Nethanel Gelernter, Amir Herzberg, and Hemi Leibowitz

Two Cents for Strong Anonymity: The Anonymous Post-office Protocol

Abstract: We introduce the *Anonymous Post-office Protocol (AnonPoP)*, a practical strongly-anonymous messaging system. AnonPoP design combines effectively known techniques such as (synchronous) mix-cascade and constant sending rate, with several new techniques including *request-pool*, *bad-server isolation* and *per-epoch mailboxes*. AnonPoP offers *strong anonymity against strong, globally eavesdropping adversaries*, that may also *control multiple AnonPoP's servers*, even all-but-one servers in a mix-cascade. Significantly, AnonPoP's anonymity holds even when clients may occasionally disconnect; this is essential to support mobile clients.

AnonPoP is *affordable*, with monthly costs of 2¢ per client, and *efficient* with respect to latency, communication, and energy, making it suitable for mobile clients. We developed an API that allows other applications to use AnonPoP for adding strong anonymity. We validated AnonPoP's functionality, reliability, efficiency and usability by experiments using web-based and mobile applications used in 'double-blinded' usability study, cloud-based deployment and simulations.

Keywords: Anonymity, Mixnets, Privacy, Annonymous communication, Tor

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

There have been many efforts to develop, analyze, deploy and use anonymous communication protocols and systems. In particular, the Tor anonymous network [23] is widely used. However, Tor focuses on minimizing latency; to achieve this, Tor design prefers reducing latency to improving anonymity, and is vulnerable to globally eavesdropping adversaries. Several works show that Tor is vulnerable even to weaker attackers, e.g., off-path attackers [28] and malicious servers/clients [3] [10].

The popularity of Tor indicates that it provides valuable service to many users and scenarios, despite its limited guarantees for anonymity (and its non-negligible overhead). However, there are also many scenarios and users that require stronger anonymity properties, even at the cost of somewhat higher latency and overhead.

Several works proposed protocols for stronger anonymity guarantees, as compared to Tor. However, existing research on strong anonymity is mostly impractical. Indeed, many seem to believe that it is infeasible to ensure strong anonymity properties in a practical system for many users, with acceptable overhead and efficiency. We show that it is feasible to have practical, efficient system providing strong anonymity at low costs, supporting large number of users.

Another limitation of Tor is that it provides a *communication channel*, but not a complete *messaging* system. A complete messaging system should also provide 'mailbox' facilities to keep messages until users pick them up; this is also required to prevent detection of a pair of users which frequently communicate with each other and may get disconnected. A naive mailbox solution, where Tor is used to communicate with a mailbox server, would allow the server to de-anonymize users by exploiting Tor's weaknesses, e.g., eavesdropping on particular (suspect) user, and correlating between messages sent/received by this user, and messages received to this mailbox or sent from this mailbox.

In summary, there is large interest in anonymity, and Tor offers some level of anonymous communication; however, neither Tor nor any other practical (existing or proposed) system allows *strongly-anonymous messaging*. This is disappointing; strong anonymity messaging is both necessary and feasible. Messaging is used more and more for business and personal communication, and anonymity is often *required* - for reasons ranging from whistle-blowing to consulting on sexual harassment. And, as we show, strongly-anonymous messaging is *feasible*, since the volume of (text) messages is not

Nethanel Gelernter: Dept. of Computer Science, College of Management Academic Studies, E-mail: nethanel.gelernter@gmail.com
Amir Herzberg: Dept. of Computer Science, Bar Ilan University, E-mail: amir.herzberg@gmail.com
Hemi Leibowitz: Dept. of Computer Science, Bar Ilan University, E-mail: leibo.hemi@gmail.com

very large, and reasonable delays are acceptable. Hence, it is frustrating that such system is not yet operative.

In this paper, we present the *Anonymous Post-office Protocol (AnonPoP)*, a practical anonymous messaging system, designed to ensure strong anonymity, even against strong attackers. We present the design, its rationale and its anonymity properties, and discuss practical deployment aspects, including:

1. Implementation and evaluation in real world environment.
2. Operating costs in cloud environment.
3. Adjustment to mobile devices, with an emphasis on user experience and energy consumption.

Anonymity loves company [22]: hence, the goal of AnonPoP is to provide strong anonymity with superb functionality, usability, reliability, efficiency and low-cost, as necessary to attract many users, and with the scalability required to support millions of users. In particular, AnonPoP uses efficient cryptographic primitives and has acceptable energy consumption, making it appropriate for use on mobile devices. Furthermore, AnonPoP is the *only* proposed anonymous messaging protocol to support client disconnections, a feature which is essential to support mobile clients.

To measure and confirm AnonPoP's low operating costs, we implemented and installed AnonPoP's servers in the cloud, and tested it on hundreds of thousands of clients, communicating anonymously with each other using AnonPoP. We found that the cost of supporting such a large amount of clients is less than a *quarter per user, per year*, or *two cents per month*.

We provide an API for messaging applications to easily add an option for strong anonymity using AnonPoP. With this API, clients of different applications can form one large anonymity set. For example, we used the API to rapidly develop an anonymous Eliza [49] client, demonstrating the use of anonymous messaging for sensitive consulting services.

Readers are encouraged to try out AnonPoP using a mobile application, web-interface and the API [1], to experience the efficiency, acceptable overhead and usability.

Contributions

Our main contribution is the design, development and evaluation of strongly-anonymous messaging protocol, secure against strong adversaries, and showing that it is practical even for mobile clients. We show that AnonPoP ensures better anonymity properties than every other proposed protocols for anonymous messaging, with acceptable overhead and extremely low yearly cost (< 0.25\$ per client). Furthermore, AnonPoP is the first protocol that was designed and shown to be suitable also for mobile devices.

Beyond that, this paper makes the following contributions:

- The *request-pool* technique, allowing 'masking' of periods when a client is disconnected.
- *Per-epoch mailboxes*, limiting the exposure due to client disconnections and due to active tagging attacks.
- The *bad-server isolation* mechanism, allowing isolation of corrupt server (mix or PO), involved in aggressive (non-stealthy) tagging attacks. Other techniques, *timestamps* and *de-duplication*, are used to prevent and/or detect tagging attacks.
- We fine-tune AnonPoP to allow its use in mobile devices, including energy-savings considerations. We conducted double-blinded user-sensitivity experiment to validate acceptable energy use.
- An open-source prototype of AnonPoP, including API for applications, and an Android messaging application that uses this API.

Paper Layout and Organization

In section 2, we start with an overview of AnonPoP architecture and main building blocks. We then define and explain the adversary model and all anonymity notions and properties achieved by AnonPoP in section 3. Sections 4-6 will explain how AnonPoP design and mechanisms handle attacks against wide range of attackers with different resources and capabilities. Section 7 will detail a *'double-blinded'* usability study, showing that AnonPoP is suitable for mobile devices. Section 8 will present a cloud based evaluations conducted under *'real-world'* conditions, demonstrating the feasibility of AnonPoP and discussing operating costs, and the AnonPoP API. We conclude by surveying related work in comparison to AnonPoP in section 9.

2 Architecture and Concepts

The goal of AnonPoP is to support anonymous exchange of messages between clients; messages are packed at the clients into fixed-size *envelopes*, and sent via AnonPoP's servers. AnonPoP uses two types of servers: *Post-Office (PO)* servers and *timed mixes* (see Figure 1). The PO

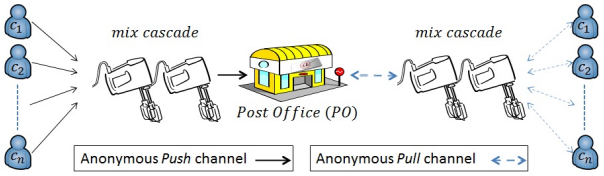


Fig. 1. System architecture of AnonPoP (Anonymous Post-office Protocol). The Post-Office (PO) maintains anonymous mailboxes; clients push and pull envelopes to/from the mailbox anonymously via mix-cascades. All communication channels (represented by arrows) use fixed rates.

and mix servers are expected to operate continuously; clients may disconnect from time to time.

AnonPoP supports multiple Post-Office (PO) servers, where each client selects arbitrarily a PO server for its use. As we later describe, AnonPoP allows detection of tagging attacks by corrupt PO, and we expect clients to move, in this case, to a different PO; however we do not discuss the process of migrating to a new PO, and, for simplicity, our discussion and figures are for a single PO. Similarly, AnonPoP supports an arbitrary number of mix servers, among which the client selects mix-cascades; AnonPoP detects tagging attacks by corrupt mix servers, and clients avoid suspect servers when selecting the mix-cascade, as discussed toward the end of this section.

Envelopes are *onion-encrypted* as they are forwarded and mixed by the mixes. Envelopes can contain either a *push request*, 'pushing' an envelope to a particular mailbox in the PO, or a *pull request*, 'pulling' an envelope from a particular mailbox.

More specifically, mixes operate in synchronized *slots* of τ seconds. Each mix collects all envelopes received in a slot, mixes their order, decrypts requests and encrypts responses, and forwards them so they are all received in the following slot. Clients send envelopes with push and pull requests every round of λ slots. If a client does not have a real message to push, she sends a dummy[43] envelope, which is recognized and discarded by the PO. However, the PO responds to dummy requests as it does for real requests. This ensures *unobservability* against eavesdroppers and even mixes, i.e, an attacker controlling mixes and eavesdropping on the communication, but not controlling the PO, cannot even distinguish between the case that no messages are sent and the case that many messages are sent between different users. Such attacker also cannot identify source or destination, or even link between incoming and forwarded envelopes.

The PO maintains anonymous *mailboxes* containing envelopes sent to different clients. Envelopes received by the PO are still encrypted; hence, a corrupt, eavesdropping PO cannot identify the recipients, senders or contents, provided that the mixes are honest, and users are always connected. (AnonPoP has defenses for anonymity when users may disconnect, as we describe in next section.)

AnonPoP mailboxes are used only for a limited period, called an *epoch*. At the end of every epoch, senders and recipients switch to a new (pseudo)random mailbox, further limiting the potential for a corrupt PO to correlate users of the same mailbox.

AnonPoP has several additional *defenses against corrupt, eavesdropping PO*. Such defenses are only needed when clients may disconnect, or mixes may actively collude with the PO (e.g., in 'tagging' attacks):

Request-Pool: a set of pre-made requests, allowing preservation of anonymity even when a client disconnects. This is especially appropriate for pull-requests, allowing the pattern of collecting envelopes from a mailbox to be independent of possible disconnections of the destination.

Tag-Prevention: mechanisms such as de-duplication and fixed forwarding times of requests and responses, which prevent active tagging attacks, where some of the mixes collude with the PO to identify sender or recipient. However, these mechanisms do not prevent all the active attacks when the attacker controls both the first mix and the PO.

Bad-Server Isolation: this mechanism detects a corrupt (mix or PO) server, or a pair of two 'consecutive' servers s.t. (at least) on of the pair is malicious (and deviates from the protocol). In such cases, clients avoid the corrupt mix or the use of the suspect pair of mixes (i.e., the 'suspect edge'). The bad-server isolation mechanism ensures a strict, low limit on the amount of de-anonymizing queries available to the adversary, in spite of its control over a significant number f of corrupt (mix and/or PO) servers.

AnonPoP has a modular two-layer design. In this work we mostly focus on the lower, *Anonymous-communication (Anon-Comm) layer*, which ensures a *strongly-anonymous communication channel* between clients and the PO. The upper *Anonymous Post-Office Box (Anon-POB) layer* of AnonPoP handles mailboxes for recipients. In this paper, we present only the simple *per-epoch mailbox* design. We believe that further improvements in security, anonymity and efficiency, may

be possible, by more elaborate Anon-POB layer mechanisms, which are beyond the scope of this paper.

Mix selection. For simplicity, in this paper, we assume that AnonPoP clients and servers use a trusted, reliable *server directory service* for path selection; this is merely a simplification, as it is straightforward to implement such a directory service in a distributed manner (avoiding a single point of failure). The directory maintains list of all AnonPoP mix and PO servers, with their public keys and addresses. Furthermore, the directory lists pairs of *connected* servers; a pair of servers (x, y) is *connected* if the directory did not receive report from x claiming misbehavior of y. These misbehavior reports are facilitated AnonPoP's *bad server isolation* mechanism. As long as clients choose paths uniformly (among all valid paths), the probability of choosing a path containing only bad mixes remains very small.

We assume a known upper bound f to the number of malicious (faulty) mix servers; once the directory contains more than f reports for a specific mix or PO server, this server must be indeed corrupt, and is disconnected from all other servers. Clients pick a cascade mix uniformly among all paths consisting of pairs of connected mixes, ending in the desired PO.

Mailbox setup. AnonPoP automatically assigns clients to random mailboxes and generates proper keys. In this paper, we assume that clients have a secure communication channel to perform the initial key exchange. Anonymous key-exchange, e.g. in [25], is a further challenge, which is beyond the scope of this paper.

3 Anonymity Properties

This section presents the anonymity properties ensured by AnonPoP. Section 3.1 introduces the adversary model. Section 3.2 discusses the challenge of defining anonymity properties, arguing that the existing rigorous, formal definitions are too simplified to cover the goals of AnonPoP; in this paper we focus on presenting the AnonPoP design, therefore, extending the formal definitions is left for future work. Instead, in Section 3.3 we present 'informal notions', which we use in this work. Section 3.4 describes the anonymity properties achieved by AnonPoP, using the (informal) notions.

3.1 Adversary Model

As in previous works, we focus on probabilistic polynomial time attackers; this is essential, since our design uses cryptographic mechanisms, which are only secure assuming probabilistic polynomial time attackers, e.g., encryption schemes [6].

AnonPoP design assumes that the attacker has *global eavesdropping* abilities, i.e., can (instantly) observe all communication sent between any of the parties in the system. We also consider *additional* attacker capabilities, mainly, control of the PO and/or some of the mixes, allowing complex, powerful attacks. Although we allow an attacker to controls several colluding servers simultaneously, we make the reasonable assumption that the *number adversarial servers is limited.*

We also consider the communication to be between trusting peers (sender and recipient). In fact, we already work on an advanced POB layer for AnonPoP that will defend against malicious peers, however, this is beyond the scope of this paper.

3.2 Challenges of Defining Anonymity

Intuitively, *anonymous communication* means the inability of identifying specific (communicating) entities among the set of potentially-communicating entities. Multiple variants were considered by researchers and practitioners, e.g., unobservability and sender (and/or recipient) anonymity. One widely-used interpretation [42] is that *sender (recipient) anonymity* refers to the inability of an attacker to identify the sender (recipient) of a message among a set of potential senders (recipients), and *unobservability* refers to inability of an attacker to know whether there was any communication at all. These are useful, intuitive notions; however, they are not sufficiently formal to allow rigorous proofs of security.

Transforming such informal, intuitive notions into precise, well-defined, formal definitions, is a non-trivial challenge. There have been multiple attempts to present appropriate formal definitions, including [2, 9, 26, 27, 30, 32–34, 41, 45, 47]. These definitions differ in multiple aspects, and in particular, in the capabilities of the adversary, and in what constitutes a successful attack.

Unfortunately, when trying to apply these definitions to analyze AnonPoP, we realized that each of them, fail to satisfy all of the following three important aspects of AnonPoP's design and goals:

1. AnonPoP's goal is to ensure strong anonymity properties - against *active, adaptive adversaries*, who eavesdrop the entire network and control a majority of AnonPoP's servers. Existing definitions are for passive, static adversaries.

***Honest* server**	**Anonymity property**	**Attack: Defense (claim)**	
		Without disconnections	**With disconnections**
Only *first* push mix	**Sender anonymity**	**Passive: Prevent (2)** **Active: Prevent (3)**	Heuristic defense, see Section 6.2 and Appendix A
Only *non-first* push mix		**Passive: Prevent (2)** **Active: Detect (4)**	
Only *first* pull mix	**Recipient anonymity**	**Passive: Prevent (2)** **Active: Prevent (3)**	**Passive: Prevent (5)** **Active: Detect (6)**
Only *non-first* pull mix		**Passive: Prevent (2)** **Active: Detect (4)**	
Only PO	**Unobservability**	**Passive and Active: Prevent (1)**	

Table 1. Anonymity properties achieved by AnonPoP against a globally-eavesdropping attacker, who controls all servers along the path, except as indicated in the 'honest server' column. In parentheses: number of relevant claim.

2. AnonPoP supports (limited-duration) client disconnections, an aspect which is not addressed by existing definitions, yet is crucial for a practical system, supporting mobile devices.
3. AnonPoP is provides different levels of anonymity against attackers with different capabilities; this is not compatible with current formal definitions. In particular, against some (strong) attack models, AnonPoP can 'only' ensure that attackers become increasingly isolated, hence deterring such attacks and reducing likelihood of successful attack; this is not captured by any of the existing formal definitions.

Since our focus in this work is on system design, we decided to only follow 'the spirit' of the existing definitions of anonymity, and use intuitive notions of anonymity (as we present in subsection 3.3) instead of formal definitions; future work should extend existing definitions to provide a well-defined notion of practical anonymity.

3.3 Informal Anonymity Notions

Our notions follow [27], which addressed the challenge of defining anonymity properties in the presence of active, adaptive adversaries who might control some of the protocol participants. We believe that the model of [27], which extends [33], may be a good basis for a complete formal model and analysis of AnonPoP - in future works.

We begin by presenting the notion of *unobservability.*

Notion 1. (Unobservability) *A protocol achieves* unobservability *against a globally eavesdropping attacker that controls a set S of the servers, if the adversary cannot (with significant advantage and in efficient time) distinguish between* any *pair of scenarios.*

To present the (slightly weaker) notions of sender and recipient anonymity, we first present *sender/recipient permuted pairs.* Consider two scenarios σ_0, σ_1, where all the recipients receive the same messages in σ_0 and in σ_1, but the senders in σ_0 are a constant permutation (chosen by the attacker) of the senders in σ_1. Namely, given a permutation π chosen by the attacker, when a honest client i needs to send a message to recipient r in σ_0, the honest client $\pi(i)$ needs to do the same in σ_1. We say that σ_1 and σ_0 are *sender permuted pair.*

Similarly, the term *recipient permuted pair* refers to two scenarios where the *recipients* in one scenario are a permutation of the recipients in the second scenario, and the senders are identical.

Notion 2. (Anonymity) *A protocol achieves* sender (recipient) anonymity *against a globally eavesdropping attacker that controls a set S of AnonPoP's servers, if the adversary cannot (with significant advantage and in efficient time) distinguish between any sender (recipient) permuted pair of scenarios.*

These notions pose great challenges because they also consider extreme scenarios that might not occur in reality. For example, for unobservability, the adversary must not be able to distinguish between any two scenarios, even a scenario where all parties send messages vs. a scenario where nobody sends any message. These are strong anonymity requirements. In particular, surely they do not hold for any low-latency solution such as Tor, since adversary can easily distinguish between the scenarios. Furthermore, we allow also corruption of different subsets including most of the servers, as follows.

Detection/Isolation. Existing formal definitions of anonymity, e.g., in [27, 33], require complete prevention of attacks. However, sometimes prevention is infeasible or hard/expensive, and a *detection/isolation*

approach is sufficient to deter attackers and hence to ensure anonymity. In particular, in AnonPoP - and possibly in other efficient strong-anonymity solutions - the PO may 'signal' the use of a particular mailbox, by intentionally dropping responses (or, equivalently, ignoring requests); such 'signal' seems almost unavoidable for the model where the PO keeps mailboxes, yet, we show that every such abuse is detected and isolated to a specific rogue entity (e.g., the PO).

Intuitively, our goal is to ensure that every 'bit' of information collected by the adversary has a 'high price'. We present an intuitive notion, which is somewhat tailored to the AnonPoP model.

Notion 3. *Let $X(f), Y(f)$ be two functions (in the number of corrupt servers f), and let x_c be the number of bits that the adversary can learn on (honest) user c (communicating only with honest peers), and $y_c = \max\{0, x_c - X(f)\}$. A protocol achieves* (X, Y)-attacker-isolation *if, with high probability, $\sum_c y_c \leq Y(f)$.*

Intuitively, the attacker can learn up to $X(f)$ bits 'per client', plus up to $Y(f)$ bits additional (for all clients); in AnonPoP, $X(f) = f$ and $Y(f) = f(f+1)$, as we show below.

3.4 AnonPoP's Anonymity Properties

We first consider the anonymity properties of AnonPoP, as a function of the malicious servers along the paths between the senders and recipients, as summarized in Table 1. As indicated in the bottom row, it suffices for *only the PO* to be honest for AnonPoPto ensure complete *unobservability*. Similarly, it requires *only one mix* in the push (pull) channel, to provide protection for sender (resp., recipient) anonymity, if clients never disconnect. *Recipient anonymity* is protected even when clients may disconnect (for reasonably-long periods), by using the *request-pool* mechanism of Section 6.1. In particular, AnonPoP resists *intersection and correlation attacks* [7, 8, 37, 52] between senders and recipients.

When clients may disconnect, AnonPoP cannot fully ensure sender anonymity, as an eavesdropping PO can launches intersection and correlation attacks to correlate between senders and mailboxes. Instead, AnonPoP includes a heuristic-mitigation mechanism, *per-epoch mailboxes (PEM)*, to improve sender anonymity. For discussion of the protection provided by PEM, see Section 6.2 and experimental evaluation in Appendix A.

As shown in the table, when the *first push (pull) mix* is honest, and without disconnections, AnonPoP completely prevents even active attacks on *sender (resp., recipient) anonymity*. However, if the first mix is malicious then AnonPoP can only *detect* active attacks on sender (resp., recipient) anonymity, and only when some other (non-first) push (pull) mix is honest. AnonPoP can similarly only detect, not prevent, sender anonymity, even when the first mix is honest, if disconnections are possible; recipient-anonymity is fully protected (thanks to the request-pool mechanism).

However, AnonPoP detection are very effective; in each detection, the client detects that the first mix is malicious, and/or at least one edge connecting a malicious server and another (malicious or honest) server is removed from the graph of AnonPoP servers maintained by the directory. Hence the amount of 'deanonymization queries' is very limited (specifically, $O(f^2)$).

4 AnonPoP Basic Defenses

AnonPoP uses *onion encryption* [14, 29], for both push and pull channels, as illustrated in Figure 2. Namely, requests are onion-encrypted, i.e., encrypted using the public key of the PO, and then, consecutively, by the public keys of the mixes. Onion-routing trivially protects the confidentiality of requests, since requests must be decrypted by all mixes, and finally by the PO.

To further prevent linkage between a request entering a mix, and the corresponding (decrypted) request output by the mix, each mix *buffers* all messages received during a slot, randomly permutes them and sends them in the following slot.

To similarly prevent linkage between incoming and outgoing responses, each mix *encrypts* the responses. We use authenticated-encryption [5], allowing the client to also validate that the response was sent by the PO, over the specified sequence of mixes; the PO sent the response upon receiving the corresponding request from the client. To facilitate the authenticated-encryption of responses, clients include the authenticated-encryption key to be applied to the response in each onion-encryption layer (denoted key_1, key_2, key_3 in Figure 2).

AnonPoP uses two additional layers of encryption, on top of the onion encryption. First, messages pushed to a mailbox, are encrypted for the destination; namely, when the PO decrypts the final onion layer, it finds only a mailbox identifier and an encrypted message. Second, all the communication between every pair of adjacent entities (adjacent mixes, last mix and PO, or client and first mix) is authenticated and encrypted.

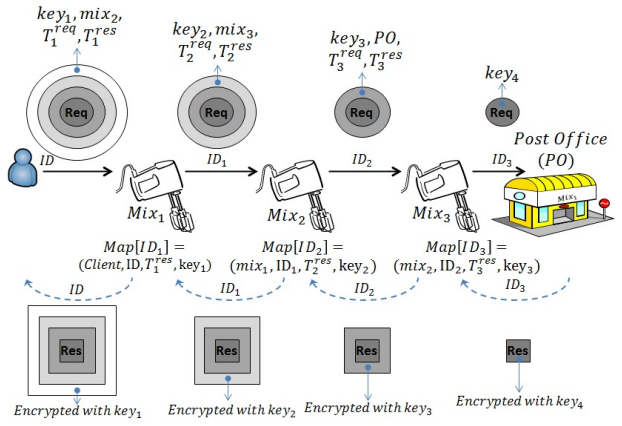


Fig. 2. Onion-encryption with cascades of mixes, as used by both push and pull channels. The circles above the straight lines mark the route of the request from the client to the PO. The response route is illustrated by squares below dashed curves. The senders encodes in each onion layer a key (key_i) which Mix_i uses to authenticate and encrypt the response, and timestamps T_i^{req}, T_i^{res} specifying when the request and the response are expected to arrive (see Section 5). The sender and mixes select each a random identifier ID (for sender) or ID_i (for i^{th} mix), and store relevant parameters (key_i, T_i^{res} and received ID) in table Map indexed by chosen ID.

To prevent traffic analysis attacks, AnonPoP uses *padding*. First, all requests and responses use fixed-sized envelopes (padding and fragmenting as needed). Furthermore, AnonPoP maintains fixed rate of transmission of envelopes, independent of the pattern of requests from the users. Namely, AnonPoP clients queue outgoing messages if their rate exceed the fixed rate, and pushes *dummy (empty)* envelopes, if no outgoing messages are queued. Dummy envelopes are treated exactly like 'real' messages; even mixes forwarding onion-encrypted messages cannot distinguish between real and dummy envelopes. Dummy envelopes are sent to a reserved mailbox identifier, allowing the PO to drops them (after decrypting the last onion layer); the PO responds with a dummy response, which is indistinguishable from a 'real' response, until decrypted and discarded by the client.

As a result of these padding mechanisms, AnonPoP's traffic is fixed, independently of the actual pattern of messages sent by clients. Together with the onion-encryption, this ensures *unobservability* as long as the PO is not corrupted, i.e., against a globally-eavesdropping attacker, which controls mixes and users. This is simply because only the clients and the PO can distinguish between dummy and real messages.

Claim 1. *AnonPoP ensures* unobservability *against a global-eavesdropping adversary which further controls any subset of mixes and users, and has the ability to disconnect users, as long as the PO is not corrupted.*

Argument: This follows by reduction to the indistinguishability property of (1) the public key encryption scheme $\mathcal{E}_{\mathcal{PK}}$, used to encrypt requests to the PO, and of (2) the shared-key encryption scheme $\mathcal{E}_{\mathcal{SK}}$, used by the PO to encrypt responses. Namely, assume some (efficient) adversary A is able to distinguish between some two scenarios of message sending S_0, S_1. We first check if A also distinguishes between scenarios S_0', S_1', which are the same as the corresponding S_0, S_1 except that responses from the PO are all (encryptions of) some fixed message m. If A succeeds, we use it as oracle to distinguish $\mathcal{E}_{\mathcal{PK}}$; otherwise, we use A to create oracle to distinguish $\mathcal{E}_{\mathcal{SK}}$ (utilizing the fact that A fails to distinguish between S_0' and S_1', yet succeeds to distinguish between S_0 and S_1). Notice that the reduction works since the pattern of transmissions is fixed, independently of input messages. □

The padding and onion-encryption mechanisms also suffice to ensure *sender and recipient anonymity*, against a *passive* ('honest-but-curious') attacker, as long as at least one of the mixes in the corresponding channel is not corrupted. However, this holds only when clients are always connected, since a (passive) corrupted PO may be able to identify a mailbox used by a client which is disconnected (e.g., using intersection attacks). We later present additional mechanisms of AnonPoP that provide anonymity also when clients may disconnect.

Claim 2. *When clients are always connected and some push (pull) mix is non-corrupted, AnonPoP ensures* sender (recipient) anonymity *against passive attackers.*

Argument: We present the argument only for sender anonymity; the argument for recipient anonymity follows similarly. The argument is by reduction to the indistinguishability of the public key encryption scheme $\mathcal{E}_{\mathcal{PK}}$, used to encrypt requests to the mixes, and in particular, to the non-corrupt mix. Namely, assume some (efficient) adversary A is able to distinguish between some two scenarios of message sending S_0, S_1, where the number and length of messages sent to each mailbox (recipient) are identical (sender anonymity). Due to AnonPoP's padding mechanisms, the pattern of transmissions is fixed, independently of input messages; and due to the operation of the (non-corrupt) mix, the order of requests arriving at the PO is random. Hence, A

provides an oracle allowing (efficient) distinguisher for $\mathcal{E}_{\mathcal{PK}}$. □

It may seem, at first sight, that the padding and onion-routing mechanisms also suffice to ensure anonymity against active attacks, if the *first* mix in the corresponding channel is non-corrupt, since a non-corrupt first mix suffices to randomize the order of requests (responses) reaching the other mixes and the PO. However, this relying would be vulnerable to PO *tagging attacks*, where the number or timing of responses are tampered to allow deanonymization. In the following section, we discuss tagging attacks and the defenses against them in AnonPoP.

Forward and proactive security. Finally, we note that it is quite simple for AnonPoP to also ensure *forward secrecy and proactive security*. Forward secrecy refers to protecting past communication from future server compromise, and have been discussed in circuit related system such Tor [39]. To ensure forward secrecy, AnonPoP simply uses forward-secure encryption scheme, e.g., [12]. Proactive security refers to recovering security (and secrecy) following a temporary compromise of a server; to ensure this, use the method of [11] to recover security for the keys used by each server.

5 Anti-Tagging Defenses

The onion-encryption and padding mechanisms described in Section 4, cannot fully protect anonymity against a rogue PO. There are few ways in which a rogue PO, possibly colluding with some mixes, can 'tag' a request and/or the corresponding response, allowing it to link between the client sending the request (and receiving the response) and the mailbox to which the request was sent. In this section, we present AnonPoP's defenses against such tagging attacks.

In the first subsection, we present AnonPoP's *timestamping and anti-duplication* mechanisms, which *prevent* tagging attacks when the first-mix is honest, and *always detect* tagging attacks. In the second subsection, we present the *bad server isolation* mechanism, ensuring that whenever a tagging attack occurs at a particular round, not only is it detected, but at least one edge connected to a malicious server is removed from AnonPoP's graph of available links. The next section extends the mechanisms presented here, to also handle disconnections of clients.

5.1 Timestamps and Anti-Duplication

The basic anti-tagging mechanism in AnonPoP, is to include *timestamps* in every layer of the onion; the request (respectively, response) timestamp for the i^{th} mix is denoted T_i^{req} (T_i^{res}).

Non-corrupt mixes always return an encrypted response *exactly* on the time specified in the timestamp field; if the expected response is not received on time, then the mix returns an appropriate error report. A response received too late (or too early) is dropped. Note that error reports sent when a response is not received on time, are indistinguishable compared to the 'real' responses. The adversary cannot learn whether an encrypted response hides either 'real' response or 'error report'.

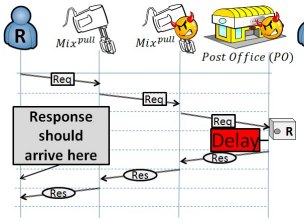
Figure 3(a) depicts a tagging attack by delaying the response; in this attack, the PO delays the response for a single request and can detect which client gets her response one slot later. Figure 3(b) depicts the effect of fixed response time; honest mixes return response in time and discard late responses, hence the attacker sees no anomaly in the communication pattern of the client.

To further detect *duplicate* requests and responses, received at the same (correct) time slot, each AnonPoP mix uses the key it receives key_i, as a unique identifier. If a mix receives multiple requests with the same key (at the same slot), then it discards all but one of them, sending back an error-report containing the plaintext and randomness, allowing previous mix to validate the collision of keys. Similarly, a mix discards all but one response for each forwarded request. (Random collisions occur with negligible probability.)

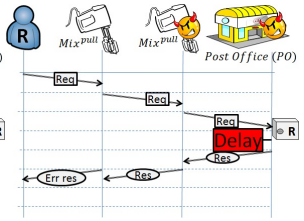
Figures 3(c,d) depict duplicate-request attack, when only a single (non-first) mix is honest, and the effect of the anti-duplication mechanism. Figure 3(e,f) depicts similar duplicate-response attack and its prevention; this attack (and defense) also applies for the case where (only) the first mix is honest.

Claim 3. *AnonPoP ensures* sender (recipient) anonymity *against active attackers, provided that the first push (pull) mix is honest and that clients are always connected.*

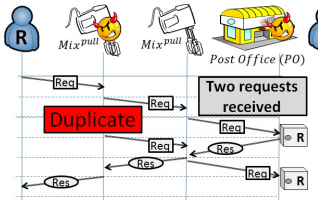
Argument: Due to the padding mechanism, requests are sent exactly once a round, with fixed size. The (honest) first mix shuffles these requests, hence, the following mixes, and the PO, cannot link between the client and a specific request from the first mix. The traffic *from* the first mix *back* to the client is also fixed, since the mix returns response *exactly* on the time specified in



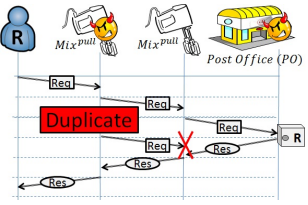
(a) Delaying the response arrival. Only the first pull mix is honest.



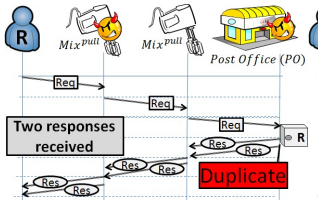
(b) Defense by returning an error report indistinguishable from a real response on the expected response time.



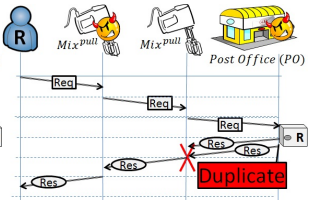
(c) Sending a duplicated request in a following slot. Only one non-first pull mix is honest.



(d) Defense by dropping the duplicated request that arrived on wrong slot.



(e) Sending a duplicated response in the same slot. Only one non-first pull mix is honest.



(f) Defense by dropping the second response received.

Fig. 3. Attacks to correlate recipient and her mailbox (on the left side) and their defenses (on the right).

the time-stamp field; due to the anti-duplication mechanisms, only a single such response is sent (and only on that slot).

The encryption applied to messages ensures that eavesdroppers and other mixes cannot link between the sender and the (encrypted) requests. Additionally, in every two scenarios that are different only in the senders (recipients), the same number of messages is pushed (pulled) to (from) mailboxes that differ only by their pseudonym, so the PO cannot distinguish between the two scenarios. Notice, that in this case, delaying or blocking of an encrypted message can be done only when all the messages are already shuffled by the honest first mix, and hence, such active attacks are not helpful and Notion 2 holds. □

5.2 Bad Server Isolation

An attacker who controls both the first mix and the PO can drop, delay or corrupt requests and/or responses to correlate between clients and mailboxes. The first push (pull) mix knows the originators of every push (pull) request, and the PO knows how many messages reach each mailbox. Even when clients do not disconnect, the PO and the first mix may try to match clients to mailboxes, by dropping or delaying requests and/or responses; this may allow an intersection attack; see Appendix A.1.

The *bad-server isolation* mechanism, allows AnonPoP to efficiently deter active attacks involving a rogue first mix and the PO. Previous works, e.g. [24] discussed the complexity of achieving such goal. The mechanism extends the fixed-response time mechanism. Suppose mix M detects that it did not receive the expected response in its specified time from the 'next server' (mix or PO), say denoted X. Then M encrypts and sends back a signed and time-stamped *problem report*, stating the relevant identities (M, X); the (signed) problem report is also deposited at the AnonPoP directory.

As a result, *all* clients and servers, quickly learn to avoid using the pair (M, X) as part of AnonPoP routes. Namely, a rogue, active server, 'loses' one of its edges to other servers, per each slot in which it uses such 'aggressive, detectable' tagging attack.

As the path is chosen uniformly among all the reliable paths, dropping connections with up to f honest mixes prevents the choice of many paths where at least one mix is honest, and by that, increases the probability of choosing a path where all the mixes are malicious. Yet, while the fraction of malicious mixes is low, the advantage gained by the attacker is not significant; see Appendix B.

Claim 4. *When clients are always connected, AnonPoP achieves* $((f, f \cdot (f+1))$*-attacker isolation.*

Argument: A server reported by $f+1$ servers, where f is bound on number of malicious servers, is definitely malicious and not used in any channel. Note that the attacker cannot *frame* an honest server. Note also, that since we assume that at least one mix along the path is honest, it follows that the PO and first mix can only signal to each other via the absence of a request/response, i.e., *one bit per round*; they cannot, for example, use content-based signaling. For each such learned bit, a disconnection occurs between one malicious mix and another mix, either due to false report by the attacker or due to a real report by the honest mix. Beyond that, each of the mixes can operate as a first mix for each

of the clients and to drop the message to tag the user without blaming the next mix. In such case, the first mix will not be disconnected from another mix, but the client knows for sure that the first mix is malicious and hence disconnects from it. Theoretically, this allows each of the f mixes to tag each of the users once. The number of bits can be learned according to each of the cases satisfies Notion 3. □

6 Handling Disconnections

Clients using mobile devices may often disconnect from the network. Such disconnections may be observable - sometimes even controlled - by the attacker, allowing an attacking PO to correlate between clients and the mailboxes they pull from, using *intersection and correlation attacks* [7, 8, 37, 52]. In particular, if a pull request reaches some mailbox, an eavesdropping PO can learn that all the clients who were offline when the request arrived are not the owner of the mailbox. By repeating this procedure over time, the adversary may correlate a single recipient with the mailbox.

Section 6.1 describes the *request-pool* mechanism that ensures recipient anonymity, provided that clients do not disconnect for excessive time periods. In Section 6.2 we discuss the challenge of ensuring sender anonymity when client might disconnect, explaining why using request-pool also for push requests may be problematic. We then introduce a simple mechanism, *per-epoch mailboxes (PEM)*, which heuristically improves the sender anonymity, although not satisfying our notion for sender anonymity (Notion 2). We believe that further research on mailbox mechanisms may further improve AnonPoP's anonymity defenses.

6.1 Request-Pool

We now explain *request-pool*, a simple yet effective technique allowing AnonPoP to extend its recipient-anonymity defenses to the case of (reasonably-limited) client disconnections, and in particular, to foil intersection and correlation attacks. We focus on pull-requests, where the defense works better, utilizing the fact that pull requests do not contain content that depends on the mailbox status. Specifically, each client prepares and sends to the first pull-mix a 'pool' of pull requests to be used in future rounds, even in rounds where the client is disconnected.

When a client is connected, the first pull-mix has this 'pool' containing μ pull requests, prepared in advance, for the μ next rounds. As long as the client remains connected, in every round, one pull request is used to retrieve a message, and the client provides a new pull request, maintaining μ requests in the 'pool'.

This 'pool' allows the first pull-mix to send a pull request for the client, even in rounds in which the client is disconnected (up to μ consecutive rounds). The mix also holds all the encrypted responses received from the PO; the mix does not know whether the responses are real or dummy.

When a client reconnects after being disconnected for $x \leq \mu$ rounds, it contacts the first pull-mix to retrieve messages kept by the mix from the previous x rounds. The client also sends to the mix $x+1$ new pull requests, to be used in future rounds, replenishing the 'pool' of μ requests. In every other round, the client sends the pull request that will be used μ rounds after the current round.

Claim 5. *AnonPoP ensures* recipient anonymity *against passive attackers when some pull mix is honest, even when clients may disconnect, if clients do not go offline for more than μ consecutive rounds.*

Argument: Since the adversary is passive, the traffic from/to the first pull mix to/from the PO is fixed (as though there were no disconnections). There might be a peak in the traffic between the first pull mix and the client immediately after the client reconnects, i.e., the rate of traffic between the client and first mix is not completely fixed. However, the traffic is still independent of the actual number of messages sent and received, and depends only on the connectivity of the clients. Since the connectivity is known to the eavesdropping attacker, then this mechanism exposes no additional information. Hence, there is no information leakage.
Since clients not offline for more than μ rounds, recipient anonymity is achieved against passive attackers according to Notion 2, provided that (at least) one pull mix is honest. This is because all the pull requests and the responses for the requests arrive to a honest mix that forwards them shuffled, and the adversary cannot correlate between incoming messages and outgoing mixed messages. □

Claim 6. *AnonPoP achieves $((f, f \cdot (f+1))$-attacker isolation, even when clients may disconnect, provided $f << n$ and clients do not go offline for more than μ consecutive rounds.*

Argument: As stated by Claim 5, while clients do not disconnect for more than μ consecutive rounds, and the traffic reaches the servers intact on time, recipient anonymity is ensured. As stated by Claim 4, the mechanism described in Section 5.2 allows detection of active attacks; under the same conditions, while the rate of pull requests remains fix, Notion 3 is satisfied with regarding to recipient anonymity against active adversaries. □

6.2 Anonymity of Disconnecting Senders

Unlike pull requests that are sent by each client to her own mailbox in a fixed rate and can be prepared in advanced, push requests are sent - to specific mailboxes - according to the current needs of the user. Therefore, AnonPoP cannot precisely predict push requests in advance. Hence, we do not use request-pool for push requests, and when clients may disconnect, the mechanisms described so far do not protect sender-anonymity against intersection and correlation attacks [7, 8, 37, 52].

Notice that in practice, when there are many clients, it might take considerable time to learn information about the sender; however, this is still feasible, hence AnonPoP does not ensure sender anonymity. For example, the adversary can choose two sender-permuted scenarios in which only a single client receives; in the first scenario, only client a sends the messages, and in the second scenario, only client b sends. Obviously, if one of the scenarios is simulated, and the adversary observes that only a or only b are online, she can detect the identity of the sender. This is by simply checking whether any message reached some mailbox or not (the adversary controls the PO). In this extreme case of a single sender, the adversary can simply correlate the incoming messages with the single sender because she knows that no other messages were sent by other clients. In reality, when there are always many clients online, and when messages are sent by many of them, it is significantly harder to detect the sender.

To defend - heuristically - sender-anonymity even in case of disconnections, AnonPoP implements *per-epoch mailboxes (PEM)*. Namely, clients change their mailboxes every fixed number δ of rounds, called *epoch*. PEM does not completely ensure sender-anonymity, but it decreases the amount of data learned by the PO. Furthermore, it ensures the anonymity guarantees of AnonPoP in an epoch, among all the clients that stay online in it.

PEM improves the resistance to sender-mailbox intersection attacks. In Appendix A, we describe a set of simulations we did to evaluate the resistance of AnonPoP to a classical intersection attack with and without PEM. The results showed that even for epochs of several hours, the use of PEM seems to provide significant resistance to the evaluated intersection attack.

7 Energy Consumption

Support for mobile clients is critical for success of anonymous messaging, however, also challenging. In particular, users of mobile devices are reluctant to use energy-hungry applications, hence, AnonPoP is optimized to save energy. We briefly describe one of our energy-optimizations, and our experimental evaluation of energy requirements.

7.1 Saving Energy with Lazy Pulling

In a naive implementation, clients maintain an open TCP connection to the first mix from sending request till receiving response , allowing the server to immediately respond to client requests, with the response reaching the client from the first mix, using that open connection. However, the open connection would prevents the mobile device from moving to the energy-saving 'sleep' mode. To reduce energy consumption, AnonPoP uses *lazy pulling*, where clients use only short connections. In lazy pulling, the first mix in every channel acts as a proxy for the responses from the PO, such that every round, the client sends requests to the first mixes in each of the channels, and on the same connection, retrieves the responses for the requests of the previous round; see Fig. 4.

Although it may not be obvious, lazy pulling, as described above, results in the same average latency as with immediate pulling. See [1] for detailed analysis.

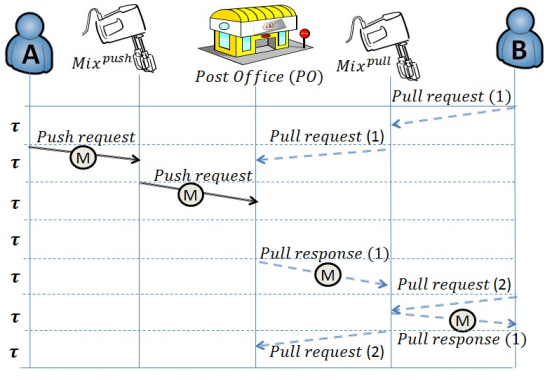


Fig. 4. Lazy pulling procedure in AnonPoP scheme with one mix in each channel; i.e., $C = 1$, and 5-slots rounds ($\lambda = 5$).

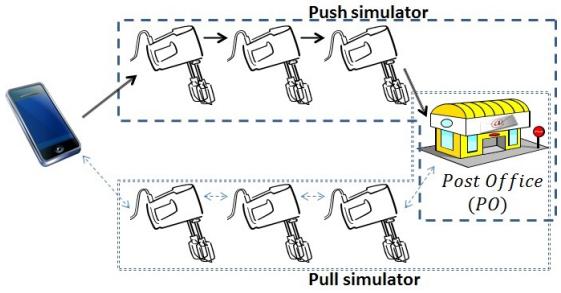


Fig. 5. Network topology in the experiment

7.2 Evaluating Energy Consumption

We implemented a prototype of the AnonPoP client application for the Android operating system, and measured its performance by simulating the communication channels. We used the prototype to perform several experiments to fine-tune AnonPoP's energy consumption, and to validate that they are unlikely to cause noticeable increase in the energy consumption of mobile devices.

We briefly describe a user study we conducted to test the impact on the user experience of Android phone users, as result of using the AnonPoP application. Details of this and other experiments appear in [1]. The topology of the network in the experiment included three mixes in each channel and one PO. Figure 5 depicts the experiment topology.

The user study was conducted with the participation of 20 smartphone users. We wanted to test how the different implementations (using asymmetric or symmetric cryptography) affect the user experience. We created an application that runs one of three states: (1) using *Asymmetric* cryptography ('real' AnonPoP), (2) using *Symmetric* cryptography, and (3) *cryptography disabled*. During (every) installation, the application randomly chose one of the three states.

The experiment participants reinstalled the application every week (to change state randomly) for eight weeks. At the end of every week the participants were asked to rate their user experience with a focus on the battery life, compared to the previous week. The experiment was conducted *double-blindly*; both our team and the participants did not know which states were assigned to each of them during the course of the experiment. In the end of the experiment, we compared the real changes in the states and the feedback by the users. The experimental results serve to strengthen our hypothesis: AnonPoP overhead does not create a significant degradation in the usability for smart-phone users.

8 Implementation & Evaluations

In this section, we first describe our implementation, focusing on AnonPoP servers and the cryptographic primitives we used. We then show that the AnonPoP implementation is practical, by evaluating it under real-world conditions, including cost analysis of the system using commercial cloud services. We conclude this section by outlining the simplicity of creating applications on top of AnonPoP using our API, including a short overview of our demo messaging application.

8.1 Implementation

AnonPoP servers implementation. We implemented AnonPoP in Java, because of its portability to different platforms. During the development process, we used several techniques to support as many clients as possible. The most significant optimization we used to support many clients, was done by dealing with the OS kernel. In order to fully utilize the server's ability, all sorts of configurations have to be made to the kernel. Tuning of TCP and other settings, significantly improved performance.

Cryptographic primitives. Our implementation for the push and pull channels uses a simple four-layer onion for each request, using a hybrid encryption scheme. See Section 4 and Figure 2.

For shared-key encryption, we use a simple authenticated encryption scheme with AES/CBC/PKCS5 padding. The key size is 48 bytes, consisting of 128-bit AES key and 256-bit HMAC-SHA key. For the public key encryption scheme we used RSA with 1024-bit key. For the push and pull request onions, the cryptographic overhead is slightly more than 1KB. The overhead for the push and the pull response onions is 256 bytes. As unique identifiers of messages and mailboxes we used 128-bit tokens.

We expect further improvements in performance by moving to elliptic-curves cryptography, by using the efficient and compact Sphinx [19] design.

8.2 Evaluation in the cloud

We used Amazon's cloud services c4.8xlarge Linux machines with 36 virtual CPUs and 60GB of memory. Our evaluation was done on the simple topology of three mixes in each channel and a single PO (see Figure 5), with extra machines that simulated the clients. We con-

figured the instances such that every pair of communicating machines will be located on different continents, so that we can emulate worst-case scenarios. The PO, the second mixes, and the machines that simulated many clients were placed in Europe. The first and third mixes in each channel were located in the US.

Throughout our experiments, we used slots of $\tau =$ 30 seconds, rounds of $\lambda = 10$ slots, and epochs of three hours.

We began to run the protocol against $100,000$ concurrent clients. We repeated the experiment, gradually increasing the number of clients until the failures rate was higher than 0.001%. Our implementation was able to support up to $500,000$ concurrent users with only sporadic failures due to failures of clients to open a connection with the first mix.

Our AnonPoP implementation uses a 1KB message size and round length of 5 minutes. The 1KB size is suitable for most textual messaging services, especially in the realm of mobile communication. The length of the rounds was also selected to trade-off latency with energy consumption, which is critical for mobile devices. Short rounds would prevent the device from sleeping, and hence, may significantly increase energy consumption as well as bandwidth. Figure 7 demonstrates the effect of payload size and round length in terms of costs, which shows that our choices are sensible.

8.3 Costs Evaluation

Running AnonPoP servers in the cloud is not expensive. Each of Amazon's instances costs are depended on several variables: location, type of payment and bandwidth usage. Significant discounts are received for reserving instances for long periods. Reserving c4.8xlarge instances for the first and third mixes in the US and for the second mixes and the PO in Europe has a yearly cost of $60K$\$.

In addition to the machine costs, there is a payment for the traffic generated by the machines. There is no need to pay for traffic coming from the Internet, but there is a changing cost for outgoing traffic. The cost begins from 0.09\$ per 1GB and decreases as the amount of outgoing traffic increases.

When a client sends push and pull requests to the first mixes, there is no cost for the system. However, each of these messages travels through the mixes and PO, generating outbound traffic of around 14.7KB per client per round. The maximal communication volume in the system for a client is 1.47GB.

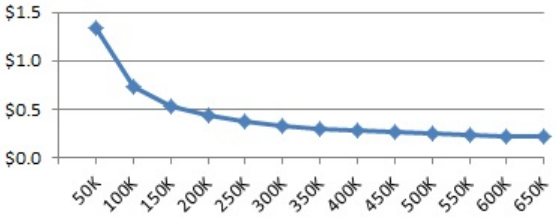


Fig. 6. Yearly cost (\$) per client as a function of the number of clients using AnonPoP.

In the calculation of the yearly cost of the system, there are two factors: (1) the yearly cost of the instances, and (2) the yearly cost for the traffic for all the clients together. While the first factor does not directly depend on the number of clients and can be referred as a constant, the second factor depends on the number of clients because the cost per GB decreases as the total amount of traffic increases. Both of the components reflect the yearly cost of running AnonPoP's servers. We divide the yearly cost for the machines and the traffic by the number of clients to get the yearly cost per client.

Using the instances we chose, the yearly cost begins from 1.4\$ per client for $50K$ clients, and decreases rapidly to less than a quarter for $500K$ clients. Figure 6 depicts the yearly cost per client as a function of the number of clients. Notice, that the calculation was based on using strong and relatively expensive instances even for low number of clients. In practice, for fewer clients, weaker and cheaper machines can be used to further decrease the cost.

8.4 Applications and API

In order to decouple any dependency in the layers of the protocol, we developed an API that relieves any direct interaction with AnonPoP layers. The API autonomously maintains the connectivity, sends dummy messages when needed, handles the encryption/decryption and generally, acts as a friendly intermediator between the application and the infrastructure of AnonPoP. The bottom line is that any application or service can use AnonPoP as a "carrier" to deliver the data anonymously, and the application only needs to encode the information on one end, in a way that it can decode it easily on the other end. Everything else is taken care of by the API. See [1] for the API, as well as the application and documentation.

AnonPoP has several envelope types, such as push/pull requests and push/pull response, etc. These envelopes are padded according to the padding mech-

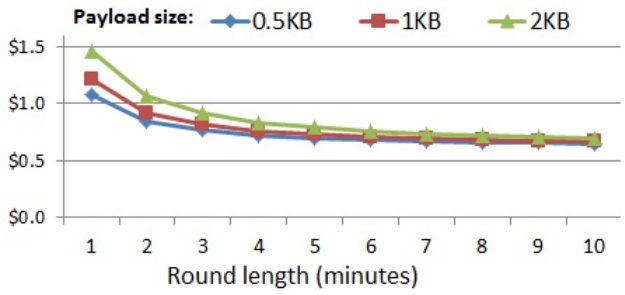


Fig. 7. Yearly cost ($) per client as a function of the payload size and round length, using c4.8xlarge machines with 100,000 concurrent users.

anism mentioned earlier, in order to prevent attackers from tracking messages end-to-end. All envelopes of the same type are of the same size[1] Short envelopes to the fixed size (e.g., 1KB), and, when necessary, AnonPoP sends 'dummy envelopes' to maintain fixed-rate sending.

We address these issues as follows:

1. Small messages can (sometimes) be merged into joint envelopes. If a user has sent a few messages to the same recipient, and these messages hasn't left the sender's device yet (because their designated slot hasn't arrived yet) they can be merged (with the 1KB limitation in affect). When the messages will arrive to the recipient's device, it will unpack the messages to their original form.
2. We use compression algorithm to enable sending actually more than 1KB of data, depend on the possible compression rate of the payload. Furthermore, oversized messages are automatically broken down to multiple envelopes and merged back at the destination.

We implemented a basic demo messaging application that communicates via AnonPoP servers to achieve unobservable messages communication. Using the API, the application is implemented both as a mobile Android application and as a standard Windows application. We also implemented an Eliza [49] AnonPoP client, allowing users to use the service anonymously.

When two people wish to interact with each other for the first time, they need to perform a basic key exchange. Instead of exchanging long identities, which can be cumbersome, the application includes an identity exchange protocol over AnonPoP servers, so the users are only required to share a shorter 128-bit symmetric key. This key can be generated and shared either via the user-interface, or using a secure key-setup mechanism, e.g., [25].

1 To further increase the anonymity set, at small price of extra bandwidth, pad all types to be same size.

9 Related Work

We now briefly compare AnonPoP to other works dealing with anonymous communication. We focus on works whose goal, like AnonPoP, is to provide anonymity against adversaries with eavesdropping capabilities; like AnonPoP, such works mostly focus on applications which can suffer significant latency, such as messaging. Note that this excludes the many works on Tor [23] and other low-latency systems, which, unlike AnonPoP, are vulnerable to eavesdropping adversaries.

Specifically, AnonPoP continues the line of mix-based mechanisms whose goal is to provide strong anonymity for messaging or email, with relatively high latency, such as Mixminion [18] and previous proposals, e.g., Babel [31], Mix-Master and Reliable [21]. Mixminion introduced new ideas like Single-Use Reply Blocks (SURBs) to allow anonymity also for the recipients, and techniques to deal with tagging and replay attacks; some of these techniques are used by AnonPoP. However, Mixminion is still vulnerable to long-term intersection attack, does not provide unobservability, and its latency can be excessive for messaging applications.

Vuvuzela [46] is a recent proposed mix-based anonymous messaging design which is most similar to AnonPoP in design and goals, including support for large number of users. Vuvuzela design is based on using a mix-net from sender to recipient; this implies vulnerability to intersection and active attacks (see Section 4). The design also has high communication overhead, because messages always flow between a pair of clients (no support for multiple senders/recipients, as in AnonPoP). Most significantly, Vuvuzela is not designed for mobile users; recipients must also be online at time of sending (no offline mailbox), and the design would consume excessive energy for use in mobile devices. There are other significant differences between the systems; see details in Appendix 9.

Other proposals for strong anonymous messaging were not really designed for practical deployment, efficient and appropriate for many users. In particular, the Busses protocol [4] ensures strong anonymity - even unobservability - by having each message sent through

all possible destinations. The Drunk Motorcyclist (DM) design achieves similar properties, e.g., , strong recipient anonymity, by sending each message randomly through the network, making it highly likely to reach the destination. Both Busses and DM are elegant designs that ensure strong anonymity - but result in excessive overhead, making them inappropriate for more than toy applications.

Verdict [17] and Dissent [50] follow the DC-net [13] design, to ensure sender anonymity. As such, the computational overhead for both the clients and the servers is relatively high, although was shown to be practical for up to thousands of users.

Riposte [16] is a recent DC-net proposal, which achieves sender anonymity against globally eavesdropping adversary for large anonymity sets, although, only minority of the users send messages. In Riposte, many clients write into a shared database, maintained by a small set of servers. To reduce the bandwidth overhead for n clients from $O(n)$ to $O(\sqrt{n})$, Riposte uses private information retrieval (PIR) [15]. However, PIR schemes, even optimized (e.g., [20]), have significant latency and bandwidth overheads, which increase as a function of the number of clients using the system, making them impractical for large-scale messaging.

Pynchon Gate [44] is another design using PIR, in this case, to retrieve pseudonymous mail. Again, due to the use of PIR, it suffers from high communication and computation overhead, making it impractical for use in mobile devices and for systems with many users.

Nipane et al. presented Mix-In-Place [38], an architecture based on Secure Function Evaluation (SFE), that supports messaging communication with a single proxy. However, SFE is even more computationally-intensive than PIR, and hence the system is not practical, certainly not when considering many users and mobile devices.

Aqua [36] is another related system; although it has higher overhead (cf. to AnonPoP), it is much more efficient than the systems discussed above. However, Aqua has a different goal: file-sharing applications such as BitTorrent. Aqua ensures k-anonymity [48], using onion routing [29] with dummy traffic via multiple paths to resist traffic analysis. Aqua does not provide anonymity against corrupt servers, and does not support disconnecting clients.

Resisting Intersection attacks. Buddies [51] offers mechanism to keep the publisher of a message on a shared board, anonymous within a set of k participants [48] for long time, to avoid intersection attacks [7, 8, 37, 52] by a global eavesdropper. However, Buddies does not mask the communication; instead, it *prevents* its clients from publishing messages when this might cause an exposure of their identity. Furthermore, by requiring many cooperating clients ('buddies') online to create a large anonymity set, Buddies requires significant overhead and latency. Hence, Buddies is not able to efficiently achieve long-term resistance to intersection attacks (see Section 5.6 in [51]).

10 Conclusions

AnonPoP demonstrates feasibility of practical anonymous messaging services against powerful attackers, with global eavesdropping capabilities and ability to control some of the servers. AnonPoP achieves this with remarkably low overhead and operational costs - about two cents a month - and is scalable, allowing support of millions of users. AnonPoP supports mobile clients, with low energy requirements and with secure support for temporary disconnections. And AnonPoP protects against both passive and active attacks, allowing for an attacker that controls multiple servers. To properly evaluate AnonPoP, we implemented it (in both mobile and desktop versions), and performed experiments; the prototype of AnonPoP-based messaging application for the Android OS is available from [1]. To measure the costs of operations and efficiency, we also deployed a prototype of AnonPoP's servers in commercial clouds, and tested them in runs with hundreds of thousands of clients.

We hope that the publication of AnonPoP will bring many messaging applications to support an option for strong anonymous messaging via the AnonPoP's API. Such a step will allow clients of different applications to form one large anonymity set and enjoy strong anonymity, not currently available.

References

[1] Anonymous. AnonPoP messaging application - demo and download site, 2016. Online at http://anonpop.weebly.com/.

[2] Michael Backes, Ian Goldberg, Aniket Kate, and Esfandiar Mohammadi. Provably secure and practical onion routing. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 369–385. IEEE, 2012.

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20. ACM, 2007.

[4] A. Beimel and S. Dolev. Buses for anonymous message delivery. *Journal of Cryptology*, 16(1):25–39, 2003.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the

generic composition paradigm. In *Advances in Cryptology—ASIACRYPT 2000*, pages 531–545. Springer, 2000.

[6] Mihir Bellare and Phillip Rogaway. Asymmetric Encryption. http://cseweb.ucsd.edu/~mihir/cse207/w-asym.pdf.

[7] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "anonymity and unobservability in the internet". In *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*, pages 57–65. ACM, 2000.

[8] Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In *Privacy Enhancing Technologies*, pages 110–128. Springer, 2003.

[9] J.M. Bohli and A. Pashalidis. Relations among privacy notions. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):4, 2011.

[10] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 92–102. ACM, 2007.

[11] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.

[12] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Advances in Cryptology—Eurocrypt 2003*, pages 255–271. Springer, 2003.

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

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

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[16] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society, 2015.

[17] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in Verdict. In *Proceedings of the 22nd USENIX conference on Security*, pages 147–162. USENIX Association, 2013.

[18] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15. IEEE, 2003.

[19] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 269–282. IEEE, 2009.

[20] Daniel Demmler, Amir Herzberg, and Thomas Schneider. Raid-pir: Practical multi-server pir. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 45–56. ACM, 2014.

[21] Claudia Díaz, Len Sassaman, and Evelyne Dewitte. Comparison between two practical mix designs. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 141–159. Springer, 2004.

[22] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *WEIS*, 2006.

[23] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[24] Roger Dingledine and Paul Syverson. Reliable mix cascade networks through reputation. In *Financial Cryptography*, pages 253–268. Springer, 2002.

[25] Michael Farb, Manish Burman, G Chandok, J McCune, and A Perrig. Safeslinger: An easy-to-use and secure approach for human trust establishment. Technical report, Technical Report CMU-CyLab-11-021, Carnegie Mellon University, 2011.

[26] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Trans. Inf. Syst. Secur.*, 15(3):14:1–14:28, November 2012.

[27] Nethanel Gelernter and Amir Herzberg. On the limits of provable anonymity. In *Proceedings of the 12th annual ACM workshop on Privacy in the electronic society*, WPES '13, 2013.

[28] Yossi Gilad and Amir Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In Simone Fischer-Hübner and Matthew Wright, editors, *Privacy Enhancing Technologies Symposium*, volume 7384 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2012.

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

[30] I Goriac. An epistemic logic based framework for reasoning about information hiding. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 286–293. IEEE, 2011.

[31] Ceki Gülcü and Gene Tsudik. Mixing email with babel. In James T. Ellis, B. Clifford Neuman, and David M. Balenson, editors, *NDSS*, pages 2–16. IEEE Computer Society, 1996.

[32] J.Y. Halpern and K.R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–514, 2005.

[33] A. Hevia and D. Micciancio. An indistinguishability-based characterization of anonymous channels. In *Privacy Enhancing Technologies*, pages 24–43. Springer, 2008.

[34] Dominic Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

[35] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. Limits of anonymity in open environments. In *Information Hiding*, pages 53–69. Springer, 2003.

[36] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 303–314. ACM, 2013.

[37] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Privacy Enhancing Technologies*, pages 17–34. Springer, 2005.

[38] Nilesh Nipane, Italo Dacosta, and Patrick Traynor. "mix-in-place" anonymous networking using secure function evaluation. In Robert H'obbes' Zakon, John P. McDermott, and Michael E. Locasto, editors, *ACSAC*, pages 63–72. ACM, 2011.

[39] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In *Privacy Enhancing Technologies*, pages 134–152. Springer, 2007.

[40] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Meet the family of statistical disclosure attacks. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 233–236. IEEE, 2013.

[41] A. Pashalidis. Measuring the effectiveness and the fairness of relation hiding systems. In *Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE*, pages 1387–1394. IEEE, 2008.

[42] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. *URL: http://dud. inf. tu-dresden. de/literatur/Anon_Terminology_v0*, 34, 2010.

[43] A. Pfitzmann, B. Pfitzmann, and M. Waidner. Isdn-mixes: Untraceable communication with very small bandwidth overhead. In *GI/ITG Conference on Communication in Distributed Systems*, volume 267, pages 451–463, 1991.

[44] Len Sassaman, Bram Cohen, and Nick Mathewson. The pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 1–9. ACM, 2005.

[45] Y. Tsukada, K. Mano, H. Sakurada, and Y. Kawabe. Anonymity, privacy, onymity, and identity: A modal logic approach. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 42–51. IEEE, 2009.

[46] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, pages 137–152. ACM, 2015.

[47] M. Veeningen, B. De Weger, and N. Zannone. Modeling identity-related properties and their privacy strength. *Formal Aspects of Security and Trust*, pages 126–140, 2011.

[48] Luis von Ahn, Andrew Bortz, and Nicholas J Hopper. K-anonymous message transmission. In *Proceedings of the 10th ACM conference on Computer and Communications Security*, pages 122–130. ACM, 2003.

[49] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[50] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. *10th OSDI*, 2012.

[51] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. Hang with your buddies to resist intersection attacks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 1153–1166, New York, NY, USA, 2013. ACM.

[52] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Passive-logging attacks against anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):3, 2008.

A Per-Epoch Mailboxes (PEM) for Sender Anonymity

This appendix discusses the improvement that Per-Epoch Mailboxes (PEM) offer to the resistance to intersection/correlation attacks on sender anonymity. As summarized in Table 1, AnonPoP's Anon-Comm layer suffices for unobservability and anonymity if the PO is honest, or if clients are always connected. Recipient anonymity is protected, even when clients may disconnect for significant (yet not excessively long) periods, using the 'pool of pull requests' mechanism (Section 6.1). In particular, these AnonPoP anonymity properties are secure against intersection and correlation attacks [7, 8, 37, 40, 52].

However, when a globally eavesdropping PO is malicious *and* clients may disconnect, then the PO may be able to correlate between the connectivity of the clients and the rate of incoming push-requests to each mailbox to compromise sender-anonymity. In this section, we empirically argue that the use of *Per-Epoch Mailboxes (PEM)*, as described in Section 6.2, strengthens the resistance to such attacks.

Additional theory is required to precisely measure the anonymity guarantees against such a strong attacker, and in particular, to measure resistance to intersection and correlation attacks. We take an empirical approach to compare and evaluate AnonPoP's resistance to intersection attack, with and without PEM. Our evaluation is intentionally *conservative*; we give the attacker additional 'hints'. This follows the practice of conservative evaluation of cryptographic security mechanisms, which weakens the mechanism and/or gives additional capabilities to the attacker. Protection will be even better in a more realistic case, where clients send to many mailboxes and may also pick up messages from several mailboxes.

We briefly describe the attack on AnonPoP with and without PEM, and then present the results of the empirical evaluation.

Our results indicate that even a relatively small user population may suffice to allow AnonPoP with PEM to provide sender-anonymity against these strong attacks; and as the number of users grow, protection improves.

A.1 Intersection attack to break sender anonymity

The attacker cannot correlate senders and recipients, due to the recipient anonymity achieved by the pool of pull requests (Section 6.1). We concentrate on the following version of an intersection attack, which attempts to detect correlation between a sending-client and a specific PO mailbox (POB) to which this client pushes messages. Given a mailbox identifier, represented by token t, the attacker wants to find a client that pushes messages with token t. We are interested in finding out how quickly the attacker succeeds in completing this attack so we can evaluate PEM's ability to improve resistance.

AnonPoP with fixes mailboxes. Even in this case, correlating a push token t and a client who sent push requests with t is challenging, since there are usually multiple senders that use the same push token t, and since the same sender may send, at different rounds, to different mailboxes, including to 'dummy' mailbox (in rounds where it has no message to send). In such a case, intersection between sets, such that each set represents potential senders of some message that was pushed to a mailbox, might result in an empty set. Generally, given all the sets of potential senders, extracting a real sender was shown to be an NP-complete problem [35].

To avoid exponential calculations and as a part of the conservative evaluation, we 'assist' the attacker with some 'hints'. Briefly, for every new set of potential clients that the attacker learns, we give the attacker an indication whether the intersection of the new set with all the previous sets will result in a set that does not contain a client that pushes messages to the mailbox.

AnonPoP with PEM. In AnonPoP with per-epoch mailboxes, the clients send messages according to tokens that are replaced every epoch. Therefore, the attacker needs to complete the attack within a single epoch. Note that even if the attacker succeeded to correlate a token to a pushing client during one epoch, she would need to relaunch the attack for the tokens of the following epochs.

A.2 Simulation results

We created a simulation to run AnonPoP with one mix in each channel, one PO, and a changing number of clients. We used rounds of $\lambda = 5$ slots and present the results for slots of $\tau = 60$ seconds.

To model the connectivity of clients, we modeled each client as a two-state machine; a client starts in the *On* state, and at each slot might move to the *Off* state with probability $\frac{1}{300}$. The probability of changing state in the opposite direction is $\frac{1}{60}$. The choice of these values is based on the mobile device's connectivity of participants in the experiment described in Section 7.2.

We divided the clients into pairs of *correspondents* and then randomly chose additional pairs as the number of clients in the simulation. When a client is online, she strikes up 3 conversations per hour with her correspondents, according to Poisson distribution. When the client receives a message, she creates a response message with a probability $\frac{1}{2}$. Namely, on average, each online client begins 3 conversations per hour, such that the number of messages in each conversation is sampled by geometric distribution with a mean 2.

We simulated the intersection attack on AnonPoP with and without PEM. For each line in the graph we simulated 100,000 attacks. We concentrated on two aspects: (1) the time required to successfully correlate between a mailbox and a sender, and (2) the size of the *anonymity set*, which is the number of potential senders for each mailbox from the attacker's perspective.

AnonPoP with fixes mailboxes. We found that the described intersection attack can quite effective deanonymize senders, if AnonPoP is used without PEM (in scenario where clients often disconnect, as in our evaluation). We simulated each of the attacks 100,000 times and counted the number of slots required to complete each of the attacks. Figure 8 shows that for even with 25,000 clients, the adversary succeeded to complete the attack for significant fraction of senders, in significant but not prohibitive time. For example, with 25,000 users, most senders are identified after 7200 minutes (120 hours). Moreover, the graph shows that increasing the number of clients yielded improvement - but rather modest, e.g., with 500 users, most senders are identified after about 3600 minutes (60 hours).

We simulated the distribution of the anonymity set for different numbers of clients. We noticed that the received distributions were very similar, if the anonymity set is presented as percentage of the number of clients, rather than the number itself. Figure 9 depicts the distribution of the anonymity set after 250, 500, 1000, and 1500 slots. It is possible to see that the size of the anonymity set decreases quickly, and after 1000 slots, the size of the anonymity set is about a tenth of the number of clients.

AnonPoP with PEM. We simulated AnonPoP with PEM for different epoch values: $\lambda = 180, 360, 720$. We found that with *all* these epoch values, the adversary could not complete the attack in a single epoch, and

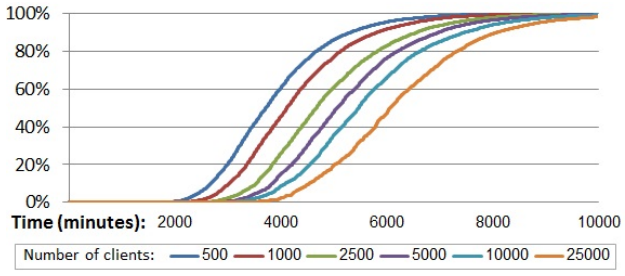


Fig. 8. Percentage of the attacks that were completed over time (x axis) for different numbers of AnonPoP clients.

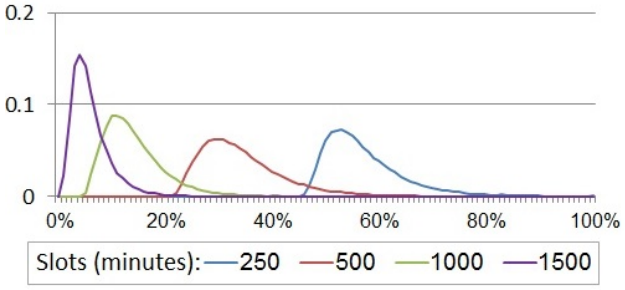


Fig. 9. The anonymity set distribution after different slots for AnonPoP without PEM. The anonymity set is presented as a fraction of the number of AnonPoP clients.

hence, failed to do it over time. To evaluate the effect of the different epoch values, we examined the anonymity set.

The goal of the adversary is to correlate a mailbox with a client. Hence, we measured over time what was the minimal anonymity set achieved for each client's mailbox. Namely, over several epochs, a client used several mailbox numbers; in each of them there was some anonymity set for potential senders. In each slot, for every client, we calculated the size of the minimal anonymity set that the adversary succeeded to create until this slot.

In addition to the inability to complete the attack in reasonable time (a year, 525,600 slots), we found that by using epochs of several hours, it is possible to keep the size of the anonymity set as a high percentage of the number of clients. Unlike the anonymity set of AnonPoP without PEM, as depicted in Figure 9, there is no significant difference in the anonymity set as time goes by. The anonymity set after 10,000 slots was not that different, from the anonymity set after 100,000 slots.

Figure 10 depicts the anonymity set as a percentage of the number of clients. As in Figure 9, we found that the anonymity set as a percentage of the number of clients is similar for different numbers of clients.

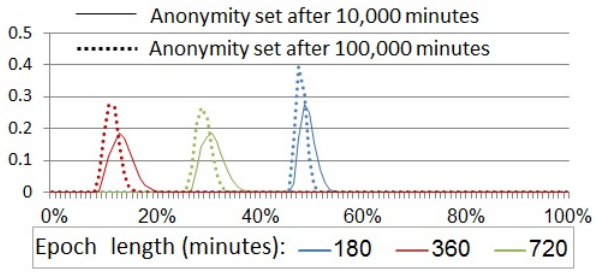


Fig. 10. The anonymity set distribution after 10,000 and 100,000 slots for AnonPoP with PEM for different epoch (δ) values. The anonymity set is presented as a fraction of the number of AnonPoP clients.

Note that these results were obtained under our 'conservative' evaluation where we provided 'hints' to the attacker, as described in the beginning of this section. The actual protection should be even better.

A.3 Sender anonymity & PEM

In spite of the results that show strong resistance to long term intersection attacks, AnonPoP with PEM fails to satisfy Notion 2. Notion 2, and surely the formal models of [27, 33], all require indistinguishability even between extreme cases, that PEM cannot prevent, e.g., only two senders. Additionally, as the model deals with probabilistic polynomial time attackers that need to get only minimal (but not negligible) advantage, even a little reduction of the anonymity set within long time (e.g., two years) is enough for the attacker.

We consider the development of definitions, tools and methods allowing rigorous analysis of AnonPoP and similar practical protocols for strong anonymity, as a significant research challenge. We believe and hope that our methodology of empirically analyzing an attack while conservatively giving the adversary additional power, gives a useful, meaningful indication showing that AnonPoP with PEM guarantees sender-anonymity, also when clients might disconnect.

B Probability of compromised channel

When the PO is corrupt, AnonPoP's sender (recipient) anonymity may fail, if *all* mixes in the push (resp., pull) channel are malicious (1). We now show that, under the

reasonable assumption that $f << n$, the probability of such 'all bad' channel is small.

To increase the probability of 'all bad' channel, the attacker may decrease the number of possible channels where at least one mix is honest, by disconnecting up to f honest servers from each malicious mix, abusing the 'bad server isolation' mechanism. However, as we show, this abuse does not significantly improve the probability of 'all bad' channel. Assume, for simplicity, that the attacker can cancel *every* connection between malicious and honest mixes; for simplicity, assume three mixes in a channel. Hence, there are $3! \cdot \binom{f}{3}$ 'all bad' channels, and $3! \cdot \binom{n-f}{3}$ 'all honest' channels. The probability of choosing an 'all bad' channel is therefore only: $\frac{\binom{f}{3}}{\binom{f}{3}+\binom{n-f}{3}}$.

C AnonPoP vs. Vuvuzela

AnonPoP and Vuvuzela [46] were designed concurrently and independently, with similar goals, and share several design decisions. However, the systems differ considerably, as we outline below. In fact, since the goals are similar, these differences are not by chance; we have considered many alternative designs via analysis and extensive experimentation, and have rejected many of the Vuvuzela's design decisions due to exactly the issues described below.

Support for multiple peers (conversations). Vuvuzela requires users to maintain constant communication rate per peer, and specifically in their evaluation they support a single conversation (peer) at any instant (e.g., see in [46, Section 3.2], under 'network traffic'). This is since user must check every round the 'dead drop' agreed with the peer. This is problematic for usage, especially considering the significant latency for setting-up a conversation (even if peer is available). Also, it allows a common friend to detect correlations between times when her peers are not available, breaking privacy. In AnonPoP, users have flexibility in their use of mailboxes and a user can efficiently correspond with multiple peers. Of course, Vuvuzela users may simply run multiple conversation all the time, but that would make the costs – especially with regarding to energy – even worse than currently.

Mobile users: energy and bandwidth. Vuvuzela was not evaluated for its operation on mobile devices and with mobile users. In fact, based on our experiments in designing and fine-tuning AnonPoP, we expect that Vuvuzela's overhead may make it inappropriate for mobile users. For example, every 'dial round', currently set at 10 minutes, every Vuvuzela user downloads and decrypts all 'invitations' sent to her invitation dead drop, shared with many other users and determined as hash of the user's public key. This overhead alone may already be unacceptable (energy-wise, and possibly also with regarding to bandwidth and processing-time). Even with only three servers, this is 7MB per (10-min) dialing round. Adding the energy costs of sending and receiving packet every few seconds, this would be unacceptable drainage for mobile battery, which users will not tolerate. Considering 'anonymity loves company', we consider this a no-go.

Support and security for offline (disconnecting) users. A major challenge with mobile users is that they sometimes disconnect. Vuvuzela allows only communication between connected (online) users. This restriction is problematic for usability. It also implies privacy exposures. In particular, consider the fixed invitation dead drop server of Alice. That server can correlate between times in which Alice is connected (and reads the invitation box) and times when Alice is disconnected. In contrast, AnonPoP's *request-pool* mechanism (Section 6.1) prevents the PO from performing such attacks.

Tagging attacks. Vuvuzela does not include a comparable mechanisms to AnonPoP's anti-tagging defenses such as *bad server isolation* (Section 5). Hence, it is vulnerable to tagging attacks.

Security evaluation. The analysis and discussion in the Vuvuzela paper focus on the relationships between users and their dead-drops, claiming that the mixnet 'unlinks users from requests', see [46, Section 4.1]. But this is a gross simplification, ignoring many challenges addressed in AnonPoP. Particular relevant aspects include tagging and duplication attacks, see Section 5, and intersection/correlations attacks exploiting client disconnections, see Section 6 and Appendix A.

Performance and costs evaluation. Vuvuzela was only evaluated in lab, rather than in real world conditions as we did. From our experiments, lab results can be misleading. In particular, in their experiments, all the machines run in the same data center, and they multiplexed several Vuvuzela clients onto a single TCP connection. Real implementation may have significantly different results, including reliability issues (not reported at all in their paper). It is also not clear if the costs claimed took into account the costs of inter-server communication – the main cost in a real deployment.