm+21], §2.2), our threat model assumes a global adversary who has control over all the servers, and can observe and manipulate all network traffic.

2. **Compromised Strangers**: Similarly, we assume that the adversary has control over all non-friend clients, and can observe and manipulate all network traffic from and to these clients.

3. **Trusted Friends**: 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 threat model, a user trusts that their devices, as well as the devices of all their friends, are uncompromised and running our client code. **[In the future, we would look into measures for handling friend compromises? —STZH]** xxx

3. **Secure Cryptographical Primitives**: Our threat model assumes the security of the standard cryptography primitives we use, including Microsoft SEAL's BFV cryptosystem and libsodium's AEAD cryptosystem.

2.3 Non-goals

1. **Not a cryptocurrency**: Anysphere's underlying PIR algorithm is unrelated to blockchains and cryptocurrencies. **[this is, unfortunately, what most people think of when they hear "private communication". —STZH]** xxx

2. **Not a plugin**: Anysphere relies on a dedicated PIR server to work. Therefore, Anysphere is not compatible with existing messaging systems like Messenger or Signal, and does not work like a google chrome extension to these apps.

[**My mom asked me this...** —STZH] xxx

3. **Not steganographic**: Anysphere does not make an attempt to hide who's using our service. [**From Tor. Is it ok?** —STZH] [**I like both of these.** —SUALEH] xxx xxx

3 Core Protocol

Consider two fictional people Alice and Bob. 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 padded encryption of that message, and if she has no message to send, she sends a random sequence of bytes encrypted with a random key. 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 i and that Bob read from outbox i, 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. Figure 2 illustrates the core protocol, and the following subsections describe the system in detail.

3.1 Private information retrieval

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

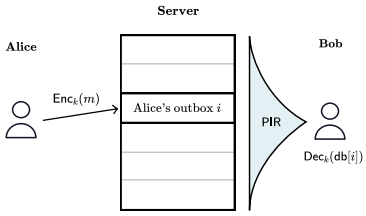

Figure 2: Alice sends to her outbox once every minute. Bob retrieves Alice's outbox using private information retrieval (PIR), which looks the same to everyone else as if he had downloaded the entire database. Alice and Bob can use standard symmetric encryption to communicate, and no one, not even the server, will learn anything at all.

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 find $\mathsf{db}[i]$. 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.

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).

3.2 Security

The simplest version of our core protocol is shown in Figure 3. In this section, we show (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.

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. [**how much information does this leak?**] xxx [**However, as long as users make their connection schedule regular(e.g. transmits from 9AM to 5PM each day), no metadata is leaked.** —STZH] xxx

3.2.2 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

Protocol 1: Anysphere Core Protocol

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

Figure 3: The simplest version of our core protocol.

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. [**link the compromised friend attack**] xxx

3.4 Chunks and ACKs

If Alice wants to send a message longer than 1 KB, she needs to separate the message into 1KB chunks and deliver all 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 a message identifier i. Each chunk of message i is further labeled with a sequence number starting from 1. When Bob receives chunk c of message i, he sends Alice a short acknowledgement(ACK) message $ACK(i, c)$ over a separate ACK PIR table. Alice listens to Bob's ACK

message using PIR, and deposits chunk $c + 1$ of message i only after reading $ACK(c)$ from Bob.

Each ACK message only needs to encode two integers i, c. Thus, reading ACKs represents a much lighter load compared to the main message database.

[**describe the separate PIR table for ACKs.**] xxx

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

Protocol 2: In-Person Trust Establishment

Encode. Alice encodes the plaintext of 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 $sk = \mathsf{DH}(kx_B^P, kx_A^S)$, and adds i_B to her set of listening indices.

Figure 4: Anysphere's protocol for establishing trust when the participants can talk to each other.

4 Trust Establishment

The Anysphere Core protocol's first phase requires participants to make contact. "Trust Establishment", [Ung+15], is a notoriously difficult problem, where users exchange long-term keys and ensure they're talking to their intended contacts. For Anysphere to always be metadata-secure, we need a metadata-secure trust establishment procedure.

In practice, Alice needs to find Bob's public key and use it to send an "Invitation" to Bob. Bob must be able to retrieve this invitation and complete a key exchange with Alice. Most importantly, the server should not learn any metadata of this exchange.

We provide two alternate methods of trust establishment designed for our users that meet our desired security profile and don't have prohibitive computational costs. The first of our methods is for security-conscious users, and the other allows the advantage of remote participants. In the following two methods, we assume Alice and Bob have a Curve25519 key exchange key pair $kx = (kx^P, kx^S)$ and aim to inform each other of their kx^P to derive a shared secret sk. [**Handle the part about why we do not include the username in the publicID.** —Sualeh] xxx

4.1 Face-to-face Invitations

Alice and Bob establish trust in a face-to-face meeting, either in-person or over zoom, and do a simple key exchange, as described in Figure 4. They share the encoding of their keys as an English story of words. All interactions between the users pass through no third party. Thus, trust establishment completes instantly, cheaply, and securely. It may seem prohibitive to set up a meeting and type manually, but we believe a privacy-conscious Alice would be willing to set up a meeting with Bob before sending him sensitive information.

Protocol 3: Remote Trust Establishment

Send Invitation. Alice initiates contact with Bob.

1. **Visibility.** Bob computes a "public id" $id_B = (ki_B^P, kx_B^P, i_B)$, (where ki_B^P is the invitation public key, kx_B^P is the key exchange public key, and i_B is the allocation). Bob posts id_B on his public profile, such as on twitter.
2. **Invitation.** Alice drafts a message m_{AB} for Bob and obtains the triple (ki_B^P, kx_B^P, i_B) from Bob's profile. She periodically sends the pair $(i_A, c_{AB} = \mathsf{Enc}(kx_B^P, id_A | m_{AB}))$ to the server, which stores it in a separate **AsyncInvitationDatabase**. When Alice has no invitations, her daemon sends $(i_A, \mathsf{Enc}(kx^P, id_A))$ for a random public key kx^P.
3. **Invitation Message.** Alice computes a shared secret $sk = \mathsf{DH}(kx_B^P, kx_A^S)$ with Bob. She then sends a message inv_{AB} via our PIR transport layer, inviting Bob to join a conversation with her. Now, she listens for Bob's ACK.

Retrieve Invitation. Bob receives Alice's invitation.

1. **Database Download.** Bob periodically downloads the entire **AsyncInvitationDatabase**.
2. **Find Invitations.** Bob computes $\mathsf{Dec}(kx_B^S, c)$ over all key-value pairs (i, c). If the decryption fails, Bob's daemon ignores this pair. If the decryption succeeds for the pair $(i, c) = (i_A, c_{AB})$, Bob decodes $i = i_A, id_A, m_{AB}$, and notices an invitation for him with message m_{AB}, and id_A.
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 Alice's invitation, he needs to respond to Alice.

1. **Decode.** Bob decodes $id_A = (ki_A^P, kx_A^P, i_A)$. He computes the shared secret $sk = \mathsf{DH}(kx_B^P, kx_A^S)$ 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. Alice sees the ACK from Bob and now they can communicate over the PIR transport layer.

Reject Invitation. If Bob rejects the invitation, no further action needs to be performed.

Figure 5: Anysphere's protocol for establishing trust when the participants are remote. Contact can be initiated only by a single participant and requires acknowledgement from both participants to establish successful contact

4.2 Asynchronous Invitations

Alice and Bob can also establish trust asynchronously. Alice privately reaches out to Bob (possibly an organization) and wishes to send her invitation privately. She sends an encrypted invitation without indicating the recipient. Bob retrieves this invitation via a full database download and informs Alice of his acceptance via an ACK message, as described in detail in Figure 5.

To send an invitation, Alice learns Bob's information through a public platform like Twitter or his website. Alice quickly imports Bob's keys and sends two messages: one is an invitation through the Invitation Database for Bob to find her information, and a second message through the PIR transport layer for Bob to acknowledge.

To retrieve an invitation, Bob periodically downloads the entire Async Invitation Database and notices if there are invitations for him. The user(s) representing Bob's account choose whether to respond to the message, as would be the case in a traditional e-mail. In the affirmative case, they acknowledge Alice's message and are set for the Core Protocol. No further action is required otherwise.

This method achieves metadata security by hiding the timing of the invitation and the recipient of Alice's request. Daemons send invitations on a fixed schedule, and we ensure that our encryption scheme is key private([**Arvid which section should we cite** —SUALEH]). ([**Actually, can't we achieve instant friend requests by continuously downloading a log of changes to the DB instead of the DB itself, and using the Poisson upload frequency idea?** —STZH1555]) xxx xxx

This method offers trust establishment on par with existing messaging platforms at the expense of increased storage costs. A **AsyncInvitationDatabase** entry requires $\approx$ 200B of storage, which for the $10M$ users, is just 2GB for the whole 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 second and can ensure incoming asynchronous invitations get detected almost instantly.

The remote method does introduce more attack vectors. For example, an attacker socially engineers Alice to make her believe that a fake id_B belongs to Bob or monitors A's traffic to B's public profile. Therefore, we recommend that privacy-conscious users use face-to-face invitations whenever possible.

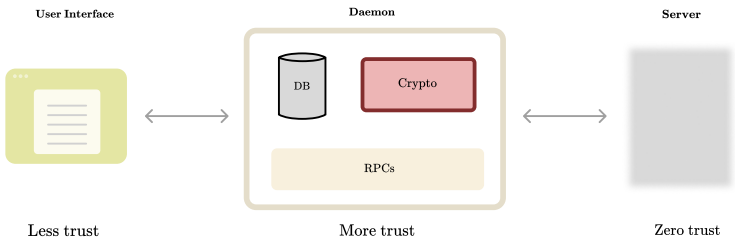

Figure 6: System architecture. The UI and the daemon run on your local computer and require some trust assumptions. The UI is cut off from the internet, and can only communicate via the daemon, meaning that it needs lower trust assumptions. The crypto module in the daemon requires the highest level of trust, and uses the widely trusted Libsodium library.

5 Practical Security

The theoretical guarantees in the previous sections — metadata security without needing to trust the server or the network — all assume that the local computer is wholly trusted. As no fancy encryption scheme prevents a pre-installed backdoor, we cannot eliminate the client-side risk. Nevertheless, we can reduce it. This section outlines the mitigations we undertake: reducing the attack surface by modularizing and supply chain protection, eliminating bugs by being open source, securing code distribution and updates, and protecting against non-privileged local malware.

We cannot achieve perfect security without our user's efforts. **PLEASE DO NOT USE ANYSPHERE FOR SECURITY-CRITICAL USE CASES ON A COMPUTER YOU DO NOT TRUST**. Our request is especially true while we are in beta, and may have bugs.

5.1 Reducing the attack surface: modularity and supply chain risk

As illustrated in Figure 6, 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 messaging goes through the daemon, which contains all security-critical code. Thus, bugs and malicious code on the front-end cannot compromise 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 Eliminating bugs: 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 — everytime you visit a website, you give attackers an opportunity to serve you malicious code. Instead, Anysphere is a local app: you can ensure 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 your local app to accept the update.

5.4 Protection against non-privileged local malware

It is unfortunately impossible to prevent compromise from softwares with root privilege. We can, nevertheless, reduce the risk of non-privileged malware. Our beta version protects against non-privileged malware[**How do we achieve this?** —STZH1555], 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. xxx

6 Related Research

Metadata-private communication has been studied for decades. In 1981, David Chaum introduced **mix-nets**, which bounce messages between a small number of servers. Combined with onion encryption, mix-nets makes it impossible to determine the destination of a given source packet. However, mix-nets require the assumption 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 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].

[I believe passive voice should be avoided. —STZH] xxx

The 2010s saw a flurry of research papers attempting a few different methods to achieve scalable metadata-privacy: Dissent and Riposte [CGF10; CGBM15] in-

troduced **DC-nets**, Vuvuzela, Atom, Talek and many others [VDH+15; Che+20; Kwo+17] introduced mix-nets with stronger security guarantees, Clarion, mcMix and Blinder [Ale+17; EB21; APY20] introduced multi-party computation techniques, and NIAR [SW21; Bün+21] introduced function-hiding functional encryption. All but NIAR's approach are less secure than our PIR-based approach; 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], 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 What's Next?

Private communication is a hard problem. We are actively doing research to increase convenience while preserving complete privacy. We also urge the research community to join us: during the first practical deployment of metadata-private communication, we have found that while many papers focus on the scalability of a core one-person messaging protocol, too few focus on problems that come up in practice, including private trust establishment, multiple-friend management, one-to-many and many-to-many communcation, and DoS protection.

7.1 Implementation milestones

7.1.1 Small Files and images.

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

7.1.2 Forward secrecy.

We hope to integrate Signal's X3DH algorithm to ensure forward secrecy.

7.1.3 Public-key infrastructure.

To facilitate the discovery and .

7.1.4 Calls.

7.1.5 Hardened daemon against local malware.

7.2 Research problems

7.2.1 Denial of service resistance.

7.2.2 One-to-many and many-to-many conversations.

An essentially important problem to solve is to allow groups of people to broadcast messages to each other, without depending on the presense of any specific user. We understand that this brings risks in itself to users because it increases the overall risk surface that a user has to trust but we believe it is crucial for large-scale pragmatic adoption.

7.2.3 More efficient trust establishment.

7.2.4 The ACK problem.

7.2.5 The multiple friends problem.

7.2.6 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; we will tackle that problem separately later.

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.

[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.

[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.