Sabot: Efficient and Strongly Anonymous Bootstrapping of Communication Channels

Christoph Coijanovic
Karlsruhe Institute of Technology
Karlsruhe, Germany
christoph.coijanovic@kit.edu

Laura Hetz
ETH Zurich
Zurich, Switzerland
laura.hetz@inf.ethz.ch

Kenneth G. Paterson
ETH Zurich
Zurich, Switzerland
kenny.paterson@inf.ethz.ch

Thorsten Strufe
Karlsruhe Institute of Technology
Karlsruhe, Germany
thorsten.strufe@kit.edu

Abstract

Anonymous communication is vital for enabling individuals to participate in social discourse without fear of marginalization or persecution. An important but often overlooked part of anonymous communication is the bootstrapping of new communication channels. If Alice wants to communicate with Bob, she must first learn his in-system identifier. In synchronous designs, message exchange is only possible once both communication partners have agreed to communicate. Thus, Alice must notify Bob of her intent, Bob must learn her in-system identifier, and Bob must acknowledge her notification. This bootstrapping process is generally assumed to occur out-of-band, but if it discloses metadata, communication partners are revealed even if the channel itself is fully anonymized. We propose Sabot, the first anonymous bootstrapping protocol that achieves both strong cryptographic privacy guarantees and bandwidth-efficient communication. In Sabot, clients cooperatively generate a private relationship matrix, which encodes who wants to contact whom. Clients communicate with $k \geq 2$ servers to obtain "their" part of the matrix and augment the received information using Private Information Retrieval (PIR) to learn about their prospective communication partners. Compared to previous solutions, Sabot achieves stronger privacy guarantees and reduces the bandwidth overhead by an order of magnitude.

Keywords

anonymous communication, bootstrapping, privacy

on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744803

1 Introduction

Today, online communication is ubiquitous and an integral part of almost everyone's life. While end-to-end encryption and confidentiality of content has become the default in most mainstream platforms, communication metadata (e.g., who talks to whom) still reveals sensitive information about the clients. Thus, the focus on private and anonymous communication is ever-increasing, in practice, e.g., Signal's *sealed sender* feature,[1] and research, where various Anonymous Communication Networks (ACNs) have been proposed [4, 10, 13, 17, 19, 30, 35]. ACNs aim to protect metadata during ongoing communication so users can communicate freely and without fear of marginalization or persecution. In this paper, we focus on *bootstrapping* of new anonymous communication channels, which existing systems generally assumed to be out-of-band.

The setting for bootstrapping is as follows: We assume a client, Alice, who wants to establish an anonymous communication channel with another client, Bob, in an ACN. Alice knows an external identifier for Bob (e.g., his email address) but needs to obtain his contact information within the ACN. In many ACN designs, communication is synchronous [1, 3, 4, 18, 25, 26, 33, 35], which requires that both Alice and Bob agree to communicate before anonymous message exchange is possible. Therefore, as part of the bootstrapping process, Alice must notify Bob, and Bob must learn her contact information and acknowledge her notification. Bootstrapping cannot occur through an existing non-anonymous channel without disclosing the relationship between Alice and Bob. A face-to-face meeting might facilitate bootstrapping if neither Alice nor Bob are under physical surveillance, but it is inconvenient. Alpenhorn [27] and Pudding [22] proposed the first standalone protocols to provide bootstrapping as described above anonymously via online channels.

In this work, we study the requirements of bootstrapping of anonymous communication and propose Sabot, a new bootstrapping protocol that satisfies the identified requirements. Like related work, Sabot utilizes multiple non-colluding servers. A bootstrapping protocol used in conjunction with an ACN must protect the metadata that the ACN itself aims to hide. Otherwise, leaked bootstrapping metadata might break the privacy goals of the ACN and

[1] https://signal.org/blog/sealed-sender/ — Accessed July 3, 2025

reveal communication relations. We thus require *communication unobservability* [23], i.e., all communication-related metadata is hidden, including how many – if any – relations a client has. Sabot satisfies this requirement, making the protocol suitable for all ACNs.

In anonymous communication, only online clients contribute to the *anonymity set*, i.e., clients are only hidden among those who are participating at the same time. To maximize the size of the anonymity set, Sabot aims to make bootstrapping as bandwidth-efficient as possible, so that even clients with limited bandwidth available can participate for extended periods of time. Anonymous communication has yet to reach mainstream adoption; hence, Sabot focuses on efficient bootstrapping in small to medium-sized deployments of up to a few hundred thousand clients.

We further identify two distinct tasks that need to occur during bootstrapping: 1) potential communication partners must agree to communicate, and 2) the contact information to set up communication in the ACN must be exchanged. The makeup of the contact information depends on the specific ACN to be bootstrapped but could, e.g., include public keys or pseudonyms. In Sabot, clients notify each other of their intent to communicate. To hide the real notifications, communication between all participants is required [24]. While this communication cost – quadratic in the number of clients – is inherent, Sabot requires only a single bit per possible relation. Relationship privacy of these bits is preserved by using secret sharing between multiple non-colluding servers. In Sabot, the exchange of contact information is realized using a protocol for Private Information Retrieval (PIR). Using PIR ensures that clients can obtain their communication partner's contact information from a central database while the servers jointly holding the database remain oblivious to the partner's identity. To hide the number of contacts per client, Sabot uses fixed PIR retrieval rates.

We introduce two variants of Sabot that differ in the trust assumptions they make about the servers: Firstly, Sabot_m, assumes *anytrust* among the servers, as assumed in related work [22, 27]. In the anytrust model, all but one server can be malicious. Sabot_m requires the use of an *authenticated* PIR scheme [11] to ensure that the malicious servers cannot send false contact information to a client without detection. Secondly, for settings where all servers are semi-honest, we introduce Sabot_h. Sabot_h is agnostic to the underlying PIR scheme (as it only requires honest-but-curious security) and can thus be more efficient than Sabot_m.

Alpenhorn, Pudding, and Sabot all provide unlinkability between communicating clients and are resistant to traffic analysis. However, Alpenhorn and Pudding only provide sketch-level proofs of their privacy guarantees. In contrast, we adapt Kuhn et al.'s privacy notion of communication unobservability [23] to the bootstrapping setting and provide a full formal proof that Sabot_h achieves this notion against a passive adversary. For Sabot_m, we provide an informal analysis that it protects the same metadata as Sabot_h.

To evaluate Sabot, we implemented a prototype in Go. Our evaluation shows that Sabot is highly bandwidth-efficient for small to medium-sized deployments: For 2^{14} participating clients, Sabot_m requires a client to exchange 7.04 KiB of data per bootstrapped communication, which is an order of magnitude less than related work. With Sabot_h, this overhead decreases further by a factor of nearly 2×. For 2^{14} clients, both Sabot_m and Sabot_h are computationally efficient with end-to-end latency of under 10 seconds.

Our main contributions in this paper are:

- Sabot, the first ACN bootstrapping protocol that is both strongly anonymous and bandwidth-efficient.
- The adaption of the formal privacy notion of communication unobservability [23] to the bootstrapping setting.
- A full formal proof that Sabot_h achieves communication unobservability with strong cryptographic privacy guarantees against a passive adversary.
- A prototype implementation and empirical evaluation of Sabot, highlighting its feasibility for bootstrapping of small to medium-sized ACN deployments.

This paper is organized as follows: § 2 discusses related work and § 3 introduces the necessary background. In § 4, Sabot's functionality, threat model, and privacy goals are stated, and in § 5, the general design of Sabot is outlined. In § 6 the protocol is described in detail and § 7 gives a formal security analysis of Sabot. § 8 presents the results of Sabot's empirical evaluation. Finally, § 9 discusses extensions to Sabot and § 10 concludes this work.

2 Related Work

Most ACNs rely on out-of-band bootstrapping to set up new communication channels. Our goal in this paper is to design a system that enables bootstrapping without metadata disclosure and without out-of-band communication. In this section, we discuss related work with the same goal.

Alpenhorn. Alpenhorn [27] was the first protocol to provide an anonymous bootstrapping functionality. In Alpenhorn, clients use Identity-Based Encryption (IBE) to encrypt their contact information without knowing the recipient's public key. The resulting ciphertexts are sent to the recipients' mailboxes through a mix network. Alpenhorn uses cover traffic generated by the mix servers to hide the number of clients trying to contact a given recipient. Alpenhorn has two significant drawbacks: First, it provides only differential privacy guarantees rather than the stronger cryptographic guarantees Sabot aims to achieve. Relying on differential privacy requires a careful selection of protocol parameters that balance long-term privacy and overhead. A client's popularity is only hidden within the bounds set by those parameters. Second, Alpenhorn's use of server-generated cover traffic leads to very high bandwidth overhead for clients: During each protocol round, a client can only initiate bootstrapping with one other client, but has to download about 7 MiB of data.

Pudding. Pudding [22] is a recent design that aims to improve on Alpenhorn's high bandwidth overhead. Pudding replaces Alpenhorn's IBE with server-issued and pre-computed packet headers. Clients can append their contact information to the header to send it through a mix network to the recipients. No prior knowledge, except for the recipient's identifier, is required for this. In Pudding, the actual message exchange for bootstrapping is very efficient, but its underlying mix network, Nym,[2] generates cover traffic at a high rate. For a client, the bootstrapping of a single communication channel incurs 7 MiB of transmitted data. Pudding achieves membership

[2]https://nymtech.net — Accessed July 3, 2025

unobservability against clients, i.e., during the bootstrapping, Alice does not learn if Bob is registered unless he decides to answer her bootstrapping request. This property cannot be achieved against servers, as they guarantee the authentication of clients.

We note that Pudding's design comes with an important privacy limitation: Since clients request the headers in plain, servers learn how many requests arrive for each receiver and can infer their popularity. We consider this leakage to be significant, as popular receivers are high-value targets for further attacks. Sabot aims to hide this leakage.

3 Background

This section provides relevant background on secret sharing (§3.1), and Private Information Retrieval (PIR) (§3.2). Throughout the paper, the following notation is used: A set $\{1, \dots, n\}$ is denoted by $[n]$, vectors are denoted by $\mathbf{v} = (x_1, \dots, x_m) \in \{0,1\}^m$ and matrices by $\mathcal{M}$ with element $\mathcal{M}_{i,j}$ for row and column indices $i, j \in [n]$. Furthermore, λ is the security parameter and functions polynomial or negligible in λ are stated as $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, respectively.

3.1 Secret Sharing

Schemes for secret sharing allow clients to split some data d into n shares, such that the original data can only be reconstructed with at least t shares. Any subset of less than t shares does not reveal any information about the data.

One can construct a simple yet computationally efficient n-out-of-n secret sharing scheme, i.e., $t = n$, by randomly generating shares $s_1, \dots, s_{n-1}$ and setting $s_n = s_1 \oplus \cdots \oplus s_{n-1} \oplus d$. The data can be recovered by computing $d = s_1 \oplus \cdots \oplus s_n$. This construction is referred to as XOR Secret Sharing (XSS) and is used in this work.

Definition 1 (XOR Secret S haring)**.** An XSS scheme with k servers, over plaintext space $\mathcal{X}$, consists of two algorithms with the following syntax:

- $\mathsf{Share}(s) \to ((s_i)_{i\in k})$: takes as input data $s \in \mathcal{X}$, and returns randomly generated shares $\{s_i\}_{i\in k}$.
- $\mathsf{Combine}((s_i)_{i\in k}) \to s$: takes as input k shares $(s_i)_{i\in k}$, and returns data $s \in \mathcal{X}$.

3.2 Private Information Retrieval

Protocols for Private Information Retrieval (PIR) allow clients to query a public database DB without revealing their requested items. One or multiple servers holding the database answer queries but remain oblivious to the requested information. Literature distinguishes between Index-PIR (I-PIR) and Keyword-PIR (KW-PIR). In settings where the servers are assumed to be malicious, Authenticated PIR (APIR) is required.

Index-based PIR. With Index-PIR (I-PIR), each database element is addressed via a unique index. Clients are assumed to know the database index of the element they wish to retrieve.

Definition 2 (Private Information Retrieval)**.** A *multi-server PIR scheme with index-based queries (I-PIR)* with k servers for a database $\{x_i\}_{i\in N}$ sampled over some element space $\mathcal{X}$ with $N \in \mathbb{N}$, consists of four algorithms with the following syntax:

- $\mathsf{Setup}(1^\lambda, (x_i)_{i\in N}) \to (\mathsf{pp}, \mathsf{DB})$: takes as input a security parameter λ and a set of elements in $\mathcal{X}$, and returns the PIR database and its public parameters pp.
- $\mathsf{Query}(\mathsf{pp}, idx) \to (\mathsf{st}, \mathbf{q})$: takes as input public parameters and an index $idx \in [N]$, and returns client state st and queries $\mathbf{q} \leftarrow (q_i)_{i\in k}$.
- $\mathsf{Answer}(\mathsf{pp}, \mathsf{DB}, q_i) \to a_i$: takes as input a database DB and client query q_i, and returns answer a_i.
- $\mathsf{Reconstruct}(\mathsf{st}, \mathbf{a}) \to x$: takes as input client state st and answers $\mathbf{a} \leftarrow (a_i)_{i\in k}$, and returns an element $x \in \mathcal{X}$.

A PIR protocol must satisfy *correctness* and *privacy*. We provide formal definitions in § A.1. Informally, correctness ensures that an honest client obtains the requested database record, i.e., $x = \mathsf{DB}[idx]$ for I-PIR, from honest-but-curious servers. Privacy guarantees that the servers do not learn any information about the requested record from answering the query.

In this work, I-PIR is instantiated with DPF-PIR, a communication-efficient two-server PIR protocol based on Distributed Point Functions (DPFs) [6, 7, 20]. In DPF-PIR, a client generates a point function for the index they want to query in the database. This function is secret-shared to both servers such that no individual share leaks any information about the queried index. Each server evaluates a function share on the database, and the client can combine all evaluated shares to recover the database record.

Keyword-based PIR. Compared to I-PIR, Keyword-PIR (KW-PIR) allows clients to privately retrieve records from a key-value store using keyword-based queries. Each database record $x_i \in \mathcal{X}$ is assigned a keyword $k_i \in \mathcal{K}$ for $i \in N$. We adapt the following algorithms in Def. 2 to allow keyword-based queries:

- $\mathsf{Setup}(1^\lambda, \{(k_i, x_i)\}_{i\in N}) \to (\mathsf{pp}, \mathsf{DB})$: takes as input a security parameter λ and a set of key-value pairs with keys in $\mathcal{K}$ and values in $\mathcal{X}$, and returns the PIR database and its public parameters pp.
- $\mathsf{Query}(\mathsf{pp}, k) \to (\mathsf{st}, \mathbf{q})$: takes as input public parameters pp and a keyword $k \in \mathcal{K}$, and returns client state st and queries $\mathbf{q} \leftarrow (q_i)_{i\in k}$.

Keyword-based PIR can be constructed by adding a "keyword-to-index" mapping function to an index-based PIR scheme. Key-value pairs are stored in a data structure where each keyword can be mapped to multiple positions using hash functions. Hence, for each KW-PIR request, multiple I-PIR queries are needed to obtain the KW-PIR record from the I-PIR answers.

In this work, we employ the hashing strategy of Binary Fuse Filter (BFF) [21] for the mapping function [9]. To allow retrieval, full key-value pairs are stored at one of the positions derived through hashing. For $N_{hash} = 3$ hash functions, a BFF needs to be $c \geq 1.125$ times larger than the number of items it contains to reduce the risk of hash collisions [21]. As the communication and computational costs of PIR depend on the database size, Sabot only uses this data store as the KW-PIR and not the I-PIR database.

Authenticated PIR. Multi-server PIR protocols generally assume honest-but-curious servers. In this setting, data integrity is guaranteed by assumption, as servers can neither deviate from the protocol nor exchange information. If, however, this assumption does not hold, even a single actively malicious server can force a chosen

output on the client. Compared to standard PIR, Authenticated PIR (APIR) [5, 11, 15, 16, 36] introduces an additional mechanism that allows clients to detect if a server deviated from the protocol and sent wrong information. We extend the definitions of I-PIR and KW-PIR for the malicious setting to allow the adversary to learn the verification result:

- $\mathsf{Setup}(1^\lambda, \mathsf{seed}, \mathit{in}) \rightarrow (\mathsf{pp}, \mathsf{DB})$: takes as input a security parameter λ, a random seed to coordinate and authenticate the database setup between servers, and an input set $\mathcal{D}$, and returns the PIR database and its public parameters pp. For I-PIR, $\mathit{in} \leftarrow \mathcal{X}^N$. For KW-PIR, $\mathit{in} \leftarrow (\mathcal{K} \times \mathcal{X})^N$.
- $\mathsf{Reconstruct}(\mathsf{st}, \mathbf{a}) \rightarrow \{x, \perp\}$: takes as input client state st, and answers $\mathbf{a} \leftarrow (a_i)_{i \in k}$, and returns an element $x \in \mathcal{X}$ if the verification is successful, else returns $\perp$.

An APIR scheme needs to satisfy the properties of correctness (as stated above) as well as *integrity* of the retrieved database record and *privacy against a malicious server*. For formal definitions see §A.1.

Colombo et al. propose an APIR scheme [11] that guarantees data integrity by combining a traditional multi-server PIR protocol with proof-of-inclusions for all database items [29]. We utilize their construction to achieve database authenticity for bootstrapping in the fully malicious case for both I-PIR and KW-PIR.

4 Model & Goals

This section presents Sabot's functionality, threat model, and its specific privacy and security goals and non-goals.

4.1 Functionality

Many prominent ACN designs [1, 3, 4, 13, 14, 18, 19, 25, 26, 30, 33–35] focus on providing client-to-client communication channels which require the exchange of contact information prior to communication. In these designs, contact information consists of public key material, unique in-system identifiers, or (random) mailbox addresses. Designs that exchange messages synchronously [1, 3, 4, 18, 25, 26, 33, 35] additionally require clients to notify intended communication partners of the upcoming message exchange. Table 1 gives an overview of these designs and their bootstrapping requirements.

Assuming Alice wants to bootstrap a communication channel with Bob, we identify two distinct tasks they need to complete:

(1) **Exchange of Contact Information.** Alice and Bob must learn each other's contact information Δ to set up subsequent communication in the bootstrapped ACN.
(2) **Notification.** Alice needs to notify Bob of her intention to communicate with him. If Bob also wishes to communicate with Alice, he must confirm her notification.

Sabot allows the anonymous bootstrapping of bi-directional one-to-one communication channels between clients from the set of registered clients U. Since it does not disclose metadata, Sabot is suitable for bootstrapping ACNs, which aim to hide metadata during communication. To participate in the bootstrapping, a client must register in Sabot using a well-known external identifier, such as an email address or phone number. The client further has to provide their *contact information* Δ for the ACN. We assume Δ to be public information independent of the contact-initiating client. The structure and content of Δ depend on the targeted ACN.

ACN	Vuvuzela [35]	Pung [3]	Stadium [33]	Karaoke [26]	XRD [25]	MCMix [1]	Clarion [18]†	Groove [4]	Dissent [13]	Verdict [14]	Sabre [34]	Express [19]	Loopix [30]
Notify	(✔)*	✔	✔	✔	✔	(✔)	(✔)	✔	✖	✖	✖	✖	✖
Required Contact Information	🔑	🔑	🔑	🔑	🔑	🪪	🪪	🔑	🔑	🔑	✉‡	✉	🔑 🪪

Table 1: Survey of prominent ACNs and their compatibility with Sabot. Protocols marked with ✔ require a notification phase, while ✖ states the opposite. ACNs denoted with (✔) propose their own subprotocol for notification. The contact information used in these systems is categorized by public keys (🔑), unique identifiers (🪪), and (random) mailbox addresses (✉).

*** Later replaced by Alpenhorn, † proposes an improvement for MCMix's anonymization technique, ‡ pseudo-randomly chosen by servers upon joining the system.**

If Alice wants to bootstrap communication with Bob, we refer to her as the *sender* and to him as the *receiver* of the bootstrapping relation. Alice *only* needs to know Bob's public identifier ID_B. Bob does not need to know anything about Alice, including the fact that she wants to communicate with him.

4.2 Threat Model

In Sabot, N clients interact with k servers over a public network. We propose two constructions of Sabot that differ in the assumed adversary model:

- Sabot_h assumes an adversary $\mathcal{A}$ with Global Passive Adversary (GPA) abilities that additionally has access to a single honest-but-curious server.
- Sabot_m assumes an adversary $\mathcal{A}'$ which can actively corrupt an arbitrary fraction of clients, controls $k-1$ malicious servers, and can actively interfere (i.e., replay, drop, delay, and modify packets) on every network link.

4.3 Privacy & Security Goals

Sabot aims to achieve the following goals:

Privacy. Sabot shall disclose *no* metadata about the activity of honest clients. This includes which and how many, if any, bootstrapping relationships a client is involved in. We formalize our privacy goal as part of the security proof in §7.1.

Authenticity. Sabot shall ensure the authenticity of honest clients. If Alice bootstraps a communication channel based on the identifier ID_B, Sabot must connect her with the owner of ID_B (i.e., Bob) rather than some impersonator. Further, if Bob receives a bootstrapping request from ID_A, he must be certain that it stems from the owner of ID_A (i.e., Alice). Like related work [22, 27], we assume that identifiers of honest clients are external (e.g., email addresses) and not controlled by the adversary.

Non-Goals. Related work [22] aims to hide whether a client is registered with the system. This *membership privacy* cannot be achieved against servers since they must verify the identifier of clients and act as trust anchors for authenticity. Since we assume that malicious clients can collude with servers, we do not aim to achieve membership privacy. Compared to attacks on privacy, a denial-of-service attack by a server is easily recognizable by any client, so servers have a strong external incentive to ensure availability. Hence, we do not aim to guarantee availability against malicious servers.

5 Design

Recall that the bootstrapping functionality consists of two parts: Notification and exchange of contact information. Neither part is allowed to reveal any metadata to ensure anonymity. In the following section, we give a high-level overview of how Sabot implements this functionality, ensuring both anonymity and authenticity. Like many anonymous communication protocols, e.g., [13, 19, 35], Sabot operates in synchronized rounds. The entire bootstrapping protocol is executed in each round; thus, it only takes one round to set up an anonymous communication channel between two parties.

Efficient and Popularity-Hiding Notification. Suppose Alice wants to bootstrap a communication channel with Bob and thus needs to inform Bob of her intention. If Bob accepts her request, he must let Alice know this. Sabot is the first bootstrapping protocol that enables this notification while *perfectly* hiding how popular a client is. This is a challenge, as, in the worst case, a client could notify *all* other clients or could be notified by *all* other clients. To make these edge cases indistinguishable from all other cases, incoming and outgoing bandwidth for every client has to be in $\mathcal{O}(N)$ [24].

Within this asymptotic constraint, Sabot proposes notification with optimal efficiency: Each client sends a vector of N bits, where each index of the vector corresponds to a specific client. Alice sets the vector elements to '1' for each client she wants to notify. The vectors of all clients are centrally collected and interpreted as the *columns* of a $N \times N$ matrix. Clients receive back a *row* of the matrix, corresponding to their own index, and can thus learn the indices of the clients who want to notify them.

This matrix cannot be constructed and transmitted in plaintext as it reveals client relationships. To hide this information, the vectors are *secret-shared* (see §3.1) among k servers. Due to the properties of the secret-sharing scheme, any $k-1$ colluding servers cannot learn any information about the matrix values as long as the remaining server is non-colluding and withholds its share.

Exchange of Contact Information. In addition to notifying each other about their intent to communicate, Alice and Bob must exchange their contact information to set up communication in the ACN to be bootstrapped. Note that Alice and Bob also need to know each other's index within the Sabot instance to effectively use the notification matrix described above.

Sabot stores the contact information together with each client's identifier and index in a logically central database. When Alice wants to contact Bob, she knows his identifier (e.g., his email address) and uses it to query the database for his index and contact information. When Bob learns Alice's index through the notification vector, he uses it to query for her identifier and contact information.

Sabot ensures that the queries do not disclose relationships through the use of Private Information Retrieval (PIR) (see §3.2) in two variants: Keyword-PIR (KW-PIR) is used for retrieval based on client identifiers, and Index-PIR (I-PIR) is used for retrieval based on indices. The k servers holding a copy of the PIR databases stay oblivious to the queried records.

Hiding Communication Patterns. To achieve communication unobservability, Sabot must ensure that observable client behavior does not depend on their communication patterns. Sabot operates in synchronized rounds. In each round, each participating client must send shares of their notification vector to the servers and make a fixed number of KW-PIR queries. If Alice wants to contact fewer clients than this fixed number, she has to add dummy queries (e.g., to her own identifier). Each client must also retrieve the shares of their row vector and make a fixed number of I-PIR queries. If Bob has been contacted by fewer clients than the fixed number, he must add dummy queries (e.g., to his own index). If more clients than the fixed number have contacted Bob, he must randomly select a subset of indices to query. Note that Bob can only accept invites from clients he selects to query. The number of I-PIR queries to make is a system parameter to be set based on expected use. Alternatively, the parameter could be set adaptively based on the actual demand, which we discuss as an extension to Sabot in §9.

We assume that Bob is a privacy-aware client who does not simply accept all requests he receives, but instead carefully selects the clients he wants to communicate with. If Bob does not want to communicate with Alice, he simply does not reply to her request. From a lack of reply, Alice does not learn if this is because a) Bob does not want to talk to her, or b) Bob currently does not have the capacity to reply. To strengthen the privacy of clients that are more naive and predictable in their answering actions, Angel et al. [2] propose additional measures to prevent non-negligible leakage.

Ensuring Authenticity. In addition to privacy, Sabot must also ensure authenticity: If Alice initiates contact based on Bob's identifier, Sabot must guarantee that she is actually talking to the real Bob. Likewise, if someone claiming to be Alice contacts Bob, Sabot must guarantee that it is indeed the real Alice. Otherwise, a malicious party could impersonate clients.

Like related work [22, 27], Sabot has a mandatory registration phase before clients can participate in the system. We assume public identifiers, e.g., an email address or phone number, to which clients have access. During registration, clients provide their identifier and contact information, which the servers verify.

During protocol execution, clients can retrieve other clients' contact information via PIR from the servers. If all servers are at least honest-but-curious (see §4.2), we can assume that the servers compute their PIR responses based on the "correct" data. As the client can be sure to receive the contact information they requested, authenticity is guaranteed. Sabot can be used with any PIR protocol in this honest-but-curious setting.

In a setting where the server(s) can be malicious, Sabot has to be used with a PIR scheme that ensures data integrity [11, 15, 16, 36]. These schemes include an integrity check for the retrieved data to verify that all servers have honestly computed their responses

and that malicious behavior is detected. Sabot$_m$ operates in the anytrust model where at least one server is assumed to be honest-but-curious. This ensures that the integrity check succeeds.

Our implementation of Sabot$_m$ uses multi-server APIR [11] based on a Merkle-Tree [29] and DPF-PIR [6, 7] construction, but can generally be used with any APIR scheme.

6 Protocol Description

This section describes the Sabot protocol in detail. It introduces two constructions: Sabot$_h$, which assumes honest-but-curious servers, and Sabot$_m$, which assumes malicious servers in the anytrust model. The differences of Sabot$_h$ to Sabot$_m$ are highlighted.

Let $U = \{u_1, \dots, u_N\}$ be the set of clients participating in Sabot and let $S_1, \dots, S_k$ be a set of k servers. Before bootstrapping can start, every client in U has to *register* at the servers. Once all clients have registered, the servers run a one-time *setup* phase. After the setup, Sabot operates in synchronized rounds, each divided into four phases: *Sender retrieval*, *sender notification*, *receiver retrieval*, and *receiver notification*. Sabot leverages an XSS scheme XSS, I-PIR and KW-PIR schemes PIR_I and PIR_K respectively.

6.1 Registration

The goal of the registration phase is for a newly joining client to provide their information to the servers. The servers verify the received information and, if successful, provide the client with the necessary information to participate. Let Alice with identifier $ID_\mathsf{A} \in \mathcal{K}$, e.g., `alice@alice.com` and contact information $\Delta_\mathsf{A} \in \mathcal{D}$ be a client who wants to register with Sabot. First, Alice establishes an encrypted and bi-directionally authenticated channel with each server. Encryption and authentication of the server can be realized, e.g., using Transport Layer Security (TLS) [31]. To authenticate Alice, the server can, for example, issue her a lightweight X.509 certificate. All future communication between Alice and the servers occurs through this channel. For Sabot$_h$, the remainder of the registration is executed between Alice and server S_0 and proceeds as follows:

(1) Alice sends $(ID_\mathsf{A}, \Delta_\mathsf{A}) \in \mathcal{K} \times \mathcal{D}$ to S_0.
(2) S_0 generates a random token and sends it to Alice via identifier ID_A, e.g., `alice@alice.com`.
(3) Alice receives the tokens via her identifier and sends them back to S_0. If Alice can reproduce the correct token, S_0 assumes that Alice indeed owns ID_A. The server keeps a list of all registered users and their contact information, i.e., $reg = \{(ID_i, \Delta_i)\}_{i \in [N]}$.
(4) Server S_0 shares reg with all other servers.

For Sabot$_m$, Alice has to register at each server individually as described above for S_0. The servers coordinate to add Alice at the same position in their list reg. Each server sends Alice her position, and she aborts in case of discrepancies.

Remark. While we introduce a basic registration, we note that Sabot is fully compatible with more advanced registration features, such as post-compromise security, re-registration mechanisms (see Alpenhorn [27]), and authentication via DKIM (see Pudding [22]).

6.2 Setup

The goal of the setup phase is for the servers to construct Sabot's *contact database(s)* based on the list of registered clients reg and to distribute the protocol's public parameters to the clients. Before setup, the servers coordinate to ensure that all have the same list reg of registered clients. For Sabot$_h$, S_0 executes the setup as follows:

(1) The server runs $\mathsf{pp}_I, \mathsf{DB}_I \leftarrow \mathsf{PIR}_I.\mathsf{Setup}(1^\lambda, X)$ where $X := \{ID_i \parallel \Delta_i\}_{(ID_i, \Delta_i) \in reg}$ to generate the I-PIR database. Each DB_I record is addressable by a unique index, i.e., $\mathsf{DB}_I = \{(idx_i, ID_i \parallel \Delta_i)\}_{i \in [N]}$.
(2) The server parses each client record in DB_I to obtain a key-value mapping from identifier to all client data, i.e., $Y \leftarrow \{(ID_i, ID_i \parallel \Delta_i \parallel idx_i)\}_{i \in [N]}$. The server runs $\mathsf{pp}_K, \mathsf{DB}_K \leftarrow \mathsf{PIR}_K.\mathsf{Setup}(1^\lambda, Y)$ to generate the KW-PIR database.
(3) The server distributes $(\mathsf{pp}_I, \mathsf{pp}_K)$ to all clients.

In Sabot$_m$, the setup described above is executed by all servers in parallel. If the setup is non-deterministic, the servers first derive a shared random seed. Clients abort if they do not receive the same $(\mathsf{pp}_I, \mathsf{pp}_K)$ from all servers.

6.3 Sender Retrieval

The goal of the sender retrieval phase is for each client to retrieve the contact information of the clients they want to contact. Assume Alice wants to contact Bob with identifier ID_B = `bob@bob.com`. Server S_0 announces the start of the sender retrieval phase, which then proceeds as follows:

(1) Alice runs $\mathsf{PIR}_K.\mathsf{Query}(\mathsf{pp}_K, ID_\mathsf{B}) \rightarrow (\mathsf{st}, \mathbf{q})$, where $\mathbf{q} = (q_1, \dots, q_k)$.
(2) Alice stores the state st for later use and sends the queries to the servers where the ith server receives $q_i \in \mathbf{q}$.
(3) Each server S_i runs $\mathsf{PIR}_K.\mathsf{Answer}(\mathsf{pp}_K, \mathsf{DB}_K, q_i) \rightarrow a_i$ and returns the answer a_i to Alice, who collects the answers into a vector $\mathbf{a}$.
(4) Once Alice has received an answer from every server, she runs $\mathsf{PIR}_K.\mathsf{Reconstruct}(\mathsf{st}, \mathbf{a}) \rightarrow res$. If $res = \perp$, the APIR integrity check has failed and Alice discards the answer.
(5) Alice stores $res = idx_\mathsf{B} \parallel ID_\mathsf{B} \parallel \Delta_\mathsf{B}$ as Bob's information for later use.

To hide how many clients a sender wants to contact, every sender must make R_send queries. Clients who want to contact fewer than R_send clients add cover queries with their own identifier as the keyword. The client learns if a keyword is not in the database, i.e., an identifier has not been registered.

6.4 Sender Notification

The goal of the sender notification phase is for clients to notify the receivers of their intent using the indices obtained in the sender retrieval phase. The phase proceeds as follows:

(1) If Alice only wants to contact Bob with index idx_B, she generates a vector $\mathbf{v}_\mathsf{A} \in \{0,1\}^N$, where

$$\mathbf{v}_\mathsf{A}[i] := \begin{cases} 1 & \text{if } i = idx_\mathsf{B} \\ 0 & \text{else.} \end{cases}$$

If she also wants to contact other clients, she sets the vector to '1' at every index that corresponds to one of the receivers.

(2) Alice runs XSS.Share($\mathbf{v}_\mathsf{A}$) $\rightarrow$ $(\mathbf{v}_\mathsf{A}^1, \ldots, \mathbf{v}_\mathsf{A}^k)$ to generate the secret shares of her vector. Share $\mathbf{v}_\mathsf{A}^i$ is sent to S_i for $i \in [k]$.

(3) Each server $i \in [k]$ assembles the incoming shares of clients into a $N \times N$ matrix M_i, where $\mathbf{v}_\mathsf{A}^i$ makes up the *column* with index idx_A. The servers wait a fixed amount of time for all clients to send their shares. Columns that did not receive a share before the timeout are filled with 0 values. Upon completion of the matrix, each server notifies all clients.

(4) Bob retrieves from each server the *row* with index idx_B, i.e., $(\mathbf{v}_\mathsf{B}^0, \cdots, \mathbf{v}_\mathsf{B}^k)$. He runs XSS.Reconstruct($(\mathbf{v}_\mathsf{B}^i)_{i\in[k]}$) $\rightarrow$ $\mathbf{v}_\mathsf{B}$ to combine the shares. Vector $\mathbf{v}_\mathsf{B}$ contains the indices of the senders who want to contact Bob. He stores the list of senders for use in the next phase.

6.5 Receiver Retrieval

The goal of the receiver retrieval phase is for receivers to learn the identifier and contact information of the senders who contacted them. If Alice contacted Bob, then $\mathbf{v}_\mathsf{B}[idx_\mathsf{A}] = 1$. Bob does not yet know which sender corresponds to idx_A.

(1) Bob runs PIR_I.Query(pp_I, idx_A) $\rightarrow$ $(\mathsf{st}, \mathbf{q} = (q_1, \ldots, q_k))$.

(2) Bob stores the state st for later use and sends the queries to the servers where the ith server receives $q_i \in \mathbf{q}$.

(3) Each server S_i runs PIR_I.Answer(pp_I, DB_I, q_i) $\rightarrow$ a_i and returns the answer a_i to Bob, who collects the answers into a vector $\mathbf{a}$.

(4) Once Bob has received an answer from every server, he runs PIR_I.Reconstruct($\mathsf{st}, \mathbf{a}$) $\rightarrow$ *res*. If $res = \perp$, the APIR integrity check has failed and Bob discards the answer.

(5) Bob stores $res = ID_\mathsf{A} \parallel \Delta_\mathsf{A}$ together with idx_A as Alice's information for later use. Based on ID_A, he decides if he wants to communicate with Alice. If so, he uses Δ_A to set up his side of the communication channel with Alice in the ACN. Otherwise, he does nothing.

Like in the sender retrieval phase, all receivers must make R_recv queries to hide their popularity, i.e., how many clients want to communicate with them. If a receiver receives less than R_recv notifications, they make cover queries to their own index. If they receive more than R_recv notifications, they randomly select a subset of size R_recv to query and discard the rest. Receivers have no additional information about the senders except their index, based on which they could prioritize their selection, thus they choose randomly.

6.6 Receiver Notification

The goal of the receiver notification phase is to enable receivers to inform senders whether they have accepted the contact request. This phase is executed analogously to the sender notification phase, but with the roles of senders and receivers reversed. Receivers set their notification vector to 1 at all indices of senders they accept, and to 0 everywhere else (i.e., at indices from senders they do not want to communicate with, or who have not contacted them). The servers receive secret shares of this vector, which they use to build their share of the notification matrix. Senders obtain and combine their shares of the matrix to learn which receiver notified them. If a sender receives an affirmative notification from an intended communication partner, they use that receiver's previously stored contact information to set up their side of the communication channel in the ACN. If a sender receives a notification from an unknown index, they discard it. If a sender did not receive a notification from an intended communication partner, they do not learn why, i.e., if this receiver did not reply because they a) decided not to or b) received too many requests.

7 Security Analysis

This section proves that Sabot meets its security goals. First, §7.1 formalizes communication unobservability $C\bar{O}$ before § 7.2 provides a formal proof that Sabot_h achieves $C\bar{O}$. This section further provides proof sketches (as done in previous work) that Sabot_m provides the same metadata protection as Sabot_h but in a malicious setting, and that Sabot_h and Sabot_m ensure authenticity (§7.3).

7.1 Formalizing Communication Unobservability

We adapt Kuhn et al.'s notion of communication unobservability $C\bar{O}$ [23] for the bootstrapping setting. The notion $C\bar{O}$ is modeled as an indistinguishability game $\mathcal{G}_{C\bar{O}}$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. At a high level, the adversary chooses two sets of protocol inputs, from which the challenger randomly selects one as input to the protocol for which privacy should be proven. Based on protocol observation, the adversary has to decide which set was selected.

In the context of the $C\bar{O}$ game, the considered bootstrapping protocol is simplified into a protocol model Π. The model takes as input client-identifier tuples, where each client should (attempt to) bootstrap a communication channel with the owner of the identifier during the observation period. The model outputs all information that an implementation of the protocol would disclose on the same inputs to the considered adversary.

Formally, we define $\mathcal{G}_{C\bar{O}}$ as follows:

Definition 3 (Privacy Game $\mathcal{G}_{C\bar{O}}$)**.** Let $\mathcal{A}$ be the adversary, $\mathcal{C}$ be the challenger, and Π the protocol model to be analyzed. Let $U_h \subseteq U$ be the set of honest participating clients, and $\mathcal{I}$ the space of all possible client identifiers not controlled by the adversary. We denote $Rel = U_h \times \mathcal{I}$ as the set of all possible bootstrapping inputs. $\mathcal{G}_{C\bar{O}}$ proceeds as follows:

(1) $\mathcal{C}$ samples a challenge bit $b \leftarrow\!\!\$\ \{0, 1\}$ uniformly at random.

(2) $\mathcal{A}$ submits a challenge $(c_0 \subseteq Rel, c_1 \subseteq Rel)$ to $\mathcal{C}$.

(3) $\mathcal{C}$ *executes* Π with input c_b. Any protocol output $\Pi(c_b)$ is forwarded to $\mathcal{A}$.

(4) Based on the received protocol output, $\mathcal{A}$ generates a guess $b' \in \{0, 1\}$.

$\mathcal{A}$ wins $\mathcal{G}_{C\bar{O}}$ if $b = b'$ and loses otherwise. Steps 2 – 3 can be repeated poly(λ) times to allow $\mathcal{A}$ to adapt its behavior based on the protocol output.

Definition 4 (Communication Unobservability ($C\bar{O}$))**.** A protocol Π achieves communication unobservability ($C\bar{O}$) if there exists no efficient polynomial-time adversary $\mathcal{A}$ who can win the game $\mathcal{G}_{C\bar{O}}$ with a non-negligible advantage in the security parameter λ over

random guessing:

$$Pr\left[b = b' \quad : \quad \begin{array}{c} b \leftarrow\$ \{0,1\} \\ \mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda) \\ c_0, c_1 \subseteq Rel \leftarrow \mathcal{A}(\mathsf{pp}) \\ b' \leftarrow \mathcal{A}(\Pi(c_b)) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

7.2 Proof of Privacy

We prove that $\textsc{Sabot}_h$ achieves communication unobservability with a series of hybrid games. In the first hybrid, $\mathcal{G}_{C\bar{O}}$ is played with $\textsc{Sabot}_h$ as the protocol model $\Pi_{\textsc{Sabot}}$ as described in §6. In subsequent games, the protocol mechanisms are replaced with random behavior until, in the last game, the protocol execution no longer depends on the intended communications. We show that no distinguisher $\mathcal{D}$ can distinguish any subsequent hybrids based on the adversary's output in $\mathcal{G}_{C\bar{O}}$ with a non-negligible advantage, which implies that no adversary can break $C\bar{O}$ with a non-negligible advantage. With this approach, $\textsc{Sabot}_h$ is the first anonymous bootstrapping protocol with a full formal proof of privacy.

Further, combining all adversarial capabilities (GPA *and* honest-but-curious servers) ensures that communication unobservability is achieved even against the strongest possible adversary within our assumptions. Furthermore, we prove the privacy of $\textsc{Sabot}_m$ informally (note that previous work [22, 27] also provide only informal security analyses in the fully malicious setting).

Theorem 1 (Honest-But-Curious Privacy)**.** $\textsc{Sabot}_h$ achieves communication unobservability against an adversary $\mathcal{A}$ who can passively observe traffic on every network link (GPA) and controls a single honest-but-curious server assuming that XSS achieves secrecy, a TLS-like channel (for all client-server communication) that is length regular IND$-CPA [32] secure, and PIR achieves query privacy as defined in Def. 6 in §A.1.

Proof. We construct the following series of hybrid games:

- H_0. $\mathcal{G}_{C\bar{O}}$ is played with Sabot as described in §6.
- H_1. Like H_0, but senders retrieve random identifiers from the identifier space $\mathcal{I}$.
- H_2. Like H_1, but senders set their notification vector $\mathbf{v} = \mathbf{0}$.
- H_3. Like H_2, but receivers retrieve random indices from $[N]$.
- H_4. Like H_3, but receivers set their notification vector $\mathbf{v} = \mathbf{0}$.

We show that no efficient distinguisher $\mathcal{D}$ exists, which can, based on $\mathcal{A}$'s view, distinguish between hybrid games H_i and H_{i+1} for $i \in \{0, \dots, 3\}$ with a non-negligible advantage over random guessing.

$H_0 \approx H_1$. H_0 and H_1 only differ in the identifiers that honest clients input into PIR_K.Query during the sender retrieval phase: In H_0, honest clients input R_{send} identifiers specified by $\mathcal{A}$ in the $\mathcal{G}_{C\bar{O}}$ challenge input c_b. If $|c_b| < R_{\text{send}}$, honest clients additionally input their own identifier to ensure exactly R_{send} identifiers are used. In H_1, honest clients input R_{send} random identifiers. During sender retrieval $\mathcal{A}$ can observer the following information:

- Through the honest-but-curious server, $\mathcal{A}$ can observe a single query q_i from each output of PIR_K.Query.
- Through its GPA capability, $\mathcal{A}$ can observe ciphertexts on the network. This includes a) ciphertexts of the output of PIR_K.Query that honest clients send to all servers and b) ciphertexts of the output of PIR_K.Answer that each server sends to the corresponding honest client.

Next, we argue that none of this observable information differs between H_0 and H_1 in a way that $\mathcal{D}$ can detect.

First, the ciphertexts, both from client to server and from server to client, do not differ in number or size between H_0 and H_1: Every honest client executes PIR_K.Query R_{send} times and thus sends R_{send} ciphertexts to each of the k servers. As PIR_K queries have a fixed size, these ciphertexts are also of identical size in H_0 and H_1. For each query each server receives, they send back a PIR_K answer of fixed size.

Second, all ciphertexts are encrypted using a length-regular IND$-CPA-secure encryption scheme. $\mathcal{A}$ only knows the key material of a single honest-but-curious server, not that of the other servers or honest clients. Thus, following from length regular IND$-CPA-security, $\mathcal{A}$ cannot distinguish ciphertexts between honest clients and servers not controlled by $\mathcal{A}$ from random data with a non-negligible advantage over random guessing. Since $\mathcal{A}$ cannot distinguish ciphertexts from random data in H_0 and H_1, it follows that ciphertext observation provides the same information to $\mathcal{D}$ in both H_0 and H_1.

Third, it follows from PIR_K's query privacy (Def. 6 in §A.1) that $\mathcal{D}$ cannot distinguish between H_0 and H_1 with a non-negligible advantage over random guessing based on the single query $\mathcal{A}$ receives of each query vector: Query privacy ensures that, for any keyword, a *simulator* can output an a subset of queries (independent of the keyword) that is computationally indistinguishable from the output of PIR_K.Query for that keyword. As query privacy holds for each query in both H_0 and H_1, it follows that the distribution that queries follow in H_0 and H_1 are both computationally indistinguishable from the same simulator's output and thus also computationally indistinguishable from each other. Thus, $\mathcal{D}$ has only a negligible advantage over random guessing in distinguishing between H_0 and H_1 based on the observation of these queries.

$H_1 \approx H_2$. The only difference between H_1 and H_2 is in the notification vector that honest clients input into XSS.Share during the sender notification phase: In H_1, honest clients set the vector to "1" in the indices derived in the sender retrieval phase, whereas they set the vector to "0" at every index in H_2. Consider the information that $\mathcal{A}$ is able to observe during the sender retrieval phase:

- Through its GPA capability, $\mathcal{A}$ can observe the ciphertexts containing the output of XSS.Share that honest clients send to all servers.
- Through the honest-but-curious server, $\mathcal{A}$ can observe a single secret share $\mathbf{v}^i$ from each output of XSS.Share.
- Through its GPA capability, $\mathcal{A}$ can observe ciphertexts containing the notification vector shares that are sent to the honest clients.

Next, we argue that none of this observable information differs between H_1 and H_2 in a way that $\mathcal{D}$ can detect. For the ciphertexts exchanged between honest clients and servers not controlled by $\mathcal{A}$, we can argue analogously to "$H_0 \approx H_1$": All notification vectors in both H_1 and H_2 are N bits in size; thus, the resulting ciphertexts are all identical in size. Both in H_1 and H_2, every honest client sends and receives exactly one notification vector share per server per

sender notification phase. As $\mathcal{A}$ does not know the key material of honest clients or other servers, and ciphertexts are encrypted using an encryption scheme that is assumed to be IND$-CPA-secure, $\mathcal{D}$ cannot distinguish these ciphertexts from random data with a non-negligible advantage over random guessing in both H_1 and H_2.

Concerning the single secret share that $\mathcal{A}$ can observe from each notification vector, we have to distinguish two cases: In the first case, $\mathcal{A}$ controls any one of the servers $S_1, \ldots, S_{k-1}$. In this case, the share $\mathbf{v}_i$ is drawn uniformly at random by definition of XSS.Share in both H_1 and H_2 and $\mathcal{A}$ can gain no information from it in either hybrid. In the second case, $\mathcal{A}$ controls the "last" server S_k. In this case, the share $\mathbf{v}_i$ is the XOR of all other corresponding shares. As all other corresponding shares are drawn uniformly at random and not known to $\mathcal{A}$, the XOR of them is indistinguishable from randomness by $\mathcal{A}$ in both H_1 and H_2.

We have shown that all information that can be observed by $\mathcal{A}$ that could potentially differ between H_1 and H_2 cannot be distinguished from randomness by $\mathcal{D}$ with a non-negligible advantage over random guessing.

$H_2 \approx H_3$. In H_2, honest clients input R_{recv} of the indices obtained from the notification vector of the previous sender notification phase. If a client obtains less than R_{recv} indices, they additionally input their own index to ensure exactly R_{recv} inputs are used. In H_3, honest clients input R_{recv} random indices. It follows analogously to "$H_0 \approx H_1$" that $H_2 \approx H_3$, with the reduction to PIR_I's query privacy rather than PIR_K's.

$H_3 \approx H_4$. H_3 and H_4 only differ in the notification vector that honest clients input into XSS.Share during the receiver notification phase: In H_3, honest clients set the vector to "1" at the indices derived in the receiver retrieval phase, whereas they set the vector to "0" at every index in H_4. As sender notification and receiver notification consist of identical protocol steps, we can argue analogously to "$H_1 \approx H_2$" that $H_3 \approx H_4$.

$\varepsilon_4 \leq \mathsf{negl}(\lambda)$. We now show that $\mathcal{A}$'s advantage in hybrid H_4 over random guessing is at most negligible, i.e., $\varepsilon_4 \leq \mathsf{negl}(\lambda)$. First note that the set U_h from which the adversary may choose challenge clients is, by definition of $\mathcal{G}_{C\bar{O}}$, identical in both challenge inputs. In H_4, the behavior of all clients in U_h during protocol execution is fully independent of $\mathcal{A}$'s input:

- During *Registration*, all clients in U_h register at the servers.
- *Setup* is based on the data provided during registration and occurs internally to the servers.
- During *Sender Retrieval*, all clients in U_h retrieve the contact information of R_{send} random identifiers in $\mathcal{I}$.
- During *Sender Notification*, all clients in U_h send a fixed $\mathbf{0}$ notification vector.
- During *Receiver Retrieval*, all clients in U_h retrieve the contact information of R_{recv} random indices.
- During *Receiver Notification*, all clients in U_h send a fixed 0 notification vector.

The behavior of servers that are not controlled by $\mathcal{A}$ is also independent of adversary input, as it fully depends on the client input: Servers compute the answers to the clients' PIR_K and PIR_I queries and distribute the notification vector shares. Thus, $\mathcal{A}$ cannot have a non-negligible advantage in winning $\mathcal{G}_{C\bar{O}}$ in H_4, i.e., $\varepsilon_4 \leq \mathsf{negl}(\lambda)$.

Deduction. We have shown that $H_0 \approx H_1 \approx H_2 \approx H_3 \approx H_4$ and that in H_4, $\mathcal{A}$ has at most a negligible advantage over random guessing. Thus, $\textsc{Sabot}_h$ achieves communication unobservability against an adversary $\mathcal{A}$ who can passively observe traffic on every network link and controls a single honest-but-curious server, assuming that XSS achieves secrecy, encryption is IND$-CPA-secure, and KW-PIR and I-PIR both achieve privacy. □

Claim 1 (Malicious Privacy)**.** An adversary $\mathcal{A}'$ who can actively interfere with traffic on every network link, actively controls $k-1$ servers, and an arbitrary number of malicious clients gains no more information about honest client behavior from $\textsc{Sabot}_m$ than $\mathcal{A}$ from $\textsc{Sabot}_h$ assuming that XSS achieves secrecy, and both authenticated I-PIR and KW-PIR achieve privacy. Further, assume that the channel between clients and servers is TLS-like in that the channel is authenticated, provides message integrity, and an adversary cannot add, drop, replay, or modify messages without the receiver being able to detect this behavior in some way.

Proof (Sketch). Consider the additional abilities that the active adversary $\mathcal{A}'$ has over the passive adversary $\mathcal{A}$: $\mathcal{A}'$ can 1) actively interfere with traffic on every network link rather than just passively observe it, 2) control an arbitrary number of malicious clients, and 3) actively control $k-1$ servers rather than passively observe at a single server. We argue that changes from $\textsc{Sabot}_h$ to $\textsc{Sabot}_m$ ensure that $\mathcal{A}'$ cannot gain any more information about the behavior of honest clients in $\textsc{Sabot}_m$ than $\mathcal{A}$ in $\textsc{Sabot}_h$.

Active Traffic Interference. While $\textsc{Sabot}_h$ assumes channels between parties to be "just" IND$-CPA-secure, $\textsc{Sabot}_m$ assumes channels to be secure against active interference. The channel guarantees that $\mathcal{A}'$ cannot insert packets in the name of any honest client or server. The channel further guarantees that any modification of a packet is detected by its receiver. $\textsc{Sabot}_m$ requires that such packets are dropped by the receiving honest client or server. Thus, any active attack by $\mathcal{A}'$ on the channel between honest clients and servers either has no effect (insertion or replay) or leads to missing packets (modification or drop). Next, we argue that missing packets do not enable $\mathcal{A}'$ to gain any information about the behavior of honest clients.

If $\mathcal{A}'$ drops any packet during registration, the affected client will not be registered and cannot participate in further protocol execution. If any packet containing the public parameters in the setup phase is dropped, the affected client also will not participate in further protocol execution. Dropping any PIR query or answer leads to the affected client not learning the requested contact information. It does not enable $\mathcal{A}'$ to gain information about the content of the query, as the client detects the drop and refuses further participation independent of the query's content. The same argument holds for the dropping of notification vector shares.

Malicious Clients. In $\textsc{Sabot}_m$, the remaining honest-but-curious server ensures that malicious clients do not gain information about the relations between honest clients: Communication with this server occurs over an authenticated channel, so malicious clients cannot pose as other clients. Thus, the honest-but-curious server will not provide a malicious client with other clients' PIR answers or notification vector shares. Without this information from the honest-but-curious server, any information the malicious client

might gain from the malicious servers is indistinguishable from random.

Malicious Servers. As in Sabot$_h$, during the notification phases in Sabot$_m$, k-of-k secret sharing ensures that even $k-1$ colluding servers cannot gain information about the honest clients' notification vectors. Compared to Sabot$_h$, Sabot$_m$ introduces two changes that allow the detection of malicious server behavior: First, registration and setup are executed by all servers in parallel based on a common random seed. If malicious servers deviate from the protocol, the derived public parameters will not match the honest-but-curious server's public parameters, and honest clients will abort. Second, Sabot$_m$ uses APIR in the retrieval phases, which lets clients verify the integrity of the retrieved information. The verification fails and the client aborts if a server does not answer the PIR query correctly.

Finally, malicious servers can modify the notification matrix and thus provide incorrect notification vectors to clients. Clients cannot detect such malicious behavior and might, therefore, miss notifications or receive false notifications. Due to the use of secret sharing and our anytrust assumption, no targeted attack on privacy can be run by malicious servers. The servers further have no way to infer information about the correct and incorrect notifications.

7.3 Proof of Authenticity

Next, we show that Sabot achieves authenticity in both the honest-but-curious setting (Claim 2) and the malicious setting (Claim 3).

Clients provide their contact information to the servers during registration; the servers use it to set up the contact database DB. In the honest-but-curious setting, we can assume that the servers honestly set up the contact database and answer PIR queries on the correct information. In the malicious setting, this assumption does not hold, and servers could provide incorrect contact information to the clients. Thus, we require parallel setup and registration phases at all servers as well as the use of APIR instead of regular PIR.

Claim 2 (Honest-But-Curious Authenticity)**.** Sabot$_h$ achieves authenticity in the honest-but-curious setting assuming PIR correctness.

Proof (Sketch). Authenticity is broken if the contact information a client receives does not match the one it requested. The only way for a client to receive contact information is from the servers during the sender or receiver retrieval phases. In the honest-but-curious setting, authenticity follows directly from correctness.

Claim 3 (Malicious Authenticity)**.** Sabot achieves authenticity in the malicious setting assuming APIR integrity.

Proof (Sketch). Authenticity in the malicious setting can be shown analogously to the honest-but-curious setting, except that we need to consider malicious servers actively deviating from the protocol. Servers can deviate from the protocol in all phases, but they can only break authenticity during setup and retrieval. During notification, malicious servers can only attack availability, which we consider out of scope for this work.

If malicious servers deviate from the protocol during setup, they will send different public parameters than the remaining honest-but-curious server. The client is able to detect this and will abort the protocol. If malicious servers deviate from the protocol during contact information retrieval, the client receives incorrect information. Using APIR to retrieve the records, the client is able to verify the integrity of the records. If the verification fails, the client aborts. For authenticated KW-PIR, we require an injective mapping function that allows the client to ensure a correct mapping from keyword to index. Using APIR for the following retrievals ensures that the client can verify integrity. Malicious server behavior that breaks authenticity will thus always be detected by the client.

8 Evaluation

In this section, we evaluate the performance of Sabot empirically and compare it to related work, namely Alpenhorn [27] and Pudding [22]. We aim to answer the following two main questions:

(1) How much communication overhead does bootstrapping impose on clients? (§ 8.2)
(2) How much computation is needed from clients and servers to enable bootstrapping? (§ 8.3)

We implemented our prototype in Go, utilizing and adapting implementations of the PIR[3, 4, 5] and data store[6] building blocks for our protocol. The benchmarking environment is containerized to allow a simplified execution of our prototype and reproducibility of our results. Our prototype is available at https://github.com/laurahetz/sabot.

8.1 Experimental Setup

We evaluate our protocol with $k = 2$ servers and one client, all executed on a single physical server with 512 GB of DDR4 memory and an AMD EPYC 7742 CPU @2.25 GHz with 64 cores. Sabot can be scaled to a larger number of servers to weaken the trust requirements, i.e., $k-1$ servers can be malicious. We note that the overhead per server remains constant, thus the client's computational overhead increases linearly with the number of servers. Furthermore, the selection of efficient (A)PIR schemes with $k > 2$ servers is limited.

To simulate the computational server overhead of N clients, the servers execute all phases N times in parallel on 50 threads. All measurements are averaged over 16 runs, except where stated otherwise. In all experiments, we assume 32 B long identifiers and 32 B of contact information per client. For Sabot and related work, we set the number of registered clients to $N \in \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\}$. Finally, we state Sabot's communication and computational overhead for retrieval rates of $R_{\text{send}} = R_{\text{recv}} \in \{1, 5, 10\}$. We consider 1–10 retrievals to be a reasonable range for evaluation, as we expect real clients to require bootstrapping relatively infrequently; in a literature survey on the structure of personal social networks, Lubbers et al. [28] found a maximum of three new acquaintances per day.

8.2 Communication Overhead

In this section, we evaluate Sabot's communication overhead and compare it to related work. We assess the impact of the total number of clients N on the communication cost of one client bootstrapping

[3] https://github.com/dedis/apir-code
[4] https://github.com/dkales/dpf-go
[5] https://github.com/dimakogan/checklist
[6] https://github.com/FastFilter/xorfilter

	Number of clients				
	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
Sabot$_h$ (1)	2.67 KiB	6.71 KiB	22.00 KiB	82.28 KiB	322.57 KiB
Sabot$_h$ (5)	1.51 KiB	2.52 KiB	5.78 KiB	18.03 KiB	66.29 KiB
Sabot$_h$ (10)	1.35 KiB	1.98 KiB	3.73 KiB	9.97 KiB	34.22 KiB
Sabot$_m$ (1)	5.44 KiB	9.98 KiB	25.76 KiB	86.54 KiB	327.34 KiB
Sabot$_m$ (5)	4.01 KiB	5.48 KiB	9.18 KiB	21.89 KiB	70.59 KiB
Sabot$_m$ (10)	3.79 KiB	4.86 KiB	7.04 KiB	13.73 KiB	38.41 KiB
Alpenhorn	7.08 MiB	7.17 MiB	7.53 MiB	8.97 MiB	14.10 MiB
Pudding w/o Nym	125.57 KiB	125.57 KiB	125.57 KiB	125.57 KiB	125.57 KiB
Pudding w/ Nym	6.99 MiB	6.99 MiB	6.99 MiB	6.99 MiB	6.99 MiB

Table 2: Bandwidth overhead per client per bootstrapped communication in Sabot and related work. Numbers in brackets behind Sabot denote the number of retrievals per round over which the cost is amortized.

one communication channel. For Sabot, we expect communication cost to scale linearly with the number of clients N, as the exchanged notification vectors are in $\{0,1\}^N$. In general, Sabot$_m$ will require more bandwidth than Sabot$_h$, as its authentication requires the client to retrieve additional information using APIR. Both of Sabot's variants are expected to require significantly less bandwidth than related work, as they do not require cover traffic from a mix network. In both Sabot$_m$ and Sabot$_h$, we measure the amount of data a single client sends and receives during one sequence of *sender retrieval*, *sender notification*, *receiver retrieval*, and *receiver notification*. We state the cost per bootstrapped communication for retrieval rates $R_{\text{recv}} = R_{\text{send}} \in \{1, 5, 10\}$.

For Pudding, we take measurements of its benchmarking implementation[7]. We note that the cover traffic generated by Pudding's underlying mix network Nym is not exclusive to Pudding, but can also protect the client's possible other communications through Nym. Thus, we evaluate Pudding in two configurations: once including the Nym cover traffic (as a worst-case measurement without any parallel use of Nym) and once without the cover traffic (as a best-case measurement). In both configurations, we use seven discovery servers and clients bootstrap a new communication channel every 30 seconds, as suggested by the authors.

For Alpenhorn, we calculate the network overhead based on the protocol parameters given in [27, Sec. 8.1]. Alpenhorn assigns each client a shared mailbox, which contains invites of 308 B each. To achieve the protocol's differential privacy guarantees, each mailbox contains an average of 12 000 cover invites per round in addition to up to 12 000 real invites. Alpenhorn's evaluation [27, Sec. 8.1] assumes that 5 % of clients send real messages each round; Alpenhorn increases the number of mailboxes once a mailbox fills up. We note that bidirectional bootstrapping in Alpenhorn requires mailbox downloads in two subsequent rounds. Thus, we compute Alpenhorn's communication overhead as

$$2 \cdot (12\,000 + \min(12\,000, N \cdot 0.05)) \cdot 308\,\text{B}.$$

Table 2 presents our measurements for Sabot, Alpenhorn, and Pudding. As expected, communication overhead for Sabot increases linearly with the number of clients, and Sabot$_m$ is $1.47 - 2.91\times$ times more expensive than Sabot$_h$. Overall, our schemes achieve a reduction in bandwidth by $151 - 6\,042\times$ compared to Pudding and Alpenhorn. For almost all parameter combinations, Sabot is even cheaper than Pudding without Nym cover traffic (and thus without privacy). However, Sabot is more expensive than Pudding without privacy for retrieval rates of 1 as well as 2^{18} clients. Sabot's communication overhead per bootstrapping decreases as the number of retrievals per round increases. This behavior can be explained by investigating the contribution of different protocol steps to the network overhead. Concretely, for 2^{18} clients and 1 retrieval, PIR accounts for 2.68 KiB of bandwidth, whereas the notification-related steps require 288.06 KiB. The PIR costs are linear in the retrieval rate, while the cost of notification is independent of this rate.

One aspect to note in Tab. 2 is that Alpenhorn's communication overhead scales linearly with the number of clients. This is due to the fact that for $N < 2^{18}$ clients, all clients share a single mailbox. For more clients, Alpenhorn increases the number of mailboxes in the system and communication overhead becomes independent of the number of clients.

Our evaluation shows that Sabot meets the requirement of low client communication cost for small to medium-sized deployments, especially compared to related work: For $N = 2^{14}$ clients, Sabot$_m$ requires an order of magnitude less communication than related work. Sabot$_h$ decreases the overhead by 2×. The overhead of both Alpenhorn and Pudding is independent of the number of clients, which is an advantage of their designs. However, they fail to hide client popularity, which is a feature of our design. We estimate that Pudding becomes more communication-efficient than Sabot at $N = 2^{21}$ and Alpenhorn at $N = 2^{22}$, making them better suited for larger deployments.

8.3 Computational Overhead

In this section, we investigate the computational cost of Sabot for clients and servers. We measure the duration of one bootstrapping round for differing numbers of clients N.

Each protocol phase described in §6, is measured independently to determine its computational cost. Note that the notification phase is split into two stages: In Notify, the client generates the notification vectors and sends them to the servers, and the servers write them to the matrix. In GetNotify, the server reads a client's row from the matrix and sends it to the client. In our prototype, the client initiates

[7] https://github.com/ckocaogullar/pudding-protocol

each phase. We measure the time between the client initiating the step and them completing the processing of the server's response. For the sender and receiver notification phases, the client does not have to process a response but waits for the servers to indicate the completion of the notification matrix. We evaluate both Sabot_h and Sabot_m with retrieval rates $R_{\text{send}} = R_{\text{recv}} \in \{1, 5, 10\}$.

In general, we expect Sabot's computation time to increase with the number of clients. Based on PIR's asymptotic complexity, we expect the sender and receiver retrieval phases to scale linearly with the number of clients. We further expect KW-PIR to be more time-intensive than I-PIR, due to the additional keyword-mapping and the need for multiple index queries per keyword. The notification phases are expected to scale quadratically in the number of clients, as the notification matrix contains an entry for every possible pair of clients. We expect the retrieval phases to scale linearly with the retrieval rates, as each retrieval is independently computed and has the same computational overhead for clients and servers. Furthermore, we expect the retrieval phases of Sabot_m to be slower than those of Sabot_h, as Sabot_m has additional integrity checks.

Table 3 presents our measurements and generally confirms our expectations: We observe that the notification phases indeed scale quadratically. We can see that the S-Notify and S-GetNotify phases show an asymmetry in execution time, despite handling the same amount of data. The same holds for the R-Notify and R-GetNotify phases. This asymmetry is caused by the more expensive write operation compared to the cheaper read operation. Computation during notification does not depend on the retrieval rates, nor does it differ between Sabot_h and Sabot_m. Hence, it is given as an average for both protocol executions and all retrieval rates.

Regarding the retrieval phases, the results confirm our expectations in three aspects: First, starting with 2^{14} clients for S-Retrieval and 2^{16} clients for R-Retrieval, we can clearly see that the computation time for the retrieval phases scales linearly with the retrieval rate. For both S-Retrieval and R-Retrieval, and for both Sabot_h and Sabot_m, a retrieval rate of 5 increases computation time by a factor of 5 compared to a single retrieval, and similarly for a rate of 10. Below 2^{14} or 2^{16} clients respectively, this correlation is obscured by run-to-run variance. Second, starting at 2^{14} clients, retrieval times are consistently higher for Sabot_m than for Sabot_h. This is due to the additional computational overhead of APIR compared to standard PIR. As (A)PIR computation generally scales with respect to the database size, the difference in runtime between Sabot_h and Sabot_m increases with the number of clients; For 2^{14} clients, S-Retrieval in Sabot_m is slower than in Sabot_h by a factor of 3.8, whereas the factor increases to 9.2 for 2^{16} clients. Third, starting with 2^{14} clients, S-Retrieval is consistently slower than R-Retrieval for both Sabot_h and Sabot_m. This is due to the additional overhead introduced by the keyword-to-index mapping to the S-Retrieval phase compared to the R-Retrieval phase.

Within the range of our evaluation, the retrieval phases are the most expensive steps, especially for more than one retrieval. However, due to the quadratic complexity of notification, we expect the notification phases to dominate computational overhead for larger deployments.

We concede that the quadratic growth in computation time for Sabot hinders scaling to large deployments. However, even for 2^{18} clients, latency is at worst around 66 min for a single retrieval per round. For higher retrieval rates, latency increases sublinear in the retrieval rate, as the notification phases scale independently of the retrieval rate. Alpenhorn evaluates their bandwidth cost based on round duration of up to 24 h [27, Fig. 6]. Thus, we can conclude that Sabot's latency is reasonable and well-suited for deployments of the expected size.

9 Discussion

Scalability. As we have evaluated in §8, Sabot's computational overhead for servers scales quadratically in the number of protocol participants. While this clearly is an obstacle to large-scale adoption, we do not see it as a fatal flaw. First, quadratic overhead in the notification phase is inherent if client popularity should be fully hidden from the servers; Only if *every* client receives enough information to be potentially notified by all other clients does the server not learn how many clients actually notify any given client. Second, the quadratic overhead only concerns servers. For clients, the overhead is linear in the number of protocol participants.

Membership Disclosure. In Sabot and Alpenhorn[8], malicious clients can learn if a given identity is registered in the system. This is a drawback compared to Pudding, where they need to collude with a server to gain the same knowledge. However, Pudding's ability to hide this information from clients comes at the cost of disclosing client popularity to the servers: In Pudding, clients request the target's contact information from the servers, but should not learn if these target clients are registered in the system. To achieve this efficiently, servers must know the targets in order to provide fake contact information for non-registered clients. This allows the servers to learn the popularity of all clients.

Active Maliciousness. Sabot is the first anonymous bootstrapping protocol for which a formal proof of privacy exists. However, the proof only covers privacy against passive adversaries in Sabot_h. For privacy against active adversaries in Sabot_m, we provide a sketch-level proof, which is in line with related work. Providing a formal proof of privacy against actively malicious adversaries is an interesting direction of future work; this requires the extension of the communication unobservability game to cover such adversaries. This extension would be of value in its own right.

Fixed Rates. Sabot requires all clients to make a fixed number of PIR queries for both sending and receiving invitations to hide popularity. Setting these parameters is not trivial: If the required number of queries is too high, unnecessary overhead is introduced. If the required number of queries is too low, some clients cannot send/receive all their invitations in a single round.

Instead of setting these parameters during protocol setup, an adaptive round-based approach, based on actual utilization, could be employed. In each round and prior to making the queries in the sender and receiver retrieval phase, clients can disclose the number of invitations they want to send/have received via a private write [12, 19]. After all clients have privately disclosed their value, the maximum is set as the respective rate. This extension allows

[8] Alpenhorn does not explicitly target membership privacy, and its current implementation does not achieve it, as it returns a descriptive error message to the client when they try to register an identifier that already exists.

		Rate	S-Retrieval	S-Notify	R-GetNotify	R-Retrieval	R-Notify	S-GetNotify	**Total**
2^{10}	SABOT_h	1	0.07 s	0.05 s	0.07 s	0.07 s	0.08 s	0.07 s	0.27 s
	SABOT_m	1	0.08 s			0.07 s			0.28 s
	SABOT_h	5	0.09 s			0.08 s			0.30 s
	SABOT_m	5	0.12 s			0.08 s			0.33 s
	SABOT_h	10	0.12 s			0.08 s			0.33 s
	SABOT_m	10	0.23 s			0.08 s			0.44 s
2^{12}	SABOT_h	1	0.31 s	0.24 s	0.30 s	0.29 s	0.31 s	0.29 s	1.15 s
	SABOT_m	1	0.32 s			0.30 s			1.17 s
	SABOT_h	5	0.50 s			0.32 s			1.36 s
	SABOT_m	5	1.22 s			0.33 s			2.09 s
	SABOT_h	10	0.95 s			0.33 s			1.82 s
	SABOT_m	10	2.40 s			0.33 s			3.27 s
2^{14}	SABOT_h	1	1.37 s	1.20 s	1.18 s	1.17 s	1.21 s	1.19 s	4.95 s
	SABOT_m	1	3.39 s			1.22 s			7.02 s
	SABOT_h	5	6.32 s			1.38 s			10.11 s
	SABOT_m	5	17.27 s			1.54 s			21.22 s
	SABOT_h	10	12.59 s			1.71 s			16.71 s
	SABOT_m	10	34.25 s			2.27 s			38.93 s
2^{16}	SABOT_h	1	18.41 s	6.71 s	4.73 s	5.33 s	6.57 s	4.73 s	37.02 s
	SABOT_m	1	70.10 s			13.08 s			96.45 s
	SABOT_h	5	95.68 s			15.48 s			124.44 s
	SABOT_m	5	336.71 s			37.50 s			387.48 s
	SABOT_h	10	191.16 s			25.71 s			230.15 s
	SABOT_m	10	653.19 s			63.02 s			729.49 s
2^{18}	SABOT_h	1	332.01 s	116.78 s	19.40 s	91.13 s	116.41 s	19.19 s	656.32 s
	SABOT_m	1	3 076.44 s			693.28 s			4 002.90 s
	SABOT_h	5	1 701.27 s			273.77 s			2 208.22 s
	SABOT_m	5	14 212.53 s			1 759.75 s			16 205.46 s
	SABOT_h	10	3 382.56 s			449.19 s			4 064.93 s
	SABOT_m	10	26 788.09 s			3 127.93 s			30 149.20 s

Table 3: Latency measurements by protocol phase in SABOT_h and SABOT_m for differing numbers of clients. The computation in the notification steps neither depends on the retrieval rate nor does it differ between SABOT_h and SABOT_m. It is thus given as an average for both protocol executions and all retrieval rates.

for the setting of optimal rates for each round. However, it comes with a slight privacy drawback: While the private write operation hides the writers, it does not hide which values are written. An adversary is, therefore, able to learn the *popularity histogram*. Since the distribution of popularity is likely to follow Zipf's law [8], the amount of information an adversary can obtain from it is limited.

10 Conclusion

In this paper, we introduced SABOT, a protocol for anonymous bootstrapping. SABOT has quadratic communication complexity, but this is tamed for moderate numbers of clients by having very small concrete constants. SABOT comes in two flavors, which present a trade-off between performance and trust in the servers. We compared SABOT to state-of-the-art alternatives (Alpenhorn, Pudding), showing that it reduces bandwidth overhead by an order of magnitude. We also provide a full formal proof of privacy for SABOT_h in the honest-but-curious setting, which makes it the first anonymous bootstrapping protocol with proven privacy guarantees. Providing a formal proof of privacy in the actively malicious setting remains future work. Overall, SABOT is intuitive, simple to analyze, and offers excellent performance at the scale of today's ACNs.

Acknowledgments

We thank Sebastian Angel, Alexander Linder, Daniel Schadt, and Kien Tuong Truong for the interesting discussions and valuable input. This work has in part been funded by the Helmholtz Association through the KASTEL Security Research Labs (HGF Topic 46.23) and by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden.

References

[1] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous Messaging via Secure Multiparty Computation. In *USENIX Security Symposium*. USENIX Association, 1217–1234.

[2] Sebastian Angel, Sampath Kannan, and Zachary B. Ratliff. 2020. Private resource allocators and their applications. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 372–391.

[3] Sebastian Angel and Srinath T. V. Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *OSDI*. USENIX Association, 551–569.

[4] Ludovic Barman, Moshe Kol, David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2022. Groove: Flexible Metadata-Private Messaging. In *OSDI*. USENIX Association, 735–750.

[5] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. 2022. Verifiable Private Information Retrieval. In *TCC (3) (Lecture Notes in Computer Science, Vol. 13749)*. Springer, 3–32.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *EUROCRYPT (2) (Lecture Notes in Computer Science, Vol. 9057)*. Springer, 337–367.
[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *CCS*. ACM, 1292–1303.
[8] Michael P Cameron. 2022. *Zipf's Law across social media.* Working Papers in Economics. University of Waikato. https://ideas.repec.org/p/wai/econwp/22-07.html
[9] Sofía Celi and Alex Davidson. 2024. Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries. In *CCS*. ACM, 4107–4121.
[10] Christoph Coijanovic, Christiane Weis, and Thorsten Strufe. 2023. Panini - Anonymous Anycast and an Instantiation. In *ESORICS (2) (Lecture Notes in Computer Science, Vol. 14345)*. Springer, 193–211.
[11] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. 2023. Authenticated Private Information Retrieval. In *USENIX Security Symposium*. USENIX Association, 3835–3851.
[12] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 321–338.
[13] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: Accountable Anonymous Group Messaging. In *CCS*. ACM, 340–350.
[14] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *22nd USENIX Security Symposium (USENIX Security 13)*. 147–162.
[15] Leo de Castro and Keewoo Lee. 2024. VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers. In *USENIX Security Symposium*. USENIX Association.
[16] Marian Dietz and Stefano Tessaro. 2024. Fully Malicious Authenticated PIR. In *CRYPTO (9) (Lecture Notes in Computer Science, Vol. 14928)*. Springer, 113–147.
[17] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*. USENIX, 303–320.
[18] Saba Eskandarian and Dan Boneh. 2022. Clarion: Anonymous Communication from Multiparty Shuffling Protocols. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society.
[19] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. 2021. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. In *USENIX Security Symposium*. USENIX Association, 1775–1792.
[20] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 8441)*. Springer, 640–658.
[21] Thomas Mueller Graf and Daniel Lemire. 2022. Binary Fuse Filters: Fast and Smaller Than Xor Filters. *ACM J. Exp. Algorithmics* 27 (2022), 1.5:1–1.5:15.
[22] Ceren Kocaoğullar, Daniel Hugenroth, Martin Kleppmann, and Alastair R Beresford. 2024. Pudding: Private User Discovery in Anonymity Networks. In *IEEE Symposium on Security and Privacy*. IEEE, 167–167.
[23] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard A. Jorswieck, and Thorsten Strufe. 2019. On Privacy Notions in Anonymous Communication. *Proc. Priv. Enhancing Technol.* 2019, 2 (2019), 105–125.
[24] Christiane Kuhn, Friederike Kitzing, and Thorsten Strufe. 2020. SoK on Performance Bounds in Anonymous Communication. *WPES@CCS* (2020).
[25] Albert Kwon, David Lu, and Srinivas Devadas. 2020. XRD: Scalable Messaging System with Cryptographic Privacy. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 759–776. https://www.usenix.org/conference/nsdi20/presentation/kwon
[26] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 711–725.
[27] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata. In *OSDI*. USENIX Association, 571–586.
[28] Miranda Jessica Lubbers, José Luis Molina, and Hugo Valenzuela-García. 2019. When networks speak volumes: Variation in the size of broader acquaintanceship networks. *Soc. Networks* 56 (2019), 55–69.
[29] Ralph C. Merkle. 1987. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO (Lecture Notes in Computer Science, Vol. 293)*. Springer, 369–378.
[30] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *USENIX Security Symposium*. USENIX Association, 1199–1216.
[31] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. doi:10.17487/RFC8446
[32] Phillip Rogaway. 2004. Nonce-Based Symmetric Encryption. In *FSE (Lecture Notes in Computer Science, Vol. 3017)*. Springer, 348–359.
[33] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. 2017. Stadium: A Distributed Metadata-Private Messaging System. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 423–440. doi:10.1145/3132747.3132783
[34] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1953–1970.
[35] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable Private messaging Rresistant to Traffic Analysis. (2015), 137–152.
[36] Yinghao Wang, Xuanming Liu, Jiawen Zhang, Jian Liu, and Xiaohu Yang. 2023. Crust: Verifiable and Efficient Private Information Retrieval With Sublinear Online Time. Cryptology ePrint Archive, Paper 2023/1607. https://ia.cr/2023/1607.

A Building Blocks

This section presents the definitions of the primitives Sabot builds upon, namely PIR and APIR (§A.1).

A.1 Private Information Retrieval

We present a definition for multi-server PIR in Def. 2 with informal descriptions of the desired properties. In this section, we provide full definitions for I-PIR and KW-PIR (considering the highlighted lines) in both a honest-but-curious and malicious setting. These definitions are adapted from [9, 11]. We refer to PIR with at most $k-1$ malicious servers as APIR.

Definition 5 (PIR Correctness)**.** A PIR scheme $\mathsf{PIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$, parameterized by the number of servers $k \in \mathbb{N}$ and database size $N \in \mathbb{N}$ satisfies correctness if for every $\mathbf{x} \in \mathcal{X}^N$ $\boxed{\mathbf{x} = \{(kw_i, x_i)\}_{i\in N} \in (\mathcal{K}\times\mathcal{X})^N}$, the following holds:

$$\Pr\left[\, x_i' = x_i \quad : \quad \begin{array}{r} (\mathsf{pp}, \mathsf{DB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x}) \\ (\mathsf{st}, \mathbf{q}) \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, i) \\ \boxed{(\mathsf{st}, \mathbf{q}) \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, kw_i)} \\ a_j \leftarrow \mathsf{PIR.Answer}(\mathsf{pp}, \mathsf{DB}, q_j) \forall j \in [k] \\ x_i' \leftarrow \mathsf{PIR.Reconstruct}(\mathsf{st}, \mathbf{a}) \end{array} \right]$$

where the probability is 1 for I-PIR and larger than $1 - \mathsf{negl}(\lambda)$ for KW-PIR. The probability is computed over all the random coins used by the schemes' algorithms.

Definition 6 (PIR Privacy (based on [9, 11]))**.** Let $\mathsf{PIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ be an unauthenticated PIR scheme for index-based keyword-based queries parameterized by a number of servers $k \in \mathbb{N}$, a set of database records $\mathbf{x} \in \mathcal{X}^N$ $\boxed{\mathbf{x} = \{(kw_i, x_i)\}_{i\in N} \in (\mathcal{K}\times\mathcal{X})^N}$ of size $N \in \mathbb{N}$. Let S be any subset of $k-1$ elements from $[k]$. For $\mathcal{I} \in [N]^T$ $\boxed{\mathcal{I} \in \mathcal{K}^T}$ and $T \in \mathsf{poly}(\lambda)$ let the distribution

$$\mathbf{REAL}_{\mathcal{I},T} = \left\{ \bigcup_{i\in S, t\in[T]} q_i^t \; : \; \begin{array}{r} (\mathsf{pp}, \mathsf{DB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x}) \\ \text{For all } t \in [T] \; : \\ (\mathsf{st}, (q_1^t, \ldots, q_k^t)) \\ \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, \mathcal{I}[t]) \end{array} \right\}.$$

Similarly, for a simulator $\mathcal{S}$, let the distribution

$$\mathbf{IDEAL}_{\mathcal{S},T} = \left\{ (q_i^t)_{i\in S, t\in[T]} \leftarrow \mathcal{S} \right\}.$$

An unauthenticated-PIR scheme PIR parametrized by a database size $N \in \mathbb{N}$ and a number of servers $k \in \mathbb{N}$ is secure if for every $\mathcal{I} \in [N]^T$ $\boxed{\mathcal{I} \in \mathcal{K}^T}$, the following holds:

$$\mathbf{REAL}_{\mathcal{J}} \approx_c \mathbf{IDEAL}_{\mathcal{S}}.$$

Definition 7 (APIR Privacy (based on [9, 1 1])). Let PIR = (Setup, Query, Answer, Reconstruct) be an authenticated PIR scheme for index-based keyword-based queries parameterized by a number of servers $k \in \mathbb{N}$, a set of database records $\mathbf{x} \in \mathcal{X}^N$ $\mathbf{x} = \{(kw_i, x_i)\}_{i\in N} \in (\mathcal{K} \times \mathcal{X})^N$ of size $N \in \mathbb{N}$. Let good $\in [k]$, and $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, define the distribution For $\mathcal{J} \in [N]^T$ $\mathcal{J} \in \mathcal{K}^T$ and $T \in \mathrm{poly}(\lambda)$ let the distribution

$$\mathbf{REAL}_{\mathcal{A},\mathrm{good},\mathcal{J},T,\lambda,\mathbf{x}} = \left\{ \hat{\beta} : \begin{array}{l} (\mathsf{pp}, \mathsf{DB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{seed}, \mathbf{x}) \\ \text{For all } t \in [T] : \\ \quad (\mathrm{st}, (q_1^t, \ldots, q_k^t)) \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, \mathcal{J}[t]) \\ \quad a_{\mathrm{good}}^t \leftarrow \mathsf{Answer}(\mathsf{pp}, \mathsf{DB}, q_{\mathrm{good}}^t) \\ (\mathrm{st}_{\mathcal{A}}, (a_i^t)_{i\neq \mathrm{good}}) \leftarrow \mathcal{A}_0(\mathbf{x}, (q_i^t)_{i\neq \mathrm{good}}) \\ y \leftarrow \mathsf{Reconstruct}(\mathrm{st}, (a_1, \ldots, a_k)) \\ b \leftarrow \mathbb{1}\{y = \perp\} \\ \text{if } b = \perp, \text{break} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathrm{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

Similarly, for a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$, let the distribution

$$\mathbf{IDEAL}_{\mathcal{A},\mathcal{S},T,\lambda,\mathbf{x}} \quad \boxed{\mathbf{IDEAL}_{\mathcal{A},\mathcal{S},\mathcal{K},T,\lambda,\mathbf{x}}} = \left\{ \beta : \begin{array}{l} (\mathrm{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_0(1^\lambda, \mathrm{seed}, \mathbf{x}) \\ (\mathrm{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(\mathrm{seed}, \mathbf{x}) \\ \text{For all } t \in [T] : \\ \quad (\mathrm{st}_{\mathcal{S}}, Q^t) \leftarrow \mathcal{S}_0(\mathrm{st}_{\mathcal{S}} \boxed{, \mathcal{K}}) \\ \quad (\mathrm{st}_{\mathcal{A}}, A^t) \leftarrow \mathcal{A}_0(\mathrm{st}_{\mathcal{A}}, Q^t \boxed{, \mathcal{K}}) \\ \quad b \leftarrow \mathcal{S}_1(\mathrm{st}_{\mathcal{S}}, A^t) \\ \quad \text{if } b = \perp, \text{break} \\ \beta \leftarrow \mathcal{A}_1(\mathrm{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

An authenticated PIR is private if for every efficient adversary $\mathcal{A}$ and for every $\mathbf{x} \in \mathcal{X}^N$ $\mathbf{x} = \{(kw_i, x_i)\}_{i\in N} \in (\mathcal{K} \times \mathcal{X})^N$, there exists a simulator $\mathcal{S}$ such that for all $\lambda \in \mathbb{N}$, $N \in \mathbb{N}$, good $\in [k]$, and $T \in \mathrm{poly}(\lambda)$ the following holds:

$$\mathbf{REAL}_{\mathcal{A},\mathrm{good},\mathcal{J},T,\lambda,\mathbf{x}} \approx_c \mathbf{IDEAL}_{\mathcal{A},\mathcal{S},T,\lambda,\mathbf{x}}$$

$$\boxed{\mathbf{REAL}_{\mathcal{A},\mathrm{good},\mathcal{J},T,\lambda,\mathbf{x}} \approx_c \mathbf{IDEAL}_{\mathcal{A},\mathcal{S},\mathcal{K},T,\lambda,\mathbf{x}}}$$

Definition 8 (APIR Integrity (based on [9, 11])). Let PIR = (Setup, Query, Answer, Reconstruct) be an unauthenticated PIR scheme for index-based keyword-based queries parameterized by a number of servers $k \in \mathbb{N}$, a set of database records $\mathbf{x} \in \mathcal{X}^N$ $\mathbf{x} = \{(kw_i, x_i)\}_{i\in N} \in (\mathcal{K} \times \mathcal{X})^N$ of size $N \in \mathbb{N}$. Let S be any subset of $k-1$ elements from $[k]$. For $idx \in [N]$ $kw \in \mathcal{K}$ let the distribution

$$\mathbf{REAL}_{idx} \quad \boxed{\mathbf{REAL}_k} = \left\{ \bigcup_{i\in S} q_i \; : \; \begin{array}{c} (\mathsf{pp}, \mathsf{DB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x}), \\ (\mathrm{st}, (q_1, \ldots, q_k)) \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, idx) \\ \boxed{(\mathrm{st}, (q_1, \ldots, q_k)) \leftarrow \mathsf{PIR.Query}(\mathsf{pp}, kw)} \end{array} \right\}.$$

Similarly, for a simulator $\mathcal{S}$, let the distribution

$$\mathbf{IDEAL}_{\mathcal{S}} = \left\{ (q_i)_{i\in S} \leftarrow \mathcal{S} \right\}.$$

An unauthenticated-PIR scheme PIR parametrized by a database size $N \in \mathbb{N}$ and a number of servers $k \in \mathbb{N}$ is secure if for every $idx \in [N]$ $kw \in \mathcal{K}$, the following holds:

$$\mathbf{REAL}_i \approx_c \mathbf{IDEAL}_{\mathcal{S}}.$$