Periscoping: Private Key Distribution for Large-Scale Mixnets

Shuhao Liu
Shenzhen Institute of Computing Sciences
shuhao@sics.ac.cn

Li Chen
University of Louisiana at Lafayette
li.chen@louisiana.edu

Yuanzhong Fu
Unaffiliated
allenfu2006@gmail.com

***Abstract*—Mix networks, or mixnets, are one of the fundamental building blocks of anonymity systems. To defend against epistemic attacks, existing free-route mixnet designs require all clients to maintain a consistent, up-to-date view of the entire key directory. This, however, inevitably raises the performance concern under system scale-out: in a larger mixnet, a client will consume more bandwidth for updating keys in the background.**

This paper presents *Periscoping*, a key distribution protocol for mixnets at scale. Periscoping relaxes the download-all requirement for clients. Instead, it allows a client to selectively download a constant number of entries of the key directory, while guaranteeing the privacy of selections. Periscoping achieves this goal via a novel Private Information Retrieval scheme, constructed based on constrained Pseudorandom Functions. Moreover, the protocol is integrated seamlessly into the mixnet operations, readily applicable to existing mixnet systems as an extension at a minimal cost. Our experiments show that, with millions of mixes, it can reduce the traffic load of a mixnet by orders of magnitude, at a minor computational and bandwidth overhead.

I. Introduction

Mixnets are one of the key cryptographic primitives for anonymous communications. Since its introduction [1], it has become an efficient technique for hiding the communication patterns among users. By routing every user message through a chain of relay servers (*a.k.a. mixes*), it provides strong unlinkability [2] between pairs of senders and receivers [3].

Most large-scale anonymity systems [4]–[10] and privacy-preserving data mining systems [11]–[13] are based on *free-route* mixnets [14], a scalable and practical variant. To send message m, the sender client routes it through a mixnet path, which consists of a *subset* of available mixes that are chosen independently randomly. Message m is onion-encrypted using the public keys of selected mixes, in the reverse order along the path. These public keys, together with other auxiliary information of available mixes (*e.g.,* IP addresses, the authentication tags and the expiration times of key pairs), are kept in a key-value database, namely the *mixnet key directory*.

For each onion encryption, the sender only needs the latest directory entries for the *chosen mixes*. Apparently, an on-demand download strategy for the key directory consumes minimal bandwidth; yet, this turns out to be insecure, being susceptible to epistemic attacks [15]. Provided that Alice has a partial view over the mixnet—say, having the directory entries of a subset $\mathcal{M}_A$ of all mixes $\mathcal{M}$—she will never choose the mix in $\mathcal{M} \setminus \mathcal{M}_A$ to relay a message. Eve, an adversary who may learn this information by corrupting mixes, can potentially exploit such observations and learn a skewed distribution over Alice's path selections. Over time, the unlinkability between Alice and her intended receivers can be compromised entirely.

To defend against epistemic attacks, each and every aforementioned mixnet system takes the conservative approach: users are required to maintain a globally consistent and up-to-date copy of the *entire* mixnet key directory. Provided that the availability of mixes changes constantly and key pairs are refreshed regularly for forward secrecy [16], updating the key directory at all clients can take a heavy toll at scale. Intuitively, distributing updates of a key directory with N mixes would generate $O(N)$ download traffic on each client, resulting in $O(N^2)$ aggregate bandwidth consumption.

Above all, it is vital to deliver key directory updates to clients timely and frequently; yet, we lack such a protocol that is *linearly scalable, secure, and efficient* at the same time.

A very similar and closely related problem has been identified in Tor [17], a widely-deployed low-latency anonymity system. Tor suffers from the same scalability issue as a free-route mixnet does, with minor differences in system configurations. Per estimation [18], [19], if Tor's user base were to grow by two orders of magnitude, network state synchronization would contribute over *half* of its network traffic.

One may attempt to adapt a scalable key distribution protocol of Tor to mixnets. Unfortunately, none previous work can fit properly. More specifically, hop-by-hop solutions (*e.g.,* WalkingOnion [19]) can reliably outsource path selection to selected relays. However, it depends heavily on the circuit-building phase of Tor, which is non-existent and expensive to introduce in mixnets. Source-routed solutions (*e.g.,* PIR-Tor [20] and ConsenSGX [21]), which allow clients to download a subset of directory entries obliviously, are applicable yet not practical. They involve either computation-heavy or network-intensive constructions. Unlike a Tor client reuses a path for a session, a mixnet client does this on a *per-message* basis, making existing protocols prohibitively expensive.

Is there an efficient method for accessing the key directory obliviously? Can we outsource some work to mixes to reduce the protocol complexity? And eventually, is it possible to design a linearly scalable mixnet, in which every client can download no more than constant-size updates?

Contributions. In this paper, we answer these questions—all in the affirmative—by proposing *Periscoping*, a novel protocol for private key distribution in a large-scale mixnet. It relieves clients from maintaining the entire copy of the mixnet key directory. Instead, a client can periodically download a specified subset of entries, free from the risk of leaking information about which entries are accessed. To the best of our knowledge, it is the most efficient protocol yet guarantees the scalability and the security of a mixnet system. In particular, we make the following concrete contributions.

(1) A private information retrieval (PIR) scheme. We propose PRFSetPIR, a novel PIR scheme based on constrained Pseudorandom Functions. As compared to prior PIR constructions, it is optimized for bandwidth usage in mixnet operations, with new query compression techniques to reduce request and response sizes; moreover, it incurs a negligible computationally overhead without using expensive cryptographic primitives.

(2) Security guarantees. We conduct comprehensive security analysis of PRFSetPIR under the standard thread model of mixnet systems. A client can download any subset of the key directory with PRFSetPIR, while ensuring that all other parties are oblivious to the entries having been accessed. In the case where a weaker ϵ-differential privacy is sufficient, it can reduce computational cost further with a minor parameter tweak.

(3) Protocol Periscoping. As a key distribution protocol, Periscoping integrates PRFSetPIR into the original mixnet message exchanges through piggybacking, incurring no additional communication session. We accommodate such an integration by allowing the reuse of mixnet message fields whenever possible, which minimizes the bandwidth overhead. Collectively, it realizes the design objectives of linear scalability, security, and efficiency, and can readily and practically work as a superior add-on to existing mixnets.

(4) Experimental study. Our prototype experiments demonstrate that in a mixnet consisting of millions of mixes, Periscoping reduces the traffic load for distributing mixnet keys at a negligible computational overhead. As compared to the state-of-the-arts, the bandwidth usage is reduced by up-to three orders of magnitude for a client.

Organization. This paper introduces Periscoping using a top-down approach. Sec. II reviews some preliminary concepts related to this work. Following a few strawman approaches, Sec. III provides an overview of Periscoping, with detailed constructions of PRFSetPIR and security analysis in Sec. IV. We demonstrate prototype evaluation results in Sec. V.

II. Preliminaries and Background

A. Mixnet

As an effective means to anonymize users online, a mixnet enables unlinkability [2] between users and their messages. It leverages the idea of onion encryption and shuffling, and distributes trust among servers.

Basic workings. Fig. 1 shows an illustrative example of Alice sending an encrypted message m to Bob anonymously through a 3-server mixnet, based on a slightly modified version of Chaum's original proposal [1]. As a *priori*, Alice learns a

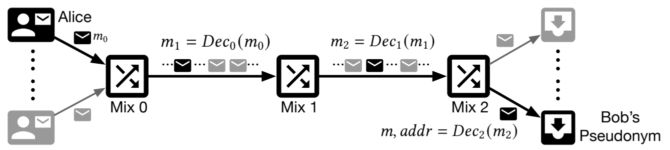

Fig. 1: Example: Alice sends a message to Bob via a mixnet.

pseudonym *addr* of Bob, a randomly assigned mailbox for Bob. Her message delivery is as follows.

1. Alice onion-encrypts $m, addr$ (via concatenation), using the public keys of Mix2, Mix1, and Mix0, and outputs $m_0 = Enc_0[Enc_1[Enc_2(m, addr), \mathsf{Mix}_2], \mathsf{Mix}_1]$.
2. Alice sends m_0 to Mix0, the first mix along the path.
3. Mix0 "peels off" the outer-most layer of encryption, by decrypting m_0 using its private key, and gets m_1.
4. Mix0 buffers a batch of decrypted messages from users, shuffles them randomly, and sends the batch to Mix1.
5. Mix1 repeats Step 3.–4., sending the batch to Mix2.
6. Mix2 repeats Step 3., recovers m and $addr$, and finally deposits m to Bob's pseudonym.

Consider a passive adversary $\mathcal{A}$ having access to all the network traffic and the internal states of corrupted mixes. The mixnet guarantees that, if at least one mix along the path is honest, with enough concurrent senders forming an anonymity set, $\mathcal{A}$ cannot decide whether Alice deposits to *addr* or not.

Free-route mixnet. In modern anonymity systems [4]–[10], mixnets are used to hide users' communication metadata (*e.g.,* the when, the where, and with whom). Another emerging use case is privacy-preserving data mining systems [11]–[13], where mixnets can unlink users from their published data. Both settings involve a massive user base, with millions of available mixes or more [13]. These systems employ the *free-route mixnet* [14], a practical and scalable variant.

Different from the basic mixnet that requires every message to go through all mixes in a pre-determined order, a free-route mixnet allows the sender of message m to draw a random subset of l mixes (*e.g.,* $l \leq 15$ [6]), and organize them as a chain in a random order. This chain will be used as the *path* for delivering m. Sending another message m' requires another draw; that is, each selected path is used in a one-off fashion. Depending on the assumed threat models, a free-route mixnet can either guarantee the perfect privacy of communication metadata [13], or make it differentially-private [5], [6].

Mixnet key directory. Onion encryption assumes the availability of the public keys of all mixes. Existing free-route mixnets maintain these keys in a version-controlled database, referred to as the *mixnet key directory*. Tor [17] has a similar entity known as ENDIVE [19], which makes a concrete reference.

More specifically, every mix has a separate entry in the key directory, indexed by its unique identifier. The attributes include its public keys, the IP addresses, two timestamps indicating its creation/expiration times, and flags of supported features or a version number. Moreover, the entry carries its own authentication tag, generated by trusted authorities. All fields are necessary for the security and the correctness of the mixnet.

Consider a mixnet with 7,000 mixes and 3 million users, at the scale of Tor [17] as of July, 2023 [18]. The key directory can be well over 2.5 MB [19]. If we require every user to maintain a copy, the average download traffic on a single mix for a round of key pair refresh is about 1 TB.

If the mixnet were to grow 10×, the overall bandwidth usage would be 100×, limiting its practicality. Worse yet, recent mixnet systems can be of an even larger scale. For example, Mycelium [13] employs millions of user devices as mixes, whose scale is three orders of magnitudes higher than that of Tor. It distributes keys to clients using the Telescoping technique [17], which is clearly one of the factors behind its high end-to-end message delivery latencies (in hours).

Key distribution in Tor. Tor has two major distinctions from a mixnet: (a) it reuses a chain of relays (called a circuit) for the entire session of a conversation, and (b) a circuit has much fewer relays (typically 3). Before sending a message, Tor has a dedicated circuit-building phase, which involves a multi-round interactive protocol to exchange keys among users and relays, *i.e.,* Telescoping [17]. Such a protocol would be unaffordable in a mixnet that has a much longer path (typically ≤ 10), used in the one-off manner. As a result, although Tor faces a similar scalability problem, the solutions are not directly applicable.

For example, the state-of-the-art WalkingOnion [19] protocol optimizes the Telescoping technique, by reliably outsourcing next-hop selection to relays. Indeed, it is efficient for circuit-building, but too expensive to apply in a mixnet. PIR-Tor [20] and ConsenSGX [21] employ computationally private information retrieval [22]—an expensive cryptographic method—to obliviously access the mixnet key directory. Despite inefficiency for mixnets, they inspire our design of Periscoping, to be discussed in Sec. III-B.

B. Private Information Retrieval

Private Information Retrieval (PIR) [23] is a technique that allows a user to access a database *obliviously*, *i.e.,* other parties, including the database owner, cannot learn *which* database entries have been accessed. More formally, consider Bob who holds a database DB with n entries. Alice wishes to download $DB[i]$, the i-th entry, while keeping the value of i private. A *trivial* solution is to download all DB entries, which is apparently too costly. Non-trivial PIR schemes incur download traffic sublinear in n, briefly reviewed as follows.

Single-server PIR. Chor *et al.* [22] show that the construction of a PIR scheme must be based on some computational hardness assumptions, if DB is hosted on a single server. The state-of-the-art practical single-server PIR schemes [24]–[26] are based on some forms of fully homomorphic encryption, such as BGV [27], BFV [28], [29], and RGSW [30], [31]. Theoretically, these encryption schemes permit arithmetic computations (*e.g.,* addition and multiplication) directly on the ciphertext; the results, after being decrypted, match the outputs had the computation been carried out on the plaintext. The drawback, however, is the computational complexity and the ciphertext size, which is several times of its plaintext size.

For this reason, existing single-server PIR schemes suffer from long response times and large request sizes, with all computation on the server side performed homomorphically on ciphertexts. Consider downloading a single entry from DB sized $n = 2^{18}$ via SpiralPIR [32], the most efficient construction to date. A user needs to send a 14KB request and it takes 24.46 seconds to respond on an 8-core server.

Multi-server PIR. PIR can be much more efficient if we assume the availability of multiple non-colluding servers, each holding a copy of DB. Also, the security guarantee can be information-theoretic: an adversary cannot recover i even with unlimited computing power. We next review a simple PIR construction [23] denoted as BasicPIR. Consider an n-entry DB replicated among l servers, $s_0, s_1, \dots, s_{l-1}$. Let $[k]$ denote the set $\{0, 1, \dots, k-1\}$ for natural number k. To access $DB[i]$, BasicPIR proceeds as follows:

1. Alice builds l requests $q_0, q_1, \dots, q_{l-1}$ initialized n-bit all-0 strings, forming an l-row n-column request matrix.
2. Alice sets bits column-wise: every bit is set to 1 with probability θ. An additional requirement is enforced such that the total number of 1s in a column must be even, except that the i-th column has an odd number of 1s. Intuitively, q_j ($j \in [l]$), the j-th row of the request matrix, is the bitset representation of a random subset of $[n]$, in which each element is included with probability θ.
3. Alice sends request q_j to server s_j, for all j in $[l]$.
4. Upon receiving q_j, s_j generates response r_j, by XORing database entries corresponding to 1s in q_j, *i.e.,* $r_j = \oplus_{t \in \{m | q_j[m]=1\}} DB[t]$, where $\oplus$ denotes the bitwise exclusive OR operation. s_j replies Alice with r_j.
5. Alice recovers $DB[i] = \oplus_{t \in [l]} r_t$.

Intuitively, BasicPIR guarantees the correct recovery of $DB[i]$, because all database entries are XORed for an even number of times except $DB[i]$. Chor *et al.* [23] prove that, given $\theta = 0.5$, the scheme leaks no information about i and tolerates at most $(l-1)$ colluding servers.

III. Periscoping

In this section, we provide an overview of the Periscoping protocol, following our design objectives and motivations.

A. Design Objectives

To ensure forward secrecy, updated keys in a mixnet must be distributed timely and frequently. An practical solution should be (1) linearly *scalable*, (2) *secure* against reasonable adversaries, and (3) *efficient*. We detail these objectives below.

Linear scalability. The central goal of Periscoping is to make a mixnet linearly scalable, *i.e.,* a mixnet consisting of N mixes should support $O(N)$ users. Clearly, the "download-all" approach does not meet this goal: given N mixes and $O(N)$ users, an average user must download $O(N)$-size key directory updates in a fixed time duration T. With Periscoping, in constrast, a client cannot download more than a constant number of entries in duration T, regardless of the mixnet size.

Threat model. To make a practical design, the adoption of Periscoping must not introduce new vulnerabilities or weaken the security/privacy guarantees of a mixnet. Therefore, we use the following threat model, the same as recent large-scale mixnet systems (*e.g.,* Karaoke [6], Groove [10], Mycelium [13]), without making any additional security assumptions.

Honest-but-curious mixes. Every available mix follows the predefined protocols correctly and answers to requests truthfully. However, it makes attempts to learn all information about the users and the processed messages. In the context of a mixnet, for instance, a mix is curious to learn the linkage between the sender and the receiver of a message.

An any-trust model. We assume a *passive* adversary $\mathcal{A}$ who may observe all the network traffic and the internal states of corrupted mixes. However, $\mathcal{A}$ can only corrupt a small fraction of mixes. If a mixnet path length is l, $\mathcal{A}$ can corrupt at most $(l-1)$ mixes on any path. Note this assumption is fundamental to the privacy of mixnets. In fact, Karaoke [6] even requires two uncorrupted mixes on every path.

Security. To keep communication metadata private [33], Periscoping must guarantee the confidentiality and the integrity of a user's knowledge about the mixnet key directory. With Periscoping operated under the assumed threat model, a user can download a subset of entries in the database. The integrity and the correctness of entries are verifiable using the authentication tags. More importantly, the protocol leaks zero (or bounded) information about which entries have been accessed.

More formally, we adapt the notion of differential privacy [34] to quantify the leakage. Say a user may send either index i or i' as query Q to DB, and O denotes all observations that are possibly made by an adversary $\mathcal{A}$. Periscoping ensure that

$$Pr(O|Q=i) \leq e^{\epsilon} P(O|Q=i'), \tag{1}$$

where ϵ is a security parameter. For $\epsilon = 0$, we have zero leakage; for $\epsilon > 0$, the protocol ensures ϵ-differential privacy.

Intuitively, Equation 1 implies plausible deniability. A user having queried index i can claim the index being query is a different i', which cannot be falsified since outside observations are equally (resp. similarly) probable for $\epsilon = 0$ (resp. $\epsilon > 0$).

Performance. Periscoping should be practically deployable. Given the large number of messages arriving at a mix, it must incur a tolerable overhead. This requirement implies that some expensive cryptographic primitives, *e.g.,* homomorphic multiplications, are no longer applicable. Ideally, the construction should introduce no additional public-key operations.

B. Strawman and Motivation

It is challenging to achieve linear scalability, security, and efficiency at the same time. In fact, meeting the former two alone is essentially a *batch random-index PIR problem* [35].

Say Alice needs to download k randomly selected entries from the n-entry key directory. Let $\mathcal{I}$ denote the set of selected indexes, where $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| = k$. The existing "download-all" approach is simply a solution based on trivial PIR. One may think allowing random selection of entries may bring opportunities for lower-complexity solutions. Unfortunately, it is not the case, and no efficient construction has yet been proposed to meet the performance goal. According to Gentry *et al.* [35], the problem is actually *equivalent* to a batch PIR problem. This observation leads us to a strawman solution:

Strawman 1 (Direct PIR)**.** Alice runs an existing batch PIR protocol to download the entries specified in $\mathcal{I}$ from one or more (*e.g.,* l) randomly-chosen mixes. The PIR protocol is executed in parallel with mixnet operations. □

If we choose to adopt a single-server PIR scheme here, the solution is equivalent to PIR-Tor [20]. As explained in Sec. II-B and in our experiments, even the most efficient construction to date (SpiralPIR [32]) is too expensive. Multi-server PIR seems plausible, with lower complexity; yet their direct adoption can introduce security vulnerabilities under our threat model. Let $\mathcal{I}$ denote the subset of entries held by Alice, who just joined the mixnet and has completed two rounds of PIR. Given the observation, an adversary $\mathcal{A}$ can conclude with confidence that $|\mathcal{I}| \leq 2k$. In the next round of PIR, Alice must communicate with l mixes directly. She now faces a dilemma: on the one hand, if she uses a small l, there is a non-negligible probability that the l selected mixes collude with each other, violating the security assumption of the PIR scheme; on the other hand, a large l (*e.g.,* $l \geq k$) can guarantee the security, while effectively revealing most elements in $\mathcal{I}$ to $\mathcal{A}$.

A natural question arises: can we mitigate the security problem, by requiring Alice to talk to fewer mixes *directly*? Fortunately, as a mixnet user, Alice does so routinely. In fact, in every mixnet round, she sends a message that will eventually pass through l mixes, which can be safely employed as a non-colluding set of PIR servers. This observation leads us to an intuitive solution described below, where the l-server PIR queries are "piggybacking" on a mixnet message.

Strawman 2 (Piggybacking PIR)**.** To start, Alice builds l queries, $q_0, q_1, \ldots, q_{l-1}$, following the l-server $\mathsf{BasicPIR}$ scheme described in Sec. II-B. The only difference is that, to access k entries in a batch, each query includes an array of k n-bit bitsets. While onion-encrypting message m for $\mathsf{Mix}_j (j \in [l])$ along the path, she appends q_j to m as an extra field. As the message goes through the chain of l mixes, Mix_j can peel off its corresponding layer of onion encryption, unpack q_j, and prepares the response r_j to q_j. After delivering m successfully, Mix_{l-1}, the last mix along the chain, initiates the returning procedure. It encrypts r_{l-1} with its ephemeral key ek_{l-1} with Alice, and returns it back to Mix_{l-2}. Mix_{l-2} appends r_{l-2}, encrypts the entire message with ek_{l-2}, and returns it to Mix_{l-3}. This continues until the message is returned to Alice by Mix_0. Alice can peel off the onion layers using the ephemeral keys, parse all the responses $r_0, r_1, \ldots, r_{l-1}$, and finally recover the entries in $\mathcal{I}$. □

Strawman 2 is clearly valid. Since $\mathsf{BasicPIR}$ is provably secure and the onion encryption makes all queries indistinguishable, it guarantees $\epsilon = 0$. The performance goal is also satisfied, as bitwise XORs are the only additional operations. However, the design is not scalable, because the query piggybacking strategy incurs a per-mix bandwidth overhead linear in n, the total number of mixes. More specifically, it has

(1) $O(nkl^2)$-size requests. To access k entries as a batch, every request must pack k n-bit bitsets, *i.e.,* $|q_j| = O(nk)$ for j in $[l]$. Therefore, the overall request bandwidth overhead is $O(nk(l + (l-1) + \ldots + 1)) = O(nkl^2)$.

(2) $O(kl^2)$ overhead in responses. Since each mix will append its response to the reply message, the overall response bandwidth overhead is $O(k(1 + 2 + \ldots + l)) = O(kl^2)$.

C. Overview of Periscoping

We design Periscoping based on the framework of Strawman 2, with techniques for reducing the bandwidth overhead.

Techniques. To reduce bandwidth overhead, we provide efficient constructions for the requests and the responses. As a major contribution of this paper, we propose a novel request compression technique that is both computationally cheap and space-efficient. To the best of our knowledge, our construction achieves the smallest request size among all the PIR schemes that fit our threat model and practical concerns. More specifically, to be applicable in a mixnet, the PIR scheme (1) cannot encode DB because it is updating constantly, and (2) must scale to l servers in which at most $(l-1)$ may collude.

Request compression. Recall that Strawman 2 generates a request that contains k n-bit bitsets. Periscoping compresses such a request by replacing bitset representations with keys of constrained Pseudorandom Functions (cPRFs) [36]–[39]. Crucially, this is the core design component of Periscoping.

(1) Condensed representation of pseudorandom sets. For starters, see how a random bitset can be represented as a keyed Pseudorandom Function (PRF). Consider a keyed PRF $F(\mathsf{msk}, x) : \mathcal{K} \times \mathbb{N} \to [n]$, where $\mathsf{msk} \in \mathcal{K}$ is a master key (*a.k.a.* seed) and x is a natural number. The intuition is that, if we relax the bitset requirement to a multiset (an extended form of set by allowing multiple occurrences of the same element), a random subset of $[n]$ with γ elements can be generated as $S = \{F(\mathsf{msk}, x) | x \in [\gamma]\}$. In other words, given a predefined PRF F, msk is a compressed representation of S.

Clearly, $\mathsf{BasicPIR}$ can work correctly with multisets. To capitalize on this observation, we still have to answer an important question for the correctness: while generating l random subsets of $[n]$, how can we guarantee that every element appears for an even number of times except i?

In Periscoping, we answer this question by making use of constrained Pseudorandom Functions (cPRFs), a special kind of cryptographic PRFs with an additional feature. For each msk and a subset ϕ of $[\mu]$, it can produce a new key ck (*a.k.a.* constrained key), such that $F(\mathsf{ck}, x)$ can evaluate on $x \in \phi$ with the same output, yet fail to evaluate on other $x \notin \phi$. As a result, $S' = \{F(\mathsf{ck}, x) | x \in \phi\}$ is a "programmed" subset of S, and ck is the compressed representation of S'.

Built on this property, we can generate effective representation of random sets for PIR requests in $\mathsf{BasicPIR}$. We denote our scheme as $\mathsf{PRFSetPIR}$. We adopt the cPRF scheme from KPTZ13 [36], which is directly derived from the original GGM PRF construction [40] and very computationally cheap. Its constrained keys are of length $O(\log n)$, small enough for the linear scalability of Periscoping. We will discuss our construction details for $\mathsf{PRFSetPIR}$ in Sec. IV-B.

(2) Batching. As an add-on optimization, we directly adopt Reverse Cuckoo Hashing [25, §4], the batching technique from SealPIR, to reduce the batching cost of Periscoping. Before processing queries, DB redistributes μ ($\mu < k$) copies of each entry into k buckets, using cuckoo hashing, such that each bucket has the same number of distinct entries. The elements in every bucket are known to all users, who can simulate the process. If Alice wishes to query k elements, she can just issue requests to k buckets in separate. There are only $\frac{\mu}{k}n$ entries in each bucket, thus reduce the overall batching cost.

Key compression. To further reduce the size of requests, Periscoping reuses the existing mixnet packet fields. As cPRFs can generate constrained keys that are indistinguishable from random strings, we use them to derive ephemeral keys for encryption, removing some unnecessary fields form requests. More details on the Periscoping message format and field reuse will be elaborated soon, with detailed operations in Fig. 2.

Response compression. Piggybacking responses can result in increasing sizes, as the reply package Π goes back along the chain of mixes. In Periscoping, we attempt to make its size constant. Given the intuition that Alice will eventually XOR all the responses she has received, we may outsource the XOR operations to the mixes, by updating Π *incrementally*. However, a security challenge arises: If Mix_0 can receive a response that can be linked back to Mix_{j-1}, we may compromise the unlinkability promise of the mixnet.

Periscoping tackles this challenge by using a stream cipher. In addition to XORing its response r_j with Π, Mix_j will add more "sauce" to it, by XORing with another string that looks like random. Alice can reproduce the random string using the ephemeral key ek_j, by remove all extra sauces and recover the final result. We next describe the details of this technique.

Protocol. With the novel compression techniques introduced above, we construct the Periscoping protocol (Fig. 2) which integrates naturally to mixnet operations, with no additional communication sessions and little computational overhead.

Request generation and onion encryption. Alice first prepares a list of l queries (Step 1) for downloading a batch of k entries from DB, following $\mathsf{PRFSetPIR}$. Each query q_j has the form of a list of k constrained keys for KPTZ13 [36]. Then, Alice starts encrypting the messages (Step 2). The original message m is end-to-end encrypted to produce its ciphertext m_{l-1}, which only Bob can decrypt. Alice proceeds to onion-encryption as usual, with a distinction in ephemeral key generation. In Periscoping, the ephemeral key ek_j for Mix_j is directly derivable from q_j, thanks to the pseudorandom nature of constrained keys of cPRF. In light of this, Alice uses the public key pk_j of Mix_j to encrypt q_j, forming the message header, and uses ek_j to encrypt the message body with a symmetric cipher. Message m_{l-1}, after l layers of onion encryption, is encoded as m_0, which Alice sends to Mix_0.

Mixnet processing. Upon receiving m_0, Mix_0 first records its

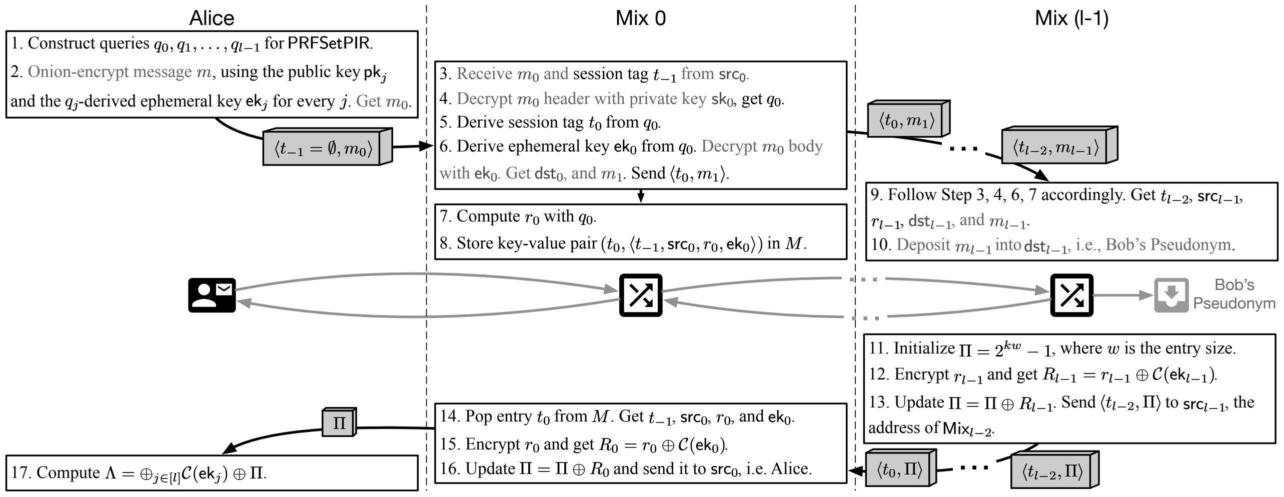
Alice
Mix 0
Mix (l-1)
1. Construct queries $q_0, q_1, \dots, q_{l-1}$ for PRFSetPIR.
2. Onion-encrypt message m, using the public key pk_j and the q_j-derived ephemeral key ek_j for every j. Get m_0.
$\langle t_{-1} = \emptyset, m_0 \rangle$
3. Receive m_0 and session tag t_{-1} from src_0.
4. Decrypt m_0 header with private key sk_0, get q_0.
5. Derive session tag t_0 from q_0.
6. Derive ephemeral key ek_0 from q_0. Decrypt m_0 body with ek_0. Get dst_0, and m_1. Send $\langle t_0, m_1 \rangle$.
7. Compute r_0 with q_0.
8. Store key-value pair $(t_0, \langle t_{-1}, \mathsf{src}_0, r_0, \mathsf{ek}_0 \rangle)$ in M.
$\langle t_0, m_1 \rangle$
$\langle t_{l-2}, m_{l-1} \rangle$
9. Follow Step 3, 4, 6, 7 accordingly. Get t_{l-2}, src_{l-1}, r_{l-1}, dst_{l-1}, and m_{l-1}.
10. Deposit m_{l-1} into dst_{l-1}, i.e., Bob's Pseudonym.
Bob's Pseudonym
11. Initialize $\Pi = 2^{kw} - 1$, where w is the entry size.
12. Encrypt r_{l-1} and get $R_{l-1} = r_{l-1} \oplus \mathcal{C}(\mathsf{ek}_{l-1})$.
13. Update $\Pi = \Pi \oplus R_{l-1}$. Send $\langle t_{l-2}, \Pi \rangle$ to src_{l-1}, the address of Mix_{l-2}.
$\langle t_{l-2}, \Pi \rangle$
$\langle t_0, \Pi \rangle$
14. Pop entry t_0 from M. Get t_{-1}, src_0, r_0, and ek_0.
15. Encrypt r_0 and get $R_0 = r_0 \oplus \mathcal{C}(\mathsf{ek}_0)$.
16. Update $\Pi = \Pi \oplus R_0$ and send it to src_0, i.e. Alice.
Π
17. Compute $\Lambda = \oplus_{j \in [l]} \mathcal{C}(\mathsf{ek}_j) \oplus \Pi$.

Fig. 2: The working of Periscoping protocol. Statements in light color are mixnet operations.

source (*i.e.,* the return address to Alice). With its private key sk_0, Mix_0 can peel off the first layer of onion encryption. It (1) decrypts the header, unpacking q_0 (Step 4); (2) generates a q_0-derived session identifier t_0 (Step 5); and (3) decrypts the body using q_0-derived ephemeral key ek_0, unpacking dst_0, the identifier for the next hop mix, and m_1, the message to relay (Step 6). Mix_0 then sends the tuple $\langle t_0, m_1 \rangle$ to Mix_1.

Note the session tag is used for identifying the session for the responding procedure. Together with a local key-value store M, a mix can maintain sufficient information (Step 8) for getting back to Alice. The protocol works similarly along the mixnet path. Eventually, Mix_{l-1}, the final mix, deposits m_{l-1} into the intended pseudonym of Bob (Step 10).

Response generation and delivery. After concluding message delivery, Mix_{l-1} initiates the responding procedure, returning the request results back to Alice. To implement *response compression*, Mix_{l-1} first initializes the result entries Π to an all-1 string (Step 11), which will be updated by each mix Mix_j along the return path (*i.e.,* $j = l-1, l-2, \dots, 1, 0$).

More specifically, each Mix_j receives a tuple $\langle t_j, \Pi \rangle$, where t_j is the session tag and Π is the intermediate result of k entries. With t_j, Mix_j looks up the persisted session information from M, recovering the upstream session tag t_{j-1}, the upstream address src_j, its computed share of the result r_j, and the ephemeral key ek_j for the session (Step 14).

Then, Mix_j encrypts r_j with a stream cipher C, using ek_j as the key (Step 15). Stream cipher is ideal in our design because the ciphertext R_j is simply produced by XORing the plaintext r_j with $C(\mathsf{ek}_j)$, a random key pad of the same length. It allows Alice to easily remove the random key pads for response reconstruction. In our construction, we use ChaCha20 [41] as C due to its maturity in industrial use and high performance.

As the last step on Mix_j, R_j, the ciphertext of the local response, is used to update Π via XOR operations (Step 16). Eventually, the response is sent back to Alice. She can recover Λ, the requested k database entries, by removing all the random key pads from Π (Step 17). As aforementioned, these random key pads can be reconstructed using C and the ephemeral key with every mix along the path.

IV. Construction Details

We describe the construction details of PRFSetPIR in Periscoping which makes one of our major contributions. We conclude this section with an analysis of Periscoping.

A. Constrained Pseudorandom Functions

A constrained Pseudorandom Function (cPRF) [37]–[39] (also known as Delegatable PRF [36]) is a special form of keyed PRFs, which allows a proxy to evaluate its output on a *subset* of its input domain, but is undefined on other inputs. Here we review its formal definition based on KPTZ13 [36], with minor adaptation to our problem context.

Definition 1 (cPRF)**.** Given key space $\mathcal{K} = \{0,1\}^\lambda$ (λ is a security parameter), input domain $\mathcal{X}$, and output range $\mathcal{Y}$, a cPRF F is defined in four polynomial-time algorithms:

- $F.\mathsf{KeyGen}() \rightarrow \mathsf{msk}$, a randomized algorithm that outputs a master key $\mathsf{msk} \in \mathcal{K}$.
- $F.\mathsf{Eval}(\mathsf{msk}, x) \rightarrow y$, a deterministic algorithm that takes as input a master key $\mathsf{msk} \in \mathcal{K}$ and an element $x \in \mathcal{X}$, and outputs an element $y \in \mathcal{Y}$. We write $F.\mathsf{Eval}(\mathsf{msk}, x)$ as $F(\mathsf{msk}, x)$ when it is clear from the context.
- $F.\mathsf{Constrain}(\mathsf{msk}, S) \rightarrow \mathsf{ck}$, a randomized algorithm that takes as input a master key $\mathsf{msk} \in \mathcal{K}$ and a subset $S \subseteq \mathcal{X}$, and outputs a constrained key $\mathsf{ck} \in \{0,1\}^{\lambda_C}$. The key ck enables the evaluation of $F(\mathsf{msk}, x)$ on all $x \in S$. S should be in the form of an integer range, *i.e.,* $[a, b] \subseteq \mathcal{X}$.
- $F.\mathsf{CEval}(\mathsf{ck}, x) \rightarrow y$, a deterministic algorithm that takes as input a constrained key $\mathsf{ck} \in \mathcal{K}_C$ and an element $x \in \mathcal{X}$, and outputs an element $y \in \mathcal{Y}$ or undefined $\perp$. Specifically, given $\mathsf{ck} = F.\mathsf{Constrain}(\mathsf{msk}, S)$, we have $F.\mathsf{CEval}(\mathsf{ck}, x) = F(\mathsf{msk}, x)$ for all $x \in S$, and $F.\mathsf{CEval}(\mathsf{ck}, x) = \perp$ if $x \notin S$. □

The cPRF in Definition 1 has the following properties.

(1) Correctness. It guarantees that, with the constrained key, one can reproduce $F(\mathsf{msk}, \cdot)$ at the specified subset S of inputs, even if msk is kept private. That is, $F.\mathsf{CEval}(F.\mathsf{Constrain}(\mathsf{msk}, S), x) = F(\mathsf{msk}, x)$ for all $x \in S$.

(2) Security. Aside from the security guarantee of a PRF, a cPRF has *constrained security*, which intuitively enforces that, given several constrained keys generated from a secret master key msk, $F(\mathsf{msk}, \cdot)$ looks like random at all points to a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ cannot compute itself. The formal definition is beyond our scope.

(3) Pseudorandomness of constrained keys. A constrained key ck in KPTZ13 looks random, provided that its associated master key msk is not known. This property is implied by the fact that ck is ultimately generated by a PRF [40].

(4) Performance. All algorithms in a cPRF are polynomial-time. In fact, KPTZ13's construction is completely based on a tree-based PRF, very efficient for its evaluations.

B. PRFSetPIR*: Request Compression*

PRFSetPIR is a novel PIR scheme, working as the core of the Periscoping protocol. Built upon BasicPIR (Sec. II-B), it reduces bandwidth usage with compressed representation pseudorandom sets, by the application of a cPRF.

Consider accessing a single DB entry. With BasicPIR, each request is an n-bit bitset, in which every bit is set with probability 0.5. From the perspective of information theory, lossless compression seems impossible. Fortunately, we make compression possible in PRFSetPIR, by taking advantage of the fact that this subset is generated randomly. Moreover, the entry to be retrieved can be random, thus it is unnecessary to go for a specific entry index. Without loss of generality, we assume l is even and DB has $n = 2^d$ entries. In the case of batching with Reverse Cuckoo Hashing (Sec. III-C), we assume each bucket has size $n = 2^d$. We also assume the availability of a cPRF F, with key space $\mathcal{K} = \{0,1\}^\lambda$, input domain $\mathcal{X} = [2^{d-1}]$, and output range $\mathcal{Y} = [2^d]$.

Representation of pseudorandom multisets. Recall from Sec. III-C that, given each DB index i in $\mathcal{Y}$, a pseudorandom multiset S with $\frac{n}{2}$ elements can be generated as $S = \{F(\mathsf{msk}, x) | x \in \mathcal{X}\}$ (Sec. III-C). We say msk is a compressed representation of S, denoted as $S = \mathcal{S}(\mathsf{msk})$. Given range $[a,b] \subseteq [2^{d-1}]$, we generate a constrained key $\mathsf{ck} = F.\mathsf{Constrain}(\mathsf{msk}, [a,b])$. Let $S' = \{F.\mathsf{CEval}(\mathsf{ck}, x) | x \in [a,b]\}$. As a result, we say ck is a compressed representation of S', a subset of S with $(b-a+1)$ elements, *i.e.,* $S' = \mathcal{S}(\mathsf{ck})$.

Request generation. With the compressed representation of pseudorandom sets, we present the procedure for generating PIR requests $q_0, q_1, \ldots, q_{l-1}$, in the following steps.

1. Set generation. Apply $F.\mathsf{KeyGen}()$ repeatedly for $\frac{l}{2}$ times to get $\mathsf{msk}_0, \mathsf{msk}_1, \ldots, \mathsf{msk}_{l/2}$. These are essentially $\frac{l}{2}$ distinct subsets of $[2^d]$, each with $[2^{d-1}]$ elements.
2. Range sampling. For each u in $[\frac{l}{2}]$, draw two numbers w_u and w'_u from binomial distribution $Binomial(2^{d-1}, \theta)$ independently (*e.g.,* $\theta = 0.5$). Generate two random ranges $[a_u, b_u]$ and $[a'_u, b'_u]$, with w_u and w'_u elements.
3. Set splitting. For each u in $[\frac{l}{2}]$, compute four constrained keys for master key msk_u:
$$\begin{aligned}\mathsf{ck}_{u,1} &= F.\mathsf{Constrain}(\mathsf{msk}_u, [a_u, b_u]),\\ \mathsf{ck}_{u,2} &= F.\mathsf{Constrain}(\mathsf{msk}_u, [2^{d-1}] \setminus [a_u, b_u]),\\ \mathsf{ck}'_{u,1} &= F.\mathsf{Constrain}(\mathsf{msk}_u, [a'_u, b'_u]),\\ \mathsf{ck}'_{u,2} &= F.\mathsf{Constrain}(\mathsf{msk}_u, [2^{d-1}] \setminus [a'_u, b'_u]),\end{aligned}$$
except that $\mathsf{ck}_{0,1} = F.\mathsf{Constrain}(\mathsf{msk}_0, [a_0+1, b_0])$.
4. Request composition. For every odd j in $[l]$, let $u = \frac{j-1}{2}$. Build request $q_j = \langle \mathsf{ck}_{u,1}, \mathsf{ck}_{u+1,2} \rangle$. For every even j in $[l]$, let $u = \frac{j}{2}$. Build request $q_j = \langle \mathsf{ck}'_{u,1}, \mathsf{ck}'_{u+1,2} \rangle$.

C. Analysis and Discussion

We next analyze the practicality of PRFSetPIR.

Correctness. PIR requests $q_0, q_1, \ldots, q_{l-1}$ are the correct requests for entry $DB[F(\mathsf{msk}_0, a_0)]$, a random entry. To see the reason intuitively, consider each request q_j as the union of two subsets, *i.e.,*$\mathcal{S}(\mathsf{ck}_{\frac{j-1}{2},1})$ and $\mathcal{S}(\mathsf{ck}_{\frac{j+1}{2},2})$ (for odd j), or $\mathcal{S}(\mathsf{ck}'_{\frac{j}{2},1})$ and $\mathcal{S}(\mathsf{ck}'_{\frac{j}{2}+1,2})$ (for even j). The union is
$$\bigcup_{j\in[l]} q_j = \bigcup_{u\in[\frac{l}{2}]} [2\mathcal{S}(\mathsf{msk}_u)] \setminus \{F(\mathsf{msk}_0, a_0)\}.$$
Since we XOR all elements to produce the result Λ, the final remaining item would be $DB[F(\mathsf{msk}_0, a_0)]$.

Security. The main result of our security analysis is as follows.

Theorem 1. PRFSetPIR *is zero-leakage for sampling probability $\theta = 0.5$. In general, it is ϵ-differentially private, where $\epsilon = 4 \cdot arctanh[(1-2\theta)]$ and $\theta = 0.5$ is a special case.*

Proof sketch. Consider l queries as an $l \times 2^d$ matrix M. Its elements are binary, and each row represents a query and each column an entry. Thus, adversary $\mathcal{A}$ is interested in the sum of each column, denoted by s_i for column i.

Let o_i denote the sum of the observable part of column i of M, and r_i denote the event that the actual queried entry has index i. We show that PRFSetPIR ensure that $\frac{P(o_i|r_i)}{P(o_i|r_{i'})} \leq e^\epsilon$ for any $i' \neq i$. Since elements are sampled independently, this is equivalent to $\frac{P(o_i|r_i)\cdot P(o_{i'}|r_i)}{P(o_i|r_{i'})\cdot P(o_{i'}|r_{i'})} \leq e^\epsilon$.

Recall that the number of elements in each request is subject to distribution $Binomial(2^d, \theta)$. Hence, we have $P(s_i$ is even$) = 0$ and $P(s_{i'}$ is even $= \frac{1}{2} + \frac{1}{2}(1-2\theta)^{2^d})$. Based on these, we can calculate the upper bound for the previous expression as $\epsilon = 4 \cdot arctanh[(1-2\theta)^{|s_i|-|o_i|}]$, where $|s_i| - |o_i| = 1$ under the any-trust model. □

Practicality. Periscoping is readily deployable as an add-on. Computational overhead. The construction of Periscoping requires cheap cryptographic primitives only. The introduced cPRF algorithms (Definition 1) are all polynomial-time; in fact, every algorithm in the KPTZ13 construction is at most $O(\lambda \log^2 n)$, where λ is the security parameter and n is the number of DB entries. As a result, the request compression technique can run in $O(kl\lambda \log^2 n)$ time, where k is the batch size and l is the mixnet path length. In addition, the response compression is even more efficient. It involves encryption with a standard stream cipher (*i.e.,* ChaCha20) and bitwise XOR operations only. Both are known to have linear complexity in the message length and θ, and to be trivially parallelly scalable.

Bandwidth overhead. The user downloads an $O(k)$-size response (*i.e.,* constant in n), meeting the linear scalability goal. For requests, Periscoping reuses some fields in the mixnet packet to reduce bandwidth usage. It introduces an $O(k \log^2 n)$-size bandwidth overhead, regardless of the mixnet path length l. Since a user has to send at least $O(k \log n)$ bytes to specify k indices, Periscoping is close to the optimum.

Limitations. Periscoping has limitations similar to the SOTA mixnets: (1) It requires a one-time bootstrapping process, where a user must hold its corresponding directory entries as *priori*. (2) It has malleable constructions, such that it cannot work correctly should an active attacker exist.

V. Experimental Study

This section evaluates Periscoping empirically. We focus on its bandwidth usage and computational complexity, while scaling the mixnet size up to tens of millions of mixes.

Settings and parameters. We assumed *w.l.o.g.* that the batch size k for DB access is always equal to the mixnet path length l. We set the entry size in the mixnet key directory to 384 bytes, aligned with the report in [19]. We vary the number of entries in DB between 2^{12}, roughly the size of Tor, and 2^{24}.

Baselines. Periscoping is evaluated in comparison with the following three secure solutions as baselines.

- Trivial: the "download-all" approach. This is the current practice in many mixnet systems. Following the settings in [19], we assume 1% of the mixes update their keys, implying that each user has to download 1% of the DB entries. This is a conservative estimation, as mixes are typically required to update their keys more frequently to guarantee strong forward secrecy. We use this solution to further illustrate the scalability problem.
- SpiralPIR-Tor: Strawman 1 with SpiralPIR [32], the state-of-the-art single-server PIR. This verifies its high cost.
- Piggybacking. Strawman 2, which shares the same protocol framework with Periscoping. It can illustrate the effectiveness of our PRFSetPIR construction.

A. Bandwidth Consumption

We first illustrate the network usage of different schemes. More specifically, we evaluate the induced user upload traffic, user download traffic, and the aggregate traffic overhead caused by a user getting regular DB updates.

User upload traffic (Fig. 3). The amount of user upload traffic depends on both the mixnet size n and the batch size k. We use two settings $k = 16$ and $k = 32$, with varying n. We skip Trivial in this experiment, as its download size is negligible.

As observed, Periscoping has the least amount of user upload traffic under all mixnet configurations. Even though it has a small growth factor as more mixes are added, its bandwidth consumption is still many magnitudes lower that other schemes with 2^{24} mixes. As compared to Piggybacking, Periscoping is much more bandwidth-efficient, thanks to the request compression technique of PRFSetPIR. SpiralPIR-Tor has a constant, yet bulky upload size. Its large request size roots in its use of BFV [29] encryption, which introduces noises to mask homomorphic arithmetic operations.

User download traffic (Fig. 4). Under our setting, Trivial requires a user to download 1% of DB each time, which is linear to n. Here we set $n = 2^{16}$, a moderate DB size. Different from user upload traffic, schemes other than Trivial induce user download traffic depending on the batch size k only. This is understandable since the more entries are batched in a single request, the more data a user needs to download. Thanks to the response compression technique of PRFSetPIR, Periscoping incurs the least amount of user download traffic, which is again magnitudes lower that other schemes. Periscoping and SpiralPIR-Tor have download sizes linear in the batch size k; Piggybacking has a super-linear growth, whose download may even exceed Trivial for $k > 25$.

Mix traffic. We evaluate the average bandwidth cost caused by a single mix user accessing DB. To make a fair comparison, we calculate the *aggregate traffic overhead*, which is the additionally generated traffic beyond mixnet communications. Our evaluation is conducted under $k = 16$ (Fig. 5) and $k = 32$ (Fig. 6). Note the x- and the y-axis are in log scale.

Trivial shows no difference for the two settings of k, and the traffic overhead grows linearly with n. Because SpiralPIR-Tor is a two-party protocol between the user and a single mix, its generated traffic overhead is equal to the sum of user upload and user download. Although Periscoping involves all mixes along the path, it requires a similar traffic overhead on mixes to SpiralPIR-Tor. Considering its much cheaper computational cost (verified below), Periscoping is a better choice.

B. Computational Overhead

Implementation and deployment. We implement Periscoping's core functionalities with over 1k lines of C++ code. More specifically, we implement KPTZ13 cPRF based on the GGM PRF construction [40], with a pseudorandom generator from OpenSSL. Moreover, Piggybacking is realized with its subset of features, and we evaluate SpiralPIR-Tor based on an open-source implementation of SpiralPIR. We deploy our prototypes on an HPC server, with 36-core 3.10GHz Intel Xeon Gold CPU and 250GB memory.

Performace. The execution times of different protocols are summarized in Table I. Please note that the statistics in the SpiralPIR-Tor column are in seconds, while the running times of the other two schemes are measured in milliseconds.

Zero leakage. As compared to Piggybacking and Periscoping, SpiralPIR-Tor's computational cost is at least three magnitudes higher, due mainly to the expensive homomorphic cryptography. Its long running times make it not applicable in practice. Among the three schemes, Piggybacking is more efficient computationally. Periscoping is built upon the Piggybacking protocol framework, with optimizations that compress requests and responses. Considering the reduced bandwidth consumption, the minor CPU cost—introduced by repetitive invocation of cPRF algorithms—is well-justified.

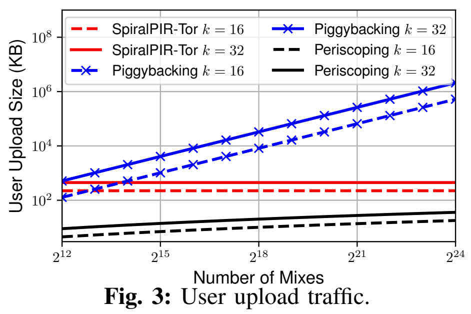

Fig. 3: User upload traffic.

Stage	# Mixes	SpiralPIR-Tor	Piggybacking	Periscoping zero-leakage	Periscoping $\epsilon = 10^{-2}$	Periscoping $\epsilon = 10^{-4}$
Request building	2^{16}	8.8 s	6.7 ms	**15.2 ms**	15.0 ms	15.0 ms
	2^{20}	11.1 s	8.3 ms	**40.8 ms**	40.8 ms	40.6 ms
Response Generation	2^{16}	60.1 s	1.4 ms	**3.6 ms**	1.6 ms (**41.7%**)	2.2 ms (**61.1%**)
	2^{20}	973.4 s	22.5 ms	**59.9 ms**	19.2 ms (**32.1%**)	29.5 ms (**49.2%**)
Response Recovery	2^{16}	20.3 s	0.6 ms	**0.4 ms**	0.4 ms	0.4 ms
	2^{20}	21.8 s	0.6 ms	**0.3 ms**	0.3 ms	0.3 ms

TABLE I: Prototype performance evaluation.

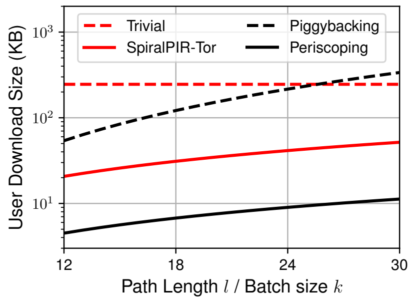

Fig. 4: User download traffic.

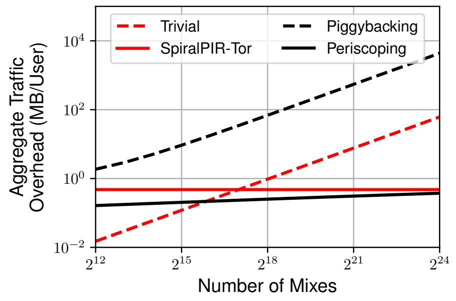

Fig. 5: Traffic overhead on a mix ($k = 16$).

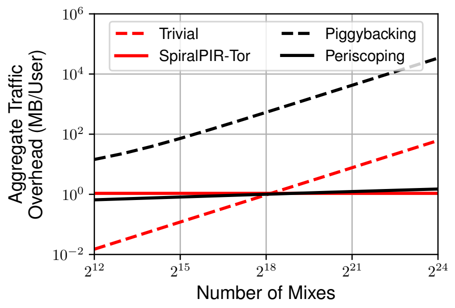

Fig. 6: Traffic overhead on a mix ($k = 32$).

Differentially-private variants. Table I also shows Periscoping for a weaker ϵ-differential privacy. Under a moderate $\epsilon = 10^{-2}$ (resp. tight $\epsilon = 10^{-4}$) bound, on average it takes only 36.9% (resp. 55.2%) time compared to the zero-leakage setup. This is a huge saving on a mix that handles thousands of queries each round, making Periscoping more attractive in practice.

VI. Related Work

Scalability in Tor. Tor [17] has a similar scalability problem in key distribution. As discussed earlier, WalkingOnion [19], the state-of-the-art Telescoping-based solution, works with an expensive circuit-building phase that is typically non-existent in a free-route mixnet. PIR-Tor [20] and ConsenSGX [21] involve expensive cryptographic primitives.

Other solutions in Tor are peer-to-peer protocols, including Crowds [42] and ShadowWalker [43]. However, both protocols are vulnerable to route-capture attacks [44], and have a relatively high risk of becoming victim to epistemic attacks [15]. They cannot fit into the security requirement of a mixnet.

PIR. We identify the key distribution problem in mixnets as a random-index PIR problem, However, existing PIR schemes are not directly or ideally applicable. which we review below.

cPRF-based PIR. CK20 [45] is an online-offline PIR, which offers cheap online request processing at the cost of an expensive amortizable offline preprocessing step. Such schemes rely on the assumption that the underlying database is static. Shi *et al.* [46] proposed a two-server scheme, while a single-machine scheme was constructed in [47]. Unfortunately, none of these constructions can work with l servers where at most $l-1$ servers may collude.

Multi-server PIR has been studied extensively. Our construction is directly based on Chor *et al.* [23], with novel techniques to compress its requests and responses. There are other l-server PIR schemes that can tolerate up-to $(l-1)$ colluding servers (*e.g.,* [48]–[51]), These schemes typically generate less communicating traffic; however, their constructions require complex encoding of DB via, *e.g.,* locally decodable codes [52], and cannot work well if DB is updating constantly.

Single-server PIR. With a single machine hosting DB, computational security is the best one can achieve [22]. Recent constructions, including XPIR [24], SealPIR [25], OnionPIR [26], and SpiralPIR [32] are all based on existing (somewhat) homomorphic encryptions, resulting in extremely high computational and bandwidth overhead. FrodoPIR [53] is an online-offline protocol. It requires expensive preprocessing, not efficiently applicable to the constantly updating DB.

Other PIR constuctions. Random-Index PIR [35] models the design objectives of Periscoping precisely. However, the proposed constructions either depend on primitives that are not yet practical (*e.g.,* fully homomorphic encryptions), or generate too much traffic for communications. Differentially-PIR [34], [54] relaxes the computational or the information-theoretic security, for a better performance. Their promise is that the information leakage is bounded by a constant factor ϵ. The security of Periscoping follows the same definition, while our PRFSetPIR provides a construction more suitable to mixnets.

VII. Conclusion

This paper proposes *Periscoping*, a protocol that addresses the scalability issue in mixnets. It achieves the three objectives: linearly scalable, secure, and efficient in both computation complexity and bandwidth consumption. Our experiments show that it exhibits superior performance in large-scale mixnets, as compared to the state-of-the-arts.

Acknowledgment

This work was supported in part by the National Science Foundation under Award Number OIA-2019511, in part by the Louisiana Board of Regents under Contract Number LEQSF(2019-22)-RD-A-21.

References

[1] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *CACM*, vol. 24, no. 2, pp. 84–90, Feb 1981.

[2] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management," https://tools.ietf.org/id/draft-hansen-privacy-terminology-00.html, 2010.

[3] C. Troncoso, M. Isaakidis, G. Danezis, and H. Halpin, "Systematizing decentralization and privacy: Lessons from 15 years of research and deployments," *Proc. Privacy Enhancing Technologies (PETS)*, vol. 2017, no. 4, pp. 404–426, 2017.

[4] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *Proc. ACM SOSP*, 2015.

[5] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proc. ACM SOSP*, 2017.

[6] D. Lazar, Y. Gilad, and N. Zeldovich, "Karaoke: Distributed private messaging immune to passive traffic analysis," in *Proc. USENIX OSDI*, 2018.

[7] D. Lazar, Y. Gilad, and N. N. Zeldovich, "Yodel: strong metadata security for voice calls," in *Proc. ACM SOSP*, 2019.

[8] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix anonymity system," in *Proc. USENIX Security*, 2017.

[9] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. Anderson, A. Krishnamurthy, and B. Parno, "Talek: Private group messaging with hidden access patterns," in *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2020, pp. 84–99.

[10] L. Barman, M. Kol, D. Lazar, Y. Gilad, and N. Zeldovich, "Groove: Flexible metadata-private messaging," in *Proc. USENIX OSDI*, 2022.

[11] A. Papadimitriou, A. Narayan, and A. Haeberlen, "Dstress: Efficient differentially private computations on distributed data," in *Proc. Eurosys*, 2017.

[12] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, "Honeycrisp: Large-scale differentially private aggregation without a trusted core," in *Proc. ACM SOSP*, 2019.

[13] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen, "Mycelium: Large-scale distributed graph queries with differential privacy," in *Proc. ACM SOSP*, 2021.

[14] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz, "A survey on routing in anonymous communication protocols," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[15] G. Danezis and P. Syverson, "Bridging and fingerprinting: Epistemic attacks on route selection," in *Proc. International Symposium on Privacy Enhancing Technologies Symposium (PETS)*. Springer, 2008.

[16] D. Lazar and N. Zeldovich, "Alpenhorn: Bootstrapping secure communication without leaking metadata," in *Proc. USENIX OSDI*, 2016.

[17] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. USENIX Security*, 2004.

[18] "Tor metrics website," https://metrics.torproject.org/, 2022.

[19] C. H. Komlo, N. Mathewson, and I. Goldberg, "Walking onions: Scaling anonymity networks while protecting users," in *Proc. USENIX Security*, 2020.

[20] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "PIR-Tor: Scalable anonymous communication using private information retrieval," in *Proc. USENIX Security*, 2011.

[21] S. Sasy and I. Goldberg, "ConsenSGX: Scaling Anonymous Communications Networks with Trusted Execution Environments." *Proc. Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 3, pp. 331–349, 2019.

[22] B. Chor and N. Gilboa, "Computationally private information retrieval," in *Proc. ACM Symposium on Theory of Computing (STOC)*, 1997.

[23] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. IEEE FOCS*, 1995.

[24] C. A. Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "XPIR: Private information retrieval for everyone," *Proc. Privacy Enhancing Technologies (PETS)*, vol. 2016, no. 2, pp. 155–174, 2016.

[25] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with compressed queries and amortized query processing," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2018.

[26] M. H. Mughees, H. Chen, and L. Ren, "OnionPIR: response efficient single-server PIR," in *Proc. ACM CCS*, 2021.

[27] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[28] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annual Cryptology Conference (Crypto)*. Springer, 2012.

[29] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[30] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Annual Cryptology Conference (Crypto)*, 2013.

[31] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[32] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server pir via fhe composition," in *Proc. IEEE S&P*, 2022.

[33] Y. Gilad, "Metadata-private communication for the 99%," *Communications of the ACM (CACM)*, vol. 62, no. 9, pp. 86–93, Aug 2019.

[34] K. D. Albab, R. Issa, M. Varia, and K. Graffi, "Batched differentially private information retrieval," in *Proc. USENIX Security*, 2022.

[35] C. Gentry, S. Halevi, B. Magri, J. B. Nielsen, and S. Yakoubov, "Random-index PIR and applications," in *Theory of Cryptography Conference*. Springer, 2021.

[36] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable pseudorandom functions and applications," in *Proc. ACM CCS*, 2013.

[37] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in *Proc. ASIACRYPT*. Springer, 2013.

[38] E. Boyle, S. Goldwasser, and I. Ivan, "Functional signatures and pseudorandom functions," in *Proc. International Workshop on Public Key Cryptography (PKC)*. Springer, 2014.

[39] A. Sahai and B. Waters, "How to use indistinguishability obfuscation: deniable encryption, and more," *SIAM Journal on Computing*, vol. 50, no. 3, pp. 857–908, 2021.

[40] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[41] D. J. Bernstein *et al.*, "Chacha, a variant of salsa20," in *Workshop record of SASC*, vol. 8. Lausanne, Switzerland, 2008, pp. 3–5.

[42] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security (TISSEC)*, vol. 1, no. 1, pp. 66–92, 1998.

[43] P. Mittal and N. Borisov, "Shadowwalker: peer-to-peer anonymous communication using redundant structured topologies," in *Proc. ACM CCS*, 2009.

[44] M. Schuchard, A. W. Dean, V. Heorhiadi, N. Hopper, and Y. Kim, "Balancing the shadows," in *Proc. Annual Workshop on Privacy in the Electronic Society*. ACM, 2010.

[45] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Proc. Eurocrypt*. Springer, 2020.

[46] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs, "Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time," in *Proc. Annual International Cryptology Conference (Crypto)*. Springer, 2021.

[47] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-server private information retrieval with sublinear amortized time," in *Proc. Eurocrypt*. Springer, 2022.

[48] C. Blundo, P. D'Arco, and A. De Santis, "A t-private k-database information retrieval scheme," *International Journal of Information Security*, vol. 1, no. 1, pp. 64–68, 2001.

[49] A. Beimel and Y. Ishai, "Information-theoretic private information retrieval: A unified construction," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2001.

[50] A. Beimel, Y. Ishai, and E. Kushilevitz, "General constructions for information-theoretic private information retrieval," *Journal of Computer and System Sciences*, vol. 71, no. 2, pp. 213–247, 2005.

[51] R. Freij-Hollanti, O. W. Gnilke, C. Hollanti, A.-L. Horlemann-Trautmann, D. Karpuk, and I. Kubjas, "t-private information retrieval schemes using transitive codes," *IEEE Transactions on Information Theory*, vol. 65, no. 4, pp. 2107–2118, 2018.

[52] S. Yekhanin *et al.*, "Locally decodable codes," *Foundations and Trends® in Theoretical Computer Science*, vol. 6, no. 3, pp. 139–255, 2012.

[53] A. Davidson, G. Pestana, and S. Celi, "Frodopir: Simple, scalable, single-server private information retrieval," *Proc. Privacy Enhancing Technologies (PETS)*, pp. 365–383, 2023.

[54] R. R. Toledo, G. Danezis, and I. Goldberg, "Lower-cost ϵ-private information retrieval," *Proc. Privacy Enhancing Technologies (PETS)*, vol. 2016, no. 2, pp. 184–201, 2016.