Formal Security Definition of Anysphere

ABSTRACT

We prove the security of Anysphere, a metadata-private messaging (MPM) system. Our main contributions are: (1) We describe a vulnerability of existing MPM implementations through a variation of the compromised-friend (CF) attack proposed by Angel et. al. Our attack can compromise the exact metadata of any conversations between honest users. (2) We present a security definition for MPM systems assuming compromised friends. (3) We prove that Anysphere's core protocol satisfies our security definition.

Contents

1 PURPOSE

Anysphere is a metadata-private messaging (MPM) system. In Anysphere's whitepaper [LZA22, Section 3], we describe our core protocol at a high level. This document contains a security definition and a proof to rigorously show that Anysphere's core protocol satisfies the metadata privacy we promise.

Existing security proofs of MPMs (such as [CGF10; CGBM15; 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 find these proofs unsatisfying 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 makes friend with a malicious user, then the metadata of conversations between honest users might be compromised even with a secure PIR system. To our knowledge, no proofs exist that show immunity against CF attacks[1]. In fact, we found a more powerful CF attack, described in Section 4, as we write this paper.
- Our system 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 other MPM systems like 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, big companies might wish rounds run faster to receive timely updates, while individual clients 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 of this decision.
- The above mentioned MPM systems also lack a mechanism to detect and retransmit lost or shuffled packages. To address this issue, We introduce an ACK mechanism in Anysphere. As we will see below, justifying the security of this mechanism is far from trivial.

Our 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. Our definition takes into consideration both asynchronous rounds and user inputs. A system satisfying our definition guarantees metadata privacy against a malicious server and an arbitrary set of clients, including potential CF attacks.

In Section 4, we describe the new CF attack which we name the PIR Replay Attack. If an honest user A has a compromised friend, our attack can compromise the metadata of any PIR requests sent by user A, even if the messaging system satisfies Pung's UO-ER security definition.

In Section 5, we describe the Anysphere core protocol in pseudocode. We also define exact security requirement on the cryptographic packages we use. In particular, we introduce a novel security definition that we require of our symmetric key cryptosystem. In Section 6 we prove that the protocol defined in Section 5 satisfies the security definition in Section 3.

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, Appendix C] assumes honest users only ever talk to honest users.

- Given an oracle $O(x, \cdot)$ and a series $\{x_i\}$, define $O(\{x_i\}, \cdot)$ as the oracle whose input takes an extra argument j and outputs $O(\{x_i\}_i, j, \text{arg}) = O(x_j, \text{arg})$.
- When we say two experiments are indistinguishable, we mean the view 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 GENERAL DEFINITIONS

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 application should hide metadata of conversations between honest users from a powerful adversary that controls the server, the network, and a subset of clients.

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

Definition 3.1. A **timestep** is a basic unit of time in our system. 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 $\mathcal{F}$ of the client.

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 GUI can simply store such messages locally and display them to the user.

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

- $\emptyset$: noop.
- $\mathsf{TrustEst}(\mathsf{reg})$: Add the user identified by reg as a friend, and enable the two parties to start a conversation.
- $\mathsf{Send}(\mathsf{reg}, \mathsf{msg})$: 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 $L_{\mathsf{msg}} \approx 1$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 will always use the registration info as the "address" in the messaging system. For example, we will 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 [LZA22, Figure 6].

Definition 3.5. A **Messaging System** consists of the following polynomial time algorithms.

Client Side Algorithms for the stateful client C.

- $C.\mathsf{Register}(1^\lambda, i, N) \to \mathsf{reg}$. 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.
- $C.\mathsf{Input}(t, \mathcal{I}) \to \mathsf{req}$. This algorithm handles a user input $\mathcal{I}$. It updates the client storage to reflect the new input, then issues a (possibly empty) request req to the server.
- $C.\mathsf{ServerRPC}(t, \mathsf{resp})$. This algorithm handles the server's response resp and updates client storage.
- $C.\mathsf{GetView}() \to V$. This algorithm outputs the view of the client (Definition 3.2). Its output is passed to the GUI and displayed to the user.

Server Side Algorithms for the stateful server S.

- $S.\mathsf{InitServer}(1^\lambda, N)$. This algorithm takes in the security parameter λ, the number of clients N, and initializes the server-side database D_S.

[2] For simplicity, we assume this holds even for adversarial inputs.

- S.ClientRPC$(t, \{\mathsf{req}_i\}_{i=1}^N) \rightarrow \{\mathsf{resp}_i\}_{i=1}^N$. 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 client.

Now we can describe some desired properties of MPM systems. In the rest of this section, we will look at three properties we wish Anysphere to satisfy: Correctness, Metadata privacy, and Integrity.

3.1 Correctness

First, we describe how the server and clients interact when all parties behave honestly. We call this scenario the Honest Server Experiment.

Definition 3.6.
The honest server experiments take 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 $\mathcal{I}_{i,t}$.

Let S denote the server machine, let $\{C_i\}_{i=1}^N$ denote the client machines. The experiment is described below.

Honest Server Experiment

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

Figure 1: The Honest Server Experiment for Messaging System

[**Arvid**: Do we want an adversary to be able to control some of the clients? Ideally, I think we would want to because we want to guarantee resistance against denial of service attacks from clients] [**stzh**: Good idea. I'll come back to this after I finish the security proof in the current iteration.]

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.

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

$$\mathcal{F}_j \cap \{\mathsf{reg}_i\}_{i\in[N]} = \{\mathsf{reg}_k : \exists t \in [T], \mathsf{TrustEst}(\mathsf{reg}_k) = \mathcal{I}_{j,t}\},$$

and

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

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

1) The definition allow $\mathcal{F}$ to contain registration infos 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 GUI 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 world experiment, any $j \in [N]$, and any positive integer $T_0 \leq T$. Let $V_j \leftarrow C_j$.GetView() be the view of client j after timestep T_0 of the Honest World Experiment, the messaging system satisfies the following two properties with probability $1 - \mathsf{negl}(\lambda)$.

1) **Consistent Prefix**: V is identical to the correct view of the client j if a prefix of user inputs have been executed on each client machine. More formally, for any $j \in [N]$, there exists a map $t : [N] \rightarrow [T_0]$ such that V is a correct view of client j under inputs $(\{\mathsf{reg}_i\}_{i\in[N]}, \{\mathcal{I}'_{i,t}\})$ where we define

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

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

3.2 Metadata Security and Integrity

Next, we define security with an active adversary. We first recap our threat model, defined in [LZA22, Section 2.2]. We enable the adversary to

1) Control all servers and the entire internet. In the definition below, the adversary read all requests from honest clients, computes any

polynomial time function over them, and returns any response to each client. Therefore, the adversary can perform most record-layer attacks considered by the security community, such as eavesdropping, traffic analysis, and active attacks like cut-and-paste, deep packet inspection, and replay attacks [WS96].

2) 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 enable the adversary to

1) Access or launch side-channel attacks on client machines such as timing attacks and Spectre [Koc+19]. We assume timing data and intermediate state of client execution are invisible to the adversary.

2) 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 definition with crucial information "redacted". We say the messaging system is secure if the views of the adversary under the two experiments are indistinguishable.

We first define the real-world experiment.

Definition 3.9. The real-world experiment uses the parameters λ, N, T in Definition 3.6. Furthermore, let $\mathcal{A}$ be a stateful p.p.t. adversary. Let $\mathcal{H}$ denote the set of honest clients the adversary chooses. Denote $\mathsf{reg}_{\mathcal{H}} = \{\mathsf{reg}_i\}_{i \in \mathcal{H}}$ to be the registration info of honest clients. The experiment $\mathsf{Real}^{\mathcal{A}}_{\mathsf{msg}}(1^\lambda)$ is described in Figure 2.

Real World Experiment $\mathsf{Real}^{\mathcal{A}}_{\mathsf{msg}}(1^\lambda)$

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

Figure 2: Real World Experiment for 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

(1) Trust establishment with compromised clients.

(2) Messages sent to the compromised clients.

The formal definition is below.

Definition 3.10. Let $\mathsf{reg}_{\mathcal{H}}$ 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** $\mathsf{Leak}(\{\mathcal{I}_{i,t}\}, \mathsf{reg}_{\mathcal{H}})$ as

$$\mathsf{Leak}(\{\mathcal{I}_{i,t}\}, \mathsf{reg}_{\mathcal{H}}) = \{\mathcal{I}_{i,t} : (i,t) \in \mathsf{Leak}_f \cup \mathsf{Leak}_m\}$$

where

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

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

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 $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}_{\mathsf{msg}}(1^\lambda)$ is described in Figure 3.

Ideal World Experiment $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}_{\mathsf{msg}}(1^\lambda)$

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

Figure 3: Ideal World Experiment for Messaging System

Finally, we define metadata security.

Definition 3.12. We say a messaging system is **SIM-metadata secure** if for any N and T polynomially bounded in λ, there exists a p.p.t simulator Sim such that for any p.p.t adversary $\mathcal{A}$, the view of $\mathcal{A}$ is indistinguishable in $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\lambda)$.

The simulator based definition is a relatively new way of writing security definitions. For readers more accustomed to indistinguishability-based security definitions, we include an equivalent security definition.

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

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

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

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

IND Experiment

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

Figure 4: IND Experiment for Messaging System

Using the argument in [SW21, Appendix A], we can show that IND-metadata security is equivalent to SIM-metadata security.

Finally, we define the notion of integrity. Informally, while a malicious server can DoS users, it shouldn't 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. For any pair of honest users $i, j \in \mathcal{H}$, define $V_j = (\mathcal{F}, \mathcal{M}) \leftarrow C_j.\mathsf{GetView}()$ at the end of the experiment. Then we say the messaging system **guarantees integrity** if with probability $1-\mathsf{negl}(\lambda)$, there exists a $t(i) \in [T]$ such that

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

3.3 A weaker Security Definition

Unfortunately, Anysphere does not satisfy the strong Definition 3.12. 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 [LZA22], 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 a small number of compromised friends, yet still theoretically guarantees security.

[**Arvid**: Another path to explore here would be to create a new leak function, say LeakCF, which contains the exact information leaked for the CF attack. For example, one potentially useful leak function would be one that contains the current number of friends, or perhaps the number of friends but rounded to the nearest 10. In the worst case (such as our current prioritization case), we would leak the timing that a message actually gets sent to the server, which might be very hard to model here... Hmmmm]

[**stzh**: I agree. This remains unresolved. We need to figure out what to do here.]

Definition 3.16. We say a set of inputs $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ satisfy **no compromised friends** if for any $i \in \mathcal{H}$, $j \in \mathcal{K}$ and $t \in [T]$, we have

$$\mathsf{TrustEst}(\mathsf{reg}_j) \neq \mathcal{I}_{i,t}.$$

We say that a set of inputs $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ satisfy 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 a messaging scheme is correct with B-bounded friends if in the honest server experiment Figure 1, if $\{\mathcal{I}_{i,t}\}$ satisfy B-bounded friends, then the scheme produces the correct views with probability $1-\mathsf{negl}(\lambda)$.

We say a messaging scheme is SIM-secure with no compromised friends / B-bounded friends if for any polynomial upper bounds on N and T, there exists a p.p.t simulator Sim such that for any p.p.t adversary $\mathcal{A}$, the view of $\mathcal{A}$ is indistinguishable in $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\lambda)$, provided that the input set $\{\mathcal{I}_{i,t}\}_{i\in\mathcal{H},t\in[T]}$ satisfies no compromised friends / B-bounded friends.

The Anysphere core protocol described below satisfies SIM-security under both B-bounded friends and no compromised friends model. The no compromised friends case essentially follows from [Ang18, Appendix C]. On the other hand, the B-bounded friends case is much more subtle. The next section explains why.

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 reveal the number of friends each honest user has, this attack can potentially reveal the sender and recipient of a PIR request 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 and Spiral reuse PIR keys. Each client generates a secret $\mathsf{sk}_{\mathsf{pir}}$ once and use them to encrypt all PIR queries $\mathsf{ct} = \mathsf{Query}(1^\lambda, \mathsf{sk}_{\mathsf{pir}}, i)$. This optimization was regarded safe since it preserves the UO-ER security definition in [AS16, Extended Version]. Now 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

- User A will have a conversation with user C at a future time $T_1 > T_0$.
- User A does not switch PIR keypair between time T_0 and T_1.

- 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).

d At time T_0, the 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 $\text{resp} = \text{Answer}^{DB'}(1^\lambda, \text{ct})$, 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 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 keypair each round, which is ok for Anysphere because of our low client-side computation requirement. However, it shows that compromised friends can do more damage to MPM systems than previously known.

Note: In [Hen+22], Henzinger et. al. discovered another attack exploiting keypair 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 oracle. In contrast, our attack is on the protocol level, applies to any public key homomorphic encryption system, and makes no assumption on the feedback the attacker gets.

5 DEFINITION OF ANYSPHERE

In this section, we formally define the Anysphere core protocol described in our whitepaper [LZA22].

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 efficient algorithms

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

with specifications

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

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 our AE scheme is **correct** if for any plaintext m of length L_{ae}, we have

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

Remark: In our implementation, we take $L_{\text{ae}} \approx 1\text{KB}$.

Definition 5.2. Given a secret key sk, and a polynomial time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^{L_{\text{ae}}}$, the **Eval oracle** $\mathsf{Eval}_f(\mathsf{sk}, \mathsf{sk}', \cdot)$ takes as input a set of ciphertexts $\text{arg}_{\text{ct}} = \{\mathsf{ct}_i\}$, and a plaintext argument arg_p. It sets

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

and outputs $\mathsf{Enc}(\mathsf{sk}, f(\{m_i\}, \text{arg}_p))$.

Definition 5.3. Let N, 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) $i, j, \text{arg}_{\text{ct}}, \text{arg}_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^0_r \leftarrow \mathsf{Eval}_f(\mathsf{sk}_i, \mathsf{sk}_j, \text{arg}_{\text{ct}}, \text{arg}_p)$.
  (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) $i, j, \text{arg}_{\text{ct}}, \text{arg}_p \leftarrow \mathcal{A}(1^\lambda)$.
  (b) $\mathsf{ct}^1_r \leftarrow \mathsf{Sim}(1^\lambda)$.
  (c) $\mathcal{A}$ stores ct^1_r.

Figure 5: Real and Ideal World Experiment for AE Scheme

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 $f : \Sigma^* \times \Sigma^* \to \Sigma^{L_{\text{ae}}}$ 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 AE Scheme

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

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

5.1.2 Symmetric Key Distribution. In our system, each pair of users shares two secret keys, used to encrypt two directions 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 [LZA22, 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

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

where DB is a length N database with 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 the PIR scheme is **correct** if for any database DB of length N with 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 N, 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 stated p.p.t adversary $\mathcal{A}$, the view of $\mathcal{A}$ under the real world experiment and the ideal world experiment are computationally indistinguishable.

For our implementation, we use libsodium's key exchange functionality as the Gen and KX functions, 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}.

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 Experiment for PIR Scheme

5.2 Messages, Sequence numbers, ACKs

This section describes how Anysphere adopts TCP's Acknowledgement system to offer 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[3] $\mathsf{msg}^{lb} = (k, \mathsf{msg})$, 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.[4] 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.[5]

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 $\Pi_{\mathsf{ae}}.\mathsf{Enc}$. While the input messages have length L_{msg}, the messages 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.

5.3 The Anysphere Core Protocol

[**TODO**: I'm going to stick to our actual implementation as closely as possible. Please point out anything that doesn't agree with the current protocol, greatly appreciate it.]

Recall some notations.

- λ is the security parameter.
- N is the number of users.
- T is the number of timesteps our protocol is run.
- L_{msg} is the length of the raw message.

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

- 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 can now formally define Anysphere's core protocol. We assume the B-bounded friends scenario in Definition 3.16.

Definition 5.7. The 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.

$\Pi_{\mathrm{asphr}}.\mathsf{C.Register}(1^{\lambda}, i, N)$

---

(1) Initialize empty map frdb. The map takes registration info as keys and the following fields as values. [6]

- sk, the secret keys.
- seqstart, the sequence number of the current message being broadcasted to the friend.
- seqend, the largest sequence number ever assigned to messages to the friend.
- seqreceived, the largest sequence number among messages received from the friends.

[**Arvid**: a bit confused between seqend and seqstart...] [**stzh**: in other words, the inbox contains messages in [seqstart, seqend].]

(2) Initialize empty maps in, out. The maps take registration info as keys and arrays of messages as values.[7]

(3) Set a transmission schedule T_{trans}. The user can customize this parameter. We assume T_{trans} is upper bounded by a constant T^{U}_{trans}.

(4) Return reg = (i).

$\Pi_{\mathrm{asphr}}.\mathsf{S.InitServer}(1^{\lambda}, N)$.

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

$\Pi_{\mathrm{asphr}}.\mathsf{C.Input}(t, \mathcal{I})$

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. [8]

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

(4) Push msg^{lb} = (frdb[reg].seqend, 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) frdb[reg] $\leftarrow$ {sk : $(\mathsf{sk}_r, \mathsf{sk}_w)$, seqstart : 1, seqend : 0, seqreceived : 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 $k \le B$ by B-bounded friends. Construct $S = [\mathsf{reg}_1, \cdots, \mathsf{reg}_k, \mathsf{reg}, \cdots, \mathsf{reg}]$, where we add $B - k$ copies of reg $= (-1)$, a dummy registration info. Sample $\mathsf{reg}_s, \mathsf{reg}_r$ uniformly and independently at random from S. [**Arvid**: this is not what we currently do. maybe we should. we should make a decision on the CF attack here and what is acceptable. i think my favorite idea is leaking a rounded version of the number of friends, or something like that. however, the way the code works now where we pick a random friend among the friends that we have outgoing messages to is quite nice because it means that messages will get delivered much faster (especially once we implement PIR batch retrieval)...]

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

(4) $\mathsf{sk}_w \leftarrow$ frdb[reg_s].sk[1]. If reg_s does not exist in frdb, randomly generate a secret key $\mathsf{sk}_w \leftarrow \Pi_{\mathrm{ae}}.\mathsf{Gen}(1^{\lambda})$.

(5) seqreceived $\leftarrow$ frdb[reg_s].seqreceived.

(6) Encrypt Messages with sk_w.

- $\mathsf{ct}_{\mathrm{msg}} \leftarrow \Pi_{\mathrm{ae}}.\mathsf{Enc}(\mathsf{sk}_w, \mathsf{msg}^{lb})$.
- $\mathsf{ct}_{\mathrm{ack}} \leftarrow \Pi_{\mathrm{ae}}.\mathsf{Enc}(\mathsf{sk}_w, \mathsf{ACK}(\mathsf{seqreceived}))$. [**Arvid**: should we talk about the ACK db here, and the fact that we always send all ACKs to everyone? maybe we shouldn't do that anymore... it was necessary for prioritization, but if we don't want to do prioritization then maybe we shouldn't do it anymore]

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

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

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

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

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

[7] 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.

[8] Skipping this step breaks consistent prefix.

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

---

for i from 1 to N

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

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

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

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

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

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

Π_{asphr}.C.ServerRPC(t, resp)

---

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

(2) Parse $\text{resp} = (a_{\text{msg}}, a_{\text{ack}})$. 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 $\text{msg}^{lb} = \perp$ or $\text{msg}^{lb}[0]$ is not $\text{frdb}[\text{reg}_r].\text{seqreceived}+1$, skip the next two steps.

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

  (d) $\text{msg} \leftarrow \text{msg}^{lb}[1]$. 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 $\text{ack} = \perp$ or ack is not the form $\text{ACK}(k)$ for some k, return.

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

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

Π_{asphr}.C.GetView()

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

6 PROOFS

In this section, we prove that our messaging scheme satisfies Definition 3.8, Definition 3.12, and Definition 3.15 under the B-bounded friend scenario. These definitions establish correctness, security, and integrity, respectively.

6.1 Proof of Correctness and Integrity

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

We first introduce some notations for convenience. Define

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

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

Consistent Prefix: First, 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 $1 - \text{negl}(\lambda)$, for any pair of honest users $i \neq j \in \mathcal{H}$ and any $t \le T$, the property below holds.*

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

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

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

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

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

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

Proof. When $t = 0$, 1) is satisfied since $C_i.\text{frdb}$ 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 $1 - \text{negl}(\lambda)$.

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

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

If 1) holds before timestep t starts, then unless

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

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

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

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

Thus we have

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

Furthermore, we have $\mathsf{msg} = \mathsf{msg}_{ij}(E_t)$, and it will be labeled with sequence number E_t by step (3) in $\Pi_{\text{asphr}}.C.\mathsf{Input}()$ 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.

$C_i.\mathsf{ServerRPC}$

The relevant variables only change during ACK decipher when a request is sent during Phase 2 of $C_i.\mathsf{Input}$. 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 $1 - \mathsf{negl}(\lambda)$, the ack variable defined in step (7a) by

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

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 j.

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 (7.c), 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.

$C_j.\mathsf{ServerRPC}$

The relevant variables only change during step (6) (message decipher). Let $i' = i_r$ be the PIR index chosen in the previous call to $C_i.\mathsf{Input}$. No relevant variable changes if $i' \neq i$, so assume $i' = i$. By EUF-CMA, with probability $1 - \mathsf{negl}(\lambda)$, 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 $1 - \mathsf{negl}(\lambda)$. □

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 $1 - \mathsf{negl}(\lambda)$ 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 $C_j.\mathsf{in}$ immediately in Phase 1 of $\Pi_{\text{asphr}}.C.\mathsf{input}$. 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.

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 $T_{cons} = 2\lambda B^2 T_{\text{trans}}^U \cdot T_1 + T_1$. We show that this T_{cons} satisfies the desired property.

We wish to show that if $T_0 \geq T_{cons}$, then with probability $1 - \mathsf{negl}(\lambda)$ we have $t(i) \geq T_1$. Let $E = |\mathsf{MSGSent}(T_1, i, j)|$(Note 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 suffice 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

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

or

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

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. 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. Analogous, 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 $R'(\lambda) = 2N'(\lambda)T(\lambda)$. Let Sim_{sym} be a simulator satisfying Definition 5.3 and with parameters $(N, R) = (N'(\lambda), R'(\lambda))$, and let Sim_{pir} be a simulator satisfying Definition 5.6 with parameters $(N, R) = (N(\lambda), R'(\lambda))$.

At step (2), (4.a), and (4.c) 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 $i \in \mathcal{H}$, which involves changing all cryptography involving unleaked information to simulations. Modifications are marked in red. 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, with $\text{sk}_{ij,r} = \text{sk}_{ji,w}$ the read key of user i.

[**TODO**: Hard-coded. Please modify if the Π_{asphr} methods get changed.]

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

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. [9]

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

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

[9]Skipping this step breaks consistent prefix.

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

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 $k \le B$. 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 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 $\text{msg} = (-1, 0^{L_{\text{msg}}})$.

(4) $\text{sk}_w \leftarrow \text{frdb}[\text{reg}_s].\text{sk}[1]$. If reg_s does not exist in frdb, randomly generate a secret key $\text{sk} \leftarrow \Pi_{\text{ae}}.\text{Gen}(1^\lambda)$.

(5) $\text{seqreceived} \leftarrow \text{frdb}[\text{reg}_s].\text{seqreceived}$.

(6) Encrypt Messages with sk.

Hybrid 1:

If $\text{reg}_s \in \text{reg}_{\mathcal{H}}$,

- $\text{ct}_{\text{msg}} \leftarrow \text{Sim}_{\text{sym}}(1^\lambda)$,
- $\text{ct}_{\text{ack}} \leftarrow \text{Sim}_{\text{sym}}(1^\lambda)$,

Else,

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

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

(9) Remember reg_r.

$\Pi_{\text{asphr}}.\text{C.ServerRPC}(t, \text{resp})$

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

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

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

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

(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 $\mathsf{msg}^{lb}[0]$ is not $\mathsf{frdb}[\mathsf{reg}_r].\mathsf{seqreceived}+1$, ignore the message.

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

  (d) Let msg be $\mathsf{msg}^{lb}[1]$. 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, ignore the ack.

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

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

We use a hybrid argument to show that the two views are indistinguishable. We start from the original implementation of the client methods $\Pi_{\mathsf{asphr}}.C$, and use a hybrid argument to transform it into the simulator methods $\mathsf{Sim}_{\mathsf{asphr}}$. 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 $\mathsf{data}_r[\mathsf{reg}_i] = (\mathsf{frdb}[\mathsf{reg}_i], \mathsf{in}[\mathsf{reg}_i], \mathsf{out}[\mathsf{reg}_i])$. $\mathcal{A}_1$ stores a log of changes to data **encrypted using** $\mathsf{sk} = \mathsf{sk}_{ij,r}$. 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 $\mathsf{data}[\mathsf{reg}]$. Finally, note that any client outputs computed from data are encrypted using $\mathsf{sk}_{ij,r}$, 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 $\mathsf{sk}_{ij,r} = \mathsf{sk}_{ji,w} = \mathsf{sk}_{i*N+j}$, where sk_* are defined 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"[10]
$$\mathsf{GenSec}(\mathsf{reg}_i, \mathsf{reg}_j) = (\mathsf{sk}_{ij,r}, \mathsf{sk}_{ji,w}).$$
$\mathcal{A}_1$ independently generates all other shared secrets using $\Pi_{\mathsf{ae}}.\mathsf{Gen}(1^\lambda)$.
- On step (4.a), $\mathcal{A}_1$ iterates over $i \in \mathcal{H}$. $\mathcal{A}_1$ simulates client i's action in $\Pi_{\mathsf{asphr}}.\mathsf{Input}$ verbatim until Phase 2 Step (6). In Step (6), $\mathcal{A}_1$ do casework based on if $\mathsf{reg}_s \in \mathsf{reg}_{\mathcal{H}}$.
  - If $\mathsf{reg}_s \notin \mathsf{reg}_{\mathcal{H}}$, $\mathcal{A}_1$ knows the shared secret between reg_i and reg_s, so $\mathcal{A}_1$ can simulate this step verbatim.
  - If $\mathsf{reg}_s \in \mathsf{reg}_{\mathcal{H}}$, assume $\mathsf{reg}_s = \mathsf{reg}_j$ where $j \in \mathcal{H}$. $\mathcal{A}_1$ sets $(\mathsf{ct}, \mathsf{arg}_p)$ so that $\mathsf{Eval}_f(\mathsf{sk}_{ij}, \mathsf{ct}, \mathsf{arg}_p)$ can simulate the original step (6) to compute $\mathsf{ct}_{\mathsf{msg}}$, outputs $(i, j, \mathsf{ct}, \mathsf{arg}_p)$, then moves to line (4.b) in Definition 5.3. $\mathcal{A}_1$ sets $\mathsf{ct}_{\mathsf{msg}} = \mathsf{ct}_r^b$ to be the output of line (4.b). $\mathcal{A}_1$ repeats this for $\mathsf{ct}_{\mathsf{ack}}$.
- On step (4.c), $\mathcal{A}_1$ simulates client action in $\Pi_{\mathsf{asphr}}.\mathsf{ServerRPC}$ using Eval_f.

We note that in $\mathsf{Real}_{\mathsf{Eval}}^{\mathcal{A}}$, $\mathcal{A}_1$ perfectly simulates the view of $\mathcal{A}$ in Hyb_0. In $\mathsf{Ideal}_{\mathsf{Eval}}^{\mathcal{A},\mathsf{Sim}_{\mathsf{ae}}}$, $\mathcal{A}_1$ 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
$$\mathsf{Hyb}_0 \equiv_c \mathsf{Hyb}_1.$$

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

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

[10] What we mean by set is that $\mathcal{A}_1$ simulates the honest users as if $\mathsf{GenSec}(\mathsf{reg}_i, \mathsf{reg}_j) = \mathsf{sk}_{ij}$. $\mathcal{A}_1$ does not have access to sk_{ij}.

(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 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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) $m_0, m_1 \leftarrow \mathcal{A}^{\mathsf{Enc}(\{\mathsf{sk}_i\},\cdot),\mathsf{Dec}(\{\mathsf{sk}_i\},\cdot)}(1^\lambda, kx_i^P)$.
(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 key exchange plus symmetric key system Π_{ae} is **IK-CCA secure** if for any p.p.t with oracle adversary $\mathcal{A}^O$, we have

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

Let Π_{ae} be 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. We 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 $k \ge 0$, 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. *Assume Π_{ae} is IK-CCA secure. Then 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 $\mathbf{Exp}_{\mathsf{IK-CCA}}^{\mathcal{A}'}$, 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 $\mathbf{Exp}_{\mathsf{IK-CCA}}^{\mathcal{A}'}$. 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 $\mu(\lambda)$ independent of j such that for any j, we have

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

Substituting back, and summing over all $i^* \in [N]$, we conclude 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.

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.

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

[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.

[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.

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

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

[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.

[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.

[LZA22] Arvid Lunnemark, Shengtong Zhang, and Sualeh Asif. *Anysphere: Private Communication in Practice*. 2022. URL: https://anysphere.co/anysphere-whitepaper.pdf (visited on 06/16/2022).

[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), 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.