2PPS – Publish/Subscribe with Provable Privacy

Sarah Abdelwahab Gaballah
TU Darmstadt
gaballah@tk.tu-darmstadt.de

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

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

Max Mühlhäuser
TU Darmstadt
max@tk.tu-darmstadt.de

***Abstract*—Publish/Subscribe systems like Twitter and Reddit let users communicate with many recipients without requiring prior personal connections. The content that participants of these systems publish and subscribe to is typically public, but they may nevertheless wish to remain anonymous. While many existing systems allow users to omit explicit identifiers, they do not address the obvious privacy risks of being associated with content that may contain a wide range of sensitive information.**

We present 2PPS (*Twice-Private Publish-Subscribe*), the first pub/sub protocol to deliver strong provable privacy protection for both publishers and subscribers, leveraging Distributed Point Function-based secret sharing for publishing and Private Information Retrieval for subscribing. 2PPS does not require trust in other clients and its privacy guarantees hold as long as even a single honest server participant remains. Furthermore, it is scalable and delivers latency suitable for microblogging applications.

A prototype implementation of 2PPS can handle 100,000 concurrent active clients with 5 seconds end-to-end latency and significantly lower bandwidth requirements than comparable systems.

***Index Terms*—privacy, anonymity, publish/subscribe, private information retrieval**

I. Introduction

Consider the immense popularity of services like Twitter, Reddit, and Telegram. All can be classified as implementing the Publish/Subscribe (pub/sub) messaging pattern: Messages are *published* to certain *topics* (e.g., hashtags for Twitter or channels for Telegram). Users can freely *subscribe* to topics they are interested in and will receive corresponding messages. The service acts as an intermediary broker between publisher and subscribers and is responsible for managing subscriptions, sorting received messages by topic, and forwarding them to the intended subscribers.

One particularly interesting use of pub/sub systems is the organization of volunteering, political involvement, and activism. Pub/sub lends itself to this setting since it allows large numbers of people to connect without having a prior personal relationship. Protesters in Iraq, Hong Kong, and Belarus have been using Telegram and FireChat for this purpose [1]–[3]. However, the use of conventional systems can leak valuable *metadata* to an adversary:

This work was supported by funding of the German Research Foundation (DFG), research grant 317688284 and by funding of the Helmholtz Association (HGF) through the Competence Center for Applied Security Technology (KASTEL)

- An activist who is found out to be publishing or subscribing to a regime-critical topic may be facing serious legal consequences.
- If the regime finds out how many users are subscribed to a critical topic, it can determine the size of the activists' movement and deploy an overwhelming police force at the next protest.

Telegram and FireChat do not protect this kind of metadata [4]. This is where our proposed protocol, 2PPS, comes in: It offers strong provable privacy protection for both publishers and subscribers in an open pub/sub setting. History shows that a state-level adversary can have access to immense resources [5]. Thus, we aim to protect against an adversary who may not only corrupt users and servers but also observe and interfere with traffic globally. While the example of political activists impressively motivates the need for private pub/sub communication, it is not the only possible use for 2PPS. Service providers can infer sensitive information such as health problems, financial status, or sexual preferences by observing which topics a user is active in.

At a high level, 2PPS reaches its goal as follows: The broker's functionality is distributed over multiple servers, where privacy is ensured as long as at least one arbitrary server does not collude with the adversary. While the adversary inherently learns which messages are published, since she can join arbitrary groups herself, she cannot learn any further information (e.g., who publishes which message). This is achieved by using *Distributed Point Function* (DPF)-based secret sharing. Compared to prior work [6], [7], we present an improved secret sharing approach that also protects against active interference. Subscribers protect their privacy by using *Private Information Retrieval* (PIR) for receiving messages.

To the best of our knowledge, no existing protocol can provide open pub/sub communication with strong provable privacy guarantees for both publishers and subscribers. Some protocols require trusted group members [8]–[10] or trusted execution environments [11]. Others don't provide both sender and receiver anonymity [10] or don't target worst-case protection [12]. Additionally, some of them are vulnerable to traffic analysis attacks [13]. PIR-based protocols offer strong cryptographic security guarantees and hide metadata efficiently [6], [7], [14]. However, the majority of these protocols support either point-to-point communication [7] or broadcasting [6],

2575-8462/21/$31.00 ©2021 IEEE
DOI 10.1109/SRDS53918.2021.00028

[14]. PIR-based protocols that support selective multicast communication either provide strong receiver anonymity but weak sender anonymity [9], or do not scale well [15].
Designing an anonymous communication protocol always requires a trade-off between privacy protection, trust, and overhead [16]. However, we show that 2PPS, despite strong privacy protection and minimal trust requirements, manages to keep the overhead for clients at a reasonable level:

- It incurs a latency of 25 s to handle one million users where each client submits a 160 B message and receives a 10 KB block of messages.
- For 50 subscriptions per client, 2PPS requires 300× less bandwidth compared to broadcasting systems such as Riposte [6] and Blinder [14].

Contributions: In this paper, we make the following contributions:

- We introduce 2PPS, a new anonymous publish/subscribe protocol by combining private writing using Distributed Point Functions (DPF) and private reading using Information-Theoretic Private Information Retrieval (IT-PIR).
- We give formal proof to show that the 2PPS protocol reaches our stated goals of Publisher- and Subscriber Unobservability.
- We provide an evaluation of 2PPS that demonstrates its efficiency in terms of latency and bandwidth.

II. Model and Goals

A. Protocol Overview

2PPS implements the publish/subscribe model: When *publishing* a message, the sending user (i.e., the publisher) specifies a *topic* the message belongs to. The protocol then delivers this message to all *subscribers* of this topic. We assume an open environment, meaning users can freely subscribe to and unsubscribe from any topic they wish.
The 2PPS network consists of n clients and N servers with $n \gg N$. Each server stores a full copy of two databases, a *write* database D_w and a *read* database D_r . All servers collectively maintain the contents of these databases. The read database D_r is partitioned into a set of ℓ_r equal-sized blocks, with a publicly known topic for each block. The write database D_w is partitioned into ℓ_w equal-sized blocks, each sized to hold one message.
Similar to related protocols [6], [15], [17], communication in 2PPS occurs in rounds to defend against traffic analysis attacks. Each round is split into three distinct phases (Figure 1 depicts these phases at a high level). During the first phase, each client deposits exactly one message into the write database D_w, which is shared among the N servers using secret sharing. If a client has no real message to send, it generates a cover message. After the first phase has concluded, the servers collaborate to reveal all messages simultaneously. During the second phase, cover messages are discarded and the remaining messages are sorted into their corresponding topic-block of the read database. Finally, in the third phase, clients anonymously retrieve the messages from their subscribed topic.

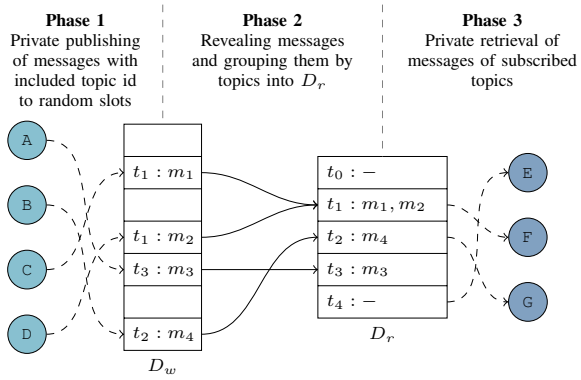


Fig. 1: Server databases (D_w and D_r) and general protocol flow.

B. Threat Model

2PPS assumes a strong adversary $\mathcal{A}$, whose goal is to compromise the privacy of honest users. $\mathcal{A}$ is assumed to be in control of all network links. Thus, she may not only passively observe all traffic on every link, but also insert, delay, drop, time, and modify arbitrary packets. Further, $\mathcal{A}$ may corrupt $N - 1$ servers and an arbitrary number of clients. Since we assume open groups, $\mathcal{A}$ may join all groups as a subscriber through corrupted clients. We assume that honest clients and servers behave as specified by the 2PPS protocol.

C. Security Goals

2PPS aims to achieve the formal privacy notions of *Publisher Unobservability* and *Subscriber Unobservability*. Kuhn et al. present a set of game-based privacy notions for unicast communication [18], which we adapt for the pub/sub scenario.
Each notion is defined by a game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:

1) $\mathcal{C}$ chooses a random challenge bit $b \in \{0, 1\}$.
2) $\mathcal{A}$ submits a challenge consisting of two self-chosen scenarios (S_0, S_1) to $\mathcal{C}$. Each secenario contains a number of communications (p, m, t), where p denotes the publisher, m the message, and t the topic that p sends m to.
3) $\mathcal{C}$ checks the received challenge for validity and, if valid, simulates the protocol execution of the communications contained in S_b.
4) Based on his abilities, $\mathcal{A}$ gets to observe and interact with the protocol execution.
5) $\mathcal{A}$ determines which of his scenarios was chosen and submits his guess $b' \in \{0, 1\}$ to $\mathcal{C}$. $\mathcal{A}$ wins if $b = b'$.

Steps 2-5 can be repeated. Instead of a challenge, the adversary can also submit a *subscription update*. The subscription update specifies for each client the topics she is subscribed to in each scenario. If differences in the communications between the two batches lead to differences in protocol behavior that $\mathcal{A}$ can

observe, then $\mathcal{A}$ gains an advantage over randomly guessing the chosen batch.

A concrete privacy notion defines which information is allowed to leak to the adversary and which should be protected by the protocol. Information that is allowed to leak may not differ between batches to ensure that $\mathcal{A}$ does not gain an unfair advantage when trying to distinguish. If $\mathcal{A}$ can still determine which of his batches was submitted to the protocol with a non-negligible advantage over random guessing, the protocol has failed to protect the information it was supposed to protect and therefore does not reach this privacy notion.

Publisher Unobservability: With publisher unobservability, the protocol aims to hide any information about active publishers. Implicitly, information about messages, topics, and subscribers is not protected. Thus, $\mathcal{A}$ is required to submit batches that only differ in the publisher of each communication: Le the ith communication of the submitted first batch be $(\mathbf{p_0^i}, t_0^i, m_0^i)$ where topic t_0^i has subscribers $r_0^{i,0}, \ldots, r_0^{i,n}$. Then, the ith communication of the submitted second batch has be of form $(\mathbf{p_1^i}, t_0^i, m_0^i)$ where t_0^i also has to have subscribers $r_0^{i,0}, \ldots, r_0^{i,n}$. If $\mathcal{A}$ can determine which of his batches was executed with a non-negligible advantage despite this restriction, the protocol does not reach publisher unobservability.

Subscriber Unobservability: Analogous to the publisher variant, subscriber unobservability aims to hide any information about active subscribers. Since information about publishers, messages, and topics is allowed to leak, $\mathcal{A}$ has to submit the *same* communications in both batches. However, which clients are subscribed to which topics may vary between the two batches of a challenge.

III. 2PPS Architecture

This section describes the 2PPS protocol in greater detail. Section III-A presents the anonymous publishing phase, Section III-B the management of published requests and Section III-C the anonymous subscription phase.

A. Phase 1: Anonymous Publishing.

Assume that client Alice wants to publish some message m to topic t without the adversary being able to link the message to her. We start by introducing a simple but inefficient method to hide sender identities from malicious servers. Then we improve the efficiency of this method and finally extend it to also protect against adversaries that are in control of the network.

Naïvely, *secret sharing* [19] can be employed for anonymous publishing: First, Alice computes a vector w of the same length as the write database D_w, which contains $(t \mid m)$ at a random index and 0 everywhere else. She then computes N secret shares $w_1, \ldots, w_N$ with the following properties:

1) $\sum_{i=1}^{N} w_i = w$
2) Any combination of $N-1$ secret shares does not reveal any information about $(t \mid m)$ or the index at which $(t \mid m)$ is located.

Alice distributes the shares to the servers, where the ith server S_i receives w_i. S_i then adds w_i to its read database state D_w^i:

$$D_w^i \leftarrow D_w^i + w_i$$

To hide sending frequencies, all clients are required to publish *exactly* one message per round. If the client has no "real" message to send, she may send a message consisting only of zeroes to a random topic as cover. After processing requests from multiple clients, the servers can collaborate to compute a combined database $D_w = \sum_i D_w^i$. As long as every client chose a unique index for her messages, D contains all original messages.

This approach is quite inefficient: For every write request, a vector with the same size as the database has to be sent. To address this issue, Riposte [6] suggested the use of distributed point functions (DPF).

Definition 1 (DPF): Let $f_{i^*,m} : \{0, \ldots, \ell_w\} \mapsto \mathbb{F}$ be a point function with

$$f_{i^*,m}(i) = \begin{cases} m & \text{for } i = i^* \\ 0 & \text{for } i \in \{0, \ldots, \ell\} \setminus i^* \end{cases}$$

$f_A, f_B : \{0, \ldots, \ell_w\} \mapsto \mathbb{F}$ are distributed point functions of $f_{i^*,m}$, iff

1) Neither f_A nor f_B by themselves reveal anything about m or i^*
2) $\forall\, i \in \{0, \ldots, \ell\} : f_A(i) + f_B(i) = f_{i^*,m}(i)$

DPFs can be used to *compress* the shares sent to the servers from the naïve approach: Alice runs $\mathsf{GenDPF}(t \mid m)$, which generates N DPF-shares $f_1, \ldots, f_N$ that contain $(t \mid m)$ at a random index. These shares are distributed among the servers, with server S_i receiving f_i. Server S_i can derive w_i by evaluating $f_i(j)$ at every point $j \in \{0, \ldots, \ell_w\}$. Current research states that sending a DPF share instead of w_i directly reduces the communication cost to $O(\lambda \cdot \log \ell_w + \log |(t \mid m)|)$ bits where λ is the security parameter [20].

As is, this approach protects against malicious servers, but not against stronger adversaries, who are also in control of the network: $\mathcal{A}$ could simply intercept Alice's shares before they reach the servers and combine them to reveal Alice's topic and message. To protect, Alice can encrypt each share with the receiving server's public key before sending it. Since we assume at least one honest server, $\mathcal{A}$ cannot gain access to all shares. Due to the public nature of messages in 2PPS, there are further active attacks possible:

Replay: To link Alice to her message by replay, $\mathcal{A}$ saves all shares Alice sends in a given round and all messages that are revealed in the same round. In the next round, $\mathcal{A}$ inserts the saved shares into the traffic. the servers will add them to their D_w state. $\mathcal{A}$ can identify which messages Alice has sent by observing which identical message was revealed in both rounds. To detect replayed messages, Alice includes a current timestamp with every share (inside of the encryption layer). An honest server can check the time stamp for currentness and refuse further participation in this round if this check fails. The share-encryption also prevents $\mathcal{A}$ from selectively

modifying the timestamp. $\mathcal{A}$ could also replay the shares in the same round as the original shares, which would corrupt Alice's message. However, this attack is easily detectable by the honest server, since it has access to all shares of the current round at once and can check for duplicates.

Modification: Our proposed protection against replay attacks also enables honest servers to detect if a received request was modified by $\mathcal{A}$. We assume that encryption used to protect the share and the timestamp provides *diffusion*, i.e., ensures that any change of a ciphertext leads to widespread and unforeseeable changes of the plaintext. Thus, any modification of a request leads to a significant change of the included timestamp with overwhelming probability. The honest server detects this invalid timestamp and refuses further participation in the current round.

Drop & Delay: Attacks based on dropping and delaying messages are both very common and hard to avoid in anonymous communication [21]. If a powerful adversary can drop the requests of all but one client, then he can unambiguously link this client to her messages once the shares are combined. A less powerful adversary might have insider knowledge of a message that will be sent in a given round. If he drops the request of the suspected sender and the message is not published, his suspicion is confirmed.

To detect a dropped request, the honest server needs to know how many messages are supposed to arrive in a given round. Related literature commonly assumes that protocol participation is static, i.e., that clients are always online [22], [23]. This is a very strong and arguably not very realistic assumption since it discards user churn. We introduce an additional mechanism that enables us to make weaker assumptions regarding client participation:

The *verifiable participation commitment* requires every client who joins the network to send a message to each server with which the client commits herself to participate in the next k rounds. The parameter k can be chosen by the client to fit his routine. A client could for example join when arriving at his office in the morning and commit to participating until his usual end of the workday. Clients can also commit to shorter periods and renew their commitment periodically. With that, the honest server knows from which clients to expect requests in any given round. If fewer requests than expected are received, the server assumes that a malicious drop must have occurred and refuses further participation to protect the senders' privacy. The adversary also needs to be prevented from replacing dropped requests with self-generated ones to circumvent the protection. This can be done by requiring the clients to include a *digital signature* with every request. That way, each request can be linked to the client who sent it and the server can *verify* that all committed clients have indeed participated.

B. Phase 2: Managing Published Messages.

When the writing epoch ends, the servers reveal the published messages among each other by combining their D_w states. As a first step, all cover messages (i.e., those which only contain zeroes) are discarded. Together, the servers choose a block size for D_r such that every topic's messages fit into a single block. Thus, the block size may be changing from round to round depending on the number of messages per topic, but at any point, all blocks of D_r have the same size. Next, the servers append each message in D_w to its corresponding topic-block in D_r. These messages are stored temporarily in D_r until the end of the communication round.

Updating the Topic List: Over time, new topics will be created and others will become inactive, thus there is a need for periodically updating the list of current topics and their corresponding blocks in D_r. To create a new topic, the client follows the same steps as when sending a message to an existing topic but includes a new topic id. The servers take notice of the unknown id and save the included message. During the next database update, a block for this new topic is created, and the topic is included in the list of topics sent to the clients. The first message from the original creator is added to this topic in the following round. Regarding deleting the inactive topics from D_r, servers will consider a topic as inactive, if there are no published messages on this topic for some configured number of rounds. The topic list and mapping from topic to block ID are updated periodically and clients are informed of the update afterward.

C. Phase 3: Anonymous Subscribing.

2PPS allows clients to anonymously subscribe to topics and get new messages without polling them. Like related protocols [9], [15], it depends on information-theoretic PIR (IT-PIR). The anonymous subscription consists of two building blocks: Subscription registration and message retrieval.

Private Subscription Registration: 2PPS requires all clients to update their subscriptions at a fixed rate to hide changes in interest. Every time a client receives a topic-list update, he has to renew his subscription. If the client is not interested in any topic, he sends a subscription request to a random topic. Clients that newly join the network need to wait until the next topic update to start participation.

Assume that Alice wants to subscribe to the jth topic. To do so, she creates a vector $q \in \{0,1\}^{\ell_r}$, which is equal to 1 at position j and equal to 0 at all other positions. Alice then computes a subscription request $req_i = Enc_{pk_i}(s_i|q_i)$ for each server S_i with $i \in \{1,\dots,N\}$. $Enk_{pk_i}(\cdot)$ is an encryption under the S_i's public key pk_i, $s_i \in \{0,1\}^{\ell_w}$ is a randomly chosen shared secret and the PIR query q_i is computed as follows:

$$q_i = \begin{cases} \text{random} & \text{for } i < N \\ q \oplus q_1 \oplus \cdots \oplus q_{K-1} & \text{for } i = N \end{cases}$$

The shared secret s_i is locally updated each round synchronously at client and server (e.g., using a cryptographic hash function or a key schedule).

To reduce the client's inbound bandwidth, related literature [9], [15] suggests the use of a random server P as a proxy for the client: Instead of sending the subscription requests $q_1,\dots,q_N$ directly to the servers, the client sends them to his proxy P. P

forwards these requests to corresponding servers where they are stored.

Remark 1 (Multiple Subscriptions): Each subscription registration may only contain a subscription to a single topic. If a client wants to be subscribed to multiple topics simultaneously, he has to send multiple subscription requests. To hide the number of topics clients are subscribed to, all clients need to send the same number of subscription requests. These can contain a mix of real subscriptions and cover subscriptions to random topics.

Private Messages Retrieval: In every round, each server S_i computes a response res_i for each stored subscription by taking the XOR of all D_r blocks that have 1 in their positions in the PIR query. Instead of sending the responses directly to the client, the servers submit them to P who computes $res \leftarrow \bigoplus_{i \in \{1,\ldots,N\}} res_i$, and forwards it to the client. Thus, the client's incoming bandwidth is reduced by a factor of N. To prevent P from learning which topic the client has subscribed to, each server has to obfuscate its response. Server S_i obfuscates its response by computing $res_i \leftarrow res_i \oplus s_i$, the client can restore the desired block of published messages by computing $res \oplus s_1 \oplus \cdots \oplus s_N$.

IV. Analysis of 2PPS security properties

In this section, we show that 2PPS reaches our formalized privacy goals as defined in Section II-C.

Theorem 1 (Publisher Unobservability): 2PPS achieves Publisher Unobservability.

Intuitively, reaching Publisher Unobservability requires unlinking senders from their messages and hiding which senders are active. To unlink senders from their messages, 2PPS employs secret sharing based on distributed point functions. Each server receives a secret share that does not reveal any information about the contained message by itself. Only once all clients have submitted their shares, the combination of all shares is revealed all at once. We strengthen the secret sharing scheme against adversaries in control of the whole network by introducing a timestamp and an additional layer of encryption around the shares. This prevents the adversary from being able to modify or replay shares. Further, we introduce the verifiable participation commitment which enables honest servers to detect dropped shares. Finally, we hide which senders are active by requiring all clients to send at a fixed rate, creating cover messages when they don't have a real message to send. A full proof of security can be found in Appendix A.

Theorem 2 (Subscriber Unobservability): 2PPS achieves Subscriber Unobservability.

In 2PPS, IT-PIR ensures that subscribers cannot be linked to the topics they're subscribed to. Both subscription requests and the responses containing the messages appear random to any adversary that is not in control of either the client itself or *all* servers. The use of cover traffic and synchronized round further ensures that the adversary cannot gain any information about the frequencies at which the client updates his subscription or receives messages. A full proof of security can be found in Appendix B.

V. Performance Evaluation

Privacy protection should not be only restricted to those with access to powerful hardware, but also it should support users who have limited bandwidth and computational power, e.g., in a mobile setting. 2PPS aims to keep the overhead at a reasonable level to enable as many clients as possible to participate. The goal of our evaluation is to investigate the impact of using DPFs (for private writing), and IT-PIR (for private reading) together on computation and network overhead on both client and server sides. One important measure of scalability is the end-to-end latency of a system. For 2PPS, we evaluate the influence of a changing number of participating clients, the number of subscribed topics per client, and the number of messages per topic on the latency of the system.

Implementation: A prototype of our protocol is implemented in C and Go. We use Go for the high-level operations of client and server. Cryptographic primitives are used from the DEDIS advanced crypto library and Go's native crypto library. We rely on the available source code of Express[1] for C implementations of the auditing protocol and DPFs. To update the shared secrets between clients and servers locally, we use keyed AES similar to Riffle [15]. We conduct the experiments on three virtual machines, each equipped with a 16-core Intel Xeon E5-2640 v2 processor and 64 GB of RAM. All three machines are located in the same data center. We operate two of them as servers and use the third one to simulate the clients. In all experiments, each client is configured to send one message per round (LS in Figure 2 refers to the length of the sent message). All clients participate in every round. Also, we test the performance of private retrieval for three different block sizes: 10 KB, 64 KB, and 256 KB (LR in Figure 2 refers to the length of the retrieved block). We adopt these block sizes from [23], [7] and [15].

Baselines: We compare 2PPS to three different PIR-based anonymous group communication protocols: Riposte [6], Pung [23], and Blinder [14]. We choose these protocols since they provide cryptographic anonymity guarantees similar to 2PPS. Riposte and Blinder support anonymous broadcasting, whereas Pung and 2PPS provide selective multicast communication (i.e., they allow the users to fetch only the messages that are interesting to them).

A. Computation Overhead

To understand the computation costs that are imposed on the client and server-side, we run a set of experiments in which every client sends one 1 KB message and retrieves one 64 KB block per round. Between experiments, we vary the number of messages processed by the servers (i.e., the number of participating clients). Since each client selects a random row to write its message into, collisions are possible and lead to the irreversible corruption of both colliding messages. In this experiment, we use a large fixed database D_w to handle this issue. This database achieves write success rates of 99.8 %, 98 %, and 82 % for 10^3, 10^4, and 10^5 messages respectively

[1] https://github.com/SabaEskandarian/Express

	# Messages Processed		
	10^3	10^4	10^5
Client CPU costs			
Generate DPF shares	229.26 μs	229.26 μs	229.26 μs
Audit	204.16 μs	204.16 μs	204.16 μs
Create PIR query	0.533 μs	6.73 μs	12.54 μs
Process PIR reply	19.33 μs	21.79 μs	22.027 μs
Server CPU costs			
Expand DPF shares	5.62 s	5.62 s	5.62 s
Audit	36.52 ms	36.52 ms	36.52 ms
Second Phase			
1. Combine D_w states	13.27 ms	14.01 ms	19.66 ms
2. Group messages	0.08 ms	0.87 ms	6.70 ms
Process PIR Query	0.17 ms	1.19 ms	10.06 ms

TABLE I: Cost of 2PPS operations under varying the number of messages stored on the server where the size of each message is 1 KB.

(according to the success rate formula in [6]). As the size of D_w is fixed, clients and servers pay fixed CPU costs for each write-request regardless of the number of messages received by the servers.

As shown in Table I, the client's operations are all comparatively inexpensive. Note that, "Create PIR query" is done only during subscription updates rather than every round. "Process PIR reply" denotes the time it takes to XOR the received PIR reply with the shared secrets to reveal the messages. The computation costs for the servers are dominated by the first phase ("Expand DPF shares" and the "Audit"). Also, the second phase introduces overhead which can be broken down further into the costs of two sub-phases: combining the shared D_w states, which accounts for the largest part of the overall time of the second phase, and the grouping of messages into their corresponding topics.

B. Network Overhead

In this section, we discuss the network overhead of 2PPS. These measures are especially important in bandwidth-restricted scenarios, such as mobile communication. We split our discussion into two parts: First, we consider the total bandwidth consumed by the operation of 2PPS versus related protocols. Second, we investigate how much "unnecessary" bandwidth in form of cover messages is required for 2PPS.

Comparative bandwidth comsumption: Figure 2a shows the total communication cost to send one message and retrieve one block by the client when the number of participating clients varies. That includes all the sent and received messages between the client and the server to achieve private writing and reading. Since we use a fixed size for D_w, the communication costs of private writing (including the auditing) are not growing when the number of clients increases.

Compared to Pung, 2PPS's anonymous writing is more expensive. However, Pung's reading phase introduces a high communication overhead that makes the total cost of communication using Pung significantly more expensive than 2PPS. For instance, when there are one million messages on the server, Pung has a total communication cost that is $1{,}263\times$ larger than 2PPS to send a 1 KB message and retrieve a 10 KB block. Note that, in the shown results, the costs of Pung include sending the PIR query by the client to retrieve the messages. If we assume that the PIR queries are stored already on the servers (similar to 2PPS), this will reduce the Pung costs to $950\times$ larger than 2PPS which is still a substantial difference. Pung's high overhead occurs because clients do not know the index of the data that they are interested in. Instead, they perform a binary search on the database through multiple CPIR queries. Further, the size of the CPIR answer increases as the CPIR recursion depth becomes higher. Angel et al. state that Pung's approach results in an increase of network overhead by a factor of $\log(n)$ for a database of n elements compared to PIR with known indices [23, Section 7.4].

Riposte also requires communication costs much higher than 2PPS. For 10^6 clients, 2PPS has $5{,}033\times$ less total cost than Riposte. 2PPS has better performance for two reasons: 1) it uses a new generation of DPFs [24] and an auditing protocol [7] that is more efficient than the one used in Riposte; 2) it sends to each client only a subset of the published messages, whereas Riposte broadcasts all messages to all clients.

Blinder introduces high total communication costs due to its need to operate by a large number of servers to achieve its anonymity guarantees (in our experiments, we ran Blinder on 5 servers). Additionally, it broadcasts all published messages to every client resulting in high communication overhead similar to Riposte. For 10^6 clients, the total network cost for each Blinder client is around 160 MB while the corresponding value in 2PPS is about 32 KB.

To further explore the performance implications of using IT-PIR on the network download overhead, Figure 2b depicts the total amount of data a client receives during 20 rounds for one subscription per client. In 2PPS, the amount of data per round equals the number of subscriptions a client has times the block size. A client who subscribes to one topic with a block size of 64 KB, downloads 1.28 MB during 20 rounds. Pung introduces a much higher communication overhead than 2PPS as well, but less than broadcasting. A Pung user downloads more data than a 2PPS user by a factor of $54-512\times$ to retrieve a 10 KB block.

To compare the download overhead in 2PPS with a broadcast-based approach such as Riposte and Blinder, we consider broadcasting under three different assumptions regarding the number of cover messages. The first case (denoted as "broadcasting 100 %") assumes that all messages that clients sent to the servers are real messages. Analogously, broadcasting 50 % and 25 % assume that 50 % and 25 % of messages are real, respectively. Note that cover messages are discarded and therefore not broadcast to the clients. For one million clients, the network download of broadcasting (100 %) is 20 GB in 20 rounds, resulting in a $15{,}625\times$ increase over 2PPS when the block size is 64 KB. Hence, this method is extremely inefficient in terms of bandwidth for popular services.

As shown in Figure 2c, 2PPS also considerably outperforms the three broadcasting variants when the number of subscriptions per client increases. Therefore, adopting IT-PIR in our

protocol to retrieve the interesting messages generally allows bandwidth-efficient communication between client and server. That makes 2PPS suitable for bandwidth-restricted users.

Average Cover Ratio: While 2PPS's retrieval method introduces no overhead for downloading a fixed amount of data, another kind of overhead occurs if topics have differing popularity: Similar to the "Counting Bound" proposed by Gelernter and Herzberg [25], hiding which topic a client is subscribed to *requires* a certain amount of overhead:

Assume that $\mathcal{A}$ knows the combined size of all messages sent to each topic and assume that each client subscribes to a single topic. To hide any information about the link between clients and their subscribed topics, *all* clients have to receive a response large enough that it could contain all messages from the most popular topic. Otherwise, $\mathcal{A}$ can eliminate topics from the list of possible subscriptions for a given client which have more messages than the client receives.

To fulfill the Receiver Counting Bound, 2PPS pads each topic to the same size as the most popular topic using cover massages; a client subscribed to any topic that receives fewer messages than the most popular one will receive some amount of cover messages with his PIR response creating network overhead. We want to evaluate how much of a client's request on average consists of cover messages. While we have no real usage data for 2PPS, we can find related literature to approximate how topic popularity might be distributed: According to Chen et al. [26], the popularity distribution of Twitter hashtags follows Zipf's law with an exponent of $\alpha \approx 0.8$. Further, Liu et al. [27] have analyzed the RSS feed characteristics and determined that the popularity of feeds sorted by the number of requests also follows Zipf's law with an exponent of $\alpha \approx 1.37$. We use Chen et al.'s result to approximate the number of messages sent to a given topic and Liu et al.'s result to determine the likelihood of a client subscribing to a given topic. In our experiment, we assumed that the most popular topic receives 1000 messages and that the ith most popular topic receives $1/i^{0.8}$ as many messages. We chose a random topic index j according to a Zipfian distribution with parameter $\alpha = 1.37$ and determined how many cover messages topic j required. Under these assumptions, clients receive approximately $54\,\%$ cover messages per retrieval on average. Note that this number highly depends on the actual usage patterns of the service: If all topics receive the same number of messages, *no* cover messages are required, if topic popularity varies widely, more than $54\,\%$ cover messages might be received on average.

C. End-to-End Latency

To evaluate the efficiency of our protocol, we are interested in computing the total time required to send one message and retrieve one block by each client. The latency in 2PPS is measured as the total time of all three protocol phases. The time of DPF evaluation represents the expensive part of the total latency, and it depends on the size of D_w. Having a fixed large database means a fixed big evaluation time for DPF shares even when the number of received write requests is small. For less latency, the evaluation time can be reduced by changing the size of D_w based on the number of expected requests. However, the database should be still large enough to handle requests successfully with high probability.

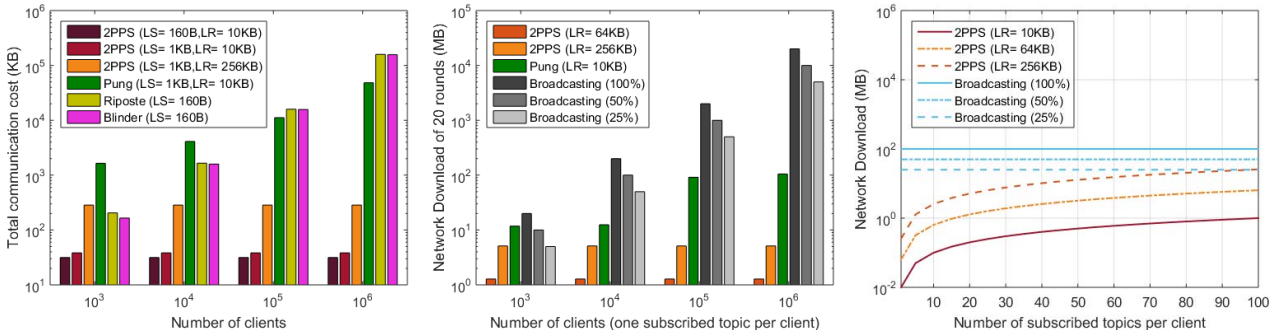
The latency of the third phase is determined by the number of topics, the block size, and the number of subscriptions per client. The more topics the read database contains or the larger each topic block is, the longer it takes to compute a PIR reply. If a client subscribes to more topics, the number of PIR replies that need to be computed increases, increasing latency.

Figure 2d illustrates how the retrieval time of one block scales whith varying block sizes and numbers of clients. In general, retrieving messages using either our method or broadcasting doesn't cause much latency. We compared our protocol to broadcasting to understand the performance implication of using PIR for distributing messages to the receivers instead of using a broadcast. 2PPS is considerably faster than Pung, even when retrieving larger blocks.

Figure 2e illustrates the increase in retrieval time when a 2PPS client subscribes to many topics. The number of participating clients influences the time required to broadcast messages more than it does with 2PPS. That is especially true when most of the published messages are real. For 10^4 clients, retrieving 12 blocks using 2PPS takes $97\,\mathrm{ms}$ more than the broadcasting, which means the difference in the latency time between the two approaches is comparatively insignificant. Therefore, 2PPS's retrieval method can reduce the bandwidth without leading in return to high latency, even if the clients have many subscriptions.

Figure 2f shows the end-to-end latency for posting one message when we vary the number of clients. Again, this figure demonstrates the negligible effect of retrieval time on the total latency, as the end-to-end latency of 2PPS is slightly smaller than the method that relies on DPF and broadcasting (instead of IT-PIR). 2PPS has significantly better performance than Riposte as it adopts a more efficient DPF version [24] and auditing method [7].

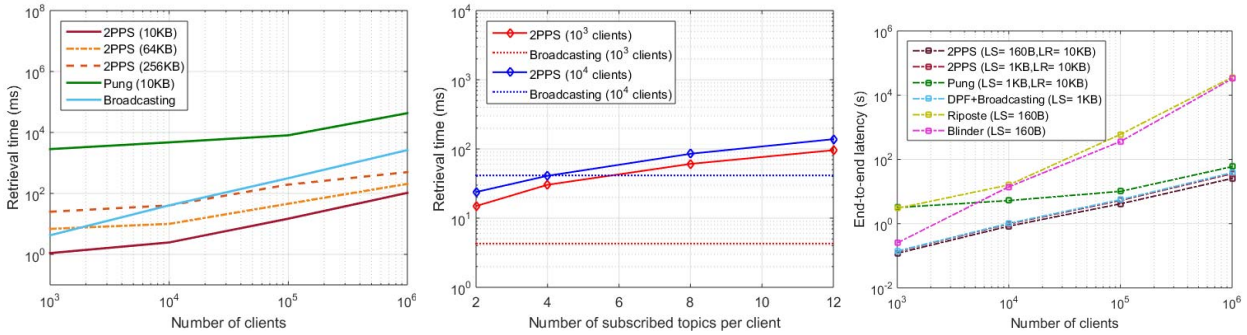
We compared the latency of 2PPS to Pung when sending one $1\,\mathrm{KB}$ message and retrieving one $10\,\mathrm{KB}$ block (10 messages in the block). As shown in Figure 2f, 2PPS outperforms Pung especially for a modest number of clients. In Pung, communication partners are required to trust each other, which enables clients to agree upon secret mailboxes for their message exchange. Since the link between clients and their mailboxes is not known to the adversary, messages can be directly written to the mailboxes, resulting in much less overhead than our DPF-based approach. The use of a single untrusted server in Pung requires much more overhead in the retrieval of messages than IT-PIR based approach of 2PPS. Overall, this expensive reading phase more than offsets any advantages in the latency of Pung's writing phase. The latency of Blinder is close to Riposte and considerably higher than 2PPS. Similar to 2PPS, the most expensive part in Blinder is the process of expanding the submitted blind write requests to add them to the database. For 10^5 clients, the total time to serve this number of clients is around 8 minutes, and the expanding process acquires $96\,\%$



(a) Total communication costs when the client sends one message and retrieves one block of messages.

(b) The total amount of downloaded data by a client after 20 rounds with varying numbers of clients and block sizes.

(c) The total amount of downloaded data by a client for 100,000 clients with varying numbers of subscribed topics and block sizes.



(d) Retrieval time of one block per client with varying number of clients and block size.

(e) Retrieval time when a client subscribes to different number of topics.

(f) End-to-end latency of message delivery when a client subscribes to one topic.

Fig. 2: Evaluation Results

of the total latency. 2PPS supports the same number of clients with less than half of Blinder's time.

VI. Related Work

2PPS provides a publish/subscribe protocol that achieves provable publisher and subscriber unobservability against a global active adversary who may also corrupt arbitrary clients and all but one server. Privacy protection is achieved via a combination of DPF-based secret sharing for publishing and PIR for subscribing to topics. In this section, we provide an overview of existing anonymous communication protocols and how they compare to 2PPS.

Mixnets-based Protocols.: Mix networks (mix nets) [28]–[30] ensure the anonymity of users by obfuscating the source of a message. They work by collecting the messages from many users and shuffle them by a set of servers called mixes before sending them out to recipients. Therefore, they make it difficult for the global adversary to correlate input and output messages and protect against traffic analysis attacks. However, malicious mixes can launch several attacks to deanonymize users; for instance, they can drop, modify, or duplicate the input messages before sending them out [31]–[33]. Verifiable shuffles techniques [34]–[36] have been proposed to protect against tampering messages by malicious servers, but these techniques introduce high computation overhead. Mixnets-based protocols like McMix [37], Atom [38] and XRD [39] induce high latency to support large numbers of users. While protocols as Vuvuzela [22], Stadium [40], Alpenhorn [41], and Karaoke [42] achieve better performance than 2PPS, but they provide weaker differential privacy guarantees compared to the cryptographic guarantees as 2PPS does. In practice, this means that an adversary learns more information the longer he observes the mixnet, which is not the case for 2PPS.

DCnets-based Protocols.: Chaum's Dining Cryptographers network (DCnet) [43], [44] is an information-theoretic anonymous broadcast method. To increase the scalability and practicality of DCnets, many protocols like Dissent [8] and Verdict [45] adopted the client-server paradigm, where n clients form the anonymity set, but only a small set of N servers implement the functionality. This adoption reduces the overall communication complexity from $O(n^2)$ in the traditional DCnet model (where there is a full graph between clients) to $O(N \cdot n)$ in the client-server DCnet model. The sender anonymity guarantees that DCnet protocols provide are the same as 2PPS does. However, the size of the sent and the received message in 2PPS is significantly smaller than in DCnet protocols. That is because 2PPS uses DPF to compress the write requests and IT-PIR to only retrieve the blocks that the client is interested in,

instead of getting all the messages through broadcasting. Also, the used primitives in our protocol allow it to support a much larger number of clients and provide faster communication time than DCnet protocols do [6], [7].

PIR-based Protocols.: Many protocols rely on PIR methods to enable anonymous communication. There are two classes of these protocols that are: 1) information theoretic-PIR-based protocols (multiserver) such as Express [7], Riposte [6], Blinder [14], and Talek [9]; and 2) computational-PIR-based protocols (one server) such as Pung [23]. Express uses DPF to allow a user to send a message anonymously to the mailbox of another user. However, this protocol does not protect the anonymity of the mailbox's owner (i.e., the receiver's anonymity). Riposte and Blinder also use DPF but broadcast all the published messages. As we have shown in our evaluation (Section V), broadcasting results in much higher network overhead than 2PPS's PIR, making it not suitable for bandwidth-restricted scenarios. DPF-based protocols in general have a comparatively low computational overhead and can scale to support millions of users. Talek provides a private publish-subscribe protocol that allows communication between small groups of trusted users. In contrast, 2PPS's overhead does not depend on the number of subscribers a topic has and users neither have to trust publishers nor other subscribers to protect their privacy. Pung operates in a single-server setting and provides strong anonymity guarantees since it can hide user's interests even if the server is malicious. However, it introduces more overhead than 2PPS and requires users to trust their communication partners.

VII. Discussion

In this section, we present some lessons learned from designing a public-message protocol based on secret sharing and private information retrieval.

Availability: As discussed in Section III, 2PPS owes its strong provable privacy protection to the combination of secret sharing and private information retrieval. Both of these techniques distribute trust by relying on multiple servers. This distribution of trust is great from a privacy perspective: As long as one server remains honest, user privacy is preserved. Users with high privacy requirements can even deploy their own servers to increase the chance of an honest one existing.

While very advantageous when it comes to privacy, secret sharing and PIR are very vulnerable as far as availability is concerned: If a *single* malicious server refuses to provide its write-database state after clients sent their messages, the protocol execution cannot continue. To reveal the messages, *all* shares are required by design. The same problem arises on the reading site: If a malicious server refuses to send his PIR response, the clients will not be able to receive the message from their subscribed topics. Thus, as related literature [6], [9], we assume that malicious servers do not target availability.

Remark 2 (Backup Servers): An obvious mitigation against availability attacks from servers would be to introduce a "backup" server for each server. The backup server would receive the same information as its corresponding main server. In case the main server refuses to submit his shares, the backup server can step in. This avoids disruption as long as not both one main server and its backup refuse to participate. However, this comes at the cost of additional trust. Instead of only requiring one server to be honest, two honest servers are required; One honest main server and one honest backup server. Further, communication overhead for publishing messages also increases twofold, since the number of servers is doubled.

Malicious clients may also target availability. While the auditing protocol prevents clients from submitting malformed requests that would corrupt the write database, there are other avenues of attack which are inherent to 2PPS' open nature: An adversary could create a large number of topics, increasing the required size of the write- and read databases and therefore also the network overhead of the whole system. Further, the adversary could also spawn a large number of clients who all submit messages to a single topic, increasing the amount of cover needed for all other topics. In scenarios where availability is of greater concern than the open nature of the systems, mechanisms can be put in place that increase the effort of adding users and topics to the system.

Mitigating Intersection Attacks: Due to the public nature of messages and topics, 2PPS is particularly vulnerable to a specific kind of intersection attack: Assume that the adversary $\mathcal{A}$ wants to find out the interests of client Alice, who only publishes to a single topic. Every time Alice is participating in a communication round, $\mathcal{A}$ records which topics are active (i.e., have messages sent to them). Over multiple rounds, $\mathcal{A}$ *intersects* the list of active topics until only a single topic remains, which is unambiguously linked to Alice.

Intersection attacks are inherently possible in any protocol that allows users to choose when they want to participate (see Appendix C for proof). While we assume constant participation for our security analysis, we also present possible mitigation techniques against intersection attacks for situations where constant participation is not obtainable:

- **Cover Traffic.** A simple mitigation that is already in use with 2PPS is cover traffic. If Alice uses cover traffic, then $\mathcal{A}$ cannot distinguish rounds where Alice is sending "real" messages from rounds where she is participating idly, increasing the number of rounds $\mathcal{A}$ needs to observe.
- **Delayed Publishing.** Alice can also require that her message is not published in the round where she sent it but later (when Alice might already be offline). To do so, Alice can include some random delay d in the message-topic tuple $t \mid m$ and *encrypt* it with the public key of one of the servers. After combining their D_w states, the corresponding server will reveal the ciphertext and delay publishing it. This solution is based on the idea of distributing knowledge which means each server knows the real publishing time of only a subset of all messages. Thus, $\mathcal{A}$ who controls one of the servers cannot accurately link every message to the set of the potential senders.

Collisions: In 2PPS, the client chooses a random row in the database to write her message into. Therefore, it is possible to have collisions, i.e., two or more clients writing their messages

in the same row which corrupts both messages. To minimize the probability of collisions during normal operation, 2PPS uses a write database that is much larger than the number of participating clients. However, this solution cannot solve the problem completely. If a client does not find her message among the published message in a given round, she has to assume that a collision occurred and may try to send her message again in a later round. Blinder [14] employs another approach to deal with collisions: If a client wants to publish message m, she computes one set of secret shares with m at a random index and another set of shares with the square of m at the same index. The servers store the received shares in separate databases. If a single collision occurred for a given index, the combined shares of both databases form a solvable system of quadratic equations, enabling the servers to reconstruct both original messages. While this reduces the required storage space on the server side, it requires more computational overhead for clients and servers.

VIII. Conclusion & Future Work

2PPS provides an anonymous publish/subscribe protocol with strong provable privacy guarantees for both publishers of messages and subscribers. This is achieved by combining secret sharing based on distributed point functions for publishing with private information retrieval for accessing subscribed topics. Compared to previous work, additional protection mechanisms are introduced to the secret sharing: A combination of timestamps and encryption allows 2PPS to provide publisher privacy not only against a malicious server but also against stronger adversaries that can observe and interfere with traffic on all network links. Our experimental evaluation shows that 2PPS can support a large number of users with latency suitable for applications such as microblogging and newsfeeds. While 2PPS reaches its stated goals, we see the following avenues for future improvement:

- Improvements to the auditing protocol to increase the number of servers without introducing high computation and communication costs.
- Enabling users to subscribe to multiple topics using one subscription request. If servers would be able to handle all the subscriptions of one client at once, more efficient packing of messages may be possible, reducing the amount of cover needed.
- Allowing the anonymous retrieval of messages from previous rounds.
- Lowering latency to enable use cases such as live streaming.

Acknowledgments

We thank Martin Byrenheid, Clemens Deusser, and Ephraim Zimmer for their valuable feedback and discussion.

References

[1] A. Hern. Firechat updates as 40,000 iraqis download 'mesh' chat app in censored baghdad. *The Guardian*, 2014.
[2] A. Bland. Firechat – the messaging app that's powering the hong kong protests. *The Guardian*, 2014.
[3] A. Herasimenka et al. There's more to belarus's 'telegram revolution' than a cellphone app. *The Washington Post*, 2020.
[4] C. Baraniuk. Firechat warns iraqis that messaging app won't protect privacy. *Wired*, 2014.
[5] M. Smithberger et al. Making sense of the $1.25 trillion national security state budget. *POGO.org*, 2019.
[6] H. Corrigan-Gibbs et al. Riposte: An anonymous messaging system handling millions of users. In *IEEE S&P*, 2015.
[7] S. Eskandarian et al. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. *ArXiv*, 2019.
[8] D. Wolinsky et al. Dissent in numbers: Making strong anonymity scale. In *USENIX OSDI*, 2012.
[9] R. Cheng et al. Talek: a private publish-subscribe protocol. Technical report, 2020.
[10] G. Perng et al. M2: Multicasting mixes for efficient and anonymous communication. In *ICDCS*, 2006.
[11] S. Arnautov et al. Pubsub-sgx: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems. In *SRDS*, 2018.
[12] G. Giakkoupis et al. Privacy-conscious information diffusion in social networks. In *DISC*, 2015.
[13] D. Lin et al. Scalable and anonymous group communication with mtor. *PETS*, 2016.
[14] I. Abraham et al. Blinder: Mpc based scalable and robust anonymous committed broadcast. *IACR Cryptol. ePrint Arch.*, 2020.
[15] A. Kwon et al. Riffle: An efficient communication system with strong anonymity. *PETS*, 2016.
[16] C. Kuhn et al. Sok on performance bounds in anonymous communication. In *WPES*, 2020.
[17] D. Wolinsky et al. Dissent in numbers: Making strong anonymity scale. In *USENIX OSDI*, 2012.
[18] C. Kuhn et al. On privacy notions in anonymous communication. *PETS*, 2019.
[19] A. Shamir. How to share a secret. *Commun. ACM*, 1979.
[20] E. Boyle et al. Function secret sharing: Improvements and extensions. *SIGSAC*, 2016.
[21] M. Ando et al. On the complexity of anonymous communication through public networks. *ArXiv*, abs/1902.06306, 2019.
[22] J. Van Den Hooff et al. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.
[23] S. Angel et al. Unobservable communication over fully untrusted infrastructure. In *USENIX OSDI*, 2016.
[24] E. Boyle et al. Function secret sharing: Improvements and extensions. In *SIGSAC*, 2016.
[25] N. Gelernter et al. On the limits of provable anonymity. In *WPES*, 2013.
[26] H. Chen et al. Scaling laws and dynamics of hashtags on twitter. *Chaos*, 2020.
[27] H. Liu et al. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *IMC*, 2005.
[28] D. Chaum et al. cmix: Anonymization by high-performance scalable mixing. In *ACNS*, 2017.
[29] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.
[30] A. Jerichow et al. Real-time mixes: A bandwidth-efficient anonymity protocol. *IEEE Journal on Selected Areas in Communications*, 1998.
[31] B. Pfitzmann et al. How to break the direct rsa-implementation of mixes. In *EUROCRYPT*, 1989.
[32] L. Nguyen et al. Breaking and mending resilient mix-nets. In *PETS*, 2003.
[33] B. Pfitzmann. Breaking an efficient anonymous channel. In *EUROCRYPT*, 1994.
[34] S. Bayer et al. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, 2012.
[35] J. Furukawa et al. An efficient scheme for proving a shuffle. In *CRYPTO*, 2001.
[36] J. Brickell et al. Efficient anonymity-preserving data collection. In *SIGKDD*, 2006.
[37] N. Alexopoulos et al. Mcmix: Anonymous messaging via secure multiparty computation. In *USENIX Security*, 2017.
[38] A. Kwon et al. Atom: Horizontally scaling strong anonymity. In *SOSP*, 2017.
[39] A. Kwon et al. Xrd: Scalable messaging system with cryptographic privacy. In *USENIX NSDI*, 2020.
[40] N. Tyagi et al. Stadium: A distributed metadata-private messaging system. In *SOSP*, 2017.

[41] D. Lazar et al. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *USENIX OSDI*, 2016.
[42] D. Lazar et al. Karaoke: Distributed private messaging immune to passive traffic analysis. In *USENIX OSDI*, 2018.
[43] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1988.
[44] P. Golle et al. Dining cryptographers revisited. In *EUROCRYPT*, 2004.
[45] H. Corrigan-Gibbs et al. Proactively accountable anonymous messaging in verdict. In *USENIX Security*, 2013.
[46] D. Wolinsky et al. Hang with your buddies to resist intersection attacks. *SIGSAC*, 2013.

APPENDIX

A. Proving Sender Unobservability

Lemma 1 (Message Unlinkability): $\mathcal{A}$ cannot identify which sender sent a given message.

Proof: We proof Lemma 1 by showing that $\mathcal{A}$ cannot use any of his abilities to do so.

- **Passive Observation.** Every client sends one request per round to each server. All requests are encrypted under the receiving servers public key. The servers collect the incoming requests and reveal all messages from the current round at once. Thus, $\mathcal{A}$ cannot link messages to senders by passively observing requests between clients and servers.
- **Server Corruption.** According to our assumptions, $\mathcal{A}$ is able to corrupt all but one 2PPS servers. $\mathcal{A}$ can only match a client to the request he sends prior to adding it to the db_w state. However, at this point, $\mathcal{A}$ can learn at most $N-1$ of the clients N DPF-shares. Related literature has formal proves that combining $N-1$ shares does not reveal any information about the enclosed message [24].
- **Replay.** $\mathcal{A}$ could either replay request during the same or a later round. If $\mathcal{A}$ replays a request during the same round as the original request was sent, the honest server will detect identical requests arriving and discard all but one. If $\mathcal{A}$ replays a request during some later round, the honest server will notice that the included timestamp is not valid and discard the request. $\mathcal{A}$ is not able to update the timestamp of the replayed request, since it is protected by an encryption layer prior to the honest server.
- **Modification.** To be able to link a request to the client who sent it, $\mathcal{A}$ has to modify the request prior to the honest server. Since the contained DPF-share is encrypted, any modification at this point will lead to unpredictable changes of the share. Such a share will be rejected by the honest server's auditing protocol with overwhelming probability.
- **Dropping.** We assume that the honest server can detect a dropped request and refuse further protocol participation.

■

Theorem 3 (Sender Unobservability): 2PPS achieves Sender Unobservability.

Proof: We define a series of hybrid games:

- H_0: The original $S\overline{O}$ game
- H_1: H_0, but clients can only publish 0-messages to a random topic
- H_2: H_1, but clients can only publish cover messages
- H_3: Identical Scenarios

In the following, we show that any adversary that can win H_i non-negligible advantage can also win H_{i+1} with non-negligible advantage.

- $H_0 \approx H_1$: We assume that $\mathcal{A}$ can win H_0. Lemma 1 states that $\mathcal{A}$ does so without being able to link any revealed message to a sender.
- $H_1 \approx H_2$: The only difference between a cover message and a "real" message is it's content: While a real message is sent to a chosen topic and can contain an arbitrary plaintext, a cover message is sent to a random topic and contains only 0s. We note that H_1 already requires real messages to have identical content to cover messages. Thus, all messages in H_1 are indistinguishable from cover messages to $\mathcal{A}$. If $\mathcal{A}$ can win H_1, he can therefore also win H_2.
- $H_2 \approx H_3$: As mentioned previously, we assume that all clients participate in every round. Further, every client sends exactly one message in each round. In H_2, all messages are sent to a random topic and contain only 0. Thus, $\mathcal{A}$ already has no influence on protocol activity in H_2 and the scenarios therefore appear identical to him.

We have shown that any adversary who can win H_0 can also win H_3 with a non-negligible advantage. Since $\mathcal{A}$ is only allowed to submit identical scenarios in H_3, there cannot be any such $\mathcal{A}$. Therefore, no $\mathcal{A}$ can win H_0, which is equivalent to the $S\overline{O}$ game. ■

B. Proving Receiver Unobservability

Theorem 4 (Receiver Unobservability): 2PPS achieves Receiver Unobservability.

Proof: With Receiver Unobservability, $\mathcal{A}$ may not learn any information about receiver activity. $\mathcal{A}$ can either gain information about receivers by observing subscription registrations or communication.

- **Subscription Registration.** Each client sends his subscription registrations at a fixed rate to each server. Prior to the receiving server, the registration is always encrypted under the public key of this server. An active adversary may be able to corrupt a clients subscription registration. However, this does not lead to differing behavior based on the topic the client wants to subscribe to, since receivers show do not react to the messages they receive.
  We assume that $\mathcal{A}$ can corrupt $N-1$ servers. This does not help him in determining the topic a client is subscribing to, since any combination of $N-1$ requests per definition appears random to $\mathcal{A}$. Only the combination of all N requests reveals the topic.
- **Communication.** Since the subscription request appears random to every server, no combination of $N-1$ servers can determine which topic the client is subscribed to by computing the response res_i. Although the primary server P has access to all individual responses, it cannot reveal the messages, since they are obfuscated by the shared secret s_j between the client and the honest server.

C. Intersection Attacks ■

Definition 2 (Delivery-Guarantee): A protocol provides Delivery-Guarantee, if all messages that are sent in a given round are also published in the same round.

Definition 3 (Sending-Nonobligation): A protocol provides Sending-Nonobligation, if it does not enforce when users send messages.

Common approaches against intersection attacks [46] do *not* provide Sending-Nonobligation.

Theorem 5: A protocol that provides both Delivery-Guarantee and Sending-Nonobligation cannot provide sender-messages unlinkability against an adversary who learns which messages are sent if the protocol guarantees delivery of all messages sent in a given round.

Proof: Without Sending-Nonobligation, clients may only participate in a subset of all rounds. $\mathcal{A}$ observes a messages m being published in round r Due to Delivery-Guarantee, a client that has not participated in round r cannot have been the sender of m.

If $\mathcal{A}$ can determine that another message m' was sent by the same sender, he can further narrow down the set of possible senders of m and m' via an *intersection* attack. ■