

# Addra: Metadata-private voice communication over fully untrusted infrastructure

Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta

*University of California, Santa Barbara*

## Abstract

Metadata from voice calls, such as the knowledge of who is communicating with whom, contains rich information about people’s lives. Indeed, it is a prime target for powerful adversaries such as nation states. Existing systems that hide voice call metadata either require trusted intermediaries in the network or scale to only tens of users. This paper describes the design, implementation, and evaluation of Addra, the first system for voice communication that hides metadata over fully untrusted infrastructure and scales to tens of thousands of users. At a high level, Addra follows a template in which callers and callees deposit and retrieve messages from private mailboxes hosted at an untrusted server. However, Addra improves message latency in this architecture, which is a key performance metric for voice calls. First, it enables a caller to push a message to a callee in two hops, using a new way of assigning mailboxes to users that resembles how a post office assigns PO boxes to its customers. Second, it innovates on the underlying cryptographic machinery and constructs a new private information retrieval scheme, FastPIR, that reduces the time to process oblivious access requests for mailboxes. An evaluation of Addra on a cluster of 80 machines on AWS demonstrates that it can serve 32K users with a 99-th percentile message latency of 726 ms—a  $7\times$  improvement over a prior system for text messaging in the same threat model.

## 1 Introduction

Voice call metadata—the parties involved in the call, the duration of the call, and the time of the call—can be incredibly revealing. The former General Counsel of NSA, Stewart Baker, has said, “metadata absolutely tells you everything about somebody’s life. If you have enough metadata, you don’t really need content” [20, 21, 65]. Several academic studies [26, 53, 54] have confirmed the power of metadata. As an example, Mayer et al. [53] used telephone metadata to infer that a study participant “received a long phone call from the cardiology group at a regional medical center, talked briefly with a medical laboratory, . . . and made brief calls to a self-reporting hotline for a cardiac arrhythmia monitoring device.” The authors confirmed that the participant had a cardiac arrhythmia. A study of whistle-blowers also revealed that metadata can identify a journalist’s sources [38].

Given the information contained in metadata, a significant question is: how can one make a voice call without revealing to anyone the metadata associated with the call? Fortunately, several systems have tackled this problem [12, 33, 48, 50, 69, 71] (§7). Although these systems

hide metadata and keep message latency low, they either restrict scalability to only tens of users [33, 69], or are vulnerable to attacks by requiring trusted intermediaries in the communication infrastructure [12, 48, 50, 71]. An example of a trust assumption is that the system guarantees security only if the adversary can compromise at most a fraction (20%) of the servers that route user calls [48]. Trusting intermediaries can be risky as powerful adversaries like nation states are the ones that try to collect metadata. Such adversaries have been known to wield their vast political, technical, and financial power to gain access to metadata [11, 52, 58, 66].

A system that can withstand strong adversaries while serving more than tens of users is Pung [6, 9]. Pung makes *no* assumptions about the communication infrastructure—the adversary may compromise a part or all of the infrastructure. However, Pung targets applications such as email and chat with long-lived messages that are retrieved asynchronously. Indeed, a Pung client makes  $\lceil \log_2(n + 1) \rceil$  round trips to a remote server to obliviously search and retrieve a message ( $n$  is the number of users), thereby incurring several seconds of message latency (§6.1). In contrast, voice calls have a strict time budget. If a user sends a packet every few hundred milliseconds, then each hop in the communication infrastructure must not spend longer than this time period to process and forward the packet, to avoid an unbounded packet build up.

We present Addra, the first system that provably hides metadata for voice calls, makes no assumptions about the underlying infrastructure, and scales to tens of thousands of users. In terms of privacy guarantees, Addra provides relationship unobservability—an adversary cannot detect whether a relationship (voice call) exists between any two users of the system [62] (§2.1). These privacy guarantees are achieved with practical latency performance of under 750 ms, and for low-bandwidth voice synthesis at a rate of 1.6 Kbit/s as in the Mozilla LPCNet voice codec [56, 73, 74].

Addra, like Pung, relies on a set of mailboxes hosted at an untrusted server. Callers deposit messages and callees retrieve messages from these mailboxes using a private information retrieval (PIR) cryptographic protocol [18, 19, 42] (§3.2). This protocol ensures that the untrusted server does not learn which mailbox a callee is accessing, thereby unlinking the callee from the caller. However, Addra must address two challenges in this architecture to support low-latency voice calls (§2.3). First, it must reduce the number of round trips a caller or callee makes to the server to transfer or retrieve a voice packet. Second, Addra must reduce the time the server takes to process caller and callee requests, particularly, the PIR requests.

Addra addresses the first challenge through a new, and remarkably simple, use of mailboxes (§3). When someone rents a conventional post office box, or PO box, at a post office, they get a mailbox with a unique and fixed address into which the mailman deposits incoming mail. Addra inverts this architecture. In Addra, a caller (rather than a recipient or callee) gets a dedicated mailbox with a fixed address or “phone number”. The caller deposits its outgoing messages into this mailbox—independent of who the caller is calling. (Thus, an adversary cannot tell whom the caller is calling.) Meanwhile, a callee retrieves a message from the mailbox tied to the caller’s phone number using a PIR protocol. Crucially, to transmit a message, a caller makes one push request to the server, and the server makes one push request to the callee—a hop count of two. In contrast, prior work requires multiple round trips between the server and the callees.

Addra addresses the second challenge mentioned above, of reducing server-side processing time for PIR, by two means. First, it parallelizes PIR processing across multiple server machines and multiple CPU cores on a machine. The fact that PIR is parallelizable is known and studied [28, 36]. Second, and more saliently, Addra constructs a new PIR scheme, FastPIR, that fundamentally reduces the server-side PIR processing time relative to prior state-of-the-art schemes [4, 6] (§4). Even though FastPIR was motivated by Addra, it can be used for other applications of PIR [13, 29, 35, 55].

FastPIR builds on the homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [14, 32] (§4.1) and leverages two of its features. First, it uses the single instruction, multiple data (SIMD) capability of BFV ciphertexts to compute on compressed PIR requests. Prior state-of-the-art schemes [4, 6] also exploit SIMD capabilities but not in a way that keeps PIR requests compressed in memory. Meanwhile, such compression improves memory utilization, reduces CPU time, and eliminates the time to uncompress requests (§4.2). However, working over compressed requests naively increases PIR response size. So, second, FastPIR uses homomorphic rotation operations in BFV to pack multiple pieces of a PIR response, thereby reducing response size. Further, FastPIR reduces both the CPU time per rotation and the number of calls to this operation (§4.3, §4.4).

For completeness, Addra includes a dialing protocol that allows a callee to detect that a caller is calling and learn the caller’s phone number (mailbox address). For this purpose, Addra uses the dialing protocol from Pung (§5).

We have implemented (§5) and evaluated (§6) a prototype of Addra. Our prototype runs on Amazon EC2 where the server runs in the US East region, and the clients (callers and callees) run geographically apart in the US West region. When the server uses 80 machines, Addra supports 32K clients communicating with each other with a 99-th percentile message latency of 726 ms. In contrast, Pung (the only other system that works at scale over completely untrusted infrastructure) transmits messages for the same number of users with a message

latency of 5.2 seconds. Besides, Addra requires a network download bandwidth of 1.46 Mbps and an upload bandwidth of 30 Kbps for every client.

Although Addra achieves low message latency for a few tens of thousands of users, it does not currently scale to hundreds of thousands or a few million users due to the overhead of PIR which grows quadratically with the number of users. Furthermore, although its instantaneous bandwidth requirements are modest, the total network transfers are high as a client must remain online even if it is not participating in a call to hide call initiation patterns. Thus, Addra assumes clients with unlimited data plans. Nevertheless, Addra demonstrates, for the first time, that even over completely untrusted infrastructure, metadata for voice calls can be hidden at scale for tens of thousands of users.

## 2 Goals, threat model, and challenges

Addra’s goal, at a high level, is to enable its users to make peer-to-peer voice calls while hiding metadata from a powerful adversary that may compromise the entire communication infrastructure.

### 2.1 Goals

**Performance and scalability.** Voice calls require the communication infrastructure to transmit messages with low latency. Addra targets a sub-second message latency due to the feasibility of voice calls under such a setting [48]. Thus, if Alice sends a voice packet to Bob, then Bob should receive it within one second. Additionally, the infrastructure must not queue up voice packets indefinitely. For instance, if Alice generates a voice packet every 500 ms, then every hop in the infrastructure must spend no more than 500 ms to process the packet before sending it forward towards Bob. Addra must also provide sufficient throughput so that the transmitted voice is understandable. For this purpose, Addra targets the LPCNet voice codec [73, 74], which specializes in low-bandwidth voice synthesis at a rate of 1.6 Kbit/s. Finally, we want Addra to scale to a large number of users (for example, tens of thousands on a cluster of hundred machines).

**Content privacy.** Addra must ensure that only the caller and callee of a voice call can comprehend the content of the voice packets they send to each other.

**Metadata privacy.** Addra, similar to Pung [9], targets the guarantee of *relationship unobservability* as defined by Pfitzmann and Hansen [62]. Relationship unobservability states that it is undetectable whether a relationship (voice call) exists between a sender (caller) and a recipient (callee), unless the sender or the recipient are compromised. If either the caller or the callee is compromised, then offering privacy guarantees has little value, as the compromised party can trivially reveal the existence of communication (or lack thereof).

## 2.2 Threat model and assumptions

As motivated in the introduction (§1), Addra assumes an adversary who can compromise the entire communication infrastructure, including routers, switches, and middleboxes. The adversary can observe network traffic, perform traffic analysis, and manipulate traffic: reorder, replay, change, and inject network packets.

Callers and callees trust their own devices. More generally, the adversary can compromise a subset of end user devices. In this case, Addra must provide content and metadata privacy to the users of non-compromised devices.

The adversary may not break standard cryptographic primitives such as public-key and symmetric-key encryption.

The adversary may mount a denial-of-service attack: bring down the entire communication infrastructure or selectively drop traffic. In such cases, Addra cannot guarantee voice communication but must continue to guarantee privacy.

## 2.3 Challenges

Meeting the performance and privacy goals stated above under the threat model just described is challenging. Indeed, prior work either relaxes the threat model or does not meet the performance goals. For instance, Yodel [48] is a metadata-private voice communication system that scales to several million users but assumes that a server in the communication infrastructure is compromised with only a 20% chance. On the other hand, Pung [6, 9] works in the stronger threat model. However, it cannot push frequent messages from a caller to a callee. As mentioned earlier (§2.1), if a caller samples voice every 500 ms, then each hop of the communication infrastructure must process a voice packet within 500 ms before the arrival of the next packet to avoid packet build up. This time budget entails that a caller or a callee cannot make multiple round trips to a server in the communication infrastructure to send or receive a single packet. But Pung requires message recipients to make multiple such trips to its server. Addra addresses these challenges and meets the performance requirements for tens of thousands of users without making any trust assumptions, as described next.

# 3 Architecture and overview of design

## 3.1 Architecture

Figure 1 shows Addra’s architecture. Addra consists of a server and user (participant) devices. The server runs over untrusted infrastructure. It is logically centralized but physically distributed over multiple machines. The server’s role is to facilitate communication among the user devices in a privacy-preserving manner.

The server exposes *mailboxes*. Specifically, it exposes  $n$  mailboxes, where  $n$  is the number of user devices using the system. Each mailbox can store one message and it has an ID, which is a number between (and inclusive of) 0 and  $n - 1$ . As we will describe later (§4), it is helpful to view the  $n$

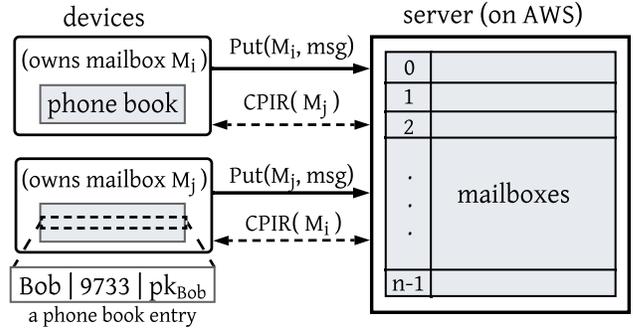


Figure 1—High-level architecture of Addra. The server runs over untrusted infrastructure and exposes mailboxes that user devices read from or write into. The mailbox identifiers (integers 0 to  $n - 1$ ) play the role of “phone numbers”. A device stores phone numbers of the device owner’s contacts in a local phone book. CPIR refers to the private information retrieval cryptographic primitive (§3.2).

mailboxes as a matrix with  $n$  rows and  $m$  columns, where each row is an individual mailbox, and the  $m$  pieces of a message are  $m$  elements of a matrix row.

The user devices run logic to enable users to initiate, pick up, and participate in calls. Each device gets assigned a mailbox ID, which acts as its phone number. Each device also contains a *phone book*, which stores information on device owner’s contacts. Each phone book entry is a tuple of a phone number of the contact, a cryptographic public key belonging to the contact, and other standard information such as the contact’s name, work place, and photograph. We assume that a device owner either knows this information or can obtain it privately through out-of-band means such as in-person meetings or personal websites.

## 3.2 Protocol

Addra relies on a cryptographic protocol called *private information retrieval* or *PIR* [18, 42]. We begin with a short background on PIR; Section 4 describes a new PIR scheme.

**A primer on PIR.** A PIR protocol [18, 42] runs between a user device and the server in Addra, where the device is interested in retrieving the message in the  $idx$ -th mailbox at the server without revealing the value of  $idx$ .

A PIR protocol has three procedures: QUERY, ANSWER, and DECODE. QUERY is run by a device. It takes as input the index  $idx$  between 0 and  $n - 1$  and returns a query,  $q$ . Typically,  $q$  is an encryption of a suitable encoding of  $idx$ . ANSWER is run by the server; it takes as input the query  $q$  and the set of  $n$  mailboxes, and returns an encoding of the message in the  $idx$ -th mailbox (without learning the value of  $idx$ ). Finally, DECODE is run by the device; it takes the output of ANSWER and returns the  $idx$ -th mailbox message.

**Addra’s protocol.** User devices in Addra participate in a round-based protocol consisting of a one-time registration step followed by synchronous rounds, each consisting of a dialing phase followed by a communication phase consisting

```

1: function RECV(key  $key$ , resp  $resp$ )
2:   //  $resp$  is the output of ANSWER PIR procedure
3:    $c \leftarrow \text{DECODE}(resp)$  // DECODE is a PIR procedure
4:    $msg \leftarrow \text{AES.DEC}(key, c)$ 
5:   play  $msg$  to user

6: function SEND(mailbox  $M$ , token  $t$ , message  $msg$ , key  $key$ )
7:    $c \leftarrow \text{AES.ENC}(key, msg)$ 
8:   send  $(M, t, c)$  to server

9: function MAIN()
10:  // Register device and obtain a mailbox ID and unique token
11:   $(M_{self}, tkn, n) \leftarrow \text{REGISTERDEVICE}()$ 
12:  while True do
13:    // Run dialing phase.  $k_{enc}$  is for encrypting content
14:     $(M_{peer}, k_{enc}) \leftarrow \text{DIAL/PICKUP}()$ 
15:     $q \leftarrow \text{QUERY}(M_{peer}, n)$  // QUERY is a PIR procedure
16:    send  $q$  to server
17:    // Asynchronously listen for server responses
18:    register callback RECV( $k_{enc}, \dots$ ) for server responses
19:    // Run communication phase consisting of  $t$  subrounds
20:    for  $r = 0$  to  $t - 1$  do
21:      wait for message generation interval
22:      call SEND( $M_{self}, tkn, msg_r, k_{enc}$ )

```

Figure 2—Pseudocode for a user device in Addra.  $n$  is the number of mailboxes at the server. QUERY, ANSWER, DECODE are procedures of a PIR scheme (§3.2, §4).

of multiple subrounds of communication. In more detail, initially a device performs a one-time *registration step* to register itself with the server and obtain its phone number (mailbox ID). This is followed by a sequence of rounds. Each round starts with a *dialing phase*, where the device initiates a call to another device or picks an incoming call. The dialing phase is followed by the *communication phase*, consisting of multiple subrounds, where each device sends exactly one message to the server and receives one message from the server. Notably, a device always writes a message to its assigned mailbox, while it receives a message from its peer’s mailbox.

We now describe Addra’s protocol in more detail. Figure 2 shows the pseudocode for a user device. A device starts executing the MAIN function (line 9 in Figure 2).

**One-time registration step.** When a user device joins Addra, it registers itself with the server and obtains three pieces of information: a mailbox ID, a unique authentication token  $tkn$ , and the number  $n$  of mailboxes (line 11 in Figure 2). As mentioned above, the mailbox ID acts as the phone number assigned to the device. Meanwhile, the authentication token is a 128-bit uniformly generated string shared between the server and the device that enables the server to verify that a device is writing messages to its assigned mailbox (and not to a mailbox assigned to another device). One may use digital signatures instead of authentication tokens, but Addra prefers the *symmetric* tokens due to their better efficiency. The number of mailboxes  $n$  may increase if new devices join the system; when this happens, the server broadcasts an updated value of  $n$ .

Addra’s server is untrusted and may assign mailbox IDs or authentication tokens incorrectly; for instance, it may reassign a previously assigned mailbox ID. Besides, it may distribute different values of  $n$  to different devices. The privacy guarantees of Addra’s protocol do not depend on the server assigning correct values for these items. However, a malicious server can deny service to system participants, which is not prevented by our threat model (§2.2). A service provider who runs the server will likely be incentivized to provide a continuous service to keep its customer base.

**Dialing phase.** Once registered, a user device, who we refer to using its phone number,  $M_{self}$ , executes the round-based protocol. At the beginning of each round,  $M_{self}$  initiates a call or picks up an incoming call (line 14 in Figure 2). If the device initiates a call, it selects the phone number of the peer device it is calling,  $M_{peer}$ , and an encryption key,  $k_{enc}$ , to hide the content of the messages it will send. On the other hand, if the device picks up an incoming call then it learns the phone number of the caller and its content encryption key. For now, we leave out the details of how a device picks up a call till later (§5). After initiating or picking up a call,  $M_{self}$  generates a PIR query  $q \leftarrow \text{QUERY}(M_{peer}, n)$  for the peer’s mailbox, and sends  $q$  to the server (lines 15 and 16 in Figure 2). The PIR query indicates, without revealing the value of  $M_{peer}$ , that  $M_{self}$  is interested in receiving messages deposited into  $M_{peer}$ ’s mailbox. The device  $M_{self}$  then registers an asynchronous callback to process PIR responses from the server (line 18 in Figure 2). Meanwhile, the server stores the PIR queries from all devices and uses them across all subrounds of the round’s communication phase.

**Communication phase.** In each subround of the communication phase, (1) a device deposits an encrypted message into its assigned mailbox at the server, (2) the server processes PIR queries from all devices and pushes the results to devices who registered these queries, and (3) each device decodes its PIR response from the server. In more detail, at the beginning of a subround, a device encrypts the message it wants to send to its peer with the key  $k_{enc}$  to create a ciphertext  $c$ . It sends the tuple  $(M_{self}, tkn, c)$  to the server (line 22 in Figure 2), where  $tkn$  is the device’s assigned authentication token obtained during the registration step. The server uses the token to validate that the messages being written to mailboxes indeed come from devices that own the mailboxes. After performing these checks, the server runs the ANSWER PIR procedure for all PIR queries. That is, for a query  $q$  sent by a device during the dialing phase, the server runs  $resp \leftarrow \text{ANSWER}(\text{mailboxes}, q)$  and pushes the PIR response  $resp$  to the device. Finally, on receiving a response, a device invokes the callback it registered during the dialing phase (line 1 in Figure 2). This callback decodes the PIR response using the DECODE PIR procedure, decrypts the underlying message sent by the device’s peer  $M_{peer}$ , and delivers the message to the user.

**Dummy participation and messages.** The protocol described so far does not address the case when a device owner does not participate in a call. During such idle periods, like prior systems for strong metadata privacy (e.g., [9, 48]), a device adds *cover traffic* (also called chaff). In particular, if a device does not initiate or pick up a call in a round’s dialing phase, it calls itself: inputs  $M_{self}$  into QUERY (line 15 in Figure 2). Besides, if a device does not have a message to send during a subround, it writes an encryption of a random message into its mailbox. Sending cover traffic is necessary, as otherwise an adversary can learn connections between users by monitoring if they join and leave at similar times.

**Security analysis.** Addra’s protocol satisfies relationship unobservability, meaning that an adversary cannot detect the existence of relationships between system users (§2.1). We provide a rigorous proof in Appendix A. Briefly, Addra’s protocol meets the property because the protocol a user device executes is independent of whom the user is communicating with or the behavior of the (malicious) server. First, a user device encrypts messages using a content encryption key known only to its peer. Further, it always writes outgoing messages at fixed intervals to its own mailbox—independent of whether the device is engaged in a call, or the identity of its peer, or the behavior of the server who may or may not deliver incoming messages to the device, or who may replay messages. Second, the security property of PIR ensures that an adversary cannot tell the IDs of the mailboxes from which devices are retrieving messages. Again, the server may process PIR queries incorrectly, or broadcast an incorrect value  $n$  for the number of mailboxes, but a user device always registers a PIR query for one of the  $n$  mailboxes, no matter the value of  $n$ . Thus, the adversary cannot detect whether a user Alice is communicating with Bob or Charlie or someone else, or even communicating at all (i.e., retrieving messages from its own mailbox).

**Performance characteristics.** Addra’s protocol exhibits two key characteristics that set it on the path to meeting its performance goals (§2.1). First, the protocol pushes messages from senders to recipients in two hops—independent of the number of users in the system. Specifically, in each subround, a sender pushes a message to the server, who then processes the PIR query provided beforehand by the recipient, and pushes the PIR response to the recipient. This two-hop communication pattern is crucial for voice calls which require low latency. Second, the protocol amortizes the cost of generating and transferring a PIR query across subrounds of a round (our prototype runs a round every five minutes, and a subround every 480 ms; §6). Thus, the server does not have to deal with PIR query management (and certain preprocessing of query) during the time-sensitive subrounds. Nevertheless, the server must complete computing ANSWER for all PIR queries in a time smaller than the voice packet generation interval (that is, the duration of a subround) to

avoid packet build up. Besides, the network transfers from the server to the devices are dictated by the size of the output of ANSWER. Thus, a low cost of the ANSWER PIR procedure is key for Addra’s performance.

## 4 FastPIR: A new CPIR scheme

As described above (§3.2), a critical component of Addra’s protocol is the ANSWER PIR procedure. It not only dictates Addra’s message latency but also the resource consumption (both CPU and network) imposed by Addra.

PIR schemes are of two types: computational PIR (CPIR) [42] and information-theoretic PIR (IT-PIR) [18, 19]. CPIR schemes assume a single (untrusted) server and rely only on cryptographic assumptions; in contrast, IT-PIR schemes are more efficient but require two or more non-colluding servers. In Addra, we use a CPIR scheme as its trust assumptions are in line with Addra’s goal of not trusting the communication infrastructure (§2.2).

One can plug in an existing CPIR scheme, either XPIR [4] or SealPIR [6], which are the state-of-the-art CPIR schemes, into Addra’s protocol (§3.2). However, these schemes exhibit a tension between the CPU time to run ANSWER (and thus the wall-clock time for ANSWER) and the output size of ANSWER (and thus the network overhead).

Suppose a CPIR client wants to privately retrieve the  $idx$ -th message from a library  $L$  of  $n$  messages (mailboxes) held at a server. In prior work, a typical way to construct a CPIR query is to treat the library as a matrix with  $n$  rows and generate a ciphertext for every row of  $L$ .<sup>1</sup> The ciphertext for the  $idx$ -th row encrypts the value 1, and the ciphertexts for the other rows encrypt 0. However, this strategy creates large queries with a number of ciphertexts that is proportional to the value of  $n$  (e.g., XPIR’s query size is  $\approx 33$  MiB for  $n=2^{15}$ , and  $\approx 1$  GiB for  $n=2^{20}$ ; §6.5). When the server processes larger queries, it consumes more memory and CPU cycles to read them into CPU caches, which slows down query processing. (SealPIR compresses the query while transferring it on the network, but expands it to the larger query at the server).

A popular technique due to Stern [70] to address the query-size issue is called *recursion*. This technique is parameterized by a depth parameter  $d$ . A value of  $d = 2$  or higher shrinks the query—it contains  $d \cdot \sqrt[d]{n}$  ciphertexts instead of  $n$ —by rearranging the library as a  $d$ -dimensional hypercube. However, this rearrangement increases the CPIR ANSWER output size exponentially with  $d$ . Thus, if we plug in existing CPIR schemes (XPIR or SealPIR), then Addra would compromise on either server CPU time or network bandwidth.

Our CPIR scheme, FastPIR, works without recursion and thus keeps the smaller CPIR answer size. However, it optimizes the computation time for ANSWER. In fact, FastPIR

<sup>1</sup>A technique called aggregation [4, 10] further combines multiple rows (messages) into wider rows, resulting in a matrix with  $n/a$  rows, where the value  $a$  depends on the size of each message.

takes less time than both XPIR and SealPIR (with or without recursion) to run ANSWER, particularly when the number of messages  $n$  in the library is greater than a threshold ( $\approx 20K$ ; §6.5), thereby improving the scalability and message latency of Addra. FastPIR may be a good fit for other applications of CPIR where costs are dominated by those of the CPIR ANSWER procedure.

FastPIR, like SealPIR [6], builds on the lattice-based homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [14, 32]. BFV offers superior efficiency than a traditional number-theoretic homomorphic encryption scheme such as Paillier [60], resists attacks by quantum computers, is implemented in mature and actively maintained codebases [2, 68], and is in the preliminary stages of being standardized (e.g., with ISO/IEC) [5]. We start with a necessary background on BFV (§4.1), and then delve into the details of FastPIR (§4.2–§4.4).

#### 4.1 Background: The BFV cryptosystem

We focus here on describing the more efficient vectorized variant of BFV in which a single homomorphic operation operates over multiple plaintext inputs (single instruction, multiple data or SIMD; also called batching in the literature).

In this BFV variant, a plaintext is a vector of dimension  $N$ , where the parameter  $N$  equals a power of two and is at least  $2^{10}$  for the security of the BFV scheme [5]. Each component of the plaintext is an integer in  $\mathbb{Z}_p = \{0, \dots, p-1\}$ , the set of integers modulo  $p$ . Sometimes, we will view a BFV plaintext as a matrix with two rows and  $N/2$  columns rather than a vector with dimension  $N$ .

A BFV ciphertext is also a vector but of dimension  $2 \cdot N$ . Each of its component is an element of  $\mathbb{Z}_q$ , where  $q \gg p$ .

The BFV encryption procedure, BFV.ENC, adds noise when it converts a plaintext vector into a ciphertext vector. This noise grows as homomorphic operations are performed on the ciphertext. If the noise grows beyond a threshold, then the ciphertext decryption procedure BFV.DEC does not produce the correct plaintext. Hence,  $q \gg p$  for enough noise budget.

The size of the plaintext vector,  $N$ , the size of the domain of each component of the plaintext,  $p$ , and the size of the domain of each component of the ciphertext,  $q$ , are all tunable parameters. Typically, one picks a combination of  $p, q, N$  depending on the application, the required noise budget, and the desired security level; we discuss concrete values for these parameters for Addra in §5.

BFV supports the following homomorphic operations that are used in FastPIR:

- **BFV.ADD**( $c_0, c_1$ ) takes as input encryptions  $c_0$  and  $c_1$  of plaintext vectors  $v_0$  and  $v_1$ , and outputs an encryption of  $v_0 + v_1$  (component-wise vector addition).
- **BFV.SCMULT**( $v_0, c_1$ ) takes as input a plaintext vector  $v_0$  and an encryption  $c_1$  of a plaintext vector  $v_1$ , and produces an encryption of the product  $v_0 \odot v_1$ , where the operator  $\odot$  denotes component-wise multiplication.
- **BFV.ROWROTATE**( $c_0, i$ ) takes as input an encryption  $c_0$  of a plaintext  $v_0$  and an integer  $0 < i < N/2 - 1$ , and produces an encryption of  $v_0$  rotated right by  $i$  positions cyclically row-wise. For instance, if plaintext dimension is  $N = 8$  and  $v_0$  is  $((a, b, c, d), (e, f, g, h))$  in its matrix representation, then a right rotation by  $i = 1$  produces an encryption of  $((d, a, b, c), (h, e, f, g))$ .
- **BFV.COLROTATE**( $c_0$ ) takes as input an encryption  $c_0$  of a plaintext  $v_0$  and returns an encryption of a plaintext produced by swapping the two rows of  $v_0$ . For the example above, the result is an encryption of  $((e, f, g, h), (a, b, c, d))$ .

The BFV homomorphic operations require public keys generated by a key generation procedure. In particular, the rotation procedures require a set of rotation keys. While BFV.COLROTATE requires one key, the size of the set of keys for BFV.ROWROTATE can vary. On the one extreme, this set can be configured to contain one key that rotates the plaintext vector by one position. Thus, to perform a rotation by  $i > 1$  positions, BFV.ROWROTATE calls itself  $i$  times, incurring  $i$  times the cost of one BFV.ROWROTATE operation. On the other extreme, the set can contain  $N/2 - 1$  keys for all possible values of  $i$  between 0 and  $N/2$ . This extreme reduces CPU time for BFV.ROWROTATE as it does not call itself recursively, but this configuration increases the key size. For the BFV parameters we choose (§5), each rotation key is 128 KiB, and the set of all possible rotation keys is 256 MiB. Thus, in practice, one generates  $\log_2(N/2)$  keys for all powers-of-two between 0 and  $N/2 - 1$ , and each invocation of BFV.ROWROTATE calls itself recursively up to  $\log_2(N/2)$  times.

#### 4.2 The FastPIR scheme

Recall the CPIR scenario (§4): a server holds a library  $L$  of  $n$  messages where each message has  $m$  components, while a client holds an integer  $0 \leq idx \leq n - 1$  and wants to retrieve the  $idx$ -th library message without revealing  $idx$  to the server.

To build intuition for FastPIR, suppose that  $L$  is an  $N \times 1$  matrix consisting of  $N$  unit length messages, where  $N$  is the plaintext vector dimension in BFV. Then, the client constructs the CPIR query  $q$  for the  $idx$ -th message by encrypting a BFV plaintext whose  $idx$ -th entry is one and the rest are zeros (this is called *one-hot encoding of  $idx$* ). For instance, if  $N = 4$  and  $idx = 1$ , the client encrypts the BFV plaintext  $(0, 1, 0, 0)$ . The server multiplies this encryption  $q$  with  $L$  by computing **BFV.SCMULT**( $L, q$ ) to obtain an encryption of the  $idx$ -th entry of  $L$ . For the example above, if  $L$  is  $(a_0, a_1, a_2, a_3)$ , **BFV.SCMULT** produces an encryption of  $(0, a_1, 0, 0)$  as the multiplication is component-wise. The client receives the output and decrypts it to get  $a_1$ .

The advantage of this strategy is that a query consumes only a component of a ciphertext for each of the  $n$  rows of  $L$  (instead of a ciphertext per row). However, a challenge is that this strategy generates one output ciphertext for each of the  $m$  columns of  $L$ . FastPIR addresses this challenge by combining ciphertexts for  $m$  columns into a single cipher-

```

1: function QUERY(index  $idx$ ,  $n$ )
2:   // Create a one-hot encoding of  $idx$ 
3:   for  $i = 0$  to  $n - 1$  do
4:      $f_i \leftarrow (i == idx) ? 1 : 0$ 
5:   // Split and encrypt the one-hot vector
6:   for  $i = 0$  to  $(n/N) - 1$  do //  $N$  is BFV plaintext dimension
7:      $q_i = \text{BFV.ENC}(pk, (f_{i \cdot N}, \dots, f_{(i+1) \cdot N - 1}))$ 
8:   return  $q = (q_0, \dots, q_{(n/N)-1})$ 

9: function ANSWER(library  $L$ , query  $q = (q_0, \dots, q_{(n/N)-1})$ )
10:  // Represent  $L$  as a matrix of elements in  $\mathbb{Z}_p$ :  $L \in \mathbb{Z}_p^{n \times m}$ 
11:  //  $q$  is an output of QUERY
12:  for  $j = 0$  to  $m - 1$  do
13:     $sum_j = \text{BFV.ENC}(pk, 0)$ 
14:    for  $i = 0$  to  $(n/N) - 1$  do
15:       $p_{i,j} \leftarrow \text{SUBMAT}(L, i \cdot N, (i + 1) \cdot N - 1, j, j)$ 
16:       $t_{i,j} = \text{BFV.SCMULT}(p_{i,j}, q_i)$ 
17:       $sum_j = \text{BFV.ADD}(sum_j, t_{i,j})$ 
18:  // Combine outputs from all columns
19:  Initialize  $s_{top}, s_{bot}$  to encryptions of zero vectors
20:  for  $j = 0$  to  $m - 1$  do
21:    if  $j < N/2$  then
22:       $sum_j \leftarrow \text{BFV.ROWROTATE}(sum_j, j)$ 
23:       $s_{top} \leftarrow \text{BFV.ADD}(s_{top}, sum_j)$ 
24:    else
25:       $sum_j \leftarrow \text{BFV.ROWROTATE}(sum_j, j - N/2)$ 
26:       $s_{bot} \leftarrow \text{BFV.ADD}(s_{bot}, sum_j)$ 
27:  return  $\text{BFV.ADD}(s_{top}, \text{BFV.COLROTATE}(s_{bot}))$ 

28: function DECODE(answer  $ans$ , index  $idx$ )
29:  //  $ans$  is an output of ANSWER
30:   $ans_{pt} \leftarrow \text{BFV.DEC}(sk, ans)$ 
31:  if  $idx \bmod N > N/2$  then
32:     $ans_{pt} \leftarrow \text{PTCOLROTATE}(ans_{pt})$ 
33:  return  $ans_{pt} \leftarrow \text{PTRROWROT}(ans_{pt}, N/2 - (idx \bmod N/2))$ 

```

Figure 3—QUERY, ANSWER, and DECODE procedures for a basic version of FastPIR.  $(pk, sk)$  are a (public, private) key pair for the BFV scheme (§4.1). SUBMAT extracts a sub-matrix of a matrix. PTRROWROT and PTCOLROTATE are like BFV.ROWROTATE and BFV.COLROTATE, respectively except they operate on BFV plaintexts rather than BFV ciphertexts.

text using the BFV rotation operations (BFV.ROWROTATE and BFV.COLROTATE), thereby reducing CPIR answer sizes.

Before describing the details of rotation, we remark that the use of vectorized operations (SIMD capabilities of BFV) is common. In fact, both XPIR and SealPIR use vectorized operations. The difference is that these prior CPIR schemes apply vectorization across columns of the matrix while FastPIR applies it across rows of the matrix, which is a more efficient use of vectorization in the PIR context (§6.5).

**Details.** Figure 3 shows the FastPIR scheme. It assumes that  $n$  is a multiple of  $N$ , i.e.,  $n = k \cdot N$  for some  $k \geq 1$ , and  $m \leq N$ . If these constraints do not hold, then the server pads  $L$  with empty rows and splits  $L$  into sets of  $N$  columns.

The QUERY procedure and the top half of ANSWER (until line 17) follow the intuition described above. That is, QUERY

creates a one-hot encoding of  $idx$  (line 4 in Figure 3), splits the encoding into multiple BFV plaintexts, and encrypts each plaintext separately (line 7 in Figure 3). The top half of ANSWER multiplies the  $k = n/N$  plaintext column vectors of each column of  $L$  with the corresponding ciphertexts in the query (line 16 in Figure 3), and adds the  $k$  output ciphertexts to get one ciphertext per column of  $L$  (line 17 in Figure 3). For instance, if  $n = 8$ ,  $N = 4$ ,  $idx = 1$ , and a column of  $L$  is  $(a_0, a_1, \dots, a_7)$ , then ANSWER computes encryptions of  $(0, a_1, 0, 0)$  and  $(0, 0, 0, 0)$  in line 16 of Figure 3, and adds them to get an encryption of  $(0, a_1, 0, 0)$  in line 17 of Figure 3.

The bottom half of ANSWER packs together outputs from each column into a single ciphertext (lines 19–27 in Figure 3). Suppose the number of columns is  $m = 4$  and the outputs corresponding to them are encryptions of  $(0, a_1, 0, 0)$ ,  $(0, b_1, 0, 0)$ ,  $(0, c_1, 0, 0)$ , and  $(0, d_1, 0, 0)$ , or equivalently encryptions of  $((0, a_1), (0, 0))$ ,  $((0, b_1), (0, 0))$ ,  $((0, c_1), (0, 0))$ , and  $((0, d_1), (0, 0))$ , when the underlying plaintexts are viewed in their matrix form. Then, ANSWER uses the BFV.ROWROTATE and BFV.ADD operations to produce encryptions of  $((b_1, a_1), (0, 0))$  and  $((d_1, c_1), (0, 0))$  (lines 20–26 in Figure 3), before column rotating the second ciphertext, and adding the result to the first ciphertext to obtain an encryption of  $((b_1, a_1), (d_1, c_1))$  (line 27 in Figure 3). Using rotations to pack outputs from multiple columns into a single ciphertext is crucial as otherwise a CPIR answer size can contain multiple ciphertexts (instead of one).

DECODE is straightforward; it decrypts the output of ANSWER and then rotates the plaintext depending on the value of the requested index. For the example above, DECODE first obtains the plaintext matrix  $((b_1, a_1), (d_1, c_1))$ , and then performs a rotation on this matrix by  $idx = 1$  to obtain  $((a_1, b_1), (c_1, d_1))$ .

### 4.3 Reducing the CPU cost of rotations

Recall that one goal of FastPIR is to optimize the CPU time of ANSWER procedure (§3.2). A source of inefficiency in what is described above is the cost of BFV.ROWROTATE (lines 22 and 25 in Figure 3), as the CPU time taken by it depends on the value of  $i$ —the positions by which the underlying plaintext is rotated. When  $i$  is a power of two, then BFV.ROWROTATE is fast, whereas when  $i$  is not a power of two, BFV.ROWROTATE calls itself up to  $\log_2(i + 1)$  times (§4.1). For example, a call to BFV.ROWROTATE with an input  $i = 7$  translates into three rotations by amounts one, two, and four—powers of two that add to seven.

FastPIR eliminates the calls to expensive rotations whose input rotation amount is not a power of two. As intuition, suppose that the ANSWER procedure (Figure 3) needs to make two calls to BFV.ROWROTATE—one for rotating a vector by two positions and the other for rotating a vector for another matrix column by three positions. Then, the straw man design presented in the previous subsection treats each rotation separately. Particularly, it breaks down the rotation by three

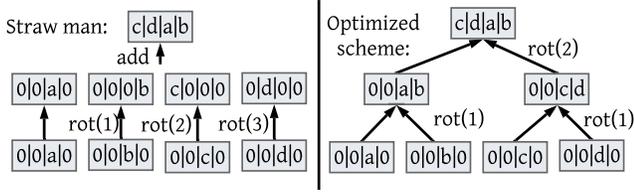


Figure 4—Illustration of optimized rotations in FastPIR. The straw man (left) performs a mix of slow rotations (with rotations amounts that are not powers of two) and fast rotations (with rotation amounts that are powers of two) to combine multiple vectors. FastPIR’s optimized scheme combines vectors using fast rotations only.

positions into a rotation by one position followed by a rotation by two positions. Instead, FastPIR first rotates the second vector by one position and adds the result to the first vector. Then, it rotates the combined vector once by two positions, thereby rotating only by powers-of-two amounts.

Figure 4 illustrates the idea for our running example with  $m = 4$  matrix columns, where the FastPIR processing for each column produces a ciphertext. FastPIR arranges the vectors to be combined as leaf nodes of a tree; it then builds up to the root of the tree. When producing a parent at a given height  $h$  of the tree, FastPIR rotates the right child by  $2^{h-1}$  positions and adds the rotated vector to the left child. The effect is that FastPIR combines  $m$  ciphertexts in lines 22 and 25 in Figure 3 using  $m$  fast rotations.

#### 4.4 Reducing the number of rotations

This optimization reduces the number of calls to `BFV.ROWROTATE` by a factor of two, and eliminates the call to `BFV.COLROTATE`, thereby further reducing the CPU cost of `ANSWER`. The trade-off is a  $2\times$  increase in CPIR query size.

The key idea is to exploit the matrix representation of a BFV plaintext (§4.1) and retrieve two elements of a matrix row (instead of one) at a time.

As motivation, suppose that the matrix  $L$  is of dimension  $N/2 \times 2$ , and the client wants the  $idx$ -th row. Then, the client sends an encryption of a vector whose  $idx$ -th and  $idx + N/2$ -th entries are one (and the rest are zeros). For instance, if  $N = 4$  and  $idx = 1$ , then the client sends an encryption of  $(0, 1, 0, 1)$ , or equivalently,  $((0, 1), (0, 1))$ . The server multiplies this query with  $L$  to get an encryption of a vector whose  $idx$ -th and  $idx + N/2$ -th entries are the desired elements from the two columns of  $L$ . As an example with  $idx = 1$ , say  $L$  is  $((a_0, a_1), (b_0, b_1))$ , then the multiplication operation produces an encryption of  $((0, a_1), (0, b_1))$ .

Figure 5 shows the procedures of FastPIR with this optimization. The procedures assume that  $n$  is a multiple of  $N/2$ , i.e.,  $n = k \cdot (N/2)$  for some  $k \geq 1$ , and  $m$  is even and  $\leq N$ . As before (§4.2), if these constraints do not hold, then the server appropriately pads and splits  $L$ .

The `QUERY` procedure encrypts a set of vectors that in total contain two non-zero entries (line 6 in Figure 5). The `ANSWER` procedure multiplies  $k$  parts of every pair of columns of  $L$  with the  $k$  ciphertexts in the query, and adds the results to

```

1: function QUERY(index  $idx$ ,  $n$ )
2:   for  $i = 0$  to  $n - 1$  do
3:      $f_i \leftarrow (i == idx) ? 1 : 0$            // one-hot encoding
4:   for  $i = 0$  to  $n/(N/2) - 1$  do
5:      $v \leftarrow (f_{i \cdot N/2}, \dots, f_{(i+1) \cdot N/2 - 1})$ 
6:      $q_i = \text{BFV.ENC}(pk, v || v)$            // || denotes concatenation
7:   return  $q = (q_0, \dots, q_{n/(N/2)-1})$ 

8: function ANSWER(library  $L$ , query  $q = (q_0, \dots, q_{n/(N/2)-1})$ )
9:   // Represent  $L$  as a matrix of elements in  $\mathbb{Z}_p$ :  $L \in \mathbb{Z}_p^{n \times m}$ 
10:  //  $q$  is an output of QUERY
11:  for  $j = 0$  to  $(m/2) - 1$  do
12:     $sum_j = \text{BFV.ENC}(pk, 0)$ 
13:    for  $i = 0$  to  $n/(N/2) - 1$  do
14:       $p_{i,j} \leftarrow \text{SUBMAT}(L, i \cdot N/2, (i+1) \cdot N/2 - 1, 2j, 2j+1)$ 
15:       $t_{i,j} = \text{BFV.SCMULT}(p_{i,j}, q_i)$ 
16:       $sum_j = \text{BFV.ADD}(sum_j, t_{i,j})$ 
17:  // Combine outputs from all pairs of columns
18:  return ROTATEANDCOMBINE( $sum_0, \dots, sum_{m/2-1}$ )

19: function DECODE(answer  $ans$ , index  $idx$ )
20:  //  $ans$  is an output of ANSWER
21:   $ans_{pt} \leftarrow \text{BFV.DEC}(sk, ans)$ 
22:  return PTROWROT( $ans_{pt}, N/2 - (idx \bmod N/2)$ )

```

Figure 5—`QUERY`, `ANSWER`, and `DECODE` procedures for FastPIR.  $(pk, sk)$  is a (public, private) key pair for the BFV scheme (§4.1). `SUBMAT` extracts a sub-matrix of a matrix. `ROTATEANDCOMBINE` refers to the optimized procedure to combine ciphertexts (§4.3). `PTROWROT` is like `BFV.ROWROTATE` except that it operates on BFV plaintexts rather than BFV ciphertexts.

get one ciphertext for every pair of columns. Then, `ANSWER` packs these outputs using the optimized scheme to combine ciphertexts described previously (§4.3). The `DECODE` procedure decrypts the output of `ANSWER` and performs a rotation on the plaintext output.

**Security analysis.** The security of a CPIR scheme requires the output of `QUERY` to not reveal any information about the requested index [19, 42]. FastPIR meets this property because its `QUERY` procedure (i) produces semantically-secure BFV ciphertexts, and (ii) outputs  $n/(N/2)$  ciphertexts independent of the value of the desired index  $idx$ .

## 5 Implementation details

**FastPIR.** Our prototype of FastPIR is  $\approx 1000$  lines of C++ and is available at <https://github.com/ishtiyaque/FastPIR>. We used the Microsoft SEAL library v3.5 [68] for the underlying cryptographic operations of the BFV scheme. Recall that FastPIR configures BFV so that it supports vectorized operations (§4.1). For vectorization, the plaintext modulus  $p$  has to be a prime number congruent to 1 (mod  $2N$ ), where  $N$  is the vector dimension of a BFV plaintext and equals  $2^{10}$  or a higher power of two (§4.1). Moreover, one needs to choose  $p \ll q$  to ensure correct decryption. For Addra, we choose  $N = 2^{12}$ ,  $p$  a 19-bit prime 270337, and  $q$  a 109-bit composite that is the product of

a 54-bit prime (18014398509309953) and a 55-bit prime (36028797018652673). These parameters provide a 128-bit security level as guided by the homomorphic encryption standard [5]. (One may choose different parameters for FastPIR based on application requirements.)

**Master-worker architecture for Addra.** We implemented Addra server using a master-worker architecture with many worker machines to distribute the PIR workload. Specifically, during the dialing phase of a round in Addra’s protocol (§3.2), the master receives CPIR queries from all devices and shards them across the workers, where a worker gets a subset of the queries. Then, during the communication phase, the master initiates each subround at a fixed schedule. During each subround, it waits to receive messages from the clients, compiles them into a message library, and broadcasts the entire message library to the workers. In case a laggard client fails to get its message to the master during the time period the master waits for incoming messages, the master buffers the laggard’s message for the next subround. If more than one message arrives at the master from a client for the same subround, the master retains the latest message. Meanwhile, to process CPIR queries, each worker computes the output of ANSWER on its assigned subset of the queries and pushes the outputs to the client devices who registered the queries.

**Dialing protocol.** Addra uses Pung’s protocol to initiate calls [10, Chapter 4.5.3] (which in turn is based on Alpenhorn [49]). Briefly, a caller sends “hello” messages encrypted with the callee’s public key to the server, who then broadcasts the set of “hello” messages from all callers to all user devices. A callee decrypts the ciphertexts using its private key and learns the content encryption key and the caller’s phone number (which are inside the hello message). This protocol is not efficient as the server broadcasts the ciphertexts to the participants (although the server could use a CDN or multicast protocols), and a callee decrypts ciphertexts from all users. Thus, Addra runs this protocol infrequently (every five minutes; §6.3). A more efficient dialing protocol in Addra’s threat model is still an open problem.

**Options for which call to pick.** A device may receive multiple incoming calls, or may make an outgoing call at the same time a call comes in. In such scenarios, Addra exposes all options to the device owner and lets them pick the call they want to participate in. However, depending on which option a user chooses, they could leak some information to the users who are on the other end in the non-chosen options. For instance, if Alice receives a call from both Bob and Charlie, and decides to pick Bob’s call, then Charlie may infer that Alice is busy. This leakage is not specific to Addra but applies to any metadata-private system [7, 8]. As efficient solutions to this problem become available, one could enhance the options-based approach currently implemented in Addra.

**Other libraries and lines of code.** Our prototype of Addra (<https://github.com/ishtiyaque/Addra>) is  $\approx 2,000$  lines of C++ on top of existing libraries, including FastPIR. Our implementation of the dialing protocol uses the libscapi [1] library for public-key encryption using the Cramer-Shoup scheme [25] with a key size of 3072 bits which provides 128 bits of security. It also uses AES-CBC implementation from OpenSSL with a 128-bit key for end-to-end content encryption with 128 bits of security. It implements the message library broadcasting mechanism from master to workers using rplib [3]. Finally, we use the open source implementation of LPCNet [56] for speech encoding/decoding.

## 6 Evaluation

Our evaluation answers the following questions:

1. What is Addra’s message latency, and how does it vary with the number of users and server machines?
2. How much resource overhead (CPU, network upload and download) does Addra impose on its server and users?
3. How does Addra compare to Pung [6, 9, 10], which is the state-of-the-art prior system for metadata-private communication over completely untrusted infrastructure?
4. How does FastPIR compare to the state-of-the-art CPIR schemes, XPIR [4] and SealPIR [6]?

A highlight of our evaluation results is as follows:

- Addra’s 99-th percentile message latency is 726 ms for 32,768 users and 80 server machines. For the same configuration, Pung’s message latency is 5.2 seconds.
- Addra’s server consumes 22.3 minutes of CPU time for a subround with 32,768 users, where a subround corresponds to 480 ms of voice call. Translated to provisioning burden, each user requires the server to provision 0.085 CPU for its call. In contrast, Pung consumes 77.1 minutes of CPU time ( $3.45\times$  higher) per subround.
- An Addra user downloads and uploads 55.1 and 1.08 MiB of data for each round when 32,768 users use Addra, where a round corresponds to five minutes of voice call. Thus, translated into bandwidth, Addra requires a download and upload bandwidth of 1.46 Mbps and 30 Kbps, respectively. In contrast, a Pung client downloads and uploads 250 MiB ( $4.6\times$  higher) and 313 MiB ( $289\times$  higher) for five minutes of voice call data.
- FastPIR has a smaller server-side CPU time *and* a smaller response size relative to XPIR and SealPIR, particularly when the number of messages in the PIR library is greater than a threshold ( $\approx 20K$ ).

**Setup and method.** We compare Addra to two variants of Pung: Pung-XPIR (P-XPIR) and Pung-SealPIR (P-SPIR). The former is the original Pung system from OSDI 2016 [9] that instantiates CPIR with the XPIR scheme [4]. The second variant replaces the XPIR scheme with the SealPIR CPIR

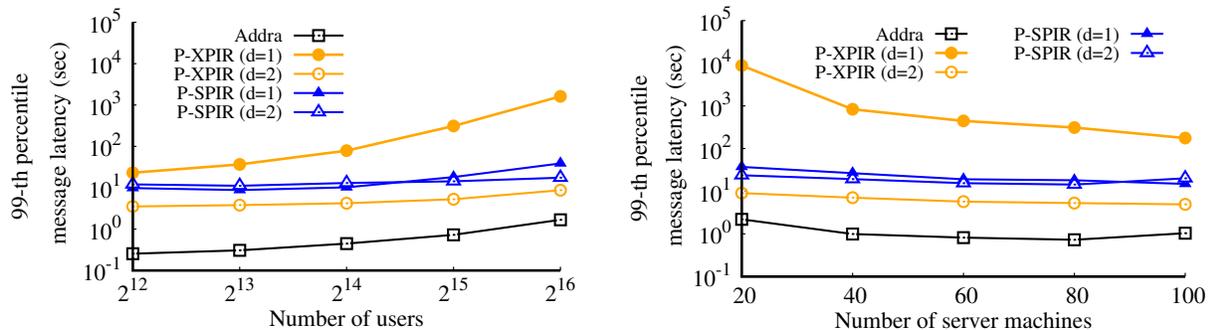


Figure 6—(Left) Message latency with a varying number of users for eighty server worker machines. (Right) Message latency with a varying number of server worker machines for 32,768 users. Messages are 96 bytes in size. The y-axis is log-scaled.  $d$  denotes CPIR recursion depth, where  $d = 1$  denotes no recursion and  $d = 2$  enables recursion. Addra does not use recursion (§4).

scheme [6]. We include both variants as there is no clear winner between them across all performance metrics. Further, we evaluate these variants without ( $d = 1$ ) and with CPIR recursion ( $d = 2$ ). We do not experiment with a recursion depth  $d > 2$  as the server CPU time and the network transfers from the server to the clients, which are the two key overhead metrics, grow significantly with depth greater than two [6].

We configure Addra and Pung to provide a security level of 128-bits. Also, we configure Pung to use its BST retrieval scheme in which a message recipient obviously searches through a tree while retrieving one message from the Pung server. This scheme is the most scalable retrieval scheme for Pung especially as the number of system users increase; we discuss other retrieval schemes Pung supports in the related work section (§7). For all of the systems, we deploy the server on a cluster of machines in AWS EC2 US East region (Ohio). Addra requires a master machine and a set of worker machines (§5). For the master, we use a machine of type `c5.24xlarge` (96 vCPU, 192 GiB of RAM and 25 Gbps of network bandwidth) which provides a high network bandwidth to enable the master to broadcast the message library (the mailboxes) to the workers. For the workers, we use the compute-optimized machines of type `c5.12xlarge` (48 vCPU, 96 GiB of RAM, and 12 Gbps of network bandwidth). Pung does not have a master and therefore we use machines of type `c5.12xlarge` as its workers. To compensate for the extra master machine assigned to Addra (relative to Pung), we assign two additional worker machines of type `c5.12xlarge` to Pung.

Addra is required to process queries from all clients in every subround to meet its security goals. Since we cannot run tens of thousands of clients in our infrastructure, we employ a combination of real and simulated clients. We deploy 256 geographically distant real clients in a machine of type `c5.24xlarge` in AWS US West (N. California). The mean network RTT, as measured by Ping, between the server and these clients is 51 ms. During each round and subround, real clients send their queries and messages to the server, and the server inserts the queries and messages of the remaining simulated clients.

We configure Addra to run a round every five minutes and a subround every 480 ms. This configuration results in a fixed message size of 96 bytes at each subround as the LPCNet voice codec encodes a 40 ms audio frame into 8 bytes (§2.1) [73, 74]. We vary the number of users (from 4,096 to 65,536) and the number of worker machines (from 20 to 100). We repeat experiments for 10 trials. To account for tail latency, we process the queries from real clients only after processing the queries from all simulated clients. Then, we measure the 99-th percentile latency observed by the real clients over the 10 trials, the CPU time consumed by the server and the real clients, and the amount of data uploaded and downloaded by the real clients.

## 6.1 Message latency

**Variation with the number of users.** Figure 6 (left) shows the 99-th percentile message latency with a varying number of users when the server has 80 worker machines.

Addra’s message latency is 254 ms for 4,096 users and increases to 1678 ms for 65,536 users. This increase is due to three reasons. First, as the number of users increases, so does the number of mailboxes and the time to broadcast their content from the master to the workers (§5). Second, the number of CPIR queries the server processes every subround equals the number of users (§3.2). Third, the time to process a CPIR query increases with the number of mailboxes, so each worker takes longer to generate CPIR responses. For 32,768 users, the latency is 726 ms, of which 398 ms is for CPIR query processing at the workers, 186 ms is for broadcast of mailbox content from the master to the workers, and the rest is for network transfers between the client and the server. However, for 65,536 users, the latency increases to 1,678 ms, of which 1,186 ms is for CPIR query processing alone. This processing time is higher than the 480 ms subround time budget and thus voice packets start queuing up at the server for these many users.

Addra’s message latency is lower than Pung’s, specifically, that of Pung-XPIR by a factor of  $7.2\times$  for 32,768 users, due to two reasons. First, a sender in Addra pushes a message to the server, who performs CPIR processing and pushes the

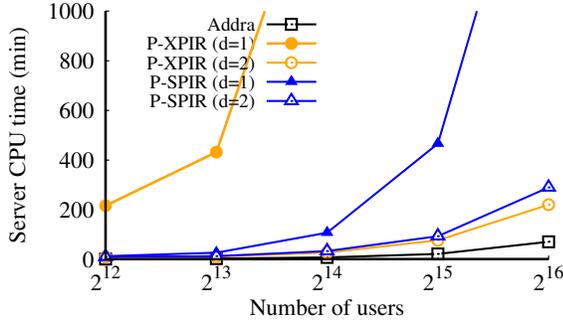


Figure 7—Server-side CPU time per subround with a varying number of users. A subround corresponds to 480 ms of voice call; in a subround, each user sends and receives one 96 byte message.

response to the recipient—in total, the message traverses two hops (§3.2). In contrast, while the sender in Pung pushes a message to the server in one hop, a recipient has to make  $\lceil \log_2(n+1) \rceil$  sequential round-trips to the server to fetch a message, where  $n$  is the number of users. Second, Addra uses FastPIR, which has lower server-side CPIR answer generation time than XPIR or SealPIR used in Pung; we will expand on this difference shortly (§6.2, §6.5).

**Variation with the number of worker machines.** Figure 6 (right) shows the 99-th percentile message latency as a function of the number of worker machines when the number of users is fixed to 32,768. Latency decreases for all systems with an increase in the number of worker machines due to increased parallelization for CPIR answer generation, but only up to an inflection point. Beyond this inflection point, adding workers does not improve latency as the time to replicate mailboxes from the master to the workers goes up, while the CPU on the workers starts to become idle. Thus, an immediate scalability bottleneck in Addra is the time to broadcast mailboxes from the master to the workers. Distributing the master or reducing the number of workers by extracting more efficiency from each may further push out the inflection point.

## 6.2 Server-side CPU consumption

Figure 7 shows that server-side CPU time increases with the number of users. This is expected as both the number of CPIR queries and the time to generate an answer for each query increases with the number of users (§3.2). Addra’s CPU consumption is lower than Pung’s. For instance, for 32,768 users, Addra takes 22.3 minutes while Pung (with XPIR and CPIR recursion depth  $d = 2$ ) takes 77.1 minutes ( $3.45\times$  higher). If we convert these times to CPU provisioning requirements, then for each subround lasting 480 ms or 0.48 seconds, Addra’s server consumes 22.3 minutes, or 1,338 seconds, of CPU, which is provided by provisioning  $1,338/0.48 = 2788$  CPUs, or 0.085 CPU per user. Similarly, each Pung user requires 0.29 CPU per user. A key reason for this difference is that FastPIR in Addra consumes lower amount of server-side CPU relative to XPIR or SealPIR in Pung (§6.5). We note that even though a recursion depth of  $d = 2$  reduces CPU consumption relative

to no recursion ( $d = 1$ ), increasing depth further ( $d = 3$ ) does not reduce CPU consumption [6]. Furthermore, a higher depth increases network overhead (§6.3). Thus, as mentioned earlier (§6), we restrict our experiments to a depth of  $d = 2$ .

## 6.3 Client-side resource overheads

**Network transfers.** Figure 8 shows the amount of data a client downloads and uploads for one round of communication (a round corresponds to five minutes of voice call).

An Addra user downloads  $\approx 55.1$  MiB in a five-minute round when 32,768 users use Addra. That is, each user requires 1.46 Mbps of network download bandwidth. Of the 55.1 MiB,  $\approx 39$  MiB is due to the communication phase of the round while the rest is due to the dialing phase (§3.2). Further, the former is independent of the number of system users, while the latter depends linearly on the number of users.

Relative to a non-private baseline which does not hide metadata, Addra’s network overhead is significantly higher due to the use of CPIR, which encrypts messages into BFV ciphertexts. For example, if the non-private baseline uses LPCNet which encodes 480 ms of speech in 96 bytes of data, then a user’s network download bandwidth will be 1.56 Kbps. In contrast, Addra encrypts the 96 bytes into a 64 KB ciphertext, which is a  $682\times$  increase.

However, relative to Pung, an Addra user downloads less data, by  $4.5\text{--}45.7\times$ , depending on the Pung variant. The improvement is due to two reasons. First, Pung requires a message recipient to make multiple CPIR queries with the server to search through the message library that is organized as a tree. Second, CPIR answer size increases with a higher CPIR recursion depth ( $d = 2$  versus  $d = 1$ ). Addra’s FastPIR, on the other hand, operates at  $d = 1$  to keep CPIR answer sizes, and thus the downloads, smaller (§4).

An Addra user uploads one CPIR query per round during its dialing phase. The Addra server then reuses the query across subrounds (§3.2). Even though the query size for Addra is larger compared to that of Pung (§6.5), unlike Pung, this cost is amortized over multiple subrounds of the communication phase. As a result, Addra’s upload network transfers are small:  $\approx 1.1$  MiB per round, or 30 Kbps.

We remark that even though Addra’s instantaneous network overhead (1.46 Mbps download and 30 Kbps upload) appears manageable, it adds up over time due to the involvement of a client in dummy calls (§3.2). Thus, Addra requires certain conditions such as unlimited network downloads for its clients to be deployable. We anticipate that in the future, as the need for privacy increases, so will advances in network technology that will provide options for unlimited data.

**CPU time.** An Addra client consumes  $\approx 27.5$  seconds of CPU time per a five-minute round when the number of users is 32,768. 94% of this time is from the dialing protocol (§5). For the same configuration, a Pung client consumes  $1.7\text{--}63\times$  higher CPU, primarily due to multiple CPIR queries with the server for transmitting each message.

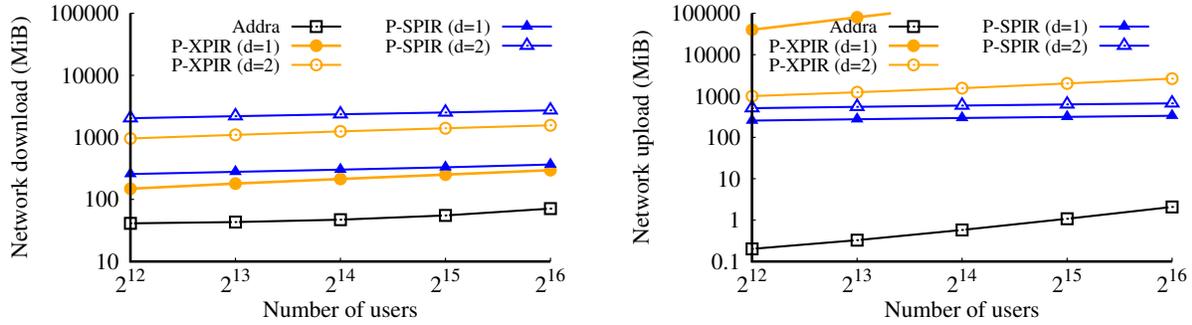


Figure 8—Data downloaded and uploaded by a user per round with varying number of users. A round corresponds to five minutes of voice call.

#### 6.4 Discussion on voice quality

The quality of voice calls and user experience depends on several factors including message transmission latency, jitter (the inconsistencies among packet arrival intervals [40]), and the effectiveness of the voice encoder that converts human speech into a digital signal. This section briefly discusses Addra’s performance on these metrics.

We reported Addra’s message transmission latency in §6.1. Specifically, latency varies with the number of users, and is lower for a lower number of users. For example, for 8K users, the latency is 306 ms, which is below the ITU-G.114 recommended value of 400ms [39]. As the number of users increases, Addra’s latency crosses the recommended value, but stays below one second for a significant number of users (32,768); this value of one second is critical as it is possible to make voice calls at this latency [48].

To measure jitter, we ran Addra for one round (i.e., 5 minutes of voice call) with 80 worker machines and a varying number of users. Ideally, a user should receive a voice packet every 480 ms, which is the duration of one subround. We measured the interval between consecutive packet arrival timestamps and calculated the absolute deviation of this value from 480 ms as jitter. Addra’s mean jitter is 4.1 ms for 4,096 users and increases to 36.8 ms for 32,768 users. This increase with the number of users is correlated with higher CPU and network load at the server.

Finally, the effectiveness of voice encoding is a property of the encoder. Addra’s current prototype uses the LPCNet [56] encoder developed by Mozilla. Conducting a user experience study on LPCNet’s quality is outside the scope of this paper, but we refer the reader to the original paper on LPCNet that discusses a subjective assessment of LPCNet’s quality based on an experiment with one hundred human listeners [74].

#### 6.5 Comparison of CPIR schemes

A core component of Addra and Pung is the CPIR cryptographic primitive. Pung uses either XPIR or SealPIR, which are also the state-of-the-art schemes. Addra uses FastPIR (§4). This section compares the cost of these CPIR schemes in isolation. Besides, since CPIR applies to several other contexts [13, 29, 35, 55], this section sheds light on which

scheme could be better for which application.

We microbenchmarked the XPIR, SealPIR, and FastPIR libraries on a single CPU of an AWS instance of type c5.12xlarge (48 vCPU, 3.6 GHz, 96 GiB RAM). We configured all three libraries for a 128-bit security level. However, XPIR does not set parameters from the homomorphic encryption standard [5], and its parameters are smaller relative to those for SealPIR and FastPIR.

We varied the number of messages in the library ( $n \in \{2^{13}, \dots, 2^{20}\}$ ) and the size of each message ( $m \in \{96\text{B}, 256\text{B}, 1024\text{B}\}$ ). The lowest value of  $n$  captures a small library with a few thousand messages, while the other extreme of  $n = 2^{20}$  demonstrates how FastPIR scales with the number of messages relative to the other CPIR libraries. Similarly, the different message sizes demonstrate performance for scenarios with small messages (for example, Addra) and also larger messages.

We measure and report both CPU and network overhead for query generation (QUERY), answer generation (ANSWER), and answer decode (DECODE) CPIR procedures, for 10 trials. Given that the CPIR cost in Addra is dominated by the cost to run the ANSWER procedure, we describe the results while focusing on ANSWER. At a high level, FastPIR keeps both the CPU cost for ANSWER and the size of ANSWER output small, while XPIR and SealPIR sacrifice one of the two.

**CPU time for ANSWER.** Figure 9 shows the CPU time for the ANSWER procedure for different values of  $n$  and  $m$ . Beyond a threshold  $n$ , and for all values of  $m$ , FastPIR consumes the least amount of CPU time for ANSWER independent of whether the baselines use recursion or not ( $d = 1$  is no recursion, and  $d = 2$  enables it). For instance, when  $n=2^{20}$  and  $m=256\text{B}$ , ANSWER in FastPIR takes  $2.5\times$  less time than XPIR ( $d = 2$ ) and  $2.7\times$  less time than SealPIR ( $d = 2$ ).

The figure also shows the impact of FastPIR’s optimizations (§4.3, §4.4) in reducing its CPU overhead. For smaller values of  $n$ , the impact of these optimizations is significant. For instance, for  $n=2^{15}$  and  $m=256\text{B}$ , FastPIR without the two optimizations (F-1 in the figure) is  $2.73\times$  more expensive than the full-fledged FastPIR, while FastPIR without its last optimization in §4.4 (F-2 in the figure) is  $1.45\times$  more expensive than FastPIR will all optimizations enabled. But,

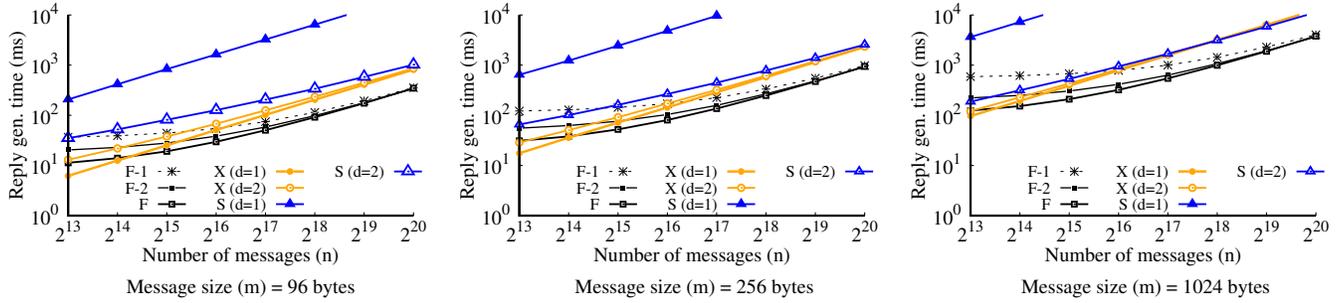


Figure 9—CPU time to run the ANSWER CPIR procedure for XPIR (X), SealPIR (S), and three variants of FastPIR (F) with a varying number of messages  $n$  in the server library and a varying size  $m$  of each message. F-1 and F-2 are intermediate baselines for FastPIR: F-1 leaves out both optimizations for the rotation operations (§4.3 and §4.4), while F-2 leaves out only the optimization in §4.4. Both axes are log-scaled.  $d$  denotes recursion depth (§4). FastPIR does not use recursion (sets  $d = 1$ , which means no recursion). For both XPIR and SealPIR, an optimization to aggregate multiple small messages into a larger one (called aggregation in the literature) is enabled.

	$n = 32,768$					$n = 1,048,576$				
	X ( $d = 1$ )	X ( $d = 2$ )	S ( $d = 1$ )	S ( $d = 2$ )	F ( $d = 1$ )	X ( $d = 1$ )	X ( $d = 2$ )	S ( $d = 1$ )	S ( $d = 2$ )	F ( $d = 1$ )
<b>query size (KiB)</b>										
$m = 96$ bytes	33,856	2,112	32	64	1,024	1,082,432	11,776	928	64	32,768
$m = 256$ bytes	95,328	3,520	96	64	1,024	3,050,432	19,776	2,752	64	32,768
$m = 1024$ bytes	524,288	8,192	512	64	1,024	16,777,216	46,368	16,384	64	32,768
<b>answer size (KiB)</b>										
$m \in \{96B, 256B, 1024B\}$	32	256	32	320	64	32	288	32	320	64
<b>client CPU costs (ms)</b>										
QUERY ( $m = 96B$ )	118.6	7.4	0.7	1.4	21.3	3801.8	41.5	19.2	1.4	679.0
QUERY ( $m = 256B$ )	335.2	12.4	2.0	1.4	21.4	10711.3	69.8	56.9	1.4	678.6
QUERY ( $m = 1024B$ )	1841.6	28.8	10.6	1.4	21.4	58990.8	164.2	338.8	1.4	678.7
DECODE ( $m \in \{96B, 256B, 1024B\}$ )	0.1	0.41	0.19	1.88	0.36	0.1	0.37	0.2	1.86	0.41

Figure 10—Network costs and client-side CPU costs for XPIR (X), SealPIR (S), and FastPIR (F) with a varying number of messages ( $n$ ) and the size of each message ( $m$ ) in the server library.  $d$  denotes recursion depth (§4). FastPIR does not use recursion (sets  $d = 1$ , which means no recursion). For both XPIR and SealPIR, an optimization to aggregate multiple small messages into a larger one (called aggregation in the literature) is enabled.

as  $n$  increases the lower CPU time benefit of the optimizations diminishes. This trend is expected as for larger  $n$  the cost for the ANSWER procedure is dominated by the time to run the BFV.SCMULT and BFV.ADD operations rather than the rotation operations, which is what the optimizations focus on (§4.2–§4.4).

**Output size of ANSWER.** Figure 10 shows the size of the CPIR response generated by the ANSWER procedure for the three CPIR schemes. When the schemes do not use recursion ( $d = 1$ ), their answer output sizes are smaller relative to when they use recursion, although FastPIR’s response size is double the size of XPIR and SealPIR. However,  $d = 1$  is not a viable solution for either XPIR or SealPIR. For XPIR, the query size is large for  $d = 1$ , which increases network bandwidth and CPU time for processing of CPIR queries (Figure 7). For SealPIR, the compressed query is smaller on the wire, but the expanded query has comparable size to that of XPIR. Furthermore, the cost to expand adds significant CPU time for SealPIR  $d = 1$ .

When the schemes use recursion ( $d = 2$ ), both XPIR and SealPIR do not have the query-size drawback, but increase

answer output size, by 8 to 10 times, relative to the  $d = 1$  setting. Overall, FastPIR produces smaller responses (answer outputs) without large queries (XPIR with  $d = 1$ ) or significant addition to computation time (SealPIR with  $d = 1$ ).

**Query-related overheads.** Query generation time and query sizes are significantly larger in FastPIR than SealPIR (especially when the latter uses recursion). For instance, query size for  $2^{15}$  items in SealPIR with  $d = 2$  is 17 times smaller than the query size in FastPIR (with  $d = 1$ ). However, FastPIR’s query sizes are either smaller or comparable to those for XPIR, depending on recursion depth and message size.

**Summary.** If ANSWER is invoked frequently for an application with a library that has over several tens of thousands of messages, then FastPIR is a better fit. However, if the application cannot be designed such that its costs are dominated by those of ANSWER, then SealPIR or XPIR may be a better fit.

## 7 Related work

**Onion-routing.** Systems such as Tor [71], which are based on onion-routing [34, 64], can support anonymous VoIP calls

with low message latency. However, they do not provide strong guarantees. Indeed, a network adversary, such as an ISP, can learn call metadata via traffic analysis [15, 37, 43, 57, 61].

**Mix-nets.** Chaum introduced a mix-net: a network of nodes in which each node (called a mix) batches incoming messages and releases them in a permuted order [17]. A mix-net based system fundamentally requires at least one mix to be trusted [44–48, 50, 51, 63, 72, 75]. Yodel [48] is a state-of-the-art system based on mix-nets that specifically targets voice calls. Yodel scales to a few million users while providing a sub-second message latency. However, Yodel assumes that a fraction of the mixes it uses (80%) are not compromised. As one relaxes this assumption, say to make the fraction of trusted mixes to be 70% or lower, Yodel increases the latency between a caller and a callee.

**DC-nets.** Unlike a mix-net, a dining cryptographers network (DC-net) provides unconditional security using a technique that requires broadcasting of messages between network participants [16]. Due to the broadcasting requirement, earlier systems based on DC-nets scaled to only tens of participants [23, 33, 69]. Later systems [24, 76] improved scalability but at the cost of relaxing the threat model. For instance, Dissent in numbers [76] scales to 5000 clients with 600 ms latency for 600-client groups, but runs a DC-net among a (smaller) group of servers while assuming that one of them is trusted. PriFi [12] is the latest DC-net based system. It improves latency for a LAN setting of a small organization with a few hundred users (latency is 100 ms for 100 users). PriFi does not scale to thousands or tens of thousands of users. It also assumes that one of its servers in the group of servers is trusted.

**Private mailboxes.** Systems based on private mailboxes either obliviously write to [22, 31] or read from [6, 9, 13, 41, 67] mailboxes hosted over untrusted servers. The state-of-the-art system based on this strategy that works over completely untrusted infrastructure is Pung [6, 9] (rest of the systems assume non-colluding servers).

We empirically compared Addra to Pung, particularly to its scalable tree-based message retrieval scheme called BST (§6). Pung offers two other retrieval schemes: one called explicit retrieval and the other based on Bloom filters. The explicit scheme requires two round trips between a message recipient and the server, and incurs comparable server-side CPU overhead as the BST scheme. However, it is not viable in terms of network overhead as the server has to frequently broadcast a mapping comparable in size to the entire message library. For instance, for 32K users, the server pushes 625 MiB of mapping data every five minutes to every user, thus adding a bandwidth requirement of 16.6 Mbps per user. The Bloom filter scheme significantly lowers the network overhead relative to the explicit scheme. However, its overhead is still linear in the number of objects (so it is not a viable solution as the system scales up to hundreds of thousands of users).

Besides, it works probabilistically: a message recipient is not guaranteed to download the message sent by the sender, thus degrading the quality of service by a non-zero amount.

Although Addra supports synchronous voice calls at scale, and Pung does not (§6.1), Addra does not replace Pung, which is designed for asynchronous applications such as email and chat. Indeed, Addra cannot retrieve long-lived messages from the server, which is a requirement for such applications.

**Private information retrieval (PIR).** Chor et al. [18, 19] introduced the problem of PIR over multiple non-colluding servers, while Kushilevitz and Ostrovsky [42] introduced CPIR over a single untrusted server. Since these decades old seminal works, there have been numerous improvements to concrete constructions of PIR. For instance, some schemes reduce PIR overheads [4, 6, 27, 28, 70], while others improve answer recovery against a byzantine server [30, 59]. In this paper, we introduced FastPIR, a new CPIR scheme that reduces the server-side computation overhead relative to the state-of-the-art CPIR schemes [4, 6] (§6.5).

## 8 Summary and future work

Metadata from voice calls contains rich information about people’s lives, and is a prime target for powerful adversaries such as nation states. Prior work that hides metadata either requires trusted intermediaries or does not scale to more than tens of users for low-latency voice calls. This paper described Addra, the first system that hides metadata for voice calls over completely untrusted infrastructure for tens of thousands of users. Addra’s current prototype supports 32,768 users on a cluster of 80 machines with a message latency of 726 ms and a voice synthesis rate of 1.6 Kbps. Addra provides its performance and privacy properties through a new, simple, and efficient protocol to access private mailboxes hosted on an untrusted server (§3), and a new private information retrieval (PIR) scheme, FastPIR (§4).

Our future work involves further scaling Addra from tens of thousands of users to hundreds of thousands or a few million users. To accelerate CPIR computation, a promising direction could be to explore efficient implementations of the master-worker architecture of Addra’s server, as well as increased efficiency for the workers using GPUs and FPGAs. For the latter, one would have to address challenges related to running PIR on a heterogeneous system. Finally, a full-fledged Addra system would require extending its support from peer-to-peer voice calls to group calls.

## Acknowledgments

We thank Sujaya Maiyya, Udit Paul, Nazmus Saquib, our shepherd Sebastian Angel, and the anonymous reviewers of OSDI 2021 for their feedback and insightful comments that helped improve this paper. This work is funded in part by NSF grants CNS-1703560 and CNS-1815733.

## References

- [1] Libscapi - the secure computation API. <https://github.com/cryptobiu/libscapi>.
- [2] PALISADE homomorphic encryption software library. <https://palisade-crypto.org/>.
- [3] rpclib - modern msgpack-rpc for C++. <http://rpclib.net/>.
- [4] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR: Private information retrieval for everyone. *Privacy Enhancing Technologies Symposium (PETS)*, 2016(2):155–174, 2016.
- [5] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, November 2018.
- [6] S. Angel, H. Chen, K. Laine, and S. Setty. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.
- [7] S. Angel, S. Kannan, and Z. Ratliff. Private resource allocators and their applications. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.
- [8] S. Angel, D. Lazar, and I. Tzialla. What’s a little leakage between friends? In *Workshop on Privacy in the Electronic Society (WPES)*, 2018.
- [9] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
- [10] S. G. Angel. *Unobservable communication over untrusted infrastructure*. PhD thesis, The University of Texas at Austin, 2018.
- [11] J. Angwin, C. Savage, J. Larson, H. Moltke, L. Poitras, and J. Risen. AT&T helped US spy on Internet on a vast scale. *The New York Times*, 2015.
- [12] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J.-P. Hubaux. PriFi: Low-latency anonymity for organizational networks. *Privacy Enhancing Technologies Symposium (PETS)*, 2020(4):24–47, 2020.
- [13] N. Borisov, G. Danezis, and I. Goldberg. DP5: A private presence service. *Privacy Enhancing Technologies Symposium (PETS)*, 2015(2):4–24, 2015.
- [14] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology—CRYPTO*, 2012.
- [15] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.
- [16] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.
- [17] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
- [18] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1995.
- [19] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.
- [20] D. Cole. We kill people based on metadata. *The New York Review*, May 2014.
- [21] D. Cole. Is privacy obsolete? *The Nation*, Mar. 2015.
- [22] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.
- [23] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.
- [24] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in Verdict. In *USENIX Security Symposium*, 2013.
- [25] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
- [26] Y.-A. De Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3(1):1–5, 2013.
- [27] D. Demmler, A. Herzberg, and T. Schneider. RAID-PIR: Practical multi-server PIR. In *ACM Workshop on Cloud Computing Security (CCSW)*, 2014.
- [28] C. Devet. Evaluating private information retrieval on the cloud. Technical report, University of Waterloo, 2013.
- [29] C. Devet and I. Goldberg. The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 63–82, 2014.
- [30] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *USENIX Security Symposium*, pages 269–283, 2012.
- [31] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *USENIX Security Symposium*, 2021.
- [32] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
- [33] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.
- [34] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.
- [35] M. Green, W. Ladd, and I. Miers. A protocol for pri-

- vately reporting ad impressions at scale. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.
- [36] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with Popcorn. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.
- [37] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):1–28, 2010.
- [38] S. Humphreys and M. De Zwart. Data retention, journalist freedoms and whistleblowers. *Media International Australia*, 165(1):103–116, 2017.
- [39] ITU-T. G.114 : One-way transmission time. <https://www.itu.int/rec/T-REC-G.114-200305-I/en>, 2003.
- [40] M. Kassim, R. A. Rahman, M. A. A. Aziz, A. Idris, and M. I. Yusof. Performance analysis of VoIP over 3G and 4G LTE network. In *2017 International Conference on Electrical, Electronics and System Engineering (ICEESE)*, pages 37–41. IEEE, 2017.
- [41] L. Kissner, A. Oprea, M. K. Reiter, D. Song, and K. Yang. Private keyword-based push and pull with applications to anonymous communication. In *International Conference on Applied Cryptography and Network Security (ACNS)*, 2004.
- [42] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1997.
- [43] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *USENIX Security Symposium*, 2015.
- [44] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.
- [45] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An efficient communication system with strong anonymity. *Privacy Enhancing Technologies Symposium (PETS)*, 2016(2):115–134, 2016.
- [46] A. Kwon, D. Lu, and S. Devadas. XRD: Scalable messaging system with cryptographic privacy. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.
- [47] D. Lazar, Y. Gilad, and N. Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.
- [48] D. Lazar, Y. Gilad, and N. Zeldovich. Yodel: Strong metadata security for voice calls. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2019.
- [49] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
- [50] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *ACM SIGCOMM Conference*, 2015.
- [51] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Balani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review*, 43(4):303–314, 2013.
- [52] R. Lenzner. ATT, Verizon, Sprint are paid cash by NSA for your private communications. *Forbes*, 2013.
- [53] J. Mayer, P. Mutchler, and J. C. Mitchell. Evaluating the privacy properties of telephone metadata. *Proceedings of the National Academy of Sciences*, 113(20):5536–5541, 2016.
- [54] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *International conference on Web search and data mining*, 2010.
- [55] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *USENIX Security Symposium*, 2011.
- [56] Mozilla. LPCNet: Efficient neural speech synthesis. <https://github.com/mozilla/LPCNet>.
- [57] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, 2005.
- [58] Office of the Director of National Intelligence. Statistical transparency report regarding the use of national security authorities. [https://www.dni.gov/files/CLPT/documents/2020\\_ASTR\\_for\\_CY2019\\_FINAL.pdf](https://www.dni.gov/files/CLPT/documents/2020_ASTR_for_CY2019_FINAL.pdf), 2020.
- [59] F. Olumofin and I. Goldberg. Revisiting the computational practicality of private information retrieval. In *International Conference on Financial Cryptography and Data Security (FC)*, 2011.
- [60] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1999.
- [61] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Workshop on Privacy in the Electronic Society (WPES)*, 2011.
- [62] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. [http://www.maroki.de/pub/dphistory/2010\\_Anon\\_Terminology\\_v0.34.pdf](http://www.maroki.de/pub/dphistory/2010_Anon_Terminology_v0.34.pdf), 2010.

- [63] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The Loopix anonymity system. In *USENIX Security Symposium*, 2017.
- [64] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.
- [65] A. Rusbridger. The Snowden leaks and the public. *The New York Review*, Nov. 2013.
- [66] D. Rushe. Yahoo \$250,000 daily fine over NSA data refusal was set to double “every week”. *The Guardian*, 2014.
- [67] L. Sassaman, B. Cohen, and N. Mathewson. The Pynchon Gate: A secure method of pseudonymous mail retrieval. In *Workshop on Privacy in the Electronic Society (WPES)*, 2005.
- [68] Microsoft SEAL (release 3.5). <https://github.com/Microsoft/SEAL>, Apr. 2020. Microsoft Research, Redmond, WA.
- [69] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding carnivores: File sharing with strong anonymity. In *Proceedings of the ACM SIGOPS European workshop*, 2004.
- [70] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 1998.
- [71] P. Syverson, R. Dingledine, and N. Mathewson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.
- [72] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 423–440, 2017.
- [73] J.-M. Valin and J. Skoglund. LPCNet: Improving neural speech synthesis through linear prediction. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.
- [74] J.-M. Valin and J. Skoglund. A real-time wideband neural vocoder at 1.6 kb/s using LPCNet. *arXiv preprint arXiv:1903.12087*, 2019.
- [75] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2015.
- [76] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

## A Security proof

In this appendix, we show that Addra’s protocol (§3.2) meets the relationship unobservability property (§2.1). Our proof

follows the proof technique of Pung [10, Appendix C], whose protocol also works over completely untrusted infrastructure and meets relationship unobservability (although with higher message latencies and overheads). We first define an abstract protocol for metadata private voice communication, then describe a cryptographic security game that captures the relationship unobservability property, and finally show why an adversary cannot win the game with non-negligible probability when the abstract protocol is instantiated with the Addra protocol.

### A.1 An abstract protocol

We define an abstract protocol for metadata private voice communication using the following three algorithms:  $\text{INIT}(1^\lambda)$ ,  $\text{RETRABSTRACT}(i, j)$ , and  $\text{SENDABSTRACT}(i, j, m_{i \rightarrow j})$ .

$\text{INIT}(1^\lambda)$  takes as input a security parameter and initializes the state of the protocol for each participant of the protocol. In particular, it establishes the content encryption key between pairs of user devices that communicate via the system.

$\text{RETRABSTRACT}(i, j)$  takes as input a user identifier  $i$  for the caller and a user identifier  $j$  for the callee, and generates a retrieval request for messages sent to  $i$  by  $j$ .

$\text{SENDABSTRACT}(i, j, m_{i \rightarrow j})$  takes as input a user identifier  $i$  for the caller, a recipient identifier  $j$  for the callee, and a message (which may be  $\perp$  if sender is idle) that the caller wants to send to the callee, and outputs a tuple that is sent to the server.

The protocol proceeds in rounds, each of which consists of  $t \geq 1$  subrounds. At the beginning of a round, each participant calls the  $\text{INIT}$  algorithm. Then, each participant calls the  $\text{RETRABSTRACT}$  algorithm to generate a request to indicate that it wants to get messages from its peer. Finally, each participant calls the  $\text{SENDABSTRACT}$  algorithm  $t$  times to send  $t$  messages to its peer.

### A.2 The security game

We define a security game that a challenger and an adversary play that captures the relationship unobservability property. We denote this game as  $\mathcal{G}_{\mathcal{A}, \pi, K, t}^b(1^\lambda)$ , where  $\mathcal{A}$  is the adversary,  $\pi$  is a round-based protocol consisting of the  $\text{INIT}$ ,  $\text{RETRABSTRACT}$ , and  $\text{SENDABSTRACT}$  algorithms,  $K$  is the number of correct users (not controlled by the adversary),  $t$  is the number of subrounds (messages exchanged per round of the protocol  $\pi$ ), and  $\lambda$  is a security parameter. The game has three phases: setup, simulation, and guess. At a high level, during setup the adversary creates two scenarios for the challenger. The challenger then during the simulation phase runs the protocol  $\pi$  for one of the scenarios selected randomly (that is, the  $b$ -th scenario) and sends the transcript of the protocol messages due to  $\pi$  to the adversary. Finally, the adversary in the guess phase guesses which scenario the challenger simulated. The adversary wins the game if the guess is correct. We expand on these three phases next.

**Setup phase.** During setup, the adversary supplies two scenarios  $M^0$  and  $M^1$  to the challenger. Each scenario has

$K$  tuples corresponding to the actions of  $K$  correct (non-compromised) users.

Each tuple has an entry for the send and the retrieve part of the protocol. For the  $k$ -th ( $k \leq K$ ) tuple, that is, for the  $k$ -th correct user, the send part of the  $b$ -th scenario,  $M^b[k].send = (i, j^b, \{m_{i \rightarrow j}^b\}_t)$  specifies that the user device with id  $i$  should send the set of  $t$  messages  $\{m_{i \rightarrow j}^b\}_t$  to user with id  $j^b$ . The messages could be dummy ( $\perp$ ). Similarly, for the  $k$ -th tuple, the retrieve part of the  $b$ -th scenario  $M^b[k].retr = (i, \ell^b)$  specifies that the user with id  $i$  should retrieve messages from the user with id  $\ell^b$ .

The adversary constructs the two scenarios  $M^0$  and  $M^1$  and supplies them to the challenger. However, the adversary has three restrictions on how it can construct the scenarios. First, both scenarios must have the same number of entries describing the actions of each user in every round. Second, both scenarios should describe the send and retrieve actions of correct users only. This is required because the challenger can simulate the actions of correct users only. Third, if  $j^b$  or  $\ell^b$  is a compromised user, then that *send* or *retr* entry should be identical across the two scenarios. Recall that relationship unobservability gives guarantees only for correct pairs of users, so such a restriction is necessary.

**Simulation phase.** During the simulation phase, the challenger simulates a protocol on one of the two scenarios that it picks randomly using a coin flip. The challenger uses the SIMULATE function described in Figure 11. The challenger then sends the output of the algorithm to the adversary.

Note that in the simulation algorithm, the challenger calls the functions exposed by the adversary. For instance, it calls GETMAILBOXIDS to learn the mailbox IDs the adversary assigns to the users. Similarly, the challenger calls GETMAILBOXACcesSTOKENS GETNUMMAILBOXES to learn mailbox access tokens and the total number of mailboxes. Finally, the challenger calls the GETRESPONSE function to learn the output of a subround, if any, for a correct user. These functions give the adversary an opportunity to set these untrusted parts of the protocol. For instance, GETRESPONSE allows adversary to drop requests or reorder them or compute the responses to requests incorrectly.

**Guess phase.** In the guess phase, the adversary  $\mathcal{A}$  outputs its guess  $b' \in \{0, 1\}$  whether the challenger simulated the  $M^0$  or  $M^1$  scenario. The adversary wins the game if its guess is correct, that is,  $b = b'$ .

### A.3 Proof

We want to show that the adversary's advantage in winning the game described above is negligible in the security parameter  $\lambda$  when the abstract protocol is instantiated with the *Addra* protocol. We use a series of hybrid games to calculate the adversary's advantage.

**Game 0.** This game is the original game as described above with  $\pi$  instantiated with the *Addra* protocol. That is, the INIT algorithm establishes a shared secret between pairs of

```

1: function SIMULATE( $\mathcal{A}, \pi, K, t, M^b$ )
2:    $requests \leftarrow \{\}$ 
3:    $responses \leftarrow \{\}$ 
4:   // Initialize content encryption key across parties
5:   //  $\mathcal{K}$  contains the shared secrets (keys)
6:    $\mathcal{K} \leftarrow \pi.INIT(1^\lambda)$ 

7:   // Obtain mailbox IDs, access tokens, number of mailboxes
8:   //  $\mathcal{M}$  contains the assigned mailbox IDs
9:    $\mathcal{M} \leftarrow \mathcal{A}.GETMAILBOXIDS()$ 
10:  //  $\mathcal{T}$  contains the assigned tokens
11:   $\mathcal{T} \leftarrow \mathcal{A}.GETMAILBOXACcesSTOKENS()$ 
12:  //  $\mathcal{N}$  contains the number of advertised mailboxes
13:   $\mathcal{N} \leftarrow \mathcal{A}.GETNUMMAILBOXES()$ 

14:  // Run retrieve part of protocol
15:  for  $k = 0$  to  $K - 1$  do
16:     $(i, j) \leftarrow M^b[k].retr$ 
17:     $req \leftarrow \pi.RETRABSTRACT(i, j)$ 
18:     $requests \xleftarrow{\text{insert}} req$ 

19:  // Run send part of protocol
20:  for  $r = 0$  to  $t - 1$  do
21:    for  $k = 0$  to  $K - 1$  do
22:       $(i, j, \{m_{i \rightarrow j}\}_t) \leftarrow M^b[k].send$ 
23:       $req \leftarrow \pi.SENDABSTRACT(i, j, \{m_{i \rightarrow j}\}_t)$ 
24:       $requests \xleftarrow{\text{insert}} req$ 
25:       $resp \leftarrow \mathcal{A}.GETRESPONSE()$ 
26:       $responses \xleftarrow{\text{insert}} resp$ 

27:  return ( $requests, responses$ )

```

Figure 11—Pseudocode for the challenger to simulate a scenario supplied by the adversary.

users. Denote the secret between users with id  $i$  and  $j$  as  $key_{i,j}$ . The RETRABSTRACT abstract algorithm is instantiated with the generation of the CIPR query in Figure 2. Specifically, RETRABSTRACT( $i, j$ ) calls  $q \leftarrow \text{QUERY}(\mathcal{M}_j, \mathcal{N}_i)$ , where  $\mathcal{M}_i$  is a mailbox ID provided by the adversary for user  $j$ , and  $\mathcal{N}_i$  is the total number of mailboxes advertised by the adversary to user  $i$ . In case,  $\mathcal{N}_i < \mathcal{M}_j$ , then the challenger generates CIPR query for index zero. The SENDABSTRACT abstract algorithm is instantiated using the SEND function in Figure 2. Specifically, SENDABSTRACT( $i, j, m_{i \rightarrow j}$ ) calls SEND( $\mathcal{M}_i, \mathcal{T}_i, m_{i \rightarrow j}, key_{i,j}$ ), where  $\mathcal{M}_i$  and  $\mathcal{T}_i$  are the mailbox ID and access token provided by the adversary for user  $i$ , and  $key_{i,j}$  is the shared secret established between users  $i$  and  $j$ .

**Game 1.** This game is same as game 0 except that the SENDABSTRACT invokes SEND over random messages rather than the ones specified in the scenario.

**Game 2.** This game is same as game 1 except that the RETR procedure generates query for a random mailbox ID. That is, it outputs a CIPR query for a random index (mailbox id) rather than  $\mathcal{M}_j$  returned by GETMAILBOXIDS.

Let  $S_0$  be the event that  $b = b'$  in game 0, where  $M_b$  is the scenario chosen by the challenger, and  $b'$  is the guess made

by the adversary. Similarly, let  $S_1$  be the event that  $b = b'$  in game 1, and  $S_2$  be the event that  $b = b'$  in game 2.

**Lemma A.1.**  $Pr[S_2] = 1/2$ .

Observe that in game 2 none of the requests or responses sent to the adversary depend on the information supplied in a scenario. Specifically, the SENDABSTRACT function generates encryptions of random messages, and similarly RETRABSTRACT generates a CPIR request for a random index. Therefore, an adversary participating in game 2 cannot distinguish between the two scenarios.

**Lemma A.2.**  $|Pr[S_1] - Pr[S_2]| \leq \epsilon_{CPIR}$

The difference between game 1 and game 2 is the input to the RETRABSTRACT algorithm. Specifically, in game 2, the index to CPIR query is random while it is the one supplied in the scenario in game 1. However, given the security of CPIR, an adversary cannot tell with non-negligible probability the index encoded in a CPIR query.

**Lemma A.3.**  $|Pr[S_0] - Pr[S_1]| \leq \epsilon_{Enc}$

The difference between game 0 and game 1 is the input to the SENDABSTRACT algorithm. Particularly, in game 1, the challenger inputs a random message while in game 0 it inputs the messages supplied in the scenario. Here, there are two sub-cases: the recipient is honest or the recipient is under the control of an adversary. If the recipient is honest, then given that the adversary does not have content encryption keys, it cannot distinguish between ciphertexts for the two scenarios (follows from the indistinguishability of ciphertexts). If the recipient is compromised, then it has the content encryption keys, but the recipient is the same across both scenarios and receives the same messages (see the restriction on scenario creation above). Again, the adversary cannot distinguish between the two scenarios.

Combining the three lemmas, we get the proof that  $|Pr[S_0] - 1/2| \leq \epsilon_{Enc} + \epsilon_{CPIR}$ . That is, the adversary does not win the security game with non-negligible probability.