Formal Security Definition of Metadata-Private Messaging

Shengtong Zhang
Anysphere, Inc.
New York, USA
stzh1555@anysphere.co

Arvid Lunnemark
Anysphere, Inc.
New York, USA
arvid@anysphere.co

Sualeh Asif
Anysphere, Inc.
New York, USA
sualeh@anysphere.co

ABSTRACT

We present a novel, complete definition of metadata-private messaging (MPM), and show that our definition is achievable and non-trivially more general than previous attempts that we are aware of. Our main contributions are:

(1) We describe a vulnerability in existing MPM implementations through a variation of the compromised-friend (CF) attack proposed by Angel et al. [ALT18]. Our attack can compromise the exact metadata of any conversations between honest users.

(2) We present a security definition for MPM systems assuming that some friends may be compromised.

(3) We present a protocol satisfying our security definition based on Anysphere, an MPM system we deployed in practice [LZA22b].

1 INTRODUCTION

This paper is motivated by our experience of developing a metadata-private messaging (MPM) service called Anysphere. In our whitepaper [LZA22b, Section 3], we describe Anysphere's threat model and core protocol at a high level. The original intent of this document was to rigorously show that Anysphere's core protocol satisfies the metadata privacy we promise. In the process, we discovered the need for a new security definition, which may be of independent interest.

Existing security proofs of MPM systems (such as [CF10; CBM15; AS16; Ahm+21]) have shown the privacy of a private-information-retrieval (PIR) system where users can deposit and retrieve information without revealing metadata to the server. We found that we wanted stronger guarantees for several reasons.

- The security of the PIR system does not guarantee the security of the messaging system as a whole. A well-known example illustrating this is the compromised-friend (CF) attack proposed by Angel, Lazar and Tzialla ([ALT18]). They show that if an honest user befriends a malicious user, then the metadata of conversations between two honest users might be compromised even with a secure PIR system. To our knowledge, no proofs exist that show immunity against CF attacks[1].

  We found a more powerful CF attack, described in Section 4, while writing this paper. This vulnerability affects existing implementations of Pung and Addra, and can result in full compromise of metadata (timing, sender and receiver of PIR queries). This attack demonstrates that CF attacks can potentially cause more damage than previously expected [ALT18]. In light of this attack, we wish to rigorously prove that our MPM system can defend against CF attacks.
- Anysphere is based on Addra [Ahm+21]. Addra is originally designed for users to hold exactly one conversation at a time. In our application, clients may hold many different conversations at the same time. We need to ensure that our adaptation does not introduce new vulnerabilities.
- Addra and Pung [AS16] assume that clients run in synchronous round, and each client sends exactly one message to the server each round. As clients have different level of resources, running synchronous rounds is not economical. For example, corporate users might wish rounds run faster to receive timely updates, while individual users might not want to participate in each round to preserve bandwidth. Anysphere uses asynchronous rounds where each client can transmit on a different schedule. We need to justify the security definition to incorporate this.
- The MPM systems mentioned above lack a mechanism to detect and retransmit lost or shuffled packets. We introduce an acknowledgement (ACK) mechanism in Anysphere to fix this issue. The security of this mechanism cannot be justified by previous security definitions. This is because the previous security definitions assume that the clients do not change their behavior based on the server's response, which is inherently not true for messaging systems.

This paper is organized as follows. In Section 3, we present a formal security definition of what it means for a whole messaging system to be correct and secure under our threat model outlined in [LZA22b]. In Section 4, we describe the new CF attack, which we name the *PIR replay attack*. In Section 5, we describe a slightly modified Anysphere core protocol in pseudocode. In Section 6 we prove that the protocol defined in Section 5 satisfies the security definition in Section 3. A technical detail in the proof is delayed to Section 7. In Appendix A, we describe and prove the security of Anysphere's real-world implementation. We focus on a technique we use called prioritization, which trades a small metadata leakage for efficiency.

2 CONVENTIONS

We use the following notational conventions.

- When we write $f(\cdot)$, the dot might hide several variables.

[1] Pung's security proof [Ang18] assumes honest users only ever talk to honest users.

- Given an oracle $O(x, \cdot)$ and a series $\{x_i\}$, we define $O(\{x_i\}, \cdot)$ as the oracle whose input contains an extra argument j and outputs $O(\{x_i\}, j, \text{arg}) = O(x_j, \text{arg})$.
- Given a series $\{a_i\}$, and a set of indices S, let a_S denote $\{a_i\}_{i \in S}$.
- Experiments are always parametrized by the security parameter λ. When we say two experiments are indistinguishable, we mean the views of the adversary in the two experiments are indistinguishable. The view of the adversary consists of all inputs, outputs, and internal randomness of the adversary.
- When machines "return" in a method, they do not execute any subsequent commands and exit the method immediately.

3 A NEW SECURITY DEFINITION

In this section, we design a general security definition for MPMs. Our definition shares many similarities with Canetti and Krawczyk's foundational CK models [CK01], which define the security of key exchange protocols over untrusted channels. However, we find it difficult to directly adapt the CK models, especially the authenticated-links (AM) model, to account for metadata privacy. Therefore, we design our security definition from scratch.

We start from the following basic principles.

(1) The messaging system has a centralized server in charge of storing and routing messages. We do not consider decentralized messaging systems in this paper.

(2) The messaging system has a large number of users, interacting with "client" software on their computers. The client software should allow the user to register, add friends, and send messages at any time. It should display received messages to the user.

(3) The messaging system should hide metadata of conversations between honest users from a powerful adversary that controls the server, the network, and any subset of clients.

(4) The messaging system should be reasonably robust. As long as the server functions normally, even if the adversary compromises some users, it should not disrupt conversations between honest users.

We now translate these principles into mathematical definitions that apply to a general messaging system.

Definition 3.1. A **timestep** is our system's basic unit of time. We assume that the system starts on timestep $t = 1$. Methods are executed on positive integer timesteps.

The timestep is different from "rounds" used in most MPM security definitions, since clients do not necessarily transmit real or fake messages at every timestep. Instead, a timestep plays a similar role as a clock cycle in computer hardware — think of it as being 1 nanosecond.

Definition 3.2. The **view** of a client is a tuple $(\mathcal{F}, \mathcal{M})$ consisting of

1. A list of friends $\mathcal{F}$.

2. A list of messages $\mathcal{M}$ received by the client, including the sender and content of the messages.

Remark: For simplicity, the view does not include messages sent by the client. The UI of the client can simply store such messages locally and display them to the user.

Definition 3.3. A **user input** $\mathcal{I}$ is a command the user can issue to the client. In our current protocol, it can take one of the following values.

- $\emptyset$: noop.
- $\mathsf{TrustEst}(\text{reg})$: Add the user identified by reg as a friend, and enable the two parties to start a conversation.
- $\mathsf{Send}(\text{reg}, \text{msg})$: Send the message msg to the client identified by reg. We assume that msg always has a constant length L_{msg}.[2]

Without loss of generality, we assume each user issues exactly one input per timestep.

Remark: In our implementation, we take $L_{\text{msg}} \approx 1\text{KB}$. To support variable length messages, we pad short messages and split long messages into chunks of length L_{msg}. This modification does not affect our security definition below.

Definition 3.4. The **registration information**, denoted reg in this paper, is the unique identifier of a user.

Remark: Throughout the rest of the paper, we always use the registration info as the "address" in the messaging system. For example, we use registration info as the argument in the make-friend and send-message methods. Registration info is ubiquitous in practical messaging systems: in Messenger, it is the Facebook handle; in Signal, it is the phone number. In Anysphere, it is the "public ID" as defined in [LZA22b, Figure 6].

Definition 3.5. A **messaging system** consists of the following polynomial-time algorithms.

Client-side algorithms for the stateful client C:

- $C.\mathsf{Register}(1^\lambda, i, N) \to \text{reg}$. This algorithm is called once for each client upon registration. It takes in a security parameter λ, the index i of the client, the total number of users N, and outputs a public registration info reg. It also initializes client storage and states.
- $C.\mathsf{Input}(t, \mathcal{I}) \to \text{req}$. This algorithm handles a user input $\mathcal{I}$. It updates the client storage to reflect the new input, and then issues a (possibly empty) request req to the server.

[2] For simplicity, and without loss of generality, we assume this holds even for adversarial inputs.

- C.ServerRPC(t, resp). This algorithm handles the server's response resp and updates client storage.
- C.GetView$() \rightarrow V$. This algorithm outputs the client's view (Definition 3.2). Its output is passed to the UI and displayed to the user.

Server-side algorithms for the stateful server S:

- S.InitServer$(1^\lambda, N)$. This algorithm takes in the security parameter λ, the number of clients N, and initializes the server-side database D_S.
- S.ClientRPC$(t, \{\text{req}_i\}_{i=1}^N) \rightarrow \{\text{resp}_i\}_{i=1}^N$. This algorithm responds to all client requests req_i the server received on a given timestep t. It outputs the responses resp_i that get sent back to the clients.

We have now mathematically described a general MPM system. In the rest of this section, we describe three desired properties of MPM systems: correctness, metadata privacy, and integrity.

3.1 Correctness

First, we describe how the server and clients interact when the server behaves honestly. We call this scenario the *honest server experiment*. To enforce robustness, we let an adversary control a subset of users and passively monitor all conversations.

Definition 3.6.
The **honest server experiment** takes the following parameters.

(1) λ, the security parameter.

(2) $N = N(\lambda)$, the number of clients, a polynomially-bounded function of λ.

(3) $T = T(\lambda)$, the number of timesteps, a polynomially-bounded function of λ.

(4) For each client $i \in [N]$ and timestep $t \in [T]$, a user input $\mathcal{I}_{i,t}$.

(5) A set of honest clients $\mathcal{H} \subset [N]$.

(6) A stateful probabilistic polynomial-time (p.p.t.) adversary $\mathcal{A}$.

Let S denote the server machine, and let $\{C_i\}_{i\in\mathcal{H}}$ denote the honest client machines. The experiment is described in Figure 1.

In this case, we expect the client's view to be "correct". We first need to define what "correct" means here. Informally speaking, the correct view should satisfy

(1) The list of friends contains the friends we called TrustEst on.

(2) Messages from friends should be present.

(3) No other messages should be present.

We state the formal definition.

Honest Server Experiment

(1) S.InitServer$(1^\lambda, N)$.
(2) For each $i \in \mathcal{H}$, $\text{reg}_i \leftarrow C_i$.Register$(1^\lambda, i, N)$.
(3) For t from 1 to T:
   (a) For each $i \in \mathcal{H}$, $\text{req}_i \leftarrow C_i$.Input$(t, \mathcal{I}_{i,t})$.
   (b) $\{\text{req}_i\}_{i\in[N]-\mathcal{H}} \leftarrow \mathcal{A}(\{\text{req}_i\}_{i\in\mathcal{H}})$.
   (c) $\{\text{resp}_i\}_{i=1}^N \leftarrow S$.ClientRPC$(t, \{\text{req}_i\}_{i=1}^N)$.
   (d) For each $i \in \mathcal{H}$, C_i.ServerRPC(t, resp_i).
   (e) $\mathcal{A}$ stores $\{\text{resp}_i\}_{i\in[N]}$.

Figure 1: The honest server experiment for a messaging system.

Definition 3.7. Given a set of honest clients identified by $\text{reg}_\mathcal{H}$, and user inputs $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$, a view $(\mathcal{F}_j, \mathcal{M}_j)$ of client j is **correct** if it satisfies

$$\mathcal{F}_j \cap \text{reg}_\mathcal{H} = \{\text{reg}_k : k \in \mathcal{H}, \exists t \in [T], \text{TrustEst}(\text{reg}_k) = \mathcal{I}_{j,t}\},$$

and, for each honest user $k \in \mathcal{H}$

$$\begin{aligned} \{\text{msg} :\quad & (\text{reg}_k, \text{msg}) \subset \mathcal{M}_j\} \\ =\{\text{msg} :\quad & \exists t \in [T], \text{Send}(\text{reg}_j, \text{msg}) = \mathcal{I}_{k,t} \wedge \\ & \exists t' < t, \text{TrustEst}(\text{reg}_j) = \mathcal{I}_{k,t'} \wedge \\ & \exists t'' \in [T], \text{TrustEst}(\text{reg}_k) = \mathcal{I}_{j,t''}\}. \end{aligned}$$

Remark: We comment on some subtleties implied by this definition.

(1) The definition allows $\mathcal{F}$ to contain registration info not corresponding to any client. We can rule out these "ghost" friends by sending each friend a "hello message" and waiting for an ACK before including the friend in a view.

(2) If user k tries to send user j a message before user j adds user k as a friend, user j should be able to receive the message.

Since the UI can query the client at any time, we expect the client's view to be "correct" all the time. There is a caveat: due to the lack of synchronous rounds, the clients do not immediately read all messages sent by their friends. Thus, the strongest correctness notion of sequential consistency might not be satisfied. Instead, we settle for a pair of weaker consistency models defined in [Ter13].

Definition 3.8. We say a messaging system is **correct** if for any choice of parameters of the honest server experiment, any $j \in \mathcal{H}$, and any positive integer $T_0 \leq T$, the messaging system satisfies the following two properties with probability $1 - \text{negl}(\lambda)$.

(1) **Consistent prefix**: Let $V_j \leftarrow C_j$.GetView() be the view of client j after timestep T_0 of the honest server experiment. Consistent prefix means that V_j is identical to the correct view of the client j where some prefix of user inputs have been executed on each client machine. More formally, for any $j \in \mathcal{H}$, there exists a map $t : \mathcal{H} \rightarrow [T_0]$ such that V_j

is a correct view of client j with honest users $\text{reg}_{\mathcal{H}}$ and inputs $\{\mathcal{I}'_{i,t}\}_{i\in\mathcal{H},t\in[T]}$, where we define

$$\mathcal{I}'_{i,t} = \begin{cases} \mathcal{I}_{i,t}, t \le t(i) \\ \emptyset, t > t(i) \end{cases} .$$

(2) **Eventual consistency**: For any T_1, there is a polynomial function $T_{cons} = T_{cons}(N, T_1)$ such that if $T_0 \ge T_{cons}$, such that we can take $t(i) \ge T_1$ for every $i \in \mathcal{H}$.

3.2 Metadata Security

Next, we define security against an active adversary. We first recap our threat model, defined in [LZA22b, Section 2.2]. We allow the adversary to do the following.

(1) The adversary can control all servers and the entire internet. In the definition below, the adversary reads all requests from honest clients, computes any polynomial time function over them, and returns any response to each client. Therefore, the adversary can perform all transport- and application-layer attacks considered by the security community that we know of, such as eavesdropping, traffic analysis, and active attacks like cut-and-paste, deep packet inspection, and replay attacks [WS96].

(2) The adversary can control any subset of the users. In the definition below, the adversary can compromise the friends of honest clients and schedule conversations with honest clients. This allows the adversary to launch the CF attacks defined in [ALT18] and Section 4.

We do not allow the adversary to do the following.

(1) The adversary cannot access or launch side-channel attacks on client machines such as timing attacks and Spectre [Koc+19]. We assume timing data and the intermediate state of client execution are invisible to the adversary.

(2) The adversary cannot break standard cryptography. In Section 5, we describe standard security requirements on the cryptography primitives we use.

We write our security definitions following the real-world-ideal-world paradigm used in [SW21, Section 2.2]. On a high level, the real-world experiment is the honest world experiment with an adversarial server running arbitrary code. The ideal world experiment is the real-world experiment with crucial information "redacted". We say the messaging system is secure if the adversary's views under the two experiments are indistinguishable.

We first define the real-world experiment.

Definition 3.9. The real-world experiment uses the parameters $\lambda, N, T, \mathcal{H}$ from Definition 3.6. Let $\mathcal{A}$ be a stateful p.p.t. adversary. The experiment $\text{Real}^{\mathcal{A}}_{\text{msg}}$ is described in Figure 2.

Real-World Experiment $\text{Real}^{\mathcal{A}}_{\text{msg}}$

(1) $\mathcal{H} \leftarrow \mathcal{A}(1^\lambda, N, T)$.
(2) For each $i \in \mathcal{H}$, $\text{reg}_i \leftarrow C_i.\text{Register}(1^\lambda, i, N)$.
(3) $\{\mathcal{I}_{i,1}\} \leftarrow \mathcal{A}(1^\lambda, \text{reg}_{\mathcal{H}})$.
(4) For t from 1 to T:
   (a) For each $i \in H$, $\text{req}_i \leftarrow C_i.\text{Input}(t, \mathcal{I}_{i,t})$.
   (b) $\{\text{resp}_i\}_{i\in\mathcal{H}}, \{\mathcal{I}_{i,t+1}\}_{i\in\mathcal{H}} \leftarrow \mathcal{A}(1^\lambda, \{\text{req}_i\}_{i\in\mathcal{H}})$.
   (c) For each $i \in \mathcal{H}$, $C_i.\text{ServerRPC}(t, \text{resp}_i)$.

Figure 2: Real-world experiment for a messaging system.

We next define the ideal-world experiment. As in [SW21], we first define the leakage, which describes the information the adversary is allowed to know. Informally, the adversary knows the time and contents of trust establishment with compromised clients and messages sent to compromised clients. The formal definition is below.

Definition 3.10. Let $\text{reg}_{\mathcal{H}}$ be the registration info of honest clients. Let $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ be the input from honest clients. We define the **leakage** $\text{Leak}(\{\mathcal{I}_{i,t}\}, \text{reg}_{\mathcal{H}})$ as

$$\text{Leak}(\{\mathcal{I}_{i,t}\}, \text{reg}_{\mathcal{H}}) = (\text{Leak}_f, \text{Leak}_m)$$

where

$$\text{Leak}_f = \{(i, \text{reg}, t) : \text{TrustEst}(\text{reg}) = \mathcal{I}_{i,t}, \text{reg} \notin \text{reg}_{\mathcal{H}}\}.$$

$$\text{Leak}_m = \{(i, \text{reg}, \text{msg}, t) : \text{Send}(\text{reg}, \text{msg}) = \mathcal{I}_{i,t}, \text{reg} \notin \text{reg}_{\mathcal{H}}\}.$$

Definition 3.11. We use the same parameters and notations as Definition 3.9. Furthermore, let Sim be a stateful simulator. The ideal-world experiment $\text{Ideal}^{\mathcal{A},\text{Sim}}_{\text{msg}}$ is described in Figure 3.

Ideal-World Experiment $\text{Ideal}^{\mathcal{A},\text{Sim}}_{\text{msg}}$

(1) $\mathcal{H} \leftarrow \mathcal{A}(1^\lambda, N, T)$.
(2) $\text{reg}_{\mathcal{H}} \leftarrow \text{Sim}(1^\lambda, N, T)$.
(3) $\{\mathcal{I}_{i,1}\}_{i\in\mathcal{H}} \leftarrow \mathcal{A}(1^\lambda, \text{reg}_{\mathcal{H}})$.
(4) For t from 1 to T:
   (a) $\{\text{req}_i\}_{i\in\mathcal{H}} \leftarrow \text{Sim}(t, \text{Leak}(\{\mathcal{I}_{i,t}\}, \text{reg}_{\mathcal{H}}))$.
   (b) $\{\text{resp}_i\}_{i\in\mathcal{H}}, \{\mathcal{I}_{i,t+1}\}_{i\in\mathcal{H}} \leftarrow \mathcal{A}(1^\lambda, \{\text{req}_i\}_{i\in\mathcal{H}})$.
   (c) $\text{Sim}(t, \{\text{resp}_i\}_{i\in\mathcal{H}})$.

Figure 3: Ideal-world experiment for a messaging system.

Finally, we define metadata security.

Definition 3.12. We say that a messaging system is **SIM-metadata secure** if there exists a p.p.t. simulator Sim such that for any p.p.t. adversary $\mathcal{A}$, the view of $\mathcal{A}$ in $\text{Real}^{\mathcal{A}}_{\text{msg}}$ is indistinguishable from the view of $\mathcal{A}$ in $\text{Ideal}^{\mathcal{A},\text{Sim}}_{\text{msg}}$.

Next, we describe an equivalent security definition for readers who are more accustomed to indistinguishability-based security definitions.

Definition 3.13. We use the same notations as in Definition 3.11. For each $b \in \{0,1\}$, the IND experiment $\mathsf{Ind}_b^{\mathcal{A}}$ is described in Figure 4.

IND Experiment $\mathsf{Ind}_b^{\mathcal{A}}$

(1) For each $i \in \mathcal{H}$, $\mathsf{reg}_i \leftarrow C_i.\mathsf{Register}(1^\lambda, i, N)$.
(2) $\{\mathcal{I}_{i,1}^0\}, \{\mathcal{I}_{i,1}^1\} \leftarrow \mathcal{A}(1^\lambda, \mathsf{reg}_{\mathcal{H}})$.
(3) For t from 1 to T:
  (a) For each $i \in \mathcal{H}$, $\mathsf{req}_i \leftarrow C_i.\mathsf{Input}(t, \mathcal{I}_{i,t}^b)$.
  (b) for $b' \in \{0,1\}$, $\{\mathsf{resp}_i^{b'}\}_{i\in\mathcal{H}}, \{\mathcal{I}_{i,t+1}^{b'}\}_{i\in\mathcal{H}} \leftarrow \mathcal{A}(1^\lambda, \{\mathsf{req}_i\}_{i\in\mathcal{H}})$.
  (c) For each $i \in \mathcal{H}$, $C_i.\mathsf{ServerRPC}(t, \mathsf{resp}_i^b)$.

Figure 4: Indistinguishability-based experiment for a messaging system.

We say that the adversary $\mathcal{A}$ is **admissible** if with probability 1 we have

$$\begin{aligned}&\mathsf{Leak}(\{\mathcal{I}_{i,t}^0\}_{i\in\mathcal{H},t\in[T]}, \mathsf{reg}_{\mathcal{H}})\\ =&\mathsf{Leak}(\{\mathcal{I}_{i,t}^1\}_{i\in\mathcal{H},t\in[T]}, \mathsf{reg}_{\mathcal{H}}).\end{aligned}$$

Definition 3.14. We say that a messaging system is **IND-metadata secure** if for any N and T polynomially bounded in λ, and any admissible adversary $\mathcal{A}$, the view of $\mathcal{A}$ in $\mathsf{Ind}_0^{\mathcal{A}}$ is indistinguishable from the view of $\mathcal{A}$ in $\mathsf{Ind}_1^{\mathcal{A}}$.

The argument in [SW21, Appendix A] shows that IND-metadata security is equivalent to SIM-metadata security.

3.3 Integrity

Finally, we define the notion of integrity. Informally, while a malicious server can deny service to users, it should not be able to forge messages or selectively omit messages between honest users. In other words, the client must guarantee consistent prefixes in our threat model.

Definition 3.15. Consider the real-world experiment in Figure 2. Define $V_j = (\mathcal{F}, \mathcal{M}) \leftarrow C_j.\mathsf{GetView}()$ at the end of the experiment. Then we say that the messaging system **guarantees integrity** if for any pair of honest users $i, j \in \mathcal{H}$, there exists a $t(i) \in [T]$ such that with probability $1 - \mathsf{negl}(\lambda)$,

$$\begin{aligned}\{\mathsf{msg} :\quad & (\mathsf{reg}_i, \mathsf{msg}) \subset \mathcal{M}\}\\ =\{\mathsf{msg} :\quad & \exists t \le t(i), \mathsf{Send}(\mathsf{reg}_j, \mathsf{msg}) = \mathcal{I}_{i,t} \wedge\\ & \exists t' < t, \mathsf{TrustEst}(\mathsf{reg}_j) = \mathcal{I}_{i,t'} \wedge\\ & \exists t'', \mathsf{TrustEst}(\mathsf{reg}_i) = \mathcal{I}_{j,t''}\}.\end{aligned}$$

3.4 A Weaker Security Definition

Unfortunately, Anysphere does not satisfy Definition 3.12, which is very strong. In fact, Angel et al. [ALT18] argues that this security notion is very hard to satisfy in general. For the threat model in our whitepaper [LZA22b], we argued security based on the strong assumption that no friends are compromised. In reality, this assumption cannot be guaranteed. In this section, we define a weaker security notion that allows the adversary to compromise friends while still satisfying theoretical security.

Definition 3.16. We say that a set of inputs $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ satisfies B-**bounded friends** if for any $i \in \mathcal{H}$, the set

$$\{\mathsf{reg} : \exists t, \mathsf{TrustEst}(\mathsf{reg}) = \mathcal{I}_{i,t}\}$$

has cardinality at most B.

We say that a messaging system is correct with B-bounded friends if it satisfies a modified Definition 3.8, where the only change is that the honest server experiment (Figure 1) is parametrized by a $\{\mathcal{I}_{i,t}\}$ that satisfies B-bounded friends.

We say that a messaging system is SIM-secure with B-bounded friends if it satisfies a modified Definition 3.12, where the only change is that $\mathcal{A}$ is restricted to only p.p.t. adversaries that produce an input set $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ that satisfies B-bounded friends.

Remark: As Angel et al. points out in [ALT18], the B-bounded friends hypothesis is a strategy to achieve perfect prevention against CF attacks. However, Angel et al. shows that any protocol that perfectly defends against CF attacks would be very inefficient in practice. We can modify the protocol to remove the B-bounded friends hypothesis and greatly increase efficiency if we are willing to leak a small amount of additional information. Our implementation of Anysphere described in Appendix A uses this modification.

4 THE PIR REPLAY ATTACK

In this section, we present the *PIR replay attack* mentioned in the introduction. While the CF attacks in [ALT18] can only directly reveal the number of friends each honest user has, the PIR replay attack can potentially directly reveal sender-recipient relationships, if the recipient has a compromised friend. The vulnerability affects existing implementations of both Pung and Addra.

Most PIR schemes (such as SealPIR [Ang+18], MulPIR [Ali+21], FastPIR [Ahm+21], and Spiral [MW22]) use an underlying homomorphic public key cryptosystem, typically some variation of the BFV cryptosystem [FV12]. Generating the necessary keypairs in these cryptosystems is expensive. To improve performance, real-world implementations of FastPIR[3] and Spiral[4] reuse PIR keys. Each client generates a secret $\mathsf{sk}_{\mathrm{pir}}$ once and use them to encrypt all PIR queries $\mathrm{ct} = \mathrm{Query}(1^\lambda, \mathsf{sk}_{\mathrm{pir}}, i)$. This optimization was regarded safe since it preserves the UO-ER security definition in [AS16, Extended Version]. We show how to combine this optimization with a compromised friend to leak metadata.

Suppose the adversary suspects that honest users A and B are communicating, and honest user A has a compromised friend C. On timestep T_0, user A sends a PIR request ct to the server. The adversary wishes to know if ct is a query to honest user B's mailbox at index i_B. Assume that

[3] As of 08/31/2022, the following code reuses PIR keys: https://github.com/ishtiyaque/FastPIR/tree/d50b1ba4ad4de64181bce71bccd352798dfa2bb3
[4] As of 08/31/2022, the following code reuses PIR keys: https://github.com/menonsamir/spiral-rs/tree/0f9bdc157086ea9534f70bb7d9e7f19920663e84

- user A will have a conversation with user C at a future time $T_1 > T_0$,
- user A does not switch their PIR key pair between time T_0 and T_1, and
- user A will provide "feedback" $f(m)$ to user C's message m. This could be any nonempty response to C's message, such as the ACK message in our system (see Section 5.2), or in systems without acknowledgements, a "I'm good! How are you?" in response to a "Hi! How are you?".

At time T_0, the malicious server stores ct, and continues to serve A honestly until time T_1. During A and C's conversation, the server responds to A's PIR requests with resp = $\mathsf{Answer}^{DB'}(1^\lambda, \mathsf{ct})$, where $DB'[i_B]$ is a valid message m from C to A, and $DB'[i] = 0$ for any $i \neq i_B$. If ct is a query to i_B, A will receive the message from C and send feedback $f(m)$ to C. Otherwise, A will not receive a message from C and not send feedback to C. Therefore, C can observe A's feedback and learn if ct is a query to i_B or not.

This attack can be prevented by changing the PIR key pair each round, which increases both client-side computation and the bandwidth required, because the server can no longer cache the big Galois key used in many PIR protocols. The PIR replay attack shows that compromised friends can do more damage to MPM systems than previously known.

Remark: In [Hen+22], Henzinger et al. discovered another attack exploiting key pair reuse. The two attacks are fundamentally different. The attack proposed in [Hen+22] is on the primitive level, specific to the BFV cryptosystem, and assumes the attacker has full access to the clients' PIR decryption oracles. In contrast, our attack is on the protocol level, applies to any public key homomorphic encryption system, and makes no assumption on the particular feedback the attacker gets.

5 A SECURE MPM PROTOCOL

In this section, we formally define the core protocol of a messaging system satisfying Definition 3.12 under the B-bounded friend hypothesis. Our protocol closely resembles the Anysphere core protocol described in our whitepaper [LZA22b], modified to achieve full prevention against CF attacks.

5.1 Cryptographic Primitives

Anysphere relies on two cryptographic primitives: an authenticated encryption (AE) system, and a PIR scheme. We outline formal simulator-based security definitions of the two.

5.1.1 Authenticated Encryption Scheme. Our authenticated encryption (AE) scheme Π_{ae} consists of three randomized efficient algorithms

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$$

with specifications

- $\mathsf{Gen}(1^\lambda) \to \mathsf{sk}$,
- $\mathsf{Enc}(\mathsf{sk}, m) \to \mathsf{ct}$,
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$.

We require the scheme to be correct and EUF-CMA unforgeable. We also require a variation of IK-CCA key privacy defined in [Bel+01]. Below are the precise security requirements.

Definition 5.1. Let $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$. We say that our AE scheme is **correct** if for any plaintext m of length L_{ae}, we have

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, m)) = m$$

with probability 1.

Definition 5.2. Given a secret key sk, and a polynomial-time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^{L_{\mathsf{ae}}}$, the **Eval oracle**

$$\mathsf{Eval}_f(\mathsf{sk}, \mathsf{sk}', \arg_{\mathsf{ct}}, \arg_p) \to \mathsf{ct}'$$

takes as input a set of ciphertexts $\arg_{\mathsf{ct}} = \{\mathsf{ct}_i\}$, and a plaintext argument $\arg_p$. It sets

$$m_i \leftarrow \mathsf{Dec}(\mathsf{sk}', \mathsf{ct}_i)$$

and outputs $\mathsf{ct}' = \mathsf{Enc}(\mathsf{sk}, f(\{m_i\}, \arg_p))$.

Definition 5.3. Let N and R be polynomial in λ. Consider the two experiments defined in Figure 5.

Real-World Experiment $\mathsf{Real}^{\mathcal{A}}_{\mathsf{Eval}}$

(1) For i from 1 to N, $\mathsf{sk}_i \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) For r from 1 to R
  (a) $i, j, \arg_{\mathsf{ct}}, \arg_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^0_r \leftarrow \mathsf{Eval}_f(\mathsf{sk}_i, \mathsf{sk}_j, \arg_{\mathsf{ct}}, \arg_p)$.
  (c) $\mathcal{A}$ stores ct^0_r.

Ideal-World Experiment $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}_{\mathsf{Eval}}$

(1) For i from 1 to N, $\mathsf{sk}_i \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) For r from 1 to R
  (a) $i, j, \arg_{\mathsf{ct}}, \arg_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^1_r \leftarrow \mathsf{Sim}(1^\lambda)$.
  (c) $\mathcal{A}$ stores ct^1_r.

Figure 5: Real- and ideal-world experiments for authenticated encryption (AE).

Then we say the AE scheme is **eval-secure** if there exists a p.p.t. simulator Sim such that for any polynomial-time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^{L_{\mathsf{ae}}}$ and any p.p.t. adversary with oracle $\mathcal{A}^{O}$, the real-world experiment and the ideal-world experiment are computationally indistinguishable.

We will later show that eval-security is implied by an analogy of IK-CCA [Bel+01, Definition 1] for symmetric-key cryptography. Since the proof is quite lengthy, we delay it to Section 7.

Definition 5.4. Consider the forging experiment described in Figure 6.

Forging Experiment

(1) $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) $\mathsf{ct} \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot),\mathsf{Dec}(\mathsf{sk},\cdot)}(1^\lambda)$.

Figure 6: Forging experiment for authenticated encryption (AE).

For each $i \in [N]$, let $\mathsf{ct}_{\mathsf{Query}}$ be the set of outputs of the Enc oracle. We say the AE scheme is **EUF-CMA** if for any p.p.t adversary with oracle $\mathcal{A}^O$, we have

$$\mathbb{P}(\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \neq \perp \wedge \mathsf{ct} \notin \mathsf{ct}_{\mathsf{Query}}) = \mathrm{negl}(\lambda).$$

5.1.2 Symmetric Key Distribution. In our system, each pair of users shares two secret keys, one for encrypting each direction of traffic. For two users identified by registration info reg_0 and reg_1, user reg_0 decrypts messages from reg_1 using their **read key** sk_r, and encrypts messages to reg_1 using their **write key** sk_w. Thus, reg_0's read key is reg_1's write key, and vice versa.

Previous MPM systems like Pung and Addra assume each pair of users has a shared secret distributed in advance. In this paper, we assume these keys are independently generated by a trusted third party. For users identified by registration info reg_0 and reg_1, the trusted third party delivers their shared secret keys $(\mathsf{sk}_r, \mathsf{sk}_w)$ to user reg_i when $\mathsf{GenSec}(\mathsf{reg}_i, \mathsf{reg}_{1-i})$ is called. The adversary does not have access to the shared secret between trusted users.

In [LZA22b, Section 4], we describe separate trust establishment protocols to replace the trusted third party without compromising metadata security.

5.1.3 PIR Scheme. Like previous MPM systems, Anysphere relies on a private information retrieval (PIR) scheme Π_{pir}. A PIR scheme supports three efficient algorithms

- $\mathsf{Query}(1^\lambda, i) \rightarrow (\mathsf{ct}, \mathsf{sk})$.
- $\mathsf{Answer}^{\mathsf{DB}}(1^\lambda, \mathsf{ct}) \rightarrow a$.
- $\mathsf{Dec}(1^\lambda, \mathsf{sk}, a) \rightarrow x_i$.

where DB is a database with N entries of length L_{pir}, and N is a parameter bounded by some polynomial $N(\lambda)$. It should satisfy the following standard correctness and security definitions [KO97].

Definition 5.5. We say that the PIR scheme is **correct** if for any database DB with N entries of length L_{pir}, the experiment

(1) $(\mathsf{ct}, \mathsf{sk}) \leftarrow \mathsf{Query}(1^\lambda, i)$.
(2) $a \leftarrow \mathsf{Answer}^{\mathsf{DB}}(1^\lambda, \mathsf{ct})$.
(3) $x_i \leftarrow \mathsf{Dec}(1^\lambda, \mathsf{sk}, a)$.

satisfies $x_i = DB[i]$ with probability 1.

Definition 5.6. Let R be polynomially bounded in λ. Consider the experiments in Figure 7. We say the PIR scheme is **SIM-secure** if there exists a p.p.t simulator Sim such that for any stateful p.p.t. adversary $\mathcal{A}$, the view of $\mathcal{A}$ under the real-world experiment and the ideal-world experiment are computationally indistinguishable.

Real-World Experiment $\mathsf{Real}^{\mathcal{A}}_{\mathbf{pir}}$

(1) for r from 1 to R
  (a) $i_r \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^0_r, \mathsf{sk} \leftarrow \mathsf{Query}(1^\lambda, i_r)$.
  (c) $\mathcal{A}$ stores ct^0_r.

Ideal-World Experiment $\mathsf{Ideal}^{\mathcal{A}}_{\mathbf{pir}}$

(1) for r from 1 to R
  (a) $i_r \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^1_r, \mathsf{sk} \leftarrow \mathsf{Sim}(1^\lambda)$.
  (c) $\mathcal{A}$ stores ct^1_r.

Figure 7: Real- and ideal-world experiments for private information retrieval (PIR).

For our implementation, we use Libsodium's key exchange functionality to implement the GenSec() functions as described in [LZA22b, Section 4], and Libsodium's secret key AEAD for the Enc and Dec functions [Den13]. We use Addra's FastPIR as the PIR protocol. The definition, correctness, and security proof of FastPIR can be found in [Ahm+21, Section 4] and in [Ang18] with more detail.

Throughout the rest of the document, we denote the AE scheme Π_{ae} and the PIR scheme Π_{pir}.

5.2 Messages, Sequence numbers, ACKs

This section describes how Anysphere adopts TCP's acknowledgement system to satisfy integrity (Definition 3.15).

Each client i labels all outgoing messages to another client j with a positive integer called the sequence number. The client transmits the labeled message[5] $\mathsf{msg}^{lb} = (k, \mathsf{msg})$, where k is the sequence number and msg is the actual message.

Critical to both consistent prefix and eventual consistency are the ACK messages. An ACK message denoted $\mathsf{ACK}(k)$ encodes a single integer k.[6] It means "I have read all messages up to sequence number k from you". As we will soon define rigorously, user i broadcasts the message with sequence number k until user j sends $\mathsf{ACK}(k)$, in which case they begin broadcasting the message with sequence number $k + 1$. We ensure ACK messages are distinct from labeled messages generated from user input.[7]

Finally, we deal with the subtlety of the length of messages. Let L_{msglb} denote the length of a labeled message msg^{lb}, and let L_{ct} be the length of a ciphertext generated by encoding msg^{lb} with $\Pi_{\mathrm{ae}}.\mathsf{Enc}$. While the input messages have length L_{msg}, the messages

[5] Called chunk in the implementation.
[6] In our implementation, the ACK message is slightly more complicated to take chunking into account.
[7] In our implementation, we encode ACKs and labeled messages with different protobuf structs.

the client sent to the server have length L_{ct}. We set parameter $L_{\mathsf{ae}} = L_{\mathsf{msglb}}$ in the AE scheme, and $L_{\mathsf{pir}} = L_{\mathsf{ct}}$ in the PIR scheme.

Remark: To support $T(\lambda)$ rounds of messages between each pair of users, we need to reserve $\log T(\lambda)$ bits for the label. Therefore, L_{ae} and L_{pir} depend linearly on $\log \lambda$.

5.3 The Core Protocol

Recall some notations.

- λ is the security parameter.
- $N = N(\lambda)$ is the number of users.
- $T = T(\lambda)$ is the number of timesteps our protocol is run.
- L_{msg} is the length of the raw message.
- Π_{ae} is an AE scheme satisfying Definition 5.1, Definition 5.3 and Definition 5.4.
- Π_{pir} is a PIR scheme satisfying Definition 5.5 and Definition 5.6.

We now define the core protocol. We assume the B-bounded friends scenario in Definition 3.16.

Definition 5.7. The **B-bounded Anysphere messaging system** Π_{asphr} implements the method signatures in Definition 3.5 with the pseudocode below. In each method, the caller stores all inputs for future use.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{C.Register}(1^\lambda, i, N)$

---

(1) Initialize empty map frdb (database of friends). The map takes registration info reg as keys and the following fields as values.[8]
   - sk, the secret keys shared with the friend identified by reg.
   - seqstart, the sequence number of the current message being broadcasted to the friend identified by reg.
   - seqend, the largest sequence number ever assigned to messages to the friend identified by reg.
   - seqrecv, the largest sequence number such that the client has received all messages with sequence number in $1, \cdots, \mathsf{seqrecv}$ from the friend.

(2) Initialize empty maps in and out. The maps take registration info reg as keys and arrays of messages as values.[9]

(3) Set a transmission schedule T_{trans} (the time between messages sent over the network to the server). The user can customize this parameter. We assume T_{trans} is upper bounded by a constant T^U_{trans}.

(4) Initialize a "dummy" key $\mathsf{sk}_{\mathsf{dummy}} \leftarrow \Pi_{\mathsf{ae}}.\mathsf{Gen}(1^\lambda)$.

(5) Return $\mathsf{reg} = (i)$.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{S.InitServer}(1^\lambda, N)$.

Initialize arrays msgdb, ackdb of length N with entries of length L_{ct}. Fill them with random strings.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{C.Input}(t, \mathcal{I})$

This method runs in two phases. Phase 1 handles the user input $\mathcal{I}$, and phase 2 formulates the request to the server.

Phase 1.

If $\mathcal{I} = \emptyset$, do nothing.

If $\mathcal{I} = \mathsf{Send}(\mathsf{reg}, \mathsf{msg})$,

(1) Check that reg is in frdb. If not, skip to Phase 2.

(2) If reg is the registration of C_i itself, append msg to in[reg], and skip to Phase 2.[10]

(3) Add 1 to frdb[reg].seqend.

(4) Push $\mathsf{msg}^{lb} = (\mathsf{frdb[reg].seqend}, \mathsf{msg})$ to out[reg].

If $\mathcal{I} = \mathsf{TrustEst}(\mathsf{reg})$.

(1) Check if reg is in frdb. If so, skip to Phase 2.

(2) $\mathsf{sk}_r, \mathsf{sk}_w \leftarrow \mathsf{GenSec}(\mathsf{reg}_m, \mathsf{reg})$. Here reg_m is the client's own registration information.

(3) $\mathsf{frdb[reg]} \leftarrow \{\mathsf{sk} : (\mathsf{sk}_r, \mathsf{sk}_w), \mathsf{seqstart} : 1, \mathsf{seqend} : 0, \mathsf{seqrecv} : 0\}$.

Phase 2.

(1) If t is not divisible by T_{trans}, return $\emptyset$.

(2) Let $\{\mathsf{reg}_1, \cdots, \mathsf{reg}_k\}$ be the keys of frdb, where $k \le B$ since we assume B-bounded friends. Construct

$$S = [\mathsf{reg}_1, \cdots, \mathsf{reg}_k, \mathsf{reg}, \cdots, \mathsf{reg}],$$

where we add $B - k$ copies of $\mathsf{reg} = (-1)$, a dummy registration info. Sample $\mathsf{reg}_s, \mathsf{reg}_r$ uniformly and independently at random from S.[11]

(3) Let msg^{lb} be the labeled message with sequence number $\mathsf{frdb[reg}_s\mathsf{].seqstart}$ in $\mathsf{out[reg}_s]$. If $\mathsf{out[reg}_s]$ is empty, let $\mathsf{msg} \leftarrow (-1, 0^{L_{\mathsf{msg}}})$.

(4) $\mathsf{sk}_w \leftarrow \mathsf{frdb[reg}_s\mathsf{].sk[1]}$. If $\mathsf{reg}_s = \mathsf{reg}_0$ or does not exist in frdb, set $\mathsf{sk}_w \leftarrow \mathsf{sk}_{\mathsf{dummy}}$.

(5) $\mathsf{seqrecv} \leftarrow \mathsf{frdb[reg}_s\mathsf{].seqrecv}$. If $\mathsf{reg}_s = \mathsf{reg}_0$ or does not exist in frdb, set $\mathsf{seqrecv} \leftarrow -1$.

(6) Encrypt Messages with sk_w.
   - $\mathsf{ct}_{\mathsf{msg}} \leftarrow \Pi_{\mathsf{ae}}.\mathsf{Enc}(\mathsf{sk}_w, \mathsf{msg}^{lb})$.

---

[8] These fields are currently implicit. They are made explicit here for simplicity

[9] They are named Friend, Inbox, Outbox in our code. Our code is slightly more complicated to support features like sending to multiple friends and chunking.

[10] Skipping this step breaks consistent prefix.

[11] This step is inefficient. We will change this step in Appendix A.

   - $\mathsf{ct}_{\mathsf{ack}} \leftarrow \Pi_{\mathsf{ae}}.\mathsf{Enc}(\mathsf{sk}_w, \mathsf{ACK}(\mathsf{seqrecv}))$.

(7) Let $\mathsf{reg}_r = (i_r)$. Formulate a PIR request for index i_r.

   - $\mathsf{ct}_{\mathsf{Query}}, \mathsf{sk}_{\mathsf{pir}} \leftarrow \Pi_{\mathsf{pir}}.\mathsf{Query}(1^\lambda, i_r)$.

(8) return $\mathsf{req} = (\mathsf{ct}_{\mathsf{msg}}, \mathsf{ct}_{\mathsf{ack}}, \mathsf{ct}_{\mathsf{Query}})$.

(9) Remember reg_r and $\mathsf{sk}_{\mathsf{pir}}$.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{S.ClientRPC}(t, \{\mathsf{req}_i\}_{i=1}^N)$

---

for i from 1 to N

(1) If $\mathsf{req}_i = \emptyset$, let $\mathsf{resp}_i = \emptyset$, and continue to next i.

(2) Parse $\mathsf{req}_i = (\mathsf{ct}_{\mathsf{msg}}, \mathsf{ct}_{\mathsf{ack}}, \mathsf{ct}_{\mathsf{Query}})$.

(3) $\mathsf{msgdb}[i] \leftarrow \mathsf{ct}_{\mathsf{msg}}, \mathsf{ackdb}[i] \leftarrow \mathsf{ct}_{\mathsf{ack}}$.

(4) $a_{\mathsf{msg}} \leftarrow \Pi_{\mathsf{pir}}.\mathsf{Answer}^{\mathsf{msgdb}}(1^\lambda, \mathsf{ct}_{\mathsf{Query}})$.

(5) $a_{\mathsf{ack}} \leftarrow \Pi_{\mathsf{pir}}.\mathsf{Answer}^{\mathsf{ackdb}}(1^\lambda, \mathsf{ct}_{\mathsf{Query}})$.

(6) $\mathsf{resp}_i \leftarrow (a_{\mathsf{msg}}, a_{\mathsf{ack}})$.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{C.ServerRPC}(t, \mathsf{resp})$

---

(1) If $\mathsf{resp} = \emptyset$, return.

(2) Parse $\mathsf{resp} = (a_{\mathsf{msg}}, a_{\mathsf{ack}})$. Let $\mathsf{reg}_r, \mathsf{sk}_{\mathsf{pir}}$ be defined in the last call to $\Pi_{\mathsf{asphr}}.C.\mathsf{Input}$.

(3) $\mathsf{ct}_{\mathsf{msg}} \leftarrow \Pi_{\mathsf{pir}}.\mathsf{Dec}(1^\lambda, \mathsf{sk}_{\mathsf{pir}}, a_{\mathsf{msg}})$.

(4) $\mathsf{ct}_{\mathsf{ack}} \leftarrow \Pi_{\mathsf{pir}}.\mathsf{Dec}(1^\lambda, \mathsf{sk}_{\mathsf{pir}}, a_{\mathsf{ack}})$.

(5) $\mathsf{sk}_r \leftarrow \mathsf{frdb}[\mathsf{reg}_r].\mathsf{sk}[0]$.

(6) Decipher the message.

   (a) $\mathsf{msg}^{lb} \leftarrow \Pi_{\mathsf{ae}}.\mathsf{Dec}(\mathsf{sk}_r, \mathsf{ct}_{\mathsf{msg}})$.

   (b) If $\mathsf{msg}^{lb} = \perp$ or $\mathsf{msg}^{lb}[0] \neq \mathsf{frdb}[\mathsf{reg}_r].\mathsf{seqrecv} + 1$, skip the next two steps.

   (c) Add 1 to $\mathsf{frdb}[\mathsf{reg}_r].\mathsf{seqrecv}$.

   (d) $\mathsf{msg} \leftarrow \mathsf{msg}^{lb}[1]$. Push msg to $\mathsf{in}[\mathsf{reg}_r]$.

(7) Decipher the ACK.

   (a) $\mathsf{ack} \leftarrow \Pi_{\mathsf{ae}}.\mathsf{Dec}(\mathsf{sk}_r, \mathsf{ct}_{\mathsf{ack}})$.

   (b) If $\mathsf{ack} = \perp$ or ack is not the form $\mathsf{ACK}(k)$ for some k, return.

   (c) Let $\mathsf{ack} = \mathsf{ACK}(k)$. If $k < \mathsf{frdb}[\mathsf{reg}_r].\mathsf{seqstart}$, return.

   (d) $\mathsf{frdb}[\mathsf{reg}_r].\mathsf{seqstart} \leftarrow k+1$. Remove the message with sequence number k from $\mathsf{out}[\mathsf{reg}_r]$.

$\mathbf{\Pi}_{\mathsf{asphr}}.\mathsf{C.GetView}()$

Let $\mathcal{F}$ be the set of keys in frdb. Let $\mathcal{M}$ be the set $\{(\mathsf{reg}_r, \mathsf{msg}) : \mathsf{msg} \in \mathsf{in}[\mathsf{reg}_r]\}$. Return $(\mathcal{F}, \mathcal{M})$.

6 PROOFS

In this section, we prove that under the B-bounded friends assumption, our messaging system Π_{asphr} satisfies correctness (Definition 3.8), metadata security (Definition 3.12), and integrity (Definition 3.15).

6.1 Proof of Correctness and Integrity

We first show that Π_{asphr} satisfies both Definition 3.8 and Definition 3.15.

We introduce some notations for convenience. Define

$$\begin{aligned}\mathsf{MSGSent}(t_0, i, j) = \{(t, \mathsf{msg}) : t \leq t_0 \wedge \\ \mathsf{Send}(\mathsf{reg}_j, \mathsf{msg}) = \mathcal{I}_{i,t} \wedge \\ \exists t' < t, \mathsf{TrustEst}(\mathsf{reg}_j) = \mathcal{I}_{i,t'}\}.\end{aligned}$$

Let $\mathsf{msg}_{ij}(\ell)$ be the ℓ-th message in $\mathsf{MSGSent}(t_0, i, j)$ sorted by timestep t. Let $\mathsf{msg}_{ij}^{lb}(\ell) = (\ell, \mathsf{msg}_{ij}(\ell))$.

6.1.1 Consistent prefix. We show that Π_{asphr} satisfies Definition 3.15, and the consistent prefix property in Definition 3.8.

Lemma 6.1. *Consider the real-world experiment in Figure 2. Then with probability* $1 - \mathrm{negl}(\lambda)$, *for any pair of honest users* $i \neq j \in \mathcal{H}$ *and any* $t \leq T$, *the property below holds.*

Property: *One of the following holds at the end of timestep* t.

(1) $\mathsf{reg}_j \notin C_i.\mathsf{frdb}, C_i.\mathsf{out}[\mathsf{reg}_j] = C_j.\mathsf{in}[\mathsf{reg}_i] = \emptyset$.

(2) $\mathsf{reg}_j \in C_i.\mathsf{frdb}$. *Then for any* ℓ, $\mathsf{msg}_{ij}(\ell)$ *is labeled with sequence number* ℓ. *Furthermore, denote*

$$\begin{aligned}S_t &= C_i.\mathsf{frdb}[\mathsf{reg}_j].\mathsf{seqstart}, \\ E_t &= C_i.\mathsf{frdb}[\mathsf{reg}_j].\mathsf{seqend}, \\ R_t &= C_j.\mathsf{frdb}[\mathsf{reg}_i].\mathsf{seqrecv},\end{aligned}$$

(we define $R_t = 0$ *if* $\mathsf{reg}_i \notin C_j.\mathsf{frdb}$*). Then we have* $S_t \in \{R_t, R_t + 1\}$, *and*

$$\begin{aligned}|\mathsf{MSGSent}(t, i, j)| = E_t, \\ C_i.\mathsf{out}[\mathsf{reg}_j] = \{\mathsf{msg}_{ij}^{lb}(S_t), \cdots, \mathsf{msg}_{ij}^{lb}(E_t)\}, \\ C_j.\mathsf{in}[\mathsf{reg}_i] = \{\mathsf{msg}_{ij}(1), \cdots, \mathsf{msg}_{ij}(R_t)\}.\end{aligned}$$

Proof. When $t = 0$, (1) is satisfied since $C_i.\mathsf{frdb}$ is initialized as empty. We now show if these properties hold for all timesteps before t, then they will hold at the end of timestep t with probability $1 - \mathrm{negl}(\lambda)$.

The relevant variables are only modified in Phase 1 of $C_i.\mathsf{Input}$, in $C_i.\mathsf{ServerRPC}$ and in $C_j.\mathsf{ServerRPC}$. We show that the lemma is satisfied after each method (no matter in which order they execute).

Phase 1 of $C_i.\mathsf{Input}$

If (1) holds before timestep t starts, then unless

$$\mathcal{I} = \mathsf{TrustEst}(\mathsf{reg}_j),$$

none of the variables are changed. Otherwise, we have $S_t = 1, E_t = R_t = 0$, and both $C_i.\mathsf{out}[\mathsf{reg}_j] = C_j.\mathsf{in}[\mathsf{reg}_i] = \emptyset$, which satisfies (2).

If (2) holds before timestep t starts, then unless $\mathcal{I} = \mathsf{Send}(\mathsf{reg}_j, \mathsf{msg})$ none of the variables are changed. Otherwise, $E_t = E_{t-1} + 1$. We can check that

$$\mathsf{MSGSent}(t, i, j) = \mathsf{MSGSent}(t-1, i, j) + (t, \mathsf{msg}).$$

Thus we have

$$|\mathsf{MSGSent}(t, i, j)| = |\mathsf{MSGSent}(t-1, i, j)| + 1 = E_t.$$

Furthermore, we have $\mathsf{msg} = \mathsf{msg}_{ij}(E_t)$, and it will be labeled with sequence number E_t by step (3) in $\Pi_{\mathsf{asphr}}.C.\mathsf{Input}()$ Phase 1. Thus $\mathsf{msg}^{lb} = \mathsf{msg}_{ij}^{lb}(E_t)$, so the equality with $C_i.\mathsf{out}[\mathsf{reg}_j]$ is maintained. All the other variables remain unchanged. Thus the property remains true.

C_i.ServerRPC

The relevant variables only change during ACK decipher when a request is sent during Phase 2 of C_i.Input. Let $j' = i_r$ be the PIR index chosen in that phase. No relevant variable changes if $j' \neq j$, so assume $j' = j$. We apply the EUF-CMA property of Π_{ae}. With probability $1 - \mathrm{negl}(\lambda)$, the ack variable defined in step (7a) by

$$\mathsf{ack} = \Pi_{\mathsf{ae}}.\mathsf{Dec}(\mathsf{sk}_r, \mathsf{ct}_{\mathsf{msg}})$$

is either equal to $\perp$, or equal to a message user j encrypted using sk_r. Since user j only encrypts messages and ACKs to user i using sk_r, we conclude that either $\mathsf{ack} = \mathsf{ACK}(R')$ for some $R' \leq R_{t-1}$, or ack is equal to a previous labeled message user j sent to user i.

If ack is equal to a labeled message or $\perp$, client i will skip steps (7c) and (7d), and no variable will be changed. Now consider the case

$$\mathsf{ack} = \mathsf{ACK}(R').$$

In step (7c), we have $C_i.\mathsf{frdb}[\mathsf{reg}_j].\mathsf{seqstart} = S_{t-1}$. By the induction hypothesis, we have

$$S_{t-1} \geq R_{t-1} \geq R'.$$

So no variable is changed unless $S_{t-1} = R_{t-1} = R'$, in which case $S_t = R_{t-1} + 1$, and after popping $\mathsf{msg}_{ij}^{lb}(S_{t-1})$ we get

$$C_i.\mathsf{out}[\mathsf{reg}_j] = \{\mathsf{msg}_{ij}^{lb}(S_{t-1}+1), \cdots, \mathsf{msg}_{ij}^{lb}(E_t)\}.$$

Thus, the desired properties still hold.

C_j.ServerRPC

The relevant variables only change during step (6) (message decipher). Let $i' = i_r$ be the PIR index chosen in the previous call to C_i.Input. No relevant variable changes if $i' \neq i$, so assume $i' = i$. By EUF-CMA, with probability $1 - \mathrm{negl}(\lambda)$, the deciphered message msg^{lb} is either $\perp$, or an ACK message, or a labeled message sent from user i to user j in a previous timestep $t' \leq t$. In the first two cases, steps (6c) and (6d) are skipped, and no variable is updated. We now consider the last case. In this case, we have

$$\mathsf{msg}^{lb} = \begin{cases} (-1, 0^{L_{\mathsf{msg}}}), S_{t'-1} > E_{t'}. \\ \mathsf{msg}_{ij}^{lb}(S_{t'-1}), S_{t'-1} \leq E_{t'}. \end{cases}$$

Step (6c) and (6d) is not skipped only in the second case with $S_{t'-1} = R_{t-1} + 1$. By the induction hypothesis, in this case we have $S_{t-1} = S_{t'-1} = R_{t-1} + 1$. So $R_t = R_{t-1} + 1 = S_{t-1}$, and after popping $\mathsf{msg}_{ij}(S_{t-1})$ we get

$$C_i.\mathsf{out}[\mathsf{reg}_j] = \{\mathsf{msg}_{ij}^{lb}(1), \cdots, \mathsf{msg}_{ij}^{lb}(S_{t-1})\}.$$

Thus, the desired properties still hold.

We have proven that the desired properties hold at the end of timestep t with probability $1 - \mathrm{negl}(\lambda)$. □

We now show that Π_{asphr} satisfies Definition 3.15. We use the notations in Lemma 6.1. Let $(\mathcal{F}, \mathcal{M}) = C_j.\mathsf{GetView}()$. Take $t(j) = T_0$. Unpacking the definition of $\mathcal{F}$ and $\mathcal{M}$, we need to verify that for each $i \in [N]$, with probability $1 - \mathrm{negl}(\lambda)$ there exists a $t(i) \leq T_0$ such that

$$\begin{aligned} C_j.\mathsf{in}[\mathsf{reg}_i] = \{\mathsf{msg} : \exists t \leq t(i), \mathsf{Send}(\mathsf{reg}_j, \mathsf{msg}) = \mathcal{I}_{i,t} \wedge \\ \exists t' < t, \mathsf{TrustEst}(\mathsf{reg}_j) = \mathcal{I}_{i,t'} \wedge \\ \exists t'' \leq T_0, \mathsf{TrustEst}(\mathsf{reg}_i) = \mathcal{I}_{j,t''}\}. \end{aligned}$$

If $i = j$, then the equation holds for $t(j) = T_0$ since messages to oneself are deposited to C_j.in immediately in Phase 1 of $\Pi_{\mathsf{asphr}}.C.\mathsf{input}$. Now assume $i \neq j$.

We show that $t(i)$ exists as long as Lemma 6.1 holds at the current timestep T_0. We consider which scenario in the lemma holds.

If (1) holds, let $t(i) = T_0$. In this case, both sides of the equation are $\emptyset$.

If (2) holds, let $t(i) = T_0$ if $\mathsf{reg}_i \notin C_j.\mathsf{frdb}$ at the current timestep. Otherwise, let $t(i)$ be the timestep before $C_i.\mathsf{Send}(\mathsf{reg}_j, \mathsf{msg}_{ij}(R_t + 1))$ is called(or the current timestep if $\mathsf{msg}_{ij}(R_t + 1)$ does not exist). If $\mathsf{reg}_i \notin C_j.\mathsf{frdb}$, we have $R_t = 0$, so both sides of the equation are empty sets. Otherwise, by the definition of $\mathsf{msg}_{ij}(\ell)$, both sides of the equation are equal to $\{\mathsf{msg}_{ij}(1), \cdots, \mathsf{msg}_{ij}(R_t)\}$. So the equality in Definition 3.15 holds when the property in Lemma 6.1 holds for all pair of honest users $i, j \in \mathcal{H}$. Thus, Lemma 6.1 implies Definition 3.15.

The consistent prefix property in Definition 3.8 is an easy corollary of Definition 3.15 when the server behaves honestly.

6.1.2 Eventual consistency. We now show the eventual consistency property in Definition 3.8. Take the same choice of $t(i)$ as in the previous subsection. Let $T_{cons} = 2\lambda B^2 T_{\mathsf{trans}}^U \cdot T_1 + T_1$. We show that this T_{cons} satisfies the desired property.

We wish to show that if $T_0 \geq T_{cons}$, then with probability $1 - \mathrm{negl}(\lambda)$ we have $t(i) \geq T_1$. Let $E = |\mathsf{MSGSent}(T_1, i, j)|$ (note that E is independent of the protocol execution). If $R_{T_1} \geq E$, then by casework on the definition we always have $t(i) \geq T_1$. So it suffices to show that $R_{T_1} < R$ with negligible probability.

We use the same notation as Lemma 6.1. For each $k \leq E$, let X_k be the random variable denoting the first t such that $R_t \geq k$, with $X_0 = 0$. Let Y_k denote the first t such that $S_t > k$. It suffices to show that for any $k \leq E$, we have

$$Y_k - X_k > \lambda B^2 T_{\mathsf{trans}}^U$$

or

$$X_k - \max(Y_{k-1}, T_1) > \lambda B^2 T^U_{\text{trans}}$$

with negligible probability. For each timestep t between X_k and X_{k+1} such that t divides $C_i.T_{\text{trans}}$, let j_t denote the i_r that C_i chooses, and let i_t denote the i_s that C_j chooses for their last non-empty request to the server before timestep $t+1$. If $(i_t, j_t) = (i, j)$, we must have $S_t \geq k+1$ by the protocol definition since C_i reads the ACK message C_j deposited to the server database. Furthermore, if we let $K = \lceil T^U_{\text{trans}}/C_i.T_{\text{trans}} \rceil C_i.T_{\text{trans}}$, then the random variables

$$(i_t, j_t), (i_{t+K}, j_{t+K}), (i_{t+2K}, j_{t+2K}) \cdots$$

are mutually independent since client i and client j both made a non-empty request to the server during timestep $(t, t+K]$. Therefore, the probability that $Y_k - X_k > \lambda B^2 T^U_{\text{trans}}$ is at most

$$(1 - B^{-2})^{\lambda B^2 T^U_{\text{trans}}/K} = \text{negl}(\lambda)$$

as desired. Analogously, the event $X_k - \max(Y_{k-1}, T_1) > \lambda B^2 T^U_{\text{trans}}$ is also negligible. Thus, eventual consistency holds for Π_{asphr}.

6.2 Proof of Security

In this section we show that Π_{asphr} satisfies Definition 3.12.

Let $N'(\lambda) = N(\lambda)^2$ and $R'(\lambda) = 2N'(\lambda)T(\lambda)$. Assume the simulator Sim_{ae} satisfy Definition 5.3 with parameters (N, R) equal to $(N'(\lambda), R'(\lambda))$, and let Sim_{pir} be a simulator satisfying Definition 5.6 with parameters $(N, R) = (N(\lambda), R'(\lambda))$.

At step (2), (4a), and (4c) of Figure 3, the simulator $\text{Sim}_{\text{asphr}}$ runs modified versions of the client methods in the corresponding steps of Figure 2 for each client $i \in \mathcal{H}$. Modifications are marked with a siderule.

For each pair of honest users $i, j \in \mathcal{H}$, let

$$(\text{sk}_{ij,r}, \text{sk}_{ij,w}) = \text{GenSec}(\text{reg}_i, \text{reg}_j)$$

be their shared secret. Here $\text{sk}_{ij,r} = \text{sk}_{ji,w}$ denote the read key of user i for messages from user j.

$\text{Sim}_{\text{asphr}}.\text{Register}(1^\lambda, i, N)$

No modification.

$\text{Sim}_{\text{asphr}}.\text{Input}(t, i,$ **Hybrid 3:** replace $\mathcal{I}$ with Leak$)$

Phase 1:

> **Hybrid 3**:
>
> Initialize $\mathcal{I}$.
>
> (1) If $(i, \text{reg}, t) \in \text{Leak}_f$, $\mathcal{I} \leftarrow \text{TrustEst}(\text{reg})$.
>
> (2) If $(i, \text{reg}, \text{msg}, t) \in \text{Leak}_m$, $\mathcal{I} \leftarrow \text{Send}(\text{reg}, \text{msg})$.
>
> (3) Else, $\mathcal{I} \leftarrow \emptyset$.

If $\mathcal{I} = \emptyset$, do nothing.

If $\mathcal{I} = \text{Send}(\text{reg}, \text{msg})$,

(1) Check that reg is in frdb. If not, skip to Phase 2.

(2) If reg is the registration of C_i itself, append msg to in[reg], and skip to Phase 2.[12]

(3) Add 1 to frdb[reg].seqend.

(4) Push $\text{msg}^{lb} = (\text{frdb[reg].seqend}, \text{msg})$ to out[reg].

If $\mathcal{I} = \text{TrustEst}(\text{reg})$.

(1) Check if reg is in frdb. If so, skip to Phase 2.

(2) $\text{sk}_r, \text{sk}_w \leftarrow \text{GenSec}(\text{reg}_m, \text{reg})$. Here reg_m is the user's own registration information.

(3) $\text{frdb[reg]} \leftarrow \{\text{sk} : (\text{sk}_r, \text{sk}_w), \text{seqstart} : 1, \text{seqend} : 0, \text{seqrecv} : 0\}$.

Phase 2:

(1) If t is not divisible by T_{trans}, return $\emptyset$.

(2) Let $\{\text{reg}_1, \cdots, \text{reg}_k\}$ be the keys of frdb, with $k \leq B$. Construct $S = [\text{reg}_1, \cdots, \text{reg}_k, \text{reg}, \cdots, \text{reg}]$, where we add $B-k$ copies of reg $= (-1)$, a dummy registration info . Sample $\text{reg}_s, \text{reg}_r$ uniformly and independently at random from S.

(3) Let msg^{lb} be the (labeled) message with sequence number $\text{frdb}[\text{reg}_s]$.seqstart in $\text{out}[\text{reg}_s]$. If $\text{out}[\text{reg}_s]$ is empty, let $\text{msg}^{lb} = (-1, 0^{L_{\text{msg}}})$.

(4) $\text{sk}_w \leftarrow \text{frdb}[\text{reg}_s].\text{sk}[1]$. If $\text{reg}_s = \text{reg}_0$ or does not exist in frdb, set $\text{sk}_w \leftarrow \text{sk}_{\text{dummy}}$.

(5) $\text{seqrecv} \leftarrow \text{frdb}[\text{reg}_s].\text{seqrecv}$. If $\text{reg}_s = \text{reg}_0$ or does not exist in frdb, set $\text{seqrecv} \leftarrow -1$.

(6) Encrypt Messages with sk_w.

> **Hybrid 1**:
>
> If $\text{reg}_s \in \text{reg}_{\mathcal{H}}$ or $\text{reg}_s \notin \text{frdb}$,
>
> - $\text{ct}_{\text{msg}} \leftarrow \text{Sim}_{\text{ae}}(1^\lambda)$,
> - $\text{ct}_{\text{ack}} \leftarrow \text{Sim}_{\text{ae}}(1^\lambda)$,
>
> Else,

- $\text{ct}_{\text{msg}} = \Pi_{\text{ae}}.\text{Enc}(\text{sk}_w, \text{msg}^{lb})$.
- $\text{ct}_{\text{ack}} = \Pi_{\text{ae}}.\text{Enc}(\text{sk}_w, \text{ACK}(\text{seqrecv}))$.

(7) Let $\text{reg}_r = (i_r, \_)$. Formulate a PIR request for index i_r.

> **Hybrid 2**:
>
> If $\text{reg}_r \in \text{reg}_{\mathcal{H}}$,
>
> - $\text{ct}_{\text{Query}} \leftarrow \text{Sim}_{\text{pir}}(1^\lambda)$.
>
> Else,

- $\text{ct}_{\text{Query}}, \text{sk}_{\text{pir}} \leftarrow \Pi_{\text{pir}}.\text{Query}(1^\lambda, i_r)$.

(8) return $\text{req} = (\text{ct}_{\text{msg}}, \text{ct}_{\text{ack}}, \text{ct}_{\text{Query}})$.

[12] Skipping this step breaks consistent prefix.

(9) Remember reg_r.

$\text{Sim}_{\text{asphr}}.\text{ServerRPC}(t, \text{resp})$

Hybrid 1': if $\text{reg}_r \in \text{reg}_{\mathcal{H}}$, return.

(1) If resp $= \emptyset$, return.

(2) Parse resp $= (a_{\text{msg}}, a_{\text{ack}})$. Let $\text{reg}_r, \text{sk}_{\text{pir}}$ be defined in the last call to $\Pi_{\text{asphr}}.C.\text{Input}$.

(3) $\text{ct}_{\text{msg}} \leftarrow \Pi_{\text{pir}}.\text{Dec}(1^\lambda, \text{sk}_{\text{pir}}, a_{\text{msg}})$.

(4) $\text{ct}_{\text{ack}} \leftarrow \Pi_{\text{pir}}.\text{Dec}(1^\lambda, \text{sk}_{\text{pir}}, a_{\text{ack}})$.

(5) $\text{sk}_r \leftarrow \text{frdb}[\text{reg}_r].\text{sk}[0]$.

(6) Decipher the message.

(a) $\text{msg}^{lb} \leftarrow \Pi_{\text{ae}}.\text{Dec}(\text{sk}_r, \text{ct}_{\text{msg}})$.

(b) If $\text{msg}^{lb} = \bot$ or $\text{msg}^{lb}[0]$ is not $\text{frdb}[\text{reg}_r].\text{seqrecv}+1$, ignore the message.

(c) Add 1 to $\text{frdb}[\text{reg}_r].\text{seqrecv}$.

(d) Let msg be $\text{msg}^{lb}[1]$. Push msg to $\text{in}[\text{reg}_r]$.

(7) Decipher the ACK.

(a) $\text{ack} \leftarrow \Pi_{\text{ae}}.\text{Dec}(\text{sk}_r, \text{ct}_{\text{ack}})$.

(b) If ack $= \bot$ or ack is not the form $\text{ACK}(k)$ for some k, ignore the ack.

(c) Let ack $= \text{ACK}(k)$. If $k < \text{frdb}[\text{reg}_r].\text{seqstart}$, ignore the ack.

(d) $\text{frdb}[\text{reg}_r].\text{seqstart} \leftarrow k+1$. Remove the message with sequence number k from $\text{out}[\text{reg}_r]$.

We use a hybrid argument to show that the two views are indistinguishable. We start from the original implementation of the client methods $\Pi_{\text{asphr}}.C$, and use a hybrid argument to transform it into the simulator methods $\text{Sim}_{\text{asphr}}$. We call the real-world experiment Hyb_0.

First Hybrid: We add the statements marked Hybrid 1 and run the experiment in Definition 3.9. We call this modified experiment Hyb_1. To argue this preserves indistinguishability, suppose on the contrary that an adversary $\mathcal{A}$ and a distinguisher $\mathcal{D}$ can distinguish the view before and after the modification. Then we can build an adversary $\mathcal{A}_1^O$ and a distinguisher $\mathcal{D}'$ breaking Definition 5.3.

The adversary $\mathcal{A}_1^O$ simulates the real-world experiment in Figure 2. tTe key idea is to choose a powerful function $f : \Sigma^* \times \Sigma^* \to \Sigma^L$. For each pair of honest users $i, j \in \mathcal{H}$, define $\text{data}_r[\text{reg}_i] = (\text{frdb}[\text{reg}_i], \text{in}[\text{reg}_i], \text{out}[\text{reg}_i])$. $\mathcal{A}_1$ stores a log of changes to data **encrypted using** $\text{sk} = \text{sk}_{ij,r}$. Whenever it wants to access or update these data in the client simulation, it calls Eval_f. We choose f to decrypt the log, recover the plaintext of the fields, do the corresponding simulation, then re-encrypt the outputs and update to the log (f can use arg_p to determine which line it is on). For such an f, $\mathcal{A}_1^O$ can use f to perfectly simulate client updates on the sk-encrypted data[reg]. Finally, note that any client outputs computed from data are encrypted using $\text{sk}_{ij,r}$, so f can also perfectly simulate client outputs.

With this choice of f, we describe $\mathcal{A}_1^O$. For any $1 \le i, j \le N$, denote $\text{sk}_{ij,r} = \text{sk}_{ji,w} = \text{sk}_{i*N+j}$ and $\text{sk}_{i,\text{dummy}} = \text{sk}_i$, where thes$\text{k}_*$ on the right hand side denote the secret keys generated on line (1) of Definition 5.3. The step numbers below refer to the steps in Definition 3.9 unless otherwise indicated.

- Simulate step (1), (2), (3), and (4.b) identical to Definition 3.9.
- After step (2), for each pair of $i, j \in \mathcal{H}$, $\mathcal{A}_1$ "set"[13]
$$\text{GenSec}(\text{reg}_i, \text{reg}_j) = (\text{sk}_{ij,r}, \text{sk}_{ij,w}),$$
$$C_i.\text{sk}_{\text{dummy}} = \text{sk}_{i,\text{dummy}}.$$
$\mathcal{A}_1$ independently generates all other shared secrets using $\Pi_{\text{ae}}.\text{Gen}(1^\lambda)$.
- On step (4.a), $\mathcal{A}_1$ iterates over $i \in \mathcal{H}$. $\mathcal{A}_1$ simulates client i's action in $\Pi_{\text{asphr}}.\text{Input}$ verbatim until Phase 2 Step (6). In Step (6), $\mathcal{A}_1$ do casework based on if $\text{reg}_s \in \text{reg}_{\mathcal{H}}$.
  - If $\text{reg}_s \notin \text{reg}_{\mathcal{H}}$ and $\text{reg}_s \in \text{frdb}$, $\mathcal{A}_1$ knows the shared secrets sk_r and sk_w between reg_i and reg_s, so $\mathcal{A}_1$ can simulate this step verbatim.
  - Otherwise, let $(k, k') = (j * N + i, i * N + j)$ if $\text{reg}_s = \text{reg}_j$ where $j \in \mathcal{H}$, else let $(k, k') = (i, i)$. $\mathcal{A}_1$ sets $(\text{arg}_{\text{ct}}, \text{arg}_p)$ so that
  $$\text{Eval}_f(\text{sk}_k, \text{sk}_{k'}, \text{arg}_{\text{ct}}, \text{arg}_p)$$
  simulates the original step (6) to compute ct_{msg}, outputs $(k, k', \text{arg}_{\text{ct}}, \text{arg}_p)$, then moves to line (4.b) in Definition 5.3. $\mathcal{A}_1$ sets $\text{ct}_{\text{msg}} = \text{ct}_r^b$ to be the output of line (4.b). $\mathcal{A}_1$ repeats this for ct_{ack}.
- On step (4.c), $\mathcal{A}_1$ simulates client action $\Pi_{\text{asphr}}.\text{ServerRPC}$ using Eval_f.

We note that in $\text{Real}_{\text{Eval}}^{\mathcal{A}}$, $\mathcal{A}_1$ perfectly simulates the view of $\mathcal{A}$ in Hyb_0. In $\text{Ideal}_{\text{Eval}}^{\mathcal{A},\text{Sim}_{\text{ae}}}$, $\mathcal{A}_1$ perfectly simulates the view of $\mathcal{A}$ in Hyb_1. Thus, $\mathcal{A}_1$ just need to output the view of $\mathcal{A}$ at the end of the simulation, and the same distinguisher $\mathcal{D}' = \mathcal{D}$ can distinguish between the views of $\mathcal{A}_1$ in the real- and ideal-world. This contradicts our assumption that Π_{ae} satisfies Definition 5.3. Therefore, we finally conclude that
$$\text{Hyb}_0 \equiv_c \text{Hyb}_1.$$

Corollary of First Hybrid: We add the statements marked Hybrid 1' and run the experiment in Definition 3.9. We call this modified experiment $\text{Hyb}_{1'}$.

We argue that this hybrid does not change the adversary's view at all. $\Pi_{\text{asphr}}.C.\text{ServerRPC}$ only updates fields of $\text{data}[\text{reg}_r]$. In Hyb_1, for any $\text{reg} \in \text{reg}_{\mathcal{H}}$, the fields of data[reg] does not affect the

[13] In other word, $\mathcal{A}_1$ simulates the honest users as if $\text{GenSec}(\text{reg}_i, \text{reg}_j) = \text{sk}_{ij}$ and $C_i.\text{sk}_{\text{dummy}} = \text{sk}_{i,\text{dummy}}$. $\mathcal{A}_1$ does not have access to sk_{ij}.

client's output. So adding the statement' Hybrid 1' does not affect the client's output. We conclude that

$$\mathsf{Hyb}_1 \equiv_c \mathsf{Hyb}_{1'}.$$

Second Hybrid: We add the statements marked Hybrid 2 and run the experiment in Definition 3.9. We call this modified experiment Hyb_2.

We argue by contradiction that $\mathsf{Hyb}_{1'}$ and Hyb_2 are indistinguishable. Suppose that an adversary $\mathcal{A}$ and a distinguisher $\mathcal{D}$ can $\mathsf{Hyb}_{1'}$ and Hyb_2. Then we can build an adversary $\mathcal{A}_2$ and a distinguisher $\mathcal{D}'$ breaking Definition 5.6.

$\mathcal{A}_2$ simulates a modified version of Hyb_1. $\mathcal{A}_2$ simulates line (1, 2, 3, 4b, 4c) in Definition 3.9 verbatim as in both $\mathsf{Hyb}_{1'}$ and Hyb_2. On line (4a), $\mathcal{A}_2$ simulates everything but Phase 2 step (7) verbatim. When it reaches Phase 2 step (7), if $\mathsf{reg}_r \notin \mathsf{reg}_{\mathcal{H}}$ then it simulates this step verbatim as well. If $\mathsf{reg}_r \in \mathsf{reg}_{\mathcal{H}}$, then $\mathcal{A}_2$ returns i_r and exits line (1a) of the experiment in Definition 5.6. Let ct_r^b be the return value of line (1b). $\mathcal{A}_2$ sets $\mathsf{ct}_{\mathsf{Query}} = \mathsf{ct}_r^b$ and continues simulation.

Contrary to the first hybrid, $\mathcal{A}_2$ knows all the key exchange secret keys. Furthermore, if $\mathsf{reg}_r \notin \mathsf{reg}_{\mathcal{H}}$, $\mathcal{A}_2$ knows $\mathsf{sk}_{\mathsf{pir}}$ generated by $\Pi_{\mathsf{asphr}}.C.\mathsf{Input}$. If $\mathsf{reg}_r \notin \mathsf{reg}_{\mathcal{H}}$, then $\mathcal{A}_2$ does not know the $\mathsf{sk}_{\mathsf{pir}}$, but this $\mathsf{sk}_{\mathsf{pir}}$ will never be used since Hybrid 1' ensures that $\Pi_{\mathsf{asphr}}.C.\mathsf{ServerRPC}$ is skipped. Thus, we conclude that $\mathcal{A}_2$ can complete the simulation.

In $\mathsf{Real}_{\mathsf{pir}}^{\mathcal{A}}$, $\mathcal{A}_2$ simulates $\mathsf{Hyb}_{1'}$ verbatim, while in $\mathsf{Ideal}_{\mathsf{pir}}^{\mathcal{A},\mathsf{Sim}}$, $\mathcal{A}_2$ simulates Hyb_2 verbatim. Thus, $\mathcal{A}_2$ just need to output the view of $\mathcal{A}$ at the end of the simulation, and the same distinguisher $\mathcal{D}' = \mathcal{D}$ can distinguish between the views of $\mathcal{A}_1$ in the real- and ideal-world. This contradicts our assumption that Π_{pir} satisfies Definition 5.6. Therefore, we conclude that

$$\mathsf{Hyb}_{1'} \equiv_c \mathsf{Hyb}_2.$$

Third Hybrid: We add the statements marked Hybrid 3, and run the experiment in Definition 3.9. We call this modified experiment Hyb_3. This modification does not change the view of the adversary at all. After the changes in Hybrid 1, 1', and 2, for any $i, j \in \mathcal{H}$, the contents of $C_i.\mathsf{frdb}[\mathsf{reg}_j], C_i.\mathsf{in}[\mathsf{reg}_j], C_i.\mathsf{out}[\mathsf{reg}_j]$ does not affect client i output. (In particular, whether reg_j lies in frdb or not does not affect the distribution of reg_s or reg_r). For any $\mathsf{reg} \notin \mathsf{reg}_{\mathcal{H}}$, Hybrid 3 does not affect updates to $C_i.\mathsf{frdb}[\mathsf{reg}], C_i.\mathsf{in}[\mathsf{reg}]$, and $C_i\mathsf{out}[\mathsf{reg}]$. So we conclude

$$\mathsf{Hyb}_2 \equiv_c \mathsf{Hyb}_3.$$

Finally, we conclude that

$$\mathsf{Hyb}_0 \equiv_c \mathsf{Hyb}_3$$

Note that $\mathsf{Hyb}_0 = \mathsf{Real}_{\mathsf{msg}}^{\mathcal{A}}$ and $\mathsf{Hyb}_3 = \mathsf{Ideal}_{\mathsf{msg}}^{\mathcal{A},\mathsf{Sim}_{\mathsf{asphr}}}$,. Therefore, Π_{asphr} satisfies Definition 3.12.

7 IK-CCA IMPLIES EVAL-SECURITY

In this section, we propose a symmetric key analog of the IK-CCA security introduced by Bellare et. al. in [Bel+01, Definition 1]. We then show that IK-CCA security implies the eval-security Definition 5.3 needed for the security of our system.

Recall that an AE scheme Π_{ae} consists of algorithms (Gen, Enc, Dec) with syntax defined in Section 5.1.1.

Definition 7.1 (IK-CCA)**.** Consider the following distinguishing experiment, where $N = N(\lambda)$ is polynomial in λ.

Distinguishing Game $\mathbf{Exp}_{\mathsf{IK-CCA}}^{\mathcal{A}}$

(1) $\mathsf{sk}_0, \mathsf{sk}_1 \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{Enc}(\{\mathsf{sk}_i\},\cdot),\mathsf{Dec}(\{\mathsf{sk}_i\},\cdot)}(1^\lambda)$.
(3) $b \leftarrow U(\{0,1\})$.
(4) $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}_b, m_b)$.
(5) $b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\{\mathsf{sk}_i\},\cdot),\mathsf{Dec}(\{\mathsf{sk}_i\},\cdot)}(\mathsf{ct})$.

Figure 8: Distinguishing game for IK-CCA security.

Let $\mathsf{ct}_{\mathsf{Query}}$ denote the queries $\mathcal{A}$ sent to both decryption oracles on line (5). We define the output of the experiment as 1 if both $b' = b$ and $\mathsf{ct} \notin \mathsf{ct}_{\mathsf{Query}}$, and 0 otherwise.

Then we say the AE system Π_{ae} is **IK-CCA secure** if for any p.p.t with oracle adversary $\mathcal{A}^O$, we have

$$\mathbb{P}(\mathbf{Exp}_{\mathcal{A}}^{\mathsf{IK-CCA}} = 1) \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Assume that Π_{ae} is IK-CCA. We now show that Π_{ae} is Eval-secure. We recall the relevant definitions, where the simulator simply outputs $\mathsf{Enc}(\mathsf{sk}_0, 0^L)$ for $\mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda)$.

Real-World Experiment $\mathsf{Real}_{\mathsf{Eval}}^{\mathcal{A}}$

(1) For i from 1 to N, $\mathsf{sk}_i \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) For r from 1 to R
  (a) $i, \mathsf{ct}, \mathsf{arg}_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}_r^0 \leftarrow \mathsf{Eval}_f(\mathsf{sk}_i, \mathsf{ct}, \mathsf{arg}_p)$.
  (c) $\mathcal{A}$ stores ct_r^0.

Ideal-World Experiment $\mathsf{Ideal}_{\mathsf{Eval}}^{\mathcal{A}}$

(1) For i from 1 to N, $\mathsf{sk}_i \leftarrow \mathsf{Gen}(1^\lambda)$.
(2) For r from 1 to R
  (a) $i, \mathsf{ct}, \mathsf{arg}_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{ct}_r^1 \leftarrow \mathsf{Enc}(\mathsf{sk}_0, 0^L)$.
  (c) $\mathcal{A}$ stores ct_r^1.

Figure 9: Recap of Definition 5.3.

We use a standard hybridizing argument to show that the views of $\mathcal{A}$ are indistinguishable.

Definition 7.2. Define the Oracle $\widetilde{\mathsf{Eval}_{k,f}}(\mathsf{sk}_i, \cdot)$ as follows. For the first k time it behaves exactly the same as $\mathsf{Eval}_f(\mathsf{sk}_i, \cdot)$. After k calls, it outputs $\mathsf{Enc}(\mathsf{sk}_1, 0^L)$ for $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$.

For each $k \geq 0$, define $\mathsf{Hyb}_{\mathsf{Eval},j}$ as the Real World Experiment $\mathsf{Real}_{\mathsf{Eval}}^{\mathcal{A}}$ with $\mathsf{Eval}_f(\{\mathsf{sk}_i\}, \cdot)$ replaced by $\widetilde{\mathsf{Eval}_{k,f}}(\{\mathsf{sk}_i\}, \mathsf{Eval}, \cdot)$.

Lemma 7.3. *For any $0 \le k \le R$, we have*

$$\mathsf{Hyb}_{\mathsf{Eval},k+1} \equiv_c \mathsf{Hyb}_{\mathsf{Eval},k}.$$

Proof. Suppose the contrary. Let D be any distinguisher such that

$$\mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k+1})) - \mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k})) \ge \mathrm{poly}(\lambda)^{-1}.$$

We design an adversary $\mathcal{A}'$ to win the IK-CCA game. Let $i^* \in [N]$ be any index. On line (2) of $\mathbf{Exp}^{\mathcal{A}'}_{\mathsf{IK-CCA}}$, the adversary $\mathcal{A}'$ simulates $\mathsf{Hyb}_{\mathsf{Eval},k}$ with $\mathsf{sk}_{i^*} = \mathsf{sk}_1$, and all other sk_i randomly generated. $\mathcal{A}'$ simulates $\mathcal{A}$ until the $k+1$-th call to the oracle $\widetilde{\mathsf{Eval}}_{j,f}$. Let $i_{k+1}, j_{k+1}, \mathsf{ct} = \{\mathsf{ct}_\ell\}, \mathsf{arg}_p$ be the output of $\mathcal{A}$. If $i_{k+1} \ne i^*$, then $\mathcal{A}'$ just guess randomly. Otherwise, $\mathcal{A}'$ use the Dec oracle to compute $m'_\ell = \mathsf{Dec}(\mathsf{sk}_{j^*}, \mathsf{ct}_\ell)$. Then it outputs $m_0 = 0^L$, $m_1 = f(\{m'_\ell\}, \mathsf{arg}_p)$, and exits line (2) of $\mathbf{Exp}^{\mathcal{A}'}_{\mathsf{IK-CCA}}$. Let ct be the output of line (5). $\mathcal{A}'$ uses ct as the output of $\widetilde{\mathsf{Eval}}_{k,f}$, then continue to simulate $\mathsf{Hyb}_{\mathsf{Eval},k}$ until the end. $\mathcal{A}'$ returns $b' = 1$ iff D accepts the resulting view.

We condition on $i_{k+1} = i^*$. If $b = 1$, then $\mathcal{A}'$ perfectly simulates $\mathsf{Hyb}_{\mathsf{Eval},k+1}$, while if $b = 0$, then $\mathcal{A}'$ perfectly simulates $\mathsf{Hyb}_{\mathsf{Eval},k}$. Furthermore, in line (6) of Definition 7.1, $\mathcal{A}'$ never use the Dec oracle, so $\mathsf{ct}_{\mathsf{Query}} = \emptyset$. Therefore, we have

$$\begin{aligned}
&\mathbb{P}(\mathbf{Exp}^{\mathcal{A}'}_{\mathsf{IK-CCA}} = 1) \\
&= \frac{1}{2} + \mathbb{P}(b' = 1, b = 1, i_{k+1} = i^*) - \mathbb{P}(b' = 1, b = 0, i_{k+1} = i^*) \\
&= \frac{1}{2} + \frac{1}{2}\mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k+1}), i_{k+1} = i^*) \\
&\quad - \frac{1}{2}\mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k}), i_{k+1} = i^*).
\end{aligned}$$

By IK-CCA, there exists a negligible function $\mu(\lambda)$ such that

$$\mathbb{P}(\mathbf{Exp}^{\mathcal{A}'}_{\mathsf{IK-CCA}} = 1) \le \frac{1}{2} + \mu(\lambda).$$

Summing over all $i^* \in [N]$, we obtain a contradiction

$$\mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k+1})) - \mathbb{P}(D(\mathsf{Hyb}_{\mathsf{Eval},k})) \le \mathrm{negl}(\lambda).$$

□

Since $\mathsf{Hyb}_{\mathsf{Eval},0}$ is identical to $\mathsf{Ideal}^{\mathcal{A}}_{\mathsf{Eval}}$, and $\mathsf{Hyb}_{\mathsf{Eval},R}$ is identical to $\mathsf{Real}^{\mathcal{A}}_{\mathsf{Eval}}$, we conclude that $\mathsf{Ideal}^{\mathcal{A}}_{\mathsf{Eval}}$ is indistinguishable with $\mathsf{Real}^{\mathcal{A}}_{\mathsf{Eval}}$. So Π_{ae} is Eval-Secure as desired.

8 CONCLUSION AND OPEN QUESTIONS

Our paper is an attempt to rigorously justify the security of a real-world metadata-private messaging system. The proof can convince users that our messaging service satisfies the security properties we promise, as well as help us find and fix existing security vulnerabilities in our implementation.

Nevertheless, many open problems remain unexplored in this paper. Below, we give a few examples.

- In Section 5.1.2, we assume that pairs of honest users can obtain pre-distributed symmetric keys from a trusted third party. We believe that the key exchange protocols proposed in [LZA22b] can replace the third party, but we have no rigorous proof of this property. Is it possible to develop a theory similar to the CK model [CK01] that addresses the metadata security of key exchange protocols?
- To prevent CF attacks, the core protocol described in Section 5.3 is very inefficient in practice. In Appendix A, we describe a method we call prioritization that speeds up the core protocol in exchange for some limited metadata leakage. Can we find more efficient protocols that defend against CF attacks fully?
- Is it possible to integrate PIR protocols into the universal composability framework [Can20]? This would greatly simplify the security proofs of future real-world MPM systems, but we are unaware of any attempt at this.
- Is it possible to formalize the proof using automated proof assistants, such as Coq or Lean?

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.

[Ali+21] Asra Ali et al. "Communication–Computation Trade-offs in PIR". In: *30th USENIX Security Symposium (USENIX Security 21).* USENIX Association, Aug. 2021, pp. 1811–1828. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/ali.

[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.

[Ang18] Sebastian Angel. "Unobservable communication over untrusted infrastructure". PhD thesis. The University of Texas at Austin, 2018.

[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: *Advances in Cryptology — ASIACRYPT 2001.* Ed. by Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 566–582.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. "Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing". In: *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology.* CRYPTO '00. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 55–73. ISBN: 3540679073.

[Can20] Ran Canetti. "Universally Composable Security". In: *J. ACM* 67.5 (Sept. 2020). ISSN: 0004-5411. DOI: 10.1145/3402457. URL: https://doi.org/10.1145/3402457.

[CBM15] 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.

[CF10] 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.

[CK01] Ran Canetti and Hugo Krawczyk. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels". In: *Advances in Cryptology — EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 453–474.

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

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

[Hen+22] Alexandra Henzinger et al. *One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval*. Cryptology ePrint Archive, Paper 2022/949. https://eprint.iacr.org/2022/949. 2022. URL: https://eprint.iacr.org/2022/949.

[Ish+04] Yuval Ishai et al. "Batch Codes and Their Applications". In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '04. Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 262–271. ISBN: 1581138520. DOI: 10.1145/1007352.1007396. URL: https://doi.org/10.1145/1007352.1007396.

[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.

[Koc+19] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002.

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

[LZA22a] Arvid Lunnemark, Shengtong Zhang, and Sualeh Asif. *Anysphere Client*. Version 0.1.15. Aug. 2022. URL: https://github.com/anysphere/client.

[LZA22b] Arvid Lunnemark, Shengtong Zhang, and Sualeh Asif. *Anysphere: Private Communication in Practice*. 2022. URL: https://anysphere.co/anysphere-whitepaper.pdf.

[MW22] Samir Jordan Menon and David J. Wu. *Spiral: Fast, High-Rate Single-Server PIR via FHE Composition*. Cryptology ePrint Archive, Paper 2022/368. https://eprint.iacr.org/2022/368. 2022. URL: https://eprint.iacr.org/2022/368.

[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.

[Ter13] Doug Terry. "Replicated Data Consistency Explained through Baseball". In: *Commun. ACM* 56.12 (Dec. 2013), pp. 82–89. ISSN: 0001-0782. DOI: 10.1145/2500500. URL: https://doi.org/10.1145/2500500.

[WS96] David Wagner and Bruce Schneier. "Analysis of the SSL 3.0 Protocol". In: *Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*. WOEC'96. Oakland, California: USENIX Association, 1996, p. 4.

A PRIORITIZATION

Anysphere's client-side source code is at [LZA22a]. Our implementation mostly adheres to the core protocol described in Section 5.3. However, the core protocol is very inefficient as stated: for each pair of communicating users i, j, each PIR query from i has a probability of B^{-1} to retrieve from j's mailbox in the database, and j has a probability of B^{-1} to put the message to i there. Therefore, the expected time between j sending a message and i receiving a message is $T_{\text{trans}}B^2$. Due to the high cost of handling PIR queries, our system takes $T_{\text{trans}} \approx 30s$. If a user is allowed to have 20 friends, this results in a latency of three hours per single-trip delivery, which is not practical.

We can use efficient batch PIR retrieval [BIM00; Ish+04; AS16; LT21], which allows a client to retrieve from all friends at once with $O(N)$ server work. However, since each client can only deposit one message into their mailbox at a time, the single-trip latency is still about $B \cdot 3T_{\text{trans}} \approx 600s$, where the constant 3 accounts for the additional work the server has to do for batch retrieval. If we expand the mailbox to allow each user to deposit multiple messages at once, the cost of PIR queries scales linearly with the size of the mailbox, so it is hard to save more work.

In our code, we use an optimization we name "prioritization" to address this problem. When choosing reg_s and reg_r in Phase 2 step (2) of $\mathsf{Input}()$, the clients select a set of "prioritized users", and select reg_s and reg_r from these users. Ideally, the prioritized users are friends who are actively conversing with the client.

More formally, our clients support a **priority function**

$$C.\mathsf{getPr}(\text{reg}) \to (p_{\text{reg},s}, p_{\text{reg},r})$$

that takes in a registration info reg and outputs priorities $p_{\text{reg},s}, p_{\text{reg}_r}$ for sending and receiving messages from and to the user. Phase 2 of $\Pi_{\text{asphr}}.C.\mathsf{Input}(t, \mathcal{I})$ is modified as follows.

$\Pi^{\text{getPr}}_{\text{asphr}}$.C.Input$(t, \mathcal{I})$, Phase 2

---

(1) If t is not divisible by T_{trans}, return $\emptyset$.

(2) Let $\{\text{reg}_1, \cdots, \text{reg}_k\}$ be the keys of frdb. Let $\text{reg}_0 = (-1)$ be a dummy registration info.

  (a) For $i = 0, 1, \cdots, k$, $(p_{i,s}, p_{i,r}) \leftarrow C.\mathsf{getPr}(\text{reg}_i)$.

  (b) Sample $i_s \sim \{0, 1, \cdots, k\}$ such that $\mathbb{P}(i_s = i) \propto p_{i,s}$.

(c) Let $\mathrm{reg}_s = \mathrm{reg}_{i_s}$.

(d) Sample $i_r \sim \{0, 1, \cdots, k\}$ such that $\mathbb{P}(i_r = i) \propto p_{i,r}$.

(e) Let $\mathrm{reg}_r = \mathrm{reg}_{i_r}$.

(3) Every step below is unchanged.

We call this modified protocol the **core protocol with prioritization**, and denote it

$$\Pi^{\mathsf{getPr}}_{\mathrm{asphr}}.$$

Example 1: If the client sets $p_{i,s} = p_{i,r} = 1$ for $i \geq 1$ and $p_{0,s} = p_{0,r} = B - k$, we recover the original protocol defined in Section 5.3.

Example 2: If the client sets $p_{i,s} = p_{i,r} = 1$ for $i \geq 1$ and $p_{0,s} = p_{0,r} = 0$, we get a slightly more efficient protocol where we select a random friend to send to and retrieve from. The tradeoff is a small metadata leakage already described in [ALT18]: an adversary can learn the number of friends a friend has by measuring latency.

Example 3: Some priority functions can lead to more serious metadata leakage. For example, consider the "intuitive" optimization which assigns higher priority to friends with non-empty outbox. Suppose a user i disconnects from the internet. If user j is user i's friend and sends i at least one message after the disconnect, j's outbox to i would always be nonempty since j does not receive ACKs from i, so j's latency with other users would increase as well. The latency increase propagates across the social graph until it reaches the compromised clients. Therefore, a very powerful adversary can potentially learn the whole social graph by DoSing each client and observing how the latency of the compromised clients changes. It could then plausibly map this social graph onto any other public friend graph (say Facebook's) to determine exactly who is talking to whom.

Fortunately, we can provably prevent metadata leakage by imposing the following condition on the priority function.

Definition A.1. The priority function $C.\mathsf{getPr}(\mathrm{reg})$ is **static** if its output on timestep T is uniquely determined by the argument reg and the user inputs $\mathcal{I}_t$ to C on timesteps $t \leq T$. In other words, there exists a function getPr such that on timestep T we have

$$C.\mathsf{getPr}(\mathrm{reg}) = \mathsf{getPr}(\{\mathcal{I}_t\}_{t \in [T]}, \mathrm{reg}).$$

If $C.\mathsf{getPr}(\cdot)$ is a static priority function, we define its **leakage** as

$$\mathsf{Leak}_{\mathsf{getPr}}(\{\mathcal{I}_{i,t}\}_{i \in \mathcal{H}, t \in [T]}, \mathrm{reg}_{\mathcal{H}}) = (S_1, S_2).$$

where

$$\begin{aligned} S_1 = \{ & (i, t, \mathrm{reg}, p_s, p_r) : i \in \mathcal{H}, t \in [T], \mathrm{reg} \notin \mathrm{reg}_{\mathcal{H}} \sqcup \mathrm{reg}_0 \\ & \wedge \exists t' \leq t, \mathsf{TrustEst}(\mathrm{reg}) = \mathcal{I}_{i,t'} \\ & \wedge (p_s, p_r) = \mathsf{getPr}(\{\mathcal{I}_{i,t'}\}_{t' \in [t]}, \mathrm{reg}).\} \end{aligned}$$

and

$$\begin{aligned} S_2 = \{ & (i, t, p_{s,\mathcal{H}}, p_{r,\mathcal{H}}) : i \in \mathcal{H}, t \in [T], \\ & \wedge \exists t' \leq t, \mathrm{reg} \notin \mathrm{reg}_{\mathcal{H}} \sqcup \mathrm{reg}_0, \mathsf{TrustEst}(\mathrm{reg}) = \mathcal{I}_{i,t'} \\ & \wedge (p_{s,\mathcal{H}}, p_{r,\mathcal{H}}) = \sum_{\mathrm{reg} \in \mathrm{reg}_{\mathcal{H}} \sqcup \mathrm{reg}_0} \mathsf{getPr}(\{\mathcal{I}_{i,t'}\}_{t' \in [t]}, \mathrm{reg})\} \end{aligned}$$

where $\mathrm{reg}_0 = (-1)$ is the dummy registration info.

For each user with compromised friend, the leakage consists of the priority of each compromised friend, plus the sum of the priority of all honest friends. Most importantly, the leakage contains no information about users with no compromised friend. This is a desirable property in practice: the compromise of a single user would only affect the metadata privacy of them and their friends, instead of jeopardizing the security of the whole network.

Using the same method in Section 6, we can prove the correctness, security and integrity of the prioritized protocol $\Pi^{\mathsf{getPr}}_{\mathrm{asphr}}$.

Theorem A.2. *Suppose the priority function $C.\mathsf{getPr}(\cdot)$ is static, and its output is always in the range $[P_L, P_U]$ for positive constants P_L, P_U. Then the prioritized core protocol $\Pi^{\mathsf{getPr}}_{asphr}$ guarantees correctness(Definition 3.8) and integrity(Definition 3.15). $\Pi^{\mathsf{getPr}}_{asphr}$ is also SIM-secure (Definition 3.12), but with the leakage function in the ideal world experiment Definition 3.11 replaced by*

$$\mathsf{Leak}'(\{\mathcal{I}_{i,t}\}_{i \in \mathcal{H}, t \in [T]}, \mathrm{reg}_{\mathcal{H}}) = (\mathsf{Leak}_f, \mathsf{Leak}_m, \mathsf{Leak}_{\mathsf{getPr}})$$

where Leak_f and Leak_m are defined in Definition 3.10 and $\mathsf{Leak}_{\mathsf{getPr}}$ is defined in Definition A.1.

The proofs of correctness and integrity are the same. The proof of security is very similar, except we change Hybrid 3 to ensure that compromised friends are chosen as reg_s and reg_r with the probability defined in the modified $\Pi^{\mathsf{getPr}}_{\mathrm{asphr}}$.

We are exploring the best priority function. Currently, the client assigns higher priorities $(p_{\mathrm{reg},s}, p_{\mathrm{reg},r})$ to friend reg if the user sent message to reg more recently. During a synchronous conversation between two users, we can achieve a latency of almost $2T_{\mathrm{trans}} \approx 60s$ because the users put each other on high priority.

On the other hand, if a user has a compromised friend, the adversary can theoretically learn their number of friends and the number of active conversations they participate in using the CF attack described in [ALT18]. Angel et al. argues in the same paper that the attack is hard to execute in practice. Therefore, we believe that the amount of information leaked is small compared to the massive efficiency benefit, so our use of prioritization is justified.