Aloha: Metadata-private voice communication over fully untrusted infrastructure

Ishtiyaque Ahmad Yuntian Yang Divyakant Agrawal Amr El Abbadi 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 Aloha, 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, Aloha follows a template in which callers and callees deposit and retrieve messages from private mailboxes hosted at an untrusted server. However, Aloha 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 (PIR) scheme, QuickPIR, that reduces the time to process oblivious access requests for mailboxes. An evaluation of Aloha on a cluster of eighty machines on AWS demonstrates that it can serve 32K users with a 99-th percentile message latency of 726 ms—a 7× improvement over prior work 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" [19, 20, 62]. Several academic studies [25, 50, 51] have confirmed the power of metadata. As an example, Mayer et al. [50] 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 [37].

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 [11, 32, 45, 47, 66, 68] (§7). Although these systems hide metadata and keep message latency low, they either restrict scalability to only tens of users [32, 66], or are vulnerable to attacks by requiring trusted intermediaries in the communication infrastructure [11, 45, 47, 68]. An example of a trust assumption is that the system guarantees security only if an adversary can compromise at most a fraction (20%) of the servers that route user calls [45]. 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 [10, 49, 55, 63].

A system that can withstand strong adversaries while serving more than tens of users is Pung [6, 8]. 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 caller sends a voice packet every few hundred milliseconds, then the communication infrastructure must process and forward the packet within that time budget before the next packet arrives.

We present Aloha, 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, Aloha provides relationship unobservability—an adversary cannot detect whether a relationship (voice call) exists between any two users of the system [59] (§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 Mozilla LPCNet voice codec [53, 70, 71].

Aloha, 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 [17, 18, 39] (§3.2). This protocol ensures that the untrusted server does not learn which mailbox the callee is accessing, thereby unlinking the callee from the caller. However, Aloha 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, Aloha must reduce the time the server takes to process caller and callee requests, particularly, the PIR requests.

Aloha 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. Aloha inverts this architecture. In Aloha, 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 the mailbox tied to its phone number—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.

Aloha 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 [27, 35]. Second, and more saliently, Aloha constructs a new PIR scheme, QuickPIR, that fundamentally reduces the server-side PIR processing time relative to prior state-of-the-art schemes [4, 6] (§4). Even though QuickPIR was motivated by Aloha, it can be used for other applications of PIR [12, 28, 34, 52]. We plan to release QuickPIR as an open source library.

QuickPIR builds on the homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [13, 31] (§4.1) and leverages two of its features. First, it uses the single instruction, multiple data (SIMD or batching, as it is sometimes called in the literature) capability of BFV ciphertexts to compute on compressed PIR requests. Keeping requests compressed improves memory utilization, CPU caching, and eliminates the time to uncompress them (§4.2). However, working over compressed requests naively increases PIR response size. Second, QuickPIR uses homomorphic rotation operations in BFV to combine multiple pieces of a PIR response. QuickPIR reduces both the CPU time per rotation operation and the number of calls to the rotation operation (§4.3, §4.4).

For completeness, Aloha 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, Aloha uses the dialing protocol from Pung (§5).

We have implemented (§5) and evaluated (§6) a prototype of Aloha. Our prototype runs on Amazon EC2 where the server runs in US East region, and the clients (callers and callees) run geographically apart in the US West region. When the server uses 80 machines, Aloha 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. Aloha's CPU consumption, network uploads, and downloads are also lower than Pung's—by 3.45, 291, and 4.5 times, respectively.

Although Aloha 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. Furthermore, it does not currently support group voice calls. Nevertheless, Aloha demonstrates, for the first time, that even over completely untrusted infrastructure, metadata for voice calls can be hidden for tens of thousands of users.

2 Goals, threat model, and challenges

Aloha'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. Aloha targets a sub-second message latency: if Alice sends a voice packet to Bob, then Bob should receive it within a second. A component of this latency is the time the communication infrastructure takes to process the packet. The infrastructure must not queue up the packets. For instance, if Alice samples voice every 500 ms, then a hop in the infrastructure, for example, a server, cannot spend more than 500 ms to process the packet before sending it forward to Bob. Aloha must also provide sufficient throughput so that the transmitted voice is understandable. Aloha targets the LPC-Net voice codec [70, 71], which specializes in low-bandwidth voice synthesis at a rate of 1.6 Kbit/s. Finally, Aloha should scale to support a large number of users (for example, tens of thousands on a cluster of hundred machines).

Content privacy. Aloha 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. Aloha, similar to Pung [8], targets the guarantee of *relationship unobservability* as defined by Pfitzmann and Hansen [59]. 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), Aloha 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, Aloha 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, our system 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 is challenging. Indeed, prior work either relaxes the threat model or does not meet the performance goals. For instance, Yodel [45] 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, 8] 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. Besides, the server cannot spend more than the time budget to process a voice packet while simultaneously unlinking a caller from a callee. Aloha addresses these challenges and meets the performance requirements for tens of thousands of users without making any trust assumptions.

3 Architecture and overview of design

3.1 Architecture

Figure 1 shows Aloha's architecture. Aloha 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 has an ID and space to store a message; the ID is a number between (and inclusive of) 0 and $n - 1$. As we will describe later (§4), it is helpful to view the n 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 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

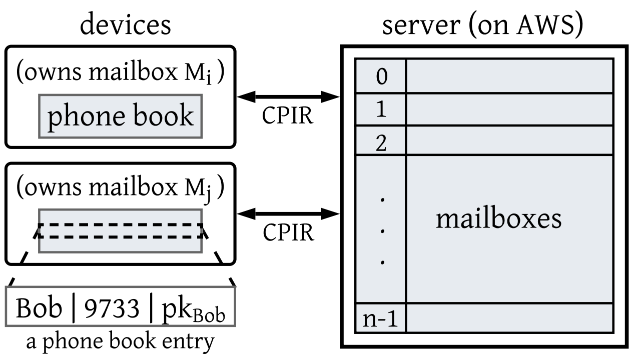


Figure 1—High-level architecture of Aloha. The server runs over untrusted infrastructure and exposes mailboxes that user devices read from or write into. The mailbox identifiers (integers 0 to $n - 1$) play the role of "phone numbers". A device stores phone numbers of the device owner's contacts in a local phone book.

contains a *phone book*, which stores information about the 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

Aloha relies on a cryptographic protocol called *private information retrieval* or *PIR* [17, 39]. We begin with a short background on PIR; §4 describes a PIR protocol in detail.

A primer on PIR. A PIR protocol [17, 39] runs between a user device and the server in Aloha, 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.

Aloha's protocol. User devices in Aloha 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 of multiple subrounds of communication. In slightly 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 up an incoming call. This is

```
 1: function RECV(key $key$, resp $resp$)
 2:     // $resp$ is the output of ANSWER PIR procedure
 3:     $c \leftarrow$ DECODE($resp$)          // DECODE is a PIR procedure
 4:     $msg \leftarrow$ AES.DEC($key, c$)
 5:     play $msg$ to user
 6: function SEND(mailbox $M$, token $t$, message $msg$, key $key$)
 7:     $c \leftarrow$ 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:     $(M_{self}, tkn, n) \leftarrow$ REGISTERDEVICE()
12:     while True do
13:         // Run dialing phase. $k_{enc}$ is for encrypting content
14:         $(M_{peer}, k_{enc}) \leftarrow$ DIAL/PICKUP()
15:         $q \leftarrow$ QUERY($M_{peer}, n$)     // QUERY is a PIR procedure
16:         send $q$ to server
17:         // Asynchronously listen for server responses
18:         register callback RECV($k_{enc}, \ldots$) 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 Aloha. n is the number of mailboxes at the server. QUERY, ANSWER, DECODE are procedures of a PIR scheme (§3.2, §4).

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 Aloha'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 Aloha, 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 that enables the server to verify that a device has written a message to its assigned mailbox. The value n changes as devices join the system; the server broadcasts this value as it changes.

Aloha'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 Aloha'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 $q \leftarrow$ 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 the corresponding devices, and finally (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. Specifically, for a query q sent by a device during the dialing phase, the server runs $resp \leftarrow$ ANSWER(mailboxes, q) and pushes the PIR response *resp* to the device. On receiving a response, a device invokes the callback it registered during the dialing phase. 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 (line 1 in Figure 2).

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., [8, 45]), 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; otherwise an adversary can learn connections between users by monitoring if they join/leave at similar times.

Security analysis. Aloha's protocol satisfies relationship unobservability, meaning that an adversary cannot detect the existence of relationships between system users (§2.1). We provide a proof in Appendix A. Briefly, Aloha's protocol meets the property for the following reasons. First, a user device only sends encrypted messages using a content encryption key known only to its peer. Besides, a device always writes an outgoing message to its own mailbox. Thus, the message sending pattern is independent of whether the device is engaged in a call or the identity of its peer. Second, the security property of the PIR protocol ensures that an adversary cannot tell the IDs of the mailboxes from which devices are retrieving messages. 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. Aloha'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, duration of a subround) to avoid packet build up. Besides, the network transfers from server to devices are dictated by the size of the output of Answer. Thus, a low cost of the Answer PIR procedure is key for Aloha's performance.

4 QuickPIR: A new CPIR scheme

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

PIR schemes are of two types: computational PIR (CPIR) [39] and information-theoretic PIR (IT-PIR) [17, 18]. 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 Aloha, we use CPIR as its deployment model is in line with Aloha's single-server design, and the 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 Aloha'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.

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 (§3.2) is to treat the library as a matrix with n rows and generate a ciphertext for every row of L. 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 as many ciphertexts as n (e.g., XPIR's query size is ≈95 MiB for $n{=}2^{15}$, and ≈3 GiB for $n{=}2^{20}$; §6.4). When the server processes larger queries, it consumes more memory and more CPU cycles to read them into CPU caches, which slows down query processing. (SealPIR compresses query on the wire, but expands it to the larger query at the server).

A popular way to address the query-size issue is a technique called *recursion* due to Stern [67]. This technique is parameterized by a depth parameter d. A value of $d = 2$ or higher, shrinks the query—it contains $d\sqrt[d]{n}$ 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 Aloha would compromise on either server CPU time or network bandwidth.

Our CPIR scheme, QuickPIR, works without recursion by default and thus keeps the smaller CPIR answer size. However, it optimizes the computation time for Answer. In fact, QuickPIR takes less time than both XPIR and SealPIR (with or without recursion) to run Answer (§6.4), and thus improves the scalability and message latency of Aloha. QuickPIR may be a good fit for other applications of CPIR where costs are dominated by those of the CPIR Answer procedure.

QuickPIR, like SealPIR [6], builds on the lattice-based homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [13, 31]. BFV has superior efficiency than a traditional number-theoretic homomorphic encryption scheme such as Paillier [57], resists attacks by quantum computers, is implemented in mature and actively maintained codebases [2, 65], 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 QuickPIR (§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; N equals a power of two and is at least 2^{10} [5]. Each component of the plaintext is an integer in $\mathbb{Z}_p = \{0,\ldots,p-1\}$, 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 a dimension of N.

A BFV ciphertext is also a vector but of dimension $2 \cdot N$. Each ciphertext 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 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 Aloha in §5.

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

- **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 $v_0 \odot v_1$. The operator $\odot$ denotes component-wise multiplication (this is an instance of single instruction, multiple data).
- **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 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 $((e, f, g, h), (a, b, c, d))$.

These BFV operations require public keys generated by a key generation procedure. Notably, the rotation procedures require a set of rotation keys. BFV.COLROTATE requires one key; however, 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. Then, 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 $0 < i < N/2$. This extreme reduces CPU time for BFV.ROWROTATE but 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 $N/2 - 1$.

4.2 The QuickPIR scheme

Recall the CPIR scenario (§3.2): a server holds a library L of n messages where each message has m components, while a client holds an integer $0 \le idx \le n - 1$ and wants to retrieve the idx-th library message without revealing idx to the server.

To build intuition for QuickPIR, suppose that L is a $N \times 1$ matrix consisting of as many messages as the plaintext vector dimension N in BFV. Then, the client constructs the CPIR query, q, for the idx-th message as follows. It one-hot encodes idx in a plaintext vector. For instance, if $N = 4$ and $idx = 1$, the client encrypts $(0, 1, 0, 0)$. The server multiplies 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), then BFV.SCMULT produces an encryption of $(0, a_1, 0, 0)$ as 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 an entire ciphertext for each row). However, a challenge is that the strategy would generates one output ciphertext for each of the m columns of L. QuickPIR addresses this challenge by combining ciphertexts for m columns into a single ciphertext using the BFV rotation operations (BFV.ROWROTATE and BFV.COLROTATE), thereby reducing CPIR answer sizes.

Figure 3 shows the QuickPIR scheme. It assumes that n is a multiple of N, i.e., $n = k \cdot N$ for some $k \ge 1$, and $m \le N$. 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 follow the intuition above. 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 (until line 17), 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 $(a_0, a_1, \ldots, a_7)$, then ANSWER computes encryptions of $(0, a1, 0, 0)$ and $(0, 0, 0, 0)$, and adds them to get an encryption of $(0, a1, 0, 0)$.

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 en-

```
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:          $q_i = \text{BFV.ENC}(pk, (f_{i \cdot N}, \ldots, f_{(i+1) \cdot N - 1}))$
8:      return $q = (q_0, \ldots, q_{(n/N)-1})$
9:  function ANSWER(library $L$, query $q = (q_0, \ldots, q_{(n/N)-1})$)
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:     if $idx \bmod N > N/2$ then
31:         $ans \leftarrow \text{BFV.COLROTATE}(ans)$
32:     $ans \leftarrow \text{BFV.ROWROTATE}(ans, N/2 - (idx \bmod N/2))$
33:     return $\text{BFV.DEC}(sk, ans)$
```

Figure 3— QUERY, ANSWER, and DECODE procedures for a basic version of QuickPIR. (pk, sk) are a (public, private) key pair for the BFV scheme (§4.1). SUBMAT extracts a sub-matrix of a matrix.

cryption of $((b_1, a_1), (d_1, c_1))$ (line 27 in Figure 3). Packing outputs from multiple columns in a single ciphertext is crucial as otherwise a CPIR answer size becomes larger.

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

A key point about the scheme is that it keeps CPIR queries compressed using BFV's vectorized operations (query does not have one ciphertext per row of library)—this reduces CPU time. The next two subsections present two further optimizations to improve the performance of this basic scheme.

4.3 Reducing the CPU cost of rotations

Recall that one goal of QuickPIR 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

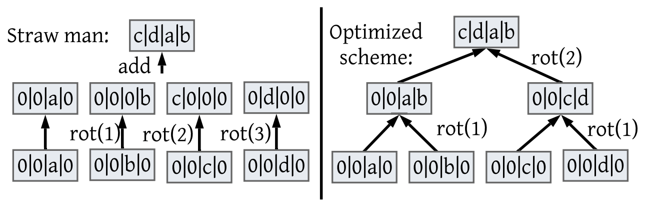


Figure 4—Illustration of optimized rotations in QuickPIR. The straw man (left) performs a mix of slow rotations (with rotations amounts that are not powers of two) and fast rotations (with rotation amounts that are powers of two) to combine multiple vectors. QuickPIR's optimized scheme combines vectors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

value of i—the positions by which the underlying plaintext is rotated. When i is a power of two, then BFV.ROWROTATE is fast. When i is a not a power of two, BFV.ROWROTATE calls itself up to $\log_2(i+1)$ times (§4.1). For example, a rotation by seven positions translates to rotations by one, two, and four positions.

QuickPIR 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 another vector by three positions. Then, the straw man design presented in the previous subsection treats each rotation separately (see the left half of Figure 4). Particularly, it breaks down the rotation by three positions into a rotation by one position followed by a rotation by two positions. Instead, QuickPIR 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 (the right half) illustrates the idea. QuickPIR 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, QuickPIR rotates the right child by 2^{h-1} positions and adds the rotated vector to the left child. The effect is that QuickPIR 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 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

```
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:          $v \leftarrow (f_{i\cdot N/2}, \ldots, f_{(i+1)\cdot N/2-1})$
6:          $q_i = \text{BFV.ENC}(pk, v||v)$     // || denotes concatenation
7:      return $q = (q_0, \ldots, q_{n/(N/2)-1})$
8:  function ANSWER(library $L$, query $q = (q_0, \ldots, q_{n/(N/2)-1})$)
9:      // Represent $L$ as a matrix of elements in $\mathbb{Z}_p$: $L \in \mathbb{Z}_p^{n\times m}$
10:     for $j = 0$ to $(m/2) - 1$ do
11:         $sum_j = \text{BFV.ENC}(pk, 0)$
12:         for $i = 0$ to $n/(N/2) - 1$ do
13:             $p_{i,j} \leftarrow \text{SUBMAT}(L, i\cdot N/2, (i+1)\cdot N/2-1, 2j, 2j+1)$
14:             $t_{i,j} = \text{BFV.SCMULT}(p_{i,j}, q_i)$
15:             $sum_j = \text{BFV.ADD}(sum_j + t_{i,j})$
16:     // Combine outputs from all pairs of columns
17:     return ROTATEANDCOMBINE$(sum_0, \ldots, sum_{m/2-1})$
18: function DECODE(answer $ans$, index $idx$)
19:     // $ans$ is an output of ANSWER
20:     $ans \leftarrow \text{BFV.ROWROTATE}(ans, N/2 - (idx \bmod N/2))$
21:     return BFV.DEC$(sk, ans)$
```

Figure 5— QUERY, ANSWER, and DECODE procedures for QuickPIR. (pk, sk) is a (public, private) key pair for the BFV scheme (§4.1). SUBMAT extracts a sub-matrix of a matrix. ROTATEANDCOMBINE is the optimized procedure to combine ciphertexts (§4.3).

the two columns of L. For the example above 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 QuickPIR 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 get one ciphertext for every pair of columns. Then, ANSWER packs these outputs using the optimized scheme to combine ciphertexts described previously (§4.3). Notably, at the implementation level, ANSWER stores at most $\lceil \log(m/2) \rceil$ ciphertexts in memory at a time by using a subtle divide and conquer strategy. The DECODE procedure performs a rotation on the output of ANSWER and decrypts the result.

Security analysis. The security of a CPIR scheme requires the output of QUERY to not reveal any information about the requested index [18, 39]. QuickPIR meets this property because its QUERY procedure (i) produces semantically-secure BFV ciphertexts, and (ii) outputs the same number of ciphertexts independent of the value of index.

5 Implementation details

QuickPIR. We implemented QuickPIR in ≈1,500 lines of C++ code. We used the Microsoft SEAL library v3.5 [65] for the underlying cryptographic operations of the BFV scheme. Recall that QuickPIR 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 $(\text{mod } 2N)$, where N is the size 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 Aloha, we choose $N = 2^{12}$, p a 19 bit prime 270337, and q a 109 bits composite number that is the product of a 54 bit prime number (18014398509309953) and a 55 bit prime number (36028797018652673). These parameters provide a 128-bit security level as guided by the homomorphic encryption standard [5]. (One may choose different parameters for QuickPIR based on application requirements.)

Master-worker architecture for Aloha. We implemented Aloha server using a master-worker architecture with many worker machines to distribute the PIR workload. Specifically, during the dialing phase of a round in Aloha's protocol (§3.2), the master receives CPIR queries from all devices and shards them across the workers (a worker gets a subset of the queries). Then, during the communication phase, master initiates a subround periodically with a pre-defined time interval. During each subround, the master receives messages from the clients and broadcasts the entire message library to the workers. Each worker uses QuickPIR to compute ANSWER for each CPIR query in its assigned subset of queries and pushes the CPIR responses to the corresponding client devices.

Dialing protocol. Aloha uses Pung's protocol to initiate connections [9, Chapter 4.5.3] (which in turn is based on Alpenhorn [46]). 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 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, Aloha runs this protocol infrequently (every five minutes; §6.3). A more efficient dialing protocol in Aloha'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, Aloha exposes all options to the device owner and lets them pick the call they want to be a part of. However, depending on which option the user chooses, they could leak some information to the users involved 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 Aloha but applies to any metadata-private system [7]. As efficient solutions to this problem become available, one could enhance the options-based approach currently implemented in Aloha.

Other libraries and lines of code. Our prototype of Aloha is ≈2,000 lines of C++ on top of existing libraries (including QuickPIR). Our implementation of dialing protocol uses the libscapi [1] library for public-key encryption using the Cramer-Shoup scheme [24] with a key size of 3072 bits. We use AES-CBC implementation from OpenSSL with a 128-bit key for end-to-end content encryption. We implement the message library broadcasting mechanism from master to workers using rpclib [3]. We use the open source implementation of LPCNet [53] for voice encoding/decoding.

6 Evaluation

Our evaluation answers the following questions:

1. What is Aloha'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 Aloha impose on the server and users?
3. How does Aloha compare to Pung [6, 8, 9], which is the state-of-the-art prior system for metadata-private communication over completely untrusted infrastructure?
4. How does QuickPIR compare to the state-of-the-art CPIR implementations, XPIR [4] and SealPIR [6]?

A highlight of our evaluation results is as follows:

- Aloha'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.
- Aloha's server consumes 22.3 minutes of CPU time for a subround for 32,768 users. A subround corresponds to 480 ms of voice call. Pung consumes 3.45× more CPU.
- An Aloha user downloads and uploads 55.1 and 1.08 MiB of data for each round when 32,768 users use Aloha. A round corresponds to five minutes of voice call. A Pung client downloads and uploads 250 MiB and 313 MiB for five minutes of voice call data.
- QuickPIR has a small server-side CPU time *and* a small response size, while XPIR and SealPIR must sacrifice on one of the two metrics.

Setup and method. We compare Aloha to two variants of Pung: Pung-XPIR (P-XPIR) and Pung-SealPIR (P-SPIR). The former is the original Pung system from OSDI '16 [8] that uses the XPIR scheme for CPIR [4]. The second variant replaces the XPIR scheme with the SealPIR CPIR scheme [6]. We include both variants as there is no clear winner between them across all performance metrics. Further, we evaluate these variants with ($d = 2$) and without recursion ($d = 1$) for the CPIR scheme (recursion is described in §4).

We configure Aloha 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 obliviously searches through a tree while retrieving one message from the Pung server. This scheme is the most scalable retrieval scheme for Pung especially as number of system users increases; 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). Since the server for both Aloha and Pung is CPU-bound, we use the compute-optimized worker machines of type `c5.12xlarge` (48 vCPU, 96 GB of RAM, and 12 Gbps of network bandwidth). For Aloha, we use a bigger machine of type `c5.24xlarge` (96 vCPU, 192 GB of RAM and 25 Gbps of network bandwidth) for its master, which broadcasts the message library (the mailboxes) to the workers and needs additional network bandwidth (§5). To compensate for this additional resource for Aloha, we assign two more worker machines to the baselines.

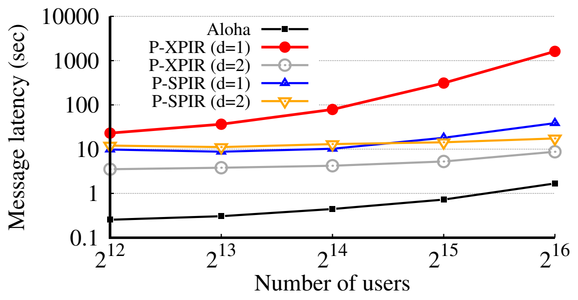
Aloha is required to process queries from all clients in every subround to meet its security goals. Since we cannot run tens of thousands of clients in our infrastructure, we employ a combination of real and simulated clients. We deploy 256 geographically distant real clients in a machine of type `c5.24xlarge` 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 Aloha 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) [70, 71]. 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.

Aloha's message latency is 254 ms for 4,096 users and increases to 1678 ms for 65,536 users. The increase is due to three reasons. First, as the number of users increases, so does the message library (mailbox) size and the time to broadcast it from the master to the workers (§5). Second, the number of CPIR queries the server processes equals the number of users (number of library rows) (§3.2). Third, the time to process



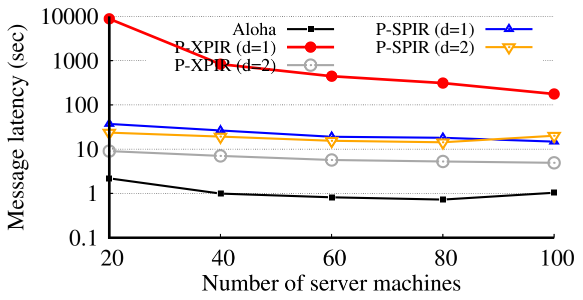


Figure 6—(Left) Message latency with varying number of users for eighty server worker machines. (Right) Message latency with varying number of server worker machines for 32,768 users. Messages are 96 bytes in size. The y-axis is log-scaled. d denotes CPIR recursion depth (described in §4); $d = 1$ denotes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

a CPIR query increases with the number of users, 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 and the rest is for the network; this latency is under one second, which is our target (§2.1). However, for 65,536 users, the latency is 1,678 ms, of which 1,186 ms is for CPIR query processing; the latter is higher than the 480 ms subround time budget and thus voice packets start queuing.

Aloha's message latency is lower than Pung's, specifically, that of Pung-XPIR—by 7.2× for 32,768 users—due to two reasons. First, a sender in Aloha pushes a message to the server, who performs CPIR processing and pushes the 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 (n is number of users). Second, Aloha uses QuickPIR, 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, §6.4).

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, up to an inflection point, with an increase in the number of worker machines due to increased parallelization for CPIR answer generation. However, beyond the inflection point, adding workers does not improve latency as the time to replicate mailboxes (message library) from the master to the workers goes up, while the CPU on worker machines start to become idle. Distributing the master or extracting more efficiency from each worker while keeping the number of workers same may further push out the inflection point (§8).

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). Aloha's CPU consumption is lower than Pung's. For instance, for 32,768 users, Aloha takes 22.3 minutes while Pung (with XPIR and CPIR recursion depth $d = 2$) takes 77.1 minutes (3.45× higher). Even though both the Aloha and Pung server do the same amount work asymptotically (privately serve one message per message recipient from a message library of same size), QuickPIR in Aloha reduces CPU time relative to XPIR or SealPIR in Pung, due to the former's more efficient use of memory and underlying homomorphic operations (§4).

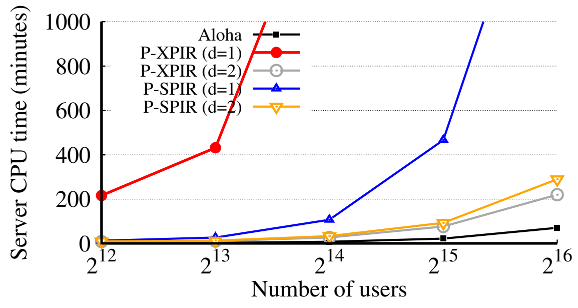


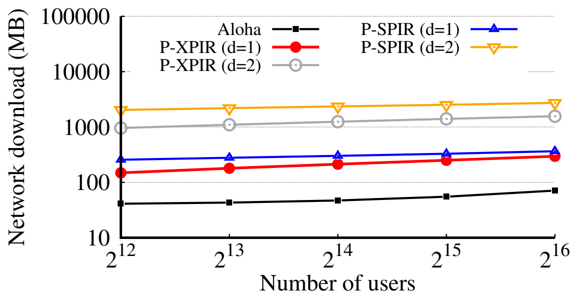
Figure 7— Server-side CPU time per subround with varying number of users. A subround corresponds to 480 ms of voice call; in a subround, each user sends and receives one 96 byte message.

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 Aloha user downloads ≈55.1 MiB in a round when 32,768 users use Aloha. Of this, ≈39 MiB is in the communication phase of the round and independent of the number of system users (§3.2), while the rest is for the dialing phase. In contrast, a Pung user downloads more data, by 4.5–45.7×, depending on the Pung variant. The increase is due to two reasons. First, Pung, unlike Aloha, makes multiple CPIR queries between a message recipient and 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 $d = 1$). Aloha's QuickPIR operates at $d = 1$ to keep CPIR answer sizes and thus the downloads smaller (§4).

An Aloha user uploads one CPIR query per round during its dialing phase. The Aloha server then reuses the query across subrounds (§3.2). Even though the query size for Ad-



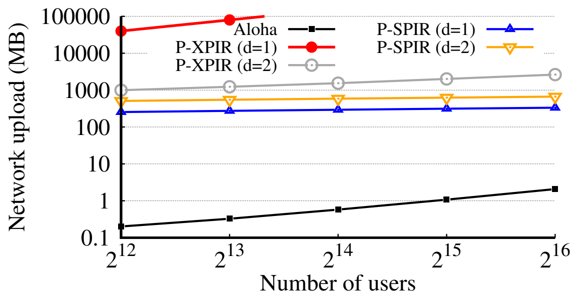


Figure 8— Data downloaded and uploaded by a user per round with varying number of users. A round corresponds to five minutes of voice call.

	$n = 32{,}768$					$n = 1{,}048{,}576$				
	X ($d=1$)	X ($d=2$)	S ($d=1$)	S ($d=2$)	Q ($d=1$)	X ($d=1$)	X ($d=2$)	S ($d=1$)	S ($d=2$)	Q ($d=1$)
client CPU costs (ms)										
QUERY	335.2	12.4	2.0	1.4	21.4	10711.3	69.8	56.9	1.4	678.6
DECODE	0.1	0.38	0.2	1.89	0.38	0.09	0.42	0.2	1.94	0.41
server CPU costs (ms)										
SETUP	89.6	88.6	370.6	232.9	68.3	2901.5	2845.5	11829.9	7404.4	2207.5
ANSWER	71.6	91.6	2433.7	161.3	52.4	2296.5	2347.2	76441.8	2554.3	942.1
network costs (KiB)										
query size	95,328	3,520	96	64	1,024	3,050,432	19,776	2,752	64	32,768
answer size	32	256	32	320	64	32	288	32	320	64

Figure 9—CPU times and network costs for XPIR (X), SealPIR (S), and QuickPIR (Q) with varying number of items (n) in the server library. Each item is 256 bytes. d denotes recursion depth (§4). QuickPIR does not use recursion (sets $d = 1$, which means no recursion).

dra is larger compared to that of Pung in some cases (§6.4), unlike Pung, this cost is amortized over multiple subrounds of communication (§3.2). As a result, Aloha's upload network transfers are small: ≈1.1 MiB per round.

CPU time. An Aloha client consumes ≈27.5 seconds of CPU time per 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 1.7–63× higher CPU.

6.4 Comparison of CPIR schemes

A core component of Aloha and the baseline systems is the CPIR cryptographic primitive. Pung uses either XPIR or SealPIR, which are also the state-of-the-art schemes. Aloha uses QuickPIR (§4). This section compares the cost of the CPIR component of these systems in isolation. Moreover, since CPIR schemes have a number of other applications [12, 28, 34, 52], this section sheds light on which scheme could be better for which application.

We microbenchmarked the XPIR, SealPIR, and QuickPIR libraries on a single CPU of an AWS instances of type `c5.12xlarge` (48 vCPU, 3.6 GHz, 96 GB RAM). We configured all three libraries for a 128-bit security level. However, XPIR does not set parameters from the homomorphic encryption standard [5]; these parameters are smaller relative to those for SealPIR and QuickPIR. We set each message to be 256 bytes and run experiments for two different message library sizes $n = \{2^{15}, 2^{20}\}$. We select 2^{15} since that is the most number of users we currently support in Aloha. Our experiments for 2^{20} messages is to demonstrate how QuickPIR scales with the number of messages compared to other CPIR libraries. We microbenchmark the query generation (QUERY), answer generation (ANSWER), and answer decode (DECODE) procedures for 10 trials. Figure 9 tabulates the means.

Given that the CPIR cost in Aloha is dominated by the cost to run the ANSWER procedure, we describe the results while focusing on ANSWER. At a high level, QuickPIR keeps both the CPU cost for ANSWER and the size of ANSWER output small, while XPIR and SealPIR have to sacrifice one of the two.

CPU time for ANSWER. QuickPIR consumes the least amount of CPU time for ANSWER (and also the least amount of time for a one time SETUP procedure to prepare library 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}, ANSWER in QuickPIR takes 2.5× less time than XPIR ($d = 2$) and 2.7× less time than SealPIR ($d = 2$).

Output size of ANSWER. When the schemes do not use recursion ($d = 1$), then the answer output sizes are smaller, although QuickPIR's is double the output 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 also increases CPU time for processing of concurrent queries; Figure 7). For SealPIR, the compressed query is smaller on the wire, but the expanded query is same size as in XPIR. Furthermore, the cost to expand adds significant

CPU time for SealPIR $d = 1$. XPIR and SealPIR with $d = 2$ do not have the query-size drawback, but they increase answer output size, by 8 to 10 times, relative to $d = 1$. Overall, QuickPIR produces smaller responses (answer outputs) without large query sizes or significant addition to computation time.

Query-related overheads. Query generation time and query sizes are significantly larger in QuickPIR 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 QuickPIR (with $d = 1$). However, QuickPIR's query sizes are within a factor of two of XPIR's even when XPIR compresses query using recursion.

Summary. If ANSWER is invoked frequently, and both the server-side CPU time and CPIR response size need to be smaller, then QuickPIR is a better fit. However, if the application cannot be designed such that the costs are dominated by those of ANSWER, then SealPIR and XPIR may be a better fit.

7 Related work

Onion-routing. Systems such as Tor [68], based on onion-routing [33, 61], can support anonymous VoIP calls with low message latency. However, they do not provide strong guarantees. Indeed, even a network adversary, such as an ISP, can learn call metadata via traffic analysis [14, 36, 40, 54, 58].

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 [16]. A mix-net based system fundamentally requires at least one mix to be trusted [41–45, 47, 48, 60, 69, 72]. Yodel [45] 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 number of mixes and 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 [15]. Due to the broadcasting requirement, earlier systems based on DC-nets scaled to only tens of participants [22, 32, 66]. Later systems [23, 73] improved scalability but at the cost of relaxing the threat model. For instance, Dissent in numbers [73] 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 [11] is the latest DC-net based system. It targets a LAN setting of a small organization with a few hundred users and reduces latency (to 100 ms for 100 users). PriFi does not scale to thousands or tens of thousands of users, and also assumes that one of its servers is trusted.

Private mailboxes. Systems based on private mailboxes either obliviously write to [21, 30] or read from [6, 8, 12, 38, 64] mailboxes hosted over untrusted servers. The state-of-the-art system based on this strategy that works over completely untrusted infrastructure is Pung [6, 8] (rest of the systems assume non-colluding servers).

We empirically compared Aloha 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 (especially for low-bandwidth voice calls). For instance, for 32K users, the server will have to push 625 MiB of mapping data every five minutes to every user. The Bloom filter scheme lowers the network overhead but works probabilistically: a message recipient is not guaranteed to download the message sent by the sender, thus affecting quality of service.

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

Private information retrieval (PIR). Chor et al. [17, 18] introduced the problem of PIR over multiple non-colluding servers, while Kushilevitz and Ostrovsky [39] 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, 26, 27, 67], while others improve answer recovery against a byzantine server [29, 56]. In this paper, we introduced QuickPIR, a new CPIR scheme that reduces the server-side computation time relative to the state-of-the-art CPIR schemes [4, 6] (§6.4).

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 Aloha, the first system that hides metadata for voice calls over completely untrusted infrastructure for tens of thousands of users. Aloha'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. Aloha 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, QuickPIR (§4).

Our future work involves further scaling Aloha from tens of thousands of users to hundreds of thousands or a few million users. To accelerate CPIR computation, we plan to explore more efficient implementations of the master-worker Aloha server architecture as well as push more work to the workers using GPUs and FPGAs. For the latter, one would have to address challenges related to running PIR on a heterogeneous system. Finally, we would like to expand Aloha's functionality from peer-to-peer voice calls to support group calls.

A Security proof

In this appendix, we show that Aloha's protocol (§3.2) meets the relationship unobservability property (§2.1). Our proof follows the proof technique of Pung [9, 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 Aloha protocol.

A.1 An abstract protocol

We define an abstract protocol for metadata private voice communication using the following three algorithms: $\textsc{Init}(1^\lambda)$, $\textsc{RetrAbstract}(i,j)$, and $\textsc{SendAbstract}(i,j,m_{i\to j})$.

$\textsc{Init}(1^\lambda)$ 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.

$\textsc{RetrAbstract}(i,j)$ 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.

$\textsc{SendAbstract}(i,j,m_{i\to j})$ 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 $\textsc{Init}$ algorithm. Then, each participant calls the $\textsc{RetrAbstract}$ algorithm to generate a request to indicate that it wants to get messages from its peer. Finally, each participant calls the $\textsc{SendAbstract}$ 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 $\mathcal{G}^b_{\mathcal{A},\pi,K,t}(1^\lambda)$, where $\mathcal{A}$ is the adversary, π is a round-based protocol consisting of the $\textsc{Init}$, $\textsc{RetrAbstract}$, and $\textsc{SendAbstract}$ 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 π), and λ 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 π for one of the scenarios selected randomly (that is, the b-th scenario) and sends the transcript of the protocol messages due to π 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, $M^b[k].send = (i, j^b, \{m^b_{i\to j}\}_t)$ specifies that the user device with id i should send the set of t messages $\{m^b_{i\to j}\}_t$ 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 ℓ^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 ℓ^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 $\textsc{Simulate}$ function described in Figure 10. 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 $\textsc{GetMailboxIDs}$ to learn the mailbox IDs the adversary assigns to the users. Similarly, the challenger calls $\textsc{GetMailboxAccessTokens}$ $\textsc{GetNumMailboxes}$ to learn mailbox access tokens and the total number of mailboxes. Finally, the challenger calls the $\textsc{GetResponse}$ 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, $\textsc{GetResponse}$ allows adversary to drop requests or reorder them or compute

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

    // Obtain mailbox IDs, access tokens, number of mailboxes
    // $\mathcal{M}$ contains the assigned mailbox IDs
    $\mathcal{M} \leftarrow \mathcal{A}$.GETMAILBOXIDS()
    // $\mathcal{T}$ contains the assigned tokens
    $\mathcal{T} \leftarrow \mathcal{A}$.GETMAILBOXACCESSTOKENS()
    // $\mathcal{N}$ contains the number of advertised mailboxes
    $\mathcal{N} \leftarrow \mathcal{A}$.GETNUMMAILBOXES()

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

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

    return $(requests, responses)$
```

Figure 10—Pseudocode for the challenger to simulate a scenario supplied by the adversary.

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 λ when the abstract protocol is instantiated with the *Aloha* 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 π instantiated with the Aloha protocol. That is, the INIT algorithm establishes a shared secret between pairs of 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 CPIR query in Figure 2. Specifically, RETRABSTRACT(i,j) calls $q \leftarrow$ 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 CPIR query for index zero. The SENDABSTRACT abstract algorithm is instantiated using the SEND function in Figure 2. Specifically, SENDABSTRACT$(i, j, m_{i \to j})$ calls SEND$(\mathcal{M}_i, \mathcal{T}_i, m_{i \to j}, key_{i,j})$, 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 CPIR 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.

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)*, pages 962–979, 2018.

[7] S. Angel, D. Lazar, and I. Tzialla. What's a little leakage between friends? In *Workshop on Privacy in the Electronic Society (WPES)*, pages 104–108, 2018.

[8] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 551–569, 2016.

[9] S. G. Angel. *Unobservable communication over untrusted infrastructure*. PhD thesis, The University of Texas at Austin, 2018.

[10] 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.

[11] 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.

[12] N. Borisov, G. Danezis, and I. Goldberg. DP5: A private presence service. *Privacy Enhancing Technologies Symposium (PETS)*, 2015(2):4–24, 2015.

[13] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology—CRYPTO*, pages 868–886, 2012.

[14] 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)*, pages 605–616, 2012.

[15] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[16] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[17] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1995.

[18] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[19] D. Cole. We kill people based on metadata. *The New York Review*, May 2014.

[20] D. Cole. Is privacy obsolete? *The Nation*, Mar. 2015.

[21] 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)*, pages 321–338. IEEE, 2015.

[22] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *ACM Conference on Computer and Communications Security (CCS)*, pages 340–350, 2010.

[23] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in verdict. In *USENIX Security Symposium*, pages 147–162, 2013.

[24] 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.

[25] 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*, 3:1376, 2013.

[26] D. Demmler, A. Herzberg, and T. Schneider. RAID-PIR: Practical multi-server PIR. In *ACM Workshop on Cloud Computing Security*, pages 45–56, 2014.

[27] C. Devet. Evaluating private information retrieval on the cloud. Technical report, University of Waterloo, 2013.

[28] 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. Springer, 2014.

[29] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *USENIX Security Symposium*, pages 269–283, 2012.

[30] 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.

[31] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[32] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[33] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[34] M. Green, W. Ladd, and I. Miers. A protocol for privately reporting ad impressions at scale. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1591–1601, 2016.

[35] 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)*, pages 91–107, 2016.

[36] 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.

[37] S. Humphreys and M. De Zwart. Data retention, journalist freedoms and whistleblowers. *Media International Australia*, 165(1):103–116, 2017.

[38] 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)*, pages 16–30, 2004.

[39] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1997.

[40] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *USENIX Security Symposium*, pages 287–302, 2015.

[41] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 406–422, 2017.

[42] 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.

[43] A. Kwon, D. Lu, and S. Devadas. XRD: Scalable messaging system with cryptographic privacy. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 759–776, 2020.

[44] 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)*, pages 711–725, 2018.

[45] D. Lazar, Y. Gilad, and N. Zeldovich. Yodel: Strong metadata security for voice calls. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 211–224, 2019.

[46] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 571–586, 2016.

[47] 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*, pages 639–652, 2015.

[48] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review*, 43(4):303–314, 2013.

[49] R. Lenzner. ATT, Verizon, Sprint are paid cash by NSA for your private communications. *Forbes*, 2013.

[50] 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.

[51] 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*, pages 251–260, 2010.

[52] 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*, page 31, 2011.

[53] Mozilla. LPCNet: Efficient neural speech synthesis. `https://github.com/mozilla/LPCNet`.

[54] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy (S&P)*, pages 183–195, 2005.

[55] 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`, 2020.

[56] F. Olumofin and I. Goldberg. Revisiting the computational practicality of private information retrieval. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 158–172, 2011.

[57] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999.

[58] 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)*, pages 103–114, 2011.

[59] 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`, 2010.

[60] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The Loopix anonymity system. In *USENIX Security Symposium*, pages 1199–1216, 2017.

[61] 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.

[62] A. Rusbridger. The Snowden leaks and the public. *The New York Review*, Nov. 2013.

[63] D. Rushe. Yahoo $250,000 daily fine over NSA data refusal was set to double “every week”. *The Guardian*, 2014.

[64] 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)*, pages 1–9, 2005.

[65] Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, Apr. 2020. Microsoft Research, Redmond, WA.

[66] 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.

[67] 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)*, pages 357–371, 1998.

[68] P. Syverson, R. Dingledine, and N. Mathewson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[69] 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.

[70] 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)*, pages 5891–5895, 2019.

[71] 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.

[72] 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)*, pages 137–152, 2015.

[73] 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)*, pages 179–182, 2012.