Farid Javani* and Alan T. Sherman

AOT: Anonymization by Oblivious Transfer

Abstract: We introduce AOT, an anonymous communication system based on mix network architecture that uses *oblivious transfer (OT)* to deliver messages. Using OT to deliver messages helps AOT resist blending $(n-1)$ attacks and helps AOT preserve receiver anonymity, even if a covert adversary controls all nodes in AOT.

AOT comprises three levels of nodes, where nodes at each level perform a different function and can scale horizontally. The sender encrypts their payload and a tag—derived from a secret shared between the sender and receiver—with the public key of a Level-2 node and sends them to a Level-1 node. On a public bulletin board, Level-3 nodes publish tags associated with messages ready to be retrieved. Each receiver checks the bulletin board, identifies tags, and receives the associated messages using OT.

A receiver can receive their messages even if the receiver is offline when messages are ready. Through what we call a "handshake" process, communicants can use the AOT protocol to establish shared secrets anonymously.

Users play an active role in contributing to the unlinkability of messages: periodically, users initiate requests to AOT to receive dummy messages, such that an adversary cannot distinguish real and dummy requests.

Keywords: Anonymity by Oblivious Transfer (AOT), anonymous communication, anonymous secret sharing, blending attack, mixnets, oblivious transfer.

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

1 Introduction

Communication systems should not only support confidentiality and authentication of messages; they should also provide untraceability and unlinkability. Failure to protect communications against traffic analysis poses serious threats to individual privacy and organizational operations. We introduce AOT [36], an *anonymous communication system (ACS)* based on *mix network (mixnet)* architecture that uses *oblivious transfer (OT)* to deliver messages. Our approach resists active attacks, supports message delivery for offline users, and provides receiver anonymity even if a covert active adversary controls the entire network.

Although Kilian [40] proved that OT is complete for two-party secure computations, to our knowledge, we are the first to demonstrate how to build an ACS with OT. The desirable properties of AOT mentioned above flow in part from the use of OT.

Figure 1 highlights the keystone of AOT—message delivery using OT. In AOT, a sender does not provide the recipient's address. Instead, they send an ordered pair (M, tag), where M is an encrypted payload (encrypted with the public key of the recipient), and *tag* is a unique tag derived from a *shared secret* between the sender and receiver. For messages ready to be retrieved, AOT posts their associated tags on a public bulletin board. Recipients monitor the bulletin board and request via OT the payloads they wish to receive. In doing so, neither AOT nor network adversaries learn which recipients receive which messages.

Instead of using OT to retrieve messages, one could use *private information retrieval (PIR)* [18]. Whereas PIR provides only receiver security, OT provides both receiver and sender security (see Sections 7.2, 9.1). Using OT helps AOT resist active attacks.

AOT is a mixnet comprising a three-level cascade of nodes, where each level performs a different function. The sender encrypts their payload and tag with the public key of a Level-2 node and sends them to a Level-1 node. Level-1 nodes strip the sender information from messages and send them to Level-2 nodes in batches. Level-2 nodes decrypt the messages, create dummy messages, and send the real and the dummy messages to Level-3 nodes in batches. Dummy messages help resist blending $(n-1)$ attacks. At each level, all nodes at that level perform the same function and can scale horizontally (more nodes can be added at each level to improve performance and reliability).

Each sender-receiver pair needs to share a secret. Through what we call a "handshake" process, communicants can use the AOT protocol to establish shared secrets, confidentially and anonymously. This handshake is of independent interest and can support other applications.

Since Chaum [11] introduced ACSs in 1981, they have evolved in terms of efficiency, network topology, communication latency, robustness, and privacy [35, 43, 51, 64]. For example, cMix by Chaum et al. [13, 15] uses precomputa-

***Corresponding Author: Farid Javani:** Cyber Defense Lab, University of Maryland, Baltimore County, USA, E-mail: javani1@umbc.edu
Alan T. Sherman: Cyber Defense Lab, University of Maryland, Baltimore County, USA, E-mail: sherman@umbc.edu

tion to support fast mixing with minimal real-time asymmetric cryptographic operations. Many recent systems, including [43, 55, 65–67], aim to resist active attacks, as does AOT. An advantage of AOT over cMix is its resistance to active attacks.

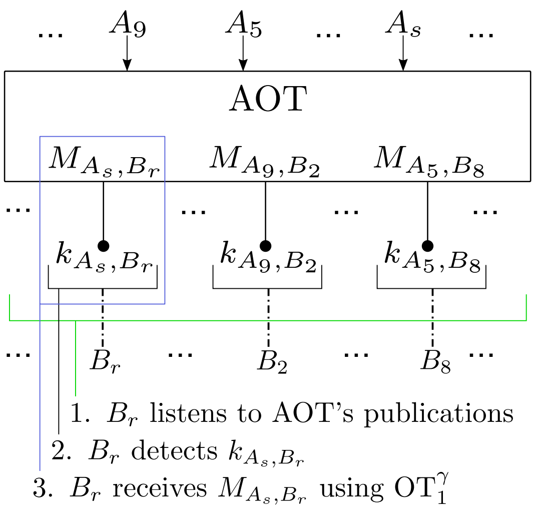

$\dots$ A_9 A_5 $\dots$ A_s $\dots$
AOT
M_{A_s,B_r} M_{A_9,B_2} M_{A_5,B_8}
k_{A_s,B_r} k_{A_9,B_2} k_{A_5,B_8}
B_r B_2 B_8
1. B_r listens to AOT's publications
2. B_r detects k_{A_s,B_r}
3. B_r receives M_{A_s,B_r} using OT_1^γ

Fig. 1. Message delivery using OT. When sender A_s sends a message through AOT to B_r, receiver B_r retrieves the message as follows. A_s sends an ordered pair $(M_{A_s,B_r}, k_{A_s,B_r})$ to AOT, where M_{A_s,B_r} is the encrypted payload and k_{A_s,B_r} a tag derived from a shared secret between A_s and B_r. The payload includes the message and some additional information, as described in Equation 2. AOT publishes k_{A_t,B_t}. Then, B_t detects k_{A_t,B_t} and asks for the corresponding encrypted payload using OT_1^γ, where γ is the number of other tags that B_t chooses to be in the OT session. Only B_r can decrypt the encrypted payload, and AOT does not learn which message B_r received.

Our contributions:

1. We introduce AOT, an anonymous communication system that uses oblivious transfer for message delivery and to resist active attacks.
2. AOT scales horizontally in that additional nodes can be added to each level to increase performance and reliability.
3. We show how two communicants can use AOT to establish a shared secret anonymously.
4. We explain how AOT resists standard active and passive mixnet attacks.

2 Background and Related work

We briefly overview *anonymous communication systems (ACSs)*, *oblivious transfer (OT)*, and applications of OT in selected cryptographic protocols. To our knowledge, we are the first to use OT to build an ACS.

2.1 Anonymous Communication Networks

In 1981, Chaum [11] introduced the concept of *mix networks*, or *mixnets* for short. Mixnets are ACSs, where a cascade of servers, called *mixnodes*, break the links between communicants. Each sender encrypts the payload and the address of the receiver using the public keys of the mixnodes, starting from the last node in the cascade, and then sends the message to the first node in the cascade. Upon receiving an encrypted message, each mixnode decrypts the outer layer of encryption using its private key. Each mixnode gathers incoming messages into a fixed-size batch, shuffles the messages within the batch, and sends the batch to the next node in the cascade. The last node in the cascade decrypts the last layer of the encryption and delivers the messages to the recipients.

To prevent an adversary from linking incoming and outgoing messages based on length, mixnets typically allow only messages of a fixed length. *Hybrid mixnets*, introduced by Pfitzmann and Waidner [54], allow arbitrary length messages. Hybrid mixnets perform bulk data encryption with symmetric-key cryptography; they use public-key cryptography to encrypt symmetric keys. Jackobson et al. [35], Ohkubo et al. [49], and Moller [44] propose combinations of public-key cryptography and symmetric-key cryptography for hybrid mixnets.

In mixnets and hybrid mixnets, message length is proportional to the number of mixnodes, because the sender has to perform one encryption for each node in the cascade. These layers of encryption increase computation on the user side and increase message length. In *re-encryption mixnets*, introduced by Park et al. [51], the sender encrypts only once using the public key of the mixnet. Instead of decrypting the outer later of encryption, each mixnode re-encrypts the ciphertext received from the previous node (using an encryption system that permits re-encryption) and forwards it to the next node. The last node in the cascade produces plaintext. Golle et al. [30] introduce universal re-encryption mixnets, where re-encrytion does not require knowledge of the public key.

Most mixnets rely heavily on users and servers to perform computationally expensive public-key operations online. Performing such operations in an offline precomputation can significantly improve the online running time. Adida and Wikstrom [1] and Jakobsson [34] study precomputation techniques, with Adida and Wikstrom focusing on server computations, and Jakobsson focusing on user computations. Chaum et al.'s *cMix* [15] is a precomputation mixnet that eliminates almost all online public-key operations.

By routing messages through a cascade of nodes, mixnets increase message latency. They also can introduce delays caused by grouping messages into the batches needed for anonymization. Therefore, mixnets, despite their strong anonymity properties, are not suitable for low-latency applications such as web browsing.

Onion routing [59] is an anonymization technique that neither uses batches nor a fixed cascade of nodes. In comparison with mixnets, they can have lower latency but often are vulnerable to traffic analysis attacks [57, 58]. Intermediary proxy nodes break the communication links, and they can vary with each communication session. Each sender may choose the communication path and which nodes will act as the proxy routers. The sender encrypts a message using the keys of the proxies in the path, starting with the key of the last node. Proxies decrypt the (onion) layers of encryption and forward the message. *Tor* [64] is a widely used system built using onion routing; other examples of ACSs built on onion routing include [10, 17, 50].

Riffle [43] is an anonymous communication and file sharing system in which users download messages or files using *private information retrieval (PIR)* [18]. Riffle focuses on communications between users within a group. To enhance anonymity, all users should send and receive messages even if they are not participating in any communications. Messages are initially broadcast to all group members. After users learn the index of the message corresponding to them within the group, they can receive messages using this index and PIR—instead of receiving all messages through broadcast.

Dissent [67] is a group ACS build on DC-nets [12] and verifiable shuffles [9, 28]. Leveraging a user-server architecture, Dissent increases the efficiency of DC-nets and tolerates user-side slowness and disruptions.

Loopix [55] is a mixnet in which nodes are grouped in different layers, where nodes in each layer can communicate with all of the nodes in the immediately previous and following layers. Loopix adds independent delays to incoming message (Poisson mixing) to obfuscate message timing. An important property of Loopix is the so called "loop message," a message that a sender sends to itself; loop messages help Loopix resist blending attacks (see Section 6.1). As explained in Section 4.4, every message in AOT is a loop message.

Vuvuzela [66] is an efficient anonymous messaging protocol that can support millions of users. Vuvuzela is built on a mixnet architecture and offers privacy guarantees based on differential privacy [25]. Users place and retrieve messages from virtual locations ("deaddrops") on mixnode memories. Vuvuzula has a "dialing protocol" (similar to AOT's handshake) that users can use to start a conversation. Unlike AOT's handshake, however, Vuvuzela's dialing protocol is distinct from its conversation protocol. Moreover, in Vuvuzela, the adversary can distinguish whether a receiver is participating in a dialing protocol or a conversation protocol, whereas in AOT the adversary cannot make this distinction. *Stadium* [65] adds horizontal scalability and verifiable mixing to Vuvuzela. Deaddrops and AOT's message tags (See Section 4.1) are similar to the rendezvous hash value of the UDM protocol of Chaum et al. [16].

2.2 Oblivious Transfer

We briefly review our main building block—*oblivious transfer (OT)*—including selected OT protocols and their security and efficiency.

Introduced in 1981 by Rabin [56], an OT protocol enables a receiver to receive a piece of information from a sequence of pieces of information from a sender, while hiding the selection of information from the sender and hiding the rest of the information from the receiver. Formally, in 1-out-of-2 OT, denoted OT_1^2, the sender has two strings s_0, s_1 and transfers s_b to the receiver, where the receiver selects $b \in \{0, 1\}$ and the following two conditions hold: (1) the sender does not know the value of b, and (2) the receiver does not learn anything about s_{1-b}.

We use the generalization 1-out-of-n OT, denoted OT_1^n: the sender has n strings and transfers one string to the receiver, without knowing which string it transferred, and the receiver does not learn anything about the other $n-1$ strings. An ideal implementation of OT might use a trusted third party: after obtaining the strings from the sender, and the index choice from the receiver, the trusted party sends the chosen string to the receiver.

OT can be implemented using public-key cryptography without a trusted third party. For example, OT_1^2 can be implemented as follows: the receiver creates two random public keys but knows the private key corresponding to only one of them. The receiver sends the two public keys to the sender. The sender encrypts each string with a different public key and sends the resulting ciphertexts to the receiver. Because the receiver knows the private key corresponding to only one of the public keys, the receiver will be able to decipher only one of the strings, and the receiver will learn nothing about the other string.

Implementing OT with public-key operations, however, is computationally expensive. Seeking faster implementations, researchers have explored the possibility of implementing

OT using symmetric-key cryptography, but Impagliazzo and Rudich [32] showed that it is unlikely to find black-box constructions of OT using one-way functions. Beaver [6], however, shows that using one-way functions, a small number of *base OTs* can be extended to any number of OTs. Namely, a small number of OTs that are built using expensive public-key operations can be extended to create a large number of OTs using symmetric cryptographic primitives. After Beaver's seminal work, more efficient extensions of oblivious transfer have been introduced that are secure against passive adversaries [2, 33], and secure against active adversaries [3, 47].

Seeking greater efficiency, Bellare and Micali [7] created an OT_1^2 that requires two rounds. Naor and Pinkas [45] reduced the number of exponentiations during run-time in Bellare and Micali from two to one on the sender's side. They also extended OT_1^2 to OT_1^n. In this OT_1^n technique, the sender performs n exponentiations in an initialization step, and uses the resulting values for all subsequent transfers.

Noar and Pinkas [46] showed how to extend any OT_1^2 protocol to an OT_1^n protocol—with $O(n \log n)$ calls to OT_1^2—that provides sender and receiver security computationally, if the underlying OT_1^2 provides sender and receiver security (see Section 7.2 for definitions). Among the most efficient OT protocols that are secure against active adversaries (including possibly the sender and receiver) are [4, 19, 52].

Chou and Orlandi [19] computed more than $10,000$ OT_1^2s per second using one thread of an Intel Core i7-3537U processor. Even with the overhead of building each OT_1^n from OT_1^2s using Noar and Pinkas's technique, the OT_1^ns for message delivery in AOT should be able to be executed sufficiently quickly for typical ACS applications. Moreover, considering the horizontal scalability of Level-3 nodes, OT_1^n should not be a bottleneck in AOT for message delivery for large numbers of users.

2.3 Applications of Oblivious Transfer

OT is a powerful primitive that alone can be used to implement any two-party or multiparty secure computation [21, 40]. We briefly point out some examples.

Nurmi et al. [48] used OT_1^n to enable a trusted election authority to distribute credentials to each of n voters such that the election authority does not learn the credential of any voter. Hence, when each voter uses their credential to cast their ballot, the election authority cannot link ballots to voters.

Even et al. [26] used OT to sign contracts. Two parties use OT_1^2 to exchange secrets, where knowledge of the other's secret implies their commitment to the contract. Here, OT guarantees that each party sends its secret correctly, and both parties simultaneously exchange their secrets.

Fagin et al. [27] used OT to enable two parties to compare their secrets without revealing them (e.g., a user wants to prove their identity using a password but does not trust the medium).

Javani and Sherman [37] use OT to provide perfect ballot secrecy and ensure correct vote casting in a self-tallying boardroom voting protocol.

OT has also been used in mental poker [20], fair computation [29], and zero-knowledge proofs [23].

3 Overview

We explain AOT's communication model, adversarial model, goals, and three-level architecture.

3.1 Communication Model

Senders and receivers communicate with AOT using a trusted user application (e.g., running on a smartphone or workstation). We refer to user applications simply as senders and receivers.

We assume users have access to a public-key infrastructure and know the public keys of the other users and the AOT nodes.

Each sender sends a message and associated tag to AOT, which posts the tag on a public bulletin board. Recipients detect tags related to them, and using OT, download the associated messages. Each pair of users who want to communicate with each other must establish a shared secret, which the sender uses to generate a tag. Users do not need to establish shared secrets with any of the mixnodes.

3.2 Adversarial Model

We consider two types of adversaries: active and covert, each with the same capabilities. Their common goal is to identify communicants, or to link senders and receivers of messages. We do not consider denial-of-service attacks.

The adversary can be any of the users, AOT nodes, or a Dolev-Yao network intruder [24]. The adversary cannot control all of the AOT users at the same time.

We assume authenticated and encrypted communications (e.g., TLS) between users and AOT, and among all nodes within AOT. We assume that the adversary cannot defeat standard cryptographic functions.

An *active adversary* can monitor traffic between users and AOT, as well as traffic among nodes within AOT. An active adversary can also delay incoming messages for an arbitrary amount of time, remove legitimate incoming messages, and inject arbitrarily many messages into the system. An active adversary can attempt to replay and modify messages.

A *covert adversary* [5] seeks to keep the execution of any attack, and their involvement in it, undetected.

3.3 Goals

Our design goal is an ACS that provide each the following properties:

1. **Unlinkability.** The adversary should not be able to link senders and receivers of the communications. AOT provides receiver anonymity even if a covert adversary compromises the entire network (see Section 7).
2. **Scalability.** The system should be able to scale with increasing numbers of users.
3. **Integrity.** Either AOT delivers the messages unaltered to receivers, or it detects message modification and identifies the source of the modification.

3.4 Architecture

As explained in Figure 2, AOT comprises a cascade of nodes organized in three levels. Each level performs a different function, and all nodes within the same level perform the same function. Each level can have a different number of nodes.

Each level can scale "horizontally" (adding more nodes to that level) and independently of the other levels. Scaling nodes horizontally serves two purposes: (1) to have higher availability and efficiency by distributing the load among multiple nodes, and (2) to make it harder for the adversary to control the entire network.

Communications between levels are digitally signed and encrypted using public-key cryptography.

We denote each node as $N_{i,j}$, where $1 \leq i \leq 3$ is the node's level, and j is the node's position within that level. For each Level i, let Q_i denote the number of nodes at Level i.

Level-1 nodes. Level-1 nodes receive messages, strip sender information from them, arrange messages into batches of size β_1, called *containers*, shuffle the messages within each container, and send the containers to Level-2 nodes.

Level-2 nodes. Each Level-2 node decrypts the messages that it receives from Level-1 nodes, arranges them in batches of size β_2, shuffles the messages within each batch, adds dummy messages, and sends the batches to Level-3 nodes. We assume $\beta_2 = Q_1\beta_1$; however, $\beta_2 > \beta_1$ can have arbitrary values. Dummy messages and dummy requests by users (see Section 6.1) help mitigate blending attacks.

Level-3 nodes. Level-3 nodes enable receivers to retrieve messages sent to them. For each round, Level-3 nodes organize themselves into "active" and "passive" nodes, and communicate this organization to Level-2 nodes. Passive nodes receive the dummy messages; active nodes receive genuine messages. Each receives messages in batches of size β_2 and delivers them.

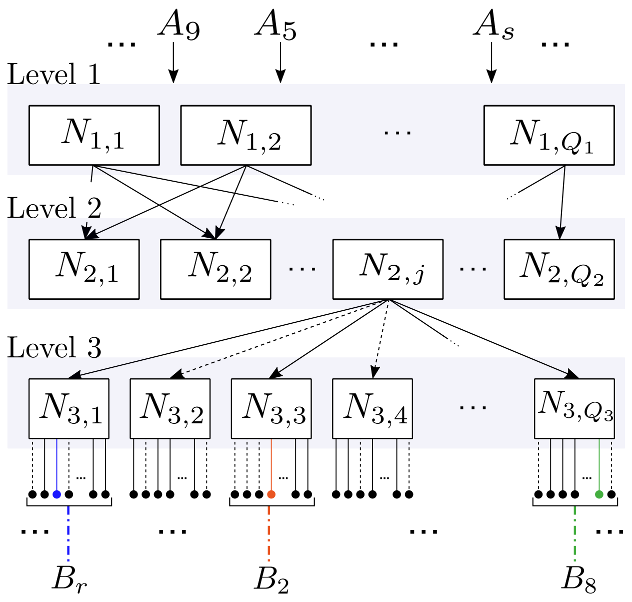


Fig. 2. Summary of how AOT works. Senders send encrypted messages with associated tags to Level-1 nodes of AOT. Level-1 nodes receive messages, remove the sender information from them, arrange the messages into batches of size β_1, shuffle the messages within each batch, and send the batches to Level-2 nodes. Level-2 nodes receive messages from Level-1 nodes, arrange them into batches of size β_2, add additional dummy messages to these batches, shuffle the real messages within each batch, send the real messages to active nodes at Level-3, and send the dummy messages (denoted by dashed lines) to passive nodes at Level-3. Level-3 nodes receive messages from Level-2 nodes, select collections of messages to publish, and publish the tags of these selected messages (denoted by small shaded circles and the end of lines). Receivers detect tags related to them and request the associated encrypted payloads from Level-3 nodes using OT. Dummy messages help mitigate blending attacks, as explained in Section 6.1.

4 Anonymization by Oblivious Transfer

We describe the AOT protocol (including how users send messages, how AOT processes them, how AOT uses message tags, and how it publishes messages), the user app for AOT, and two-way communications using AOT. Table 2 summarizes our notation.

4.1 Message Tags

AOT requires each sender and receiver to have a *shared secret*. Users may establish these secrets anonymously using any method they choose, including using AOT through a process called a *handshake* (see Section 5). Let $\sigma_{A,B}$ denote the shared secret between users A and B. Communicants use the shared secrets to compute *tags*, which AOT posts when making messages ready for delivery. Receivers recognize messages by their tags; hence each tag should be identifiable only by its intended recipient.

For each message sent from A and B, users compute a tag $k_{A,B} = f(\sigma_{A,B}, c)$, where c is a counter and f is a cryptographic key-derivation function (see [41, 42]). Specifically, c denotes the number of successful communications between A and B, where a successful communication is a communication in which the recipient of the message replies back to the sender, or sends an acknowledgment of receiving the message. Section 4.4 explains how AOT uses c, including how AOT handles synchronization issues.

4.2 Sending and Processing Messages

Figure 3 shows how messages flow through AOT. Suppose a sender A wishes to send a payload $x_{A,B}$ to a receiver B. To begin, A encrypts the payload by computing

$$M_{A,B} = E\left[p_B, (x_{A,B}, n)\right], \tag{1}$$

where p_B is the public key of B and n is a nonce. $E[p_B, M]$ denotes public-key encryption of M using the public key p_B—that provides confidentiality and ciphertext integrity—as described, for example, by Bernstein's [8] (see Appendix A.4).[1]

Next, A chooses a Level-1 node $N_{1,i}$ and a Level-2 node $N_{2,j}$ at random, for some $1 \leq i \leq Q_1$ and $1 \leq j \leq Q_2$. Sender A generates and sends the following message to $N_{1,i}$:

$$m = \left(E\left[p_{N_{2,j}}, (M_{A,B}, k_{A,B}, N_{2,j}, ts)\right], N_{2,j}\right), \tag{2}$$

where $p_{N_{2,j}}$ is the public key of node $N_{2,j}$; the value $k_{A,B}$ is the tag of $M_{A,B}$; and ts is the timestamp for when m was created. To prevent the adversary from linking incoming and outgoing messages based on their sizes, AOT uses a fixed size for all message payloads and tags.

For $1 \leq j \leq Q_2$, each Level-1 node maintains one *container* C_j of messages for each Level-2 node $N_{2,j}$. Upon receiving the messages from senders, each Level-1 node strips the sender information off the messages and puts them in the container corresponding to the specified Level-2 node. Whenever any container is filled with β_1 messages, the Level-1 node shuffles the order of messages in the container, sends the messages to the corresponding Level-2 node, and empties the container.

For $1 \leq j \leq Q_2$, node $N_{2,j}$ receives β_1 messages of the following form from a Level-1 node:

$$E\left[p_{N_{2,j}}, (M_{A,B}, k_{A,B}, N_{2,j}, ts)\right]. \tag{3}$$

$N_{2,j}$ then decrypts these messages, puts them in *batches* of size β_2, and permutes the order of the messages within each batch. Level-1 and Level-2 nodes commit to these permutations using a perfectly hiding commitment scheme [31], broadcasting the commitments to the other nodes.

To prevent replay attacks, Equation 2 includes a timestamp ts. Each Level-2 node maintains a record of the (message, tag)-pairs of the messages that it has processed during the last T minutes. If they receive a message with an old timestamp that has been created within last T minutes, they check the record of previously processed messages. If the message has been processed before, they drop the message (see Section 6.2).

Each *round* is the interval during which Level-2 nodes receive β_2 messages and send them to the Level-3 nodes. For $l = 1, 2, \ldots$, let $\mathcal{R}_l$ denote round l, and let Ψ_l denote the batch processed at round l, where the first round is $\mathcal{R}_1$.

During each round, AOT divides the Level-3 nodes into α *active* nodes and ρ *passive* nodes (see Section 4.6). Independently for each batch, the active nodes randomly partition the set of messages in the batch into α subsets. Let P denote the corresponding partition of message indices. Each active node then receives one of the subsets of β_2/α messages from the partition. AOT partitions the nodes into active and passive nodes so that a compromised Level-3 node cannot process all of the batches (see Section 9.1.6).

1 Alternatively, instead of using public-key encryption, A could encrypt the payload $(x_{A,B}, n)$ using symmetric-key encryption, where the key is derived from the shared secret (in a fashion similar to how AOT computes the message tags). Nevertheless, A would still have to encrypt their communication using the public key of a Level-2 node. For consistency, we choose to use E for both encryptions.

The batch Ψ_l that $N_{2,j}$ processes at round $\mathcal{R}_l$ comprises β_2 tuples of (encrypted payload, tag)-pairs:

$$\begin{aligned}\Psi_l = \{ & \left(M_{A_{s_1},B_{r_1}}, k_{A_{s_1},B_{r_1}}\right), \\ & \left(M_{A_{s_2},B_{r_2}}, k_{A_{s_2},B_{r_2}}\right), \\ & \vdots \\ & \left(M_{A_{s_{\beta_2}},B_{r_{\beta_2}}}, k_{A_{s_{\beta_2}},B_{r_{\beta_2}}}\right) \},\end{aligned} \tag{4}$$

where $A_{s_1}, B_{r_1}, A_{s_2}, B_{r_2}, \ldots, A_{s_{\beta_2}}, B_{r_{\beta_2}}$ denote the senders and receivers of messages in the batch.

For each batch Ψ_l that node $N_{2,j}$ processes, $N_{2,j}$ creates $\rho\beta_2/\alpha$ *dummy messages*. As specified by partition P, node $N_{2,j}$ sends β_2/α real messages from the batch to each active Level-3 node, and $N_{2,j}$ sends β_2/α dummy messages to each passive node. We call each of these sets of β_2/α messages, real or dummy, *buckets* and denote each bucket by Φ. Although dummy messages increase the communication load among the mixnodes, they do not create any additional computational load because users do not ask to receive dummy messages, except possibly in some dummy requests.

4.3 Publishing Messages

AOT enables receivers to retrieve messages through two steps: *publication* and *delivery*. Each Level-3 node maintains a *message repository*, *publication repository*, and public *bulletin board*. During publication, the Level-3 node moves messages from its message repository into its publication repository and posts the associated message tags on its bulletin board. During delivery, the recipient engages the Level-3 node in an OT to retrieve the messages it recognized from their associated tags.

As explained in Figure 4b, upon receiving buckets of messages from Level-2 nodes, each Level-3 node puts messages into its message repository. The repository consists of messages from the latest λ buckets received.

During the publication step, each Level-3 node chooses $\beta_2/\lambda\alpha$ messages randomly from each of the λ buckets in its message repository and moves them to its publication repository. The node also posts their tags on its bulletin board in a *publication list*. Thus, Level-3 nodes do not deliver messages in exactly the same order in which they receive them from Level-2 nodes.

Each Level-3 node maintains γ messages in its publication repository, deleting the oldest messages as needed. Recipients do not need to be online when their messages reach the publication repository; they can retrieve their messages when they return online (provided the messages still remain).

Figure 4a shows how a Level-3 node moves messages from its message repository into its publication repository. The node stores each message in its message repository for less than τ seconds. If the rate of incoming messages to AOT is $\beta_2\tau/\lambda$ messages per second, with λ batches in each message repository, and if Level-3 nodes perform their publication step every τ/λ seconds, the maximum delay caused by the publication step is τ seconds for each message. The parameters can be adjusted for any rate of incoming messages.

During the delivery step, a receiver recognizes a tag in the publication list of a Level-3 node. Using OT, the receiver asks for the message corresponding to the tag from the Level-3 node that published the tag. The receiver engages in an OT_1^ζ session with the node and receives the message corresponding to the chosen tag. Here, $\zeta \leq \gamma$ is a parameter set by AOT (see Section 9.3).

When delivering the chosen message, the Level-3 node sends the encrypted payload and tag together with a signed *message authentication code (MAC)* of the encrypted payload and tag. This MAC is necessary to assure message integrity (see Section 6.3).

In case of node failures, messages could be processed by other nodes at the same level. If a node at Level 1 fails, senders resend the messages to another Level-1 node. If an active Level-3 node fails, the Level-2 node that processed the batch sends the messages to another Level-3 node. If a Level-2 node fails, the sender recreates the message for another Level-2 node; the need to resend the message in this case also arises in similar situations for fixed cascades and free-routing mix networks.

4.4 User App and Its Roles

Senders and receivers interact with AOT through a trusted user app running on the user's machine (possibly a smartphone). This app helps the user carry out the AOT protocol, manage user keys and counters, and perform security-enhancing tasks.

The app facilitates message delivery by monitoring the bulletin boards. When it notices a recognizable tag, it initiates an OT_1^γ with the associated Level-3 node.

For each communicant, the app manages the associated shared secret and message counter, which it uses to compute tags. The message counters of senders and receivers might lose synchronization due, for example, to a dropped message or failed delivery. To deal with such possible issues, the app can try all values of the counter within some range (e.g., current value plus or minus up to some constant ξ, say $\xi = 2$).

The app helps senders perform two integrity checks. First, after sending each message, the sender checks that the associated tag has been posted on some bulletin board. The sender informs the appropriate Level-1 node of the result of this check. Because the user's app monitors the bulletin board for coming

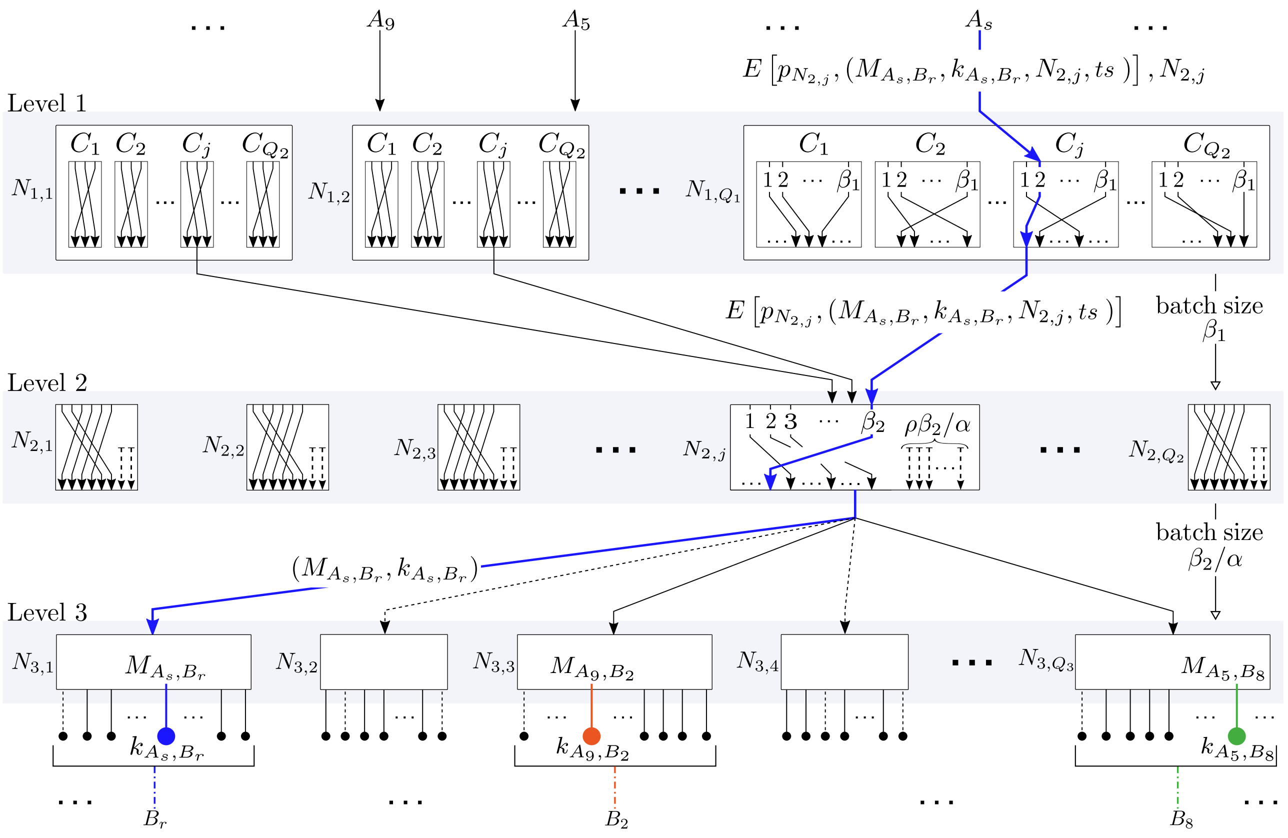


Fig. 3. Detailed view of how messages flow through AOT. Each Level-1 node maintains a container, denoted C_i, for each Level-2 node $N_{2,i}$. Crossing lines denote shuffling of messages. Level-2 nodes add dummy messages (denoted by dashed lines). Thick blue lines identify the route of a message from sender A_s to receiver B_r.

messages, this check does not add any additional overhead. As a result of this check, each message in AOT is a loop message [55], without requiring the sender to list their own address as the address of the receiver. Second, at random times, each sender asks for the messages that they have sent to the system to verify the system's integrity (see Section 6.3). These times are chosen at random from the interval $[1, T_1]$, The second check can also be performed by the app, using the same functionality for retrieving messages sent by others. Users inform Level-1 nodes if any of these checks fail; see Section 6.3.

We assume that the rate of incoming messages to AOT is $\beta_2\tau/\lambda$ messages per second. AOT should receive $\beta_1 Q_1 Q_2/2$ on average before sending a batch of size β_2 to Level-3 nodes (see Section 7). Therefore, senders should expect their messages to be published within $\tau + (Q_2\tau/(2\lambda))$ seconds.

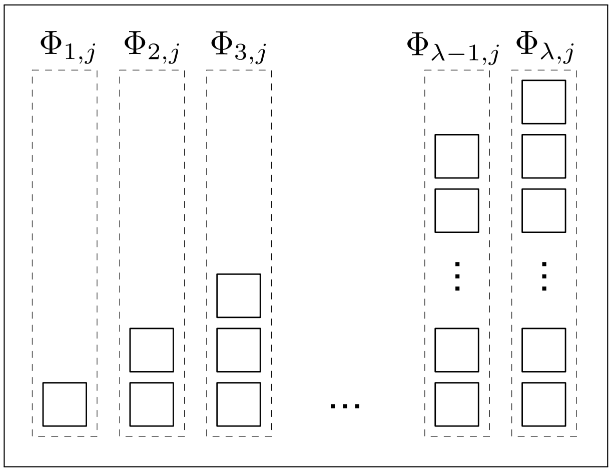
The app also initiates dummy message requests to Level-3 nodes, as explained in Section 6.1. Each user makes a dummy request at random times, where the time to the next request is chosen at random with uniform distribution from the interval $[1, T_2]$.

4.5 Return Path, Delivery Acknowledgment, and Resending Messages

A strong property of AOT is that return paths can be implemented easily without any change in the communication model, enabling two-way anonymous communications. Because users ask for messages from AOT using OT, AOT does not require the sender to specify the recipient's address. This property enables AOT to process each message in two-way communications in a stateless form, without requiring information about any previous message. None of the nodes in AOT can distinguish between forward and return messages. Upon receiving a message from a Level-3 node, the receiver can send a return message as follows: update the counter c from previous communications with the sender, create a new tag, create the message tuple with new payload and tag, and send it to any Level-1 node. Although AOT processes messages in a stateless fashion, senders and receivers operate in a stateful fashion that requires knowledge of the counter value.

```
1: procedure $\mathcal{P}(\tau, \lambda, j)$                ▷ Collect messages to publish
2:     $M \leftarrow$ Empty Message Repository
3:     Add dummy messages to $M$
4:     while Node is alive do
5:         Add the new bucket $\Phi_{\lambda,j}$ to $M$
6:         loop every $\tau/\lambda$ seconds
7:             $L \leftarrow$ Empty set of messages to be published
8:             for $1 \le i \le \lambda$ do
9:                 Choose $\beta_2/(\alpha\lambda)$ messages at random from bucket $\Phi_{i,j}$ in $M$
10:                Add chosen messages to $L$
11:                Remove chosen messages from $\Phi_{i,j}$
12:            Publish $L$
```

(a) Procedure $\mathcal{P}$ used by each Level-3 node to collect a list of messages to publish. Each node $N_{3,j}$ randomly selects messages from each of its past λ buckets $\Phi_{1,j}, \ldots, \Phi_{\lambda,j}$. The procedure guarantees that every incoming message is published in less than τ seconds from its arrival.



(b) Message repository for each Level-3 node. Representative node $N_{3,j}$ maintains a message repository of the λ most recent buckets $\Phi_{1,j}, \ldots, \Phi_{\lambda,j}$ of messages received, where $\Phi_{\lambda,j}$ is the most recent bucket. Each square denotes $\beta_2/(\alpha\lambda)$ messages. For each $1 \le i \le \lambda$, bucket $\Phi_{i,j}$ (denoted by dashed lines) has $i\beta_2/(\alpha\lambda)$ messages; the repository has a total of $\beta_2(\lambda+1)/(2\alpha)$ messages.

Fig. 4. How each Level-3 node collects messages from its message repository to publish in the AOT bulletin board. To increase the anonymity set and to obfuscate the flow of the messages within the system, each Level-3 node collects messages from its λ most recent buckets of messages received.

A recipient can acknowledge receipt of a message as follows. They send a return message, as described above. By observing the resulting posted tag, the sender learns that the recipient received the message, even if the sender does not retrieve the posted return message.

If a sender does not receive any acknowledgment of message receipt, the sender can resend the message. Because recipients request messages using OT, Level-3 nodes do not know which messages were received. Known bounds on the latency caused by AOT, and on the time tags remain in the bulletin board, facilitate such checking by senders.

4.6 Dividing Level-3 Nodes into Active and Passive Nodes

Level-3 nodes partition themselves into α active nodes and ρ passive node where $(1/3)\rho \le \alpha \le (3/4)\rho$ and $\alpha + \rho = Q_3$. Partitioning the Level-3 nodes has two phases: 1) Initiation, and 2) Division. Initiation happens when AOT first starts operating. Nodes perform the division phase for each round.

4.6.1 Initiation phase

For each $1 \le j \le Q_3$, node $N_{3,j}$ generates a random value $v_{3,j}$ and broadcasts a commitment of the value to all Level-2 nodes and other Level-3 nodes. After each $N_{3,j}$ receives the commitment from all other nodes, it broadcasts $v_{3,j}$ to all Level-2 nodes and all other Level-3 nodes. Level-2 nodes and Level-3 nodes compute $\oplus_{j=1}^{Q_3} v_{3,j}$, where $\oplus$ denotes bit-wise XOR.

4.6.2 Division phase

At each round $\mathcal{R}_l$, the Level-2 node that processes the batch, and all Level-3 nodes, compute $V_l = g(\oplus_{j=1}^{Q_3} v_{3,j}, l)$, where g is a cryptographic key-derivation function (see [41, 42]). Each node then divides V_l into Q_3 bit strings of the same length, denoted by $V_{l,j}$, such that $V_l = V_{l,1}||V_{l,2}||\ldots||V_{l,Q_3}$, where $||$ denotes concatenation. Each node creates the tuples $\left(V_{l,j}, N_{3,j}\right)$ for all $1 \le j \le Q_3$ and sorts the list of tuples based on their first elements. The first α nodes in the sorted list are actives nodes of the round.

The bit-length of V_l should be set such that the probability of having $V_{l,j_1} = V_{l,j_2}$ where $j_1 \ne j_2$ is negligible.

5 Handshake Process

We explain how the AOT protocol can be used by any two communicants to establish a shared secret anonymously, in what we call a *handshake process*. Any adversary that monitors these communications should be unable to identify the communicants or determine that they engaged in a handshake process. This process can be used by senders and receivers, as needed, to establish the shared secrets needed for message delivery in AOT (see Section 4.2), as well as for any application unrelated to AOT. The handshake process leverages the fact that the communicants know each other's public keys.

For a sender A to establish a shared secret with a receiver B, the sender generates the message

$$\left(E\left[p_{N_{2,j}}, (E\left[p_B, (B, R_A, ts)\right], k_{HS}, N_{2,j}, ts)\right], N_{2,j}\right), \quad (5)$$

and sends it to AOT as they would any other message. All such messages include the same global constant tag k_{HS}, generated by AOT and sent to all users. Here, B is the receiver's identity and R_A is a random value generated by A.

Whenever any user application detects the special tag k_{HS} on a bulletin board, they ask for the corresponding message $E\left[p_B, (B, R_A, ts)\right]$ using OT, but only B can decrypt the message. The receiver B decrypts the message and computes $\sigma_{A,B} = h(R_A)$ as the shared secret of A and B, where f is a cryptographic hash function.

Henceforth, A and B can communicate through AOT. For example, using the session shared key $k_{B,A} = f(\sigma_{A,B}, 1)$, B can reply to A with the message

$$\left(E\left[p_{N_{2,j}}, (E\left[p_A, (x_{B,A}, n)\right], k_{B,A}, N_{2,j}, ts)\right], N_{2,j}\right). \quad (6)$$

If a sender and a receiver already share a value that they could use as the shared secret, they do not need to perform the handshake process.

AOT treats messages with the special tag k_{HS} the same as all other messages. Therefore, any OT session for the handshake process is indistinguishable from any other OT session, except that anyone can recognize how many special tags were posted on the bulletin board.

6 Security Notes

We explain how AOT resists blending attacks and other standard attack against mixnets. We also explain how AOT achieves message authenticity and protocol integrity. In Section 7, we define the security of OT and analyze the anonymity of AOT, including stating and proving its receiver anonymity property.

6.1 Resisting Blending Attacks

AOT resists blending attacks with dummy messages generated by Level-2 nodes and dummy requests initiated by users. In a *blending* (or $n-1$) *attack* [61], an active adversary attempts to determine the receiver of a message by allowing only the targeted message into the system—the adversary blocks or delays all other messages, or fills the system with its own messages. The adversary then tries to determine the receiver by observing who receives the targeted message.

Let $m_{XY} = \left(E\left[p_{N_{2,j}}, (M_{X,Y}, k_{X,Y}, N_{2,j}, ts)\right], N_{2,j}\right)$ be the targeted message. The adversary knows that user X generated and sent message m_{XY}; the adversary does not know the ciphertext $M_{X,Y}$ or the tag $k_{X,Y}$.

Without dummy requests, the adversary could identify the receiver as the sole user to retrieve a message from the current publication lists. Similarly, without dummy messages, the adversary could identify—by the process of elimination—the sole message referenced in the publication lists not created by the adversary.

With dummy messages, the adversary could infer only that the targeted message is one among several (the targeted message and the dummy messages). Furthermore, with dummy requests, the adversary could infer only that the receiver is one among many (the receiver and any user who made a dummy request).[2]

Thus, under a blending attack, the adversary can reduce the size of the receiver anonymity set size for the targeted message to the number of dummy requests initiated by users during the time that the targeted message is in a publication list.

For example, if U users make dummy requests every T_2 minutes, with Q_3 level three nodes that hold each message for H hours in their publication repositories, AOT will receive $HU/(T_2 Q_3)$ dummy requests during the time that any message is in a publication repository. Considering that each user will make more than one dummy request in H hours, the adversary cannot reduce the receiver anonymity set.

6.2 Resisting Standard Attacks

We explain how AOT resists replay, traffic-analysis, tagging, and intersection attacks.

2 The chance that the tag of a real message collides with the tag of a dummy message is negligible.

6.2.1 Replay attacks

AOT resists replay attacks because Level-2 nodes detect and drop any recent message that has been previously processed. Level-2 nodes drop the messages that are not created within the last T minutes. Level-2 nodes also drop any message that has been created within the last T minutes and has a (payload, tag)-pair that is equal to a (payload, tag)-pair of some previously processed message. If a user wants to resend their message, they recreate the message with the same encrypted payload, a new tag (with a new counter—see Section 4.4), and a new timestamp.

6.2.2 Traffic-analysis attacks

AOT resists traffic-analysis attacks through standard message and batch sizes and message reordering. AOT accepts only messages of a fixed size. Each Level-1 node sends messages in batches of size β_1, and each Level-2 node sends batches of size β_2/α (dummy or real). Furthermore, each Level-1 and Level-2 node reorders all messages within each batch.

6.2.3 Tagging attacks

In a tagging attack, the adversary modifies ("tags") a message before it enters the anonymity network and observes the output messages to recognize the modified message. The only published outputs of AOT are the message tags, which senders compute from shared secrets that the adversary does not know. If the adversary could replace a message with a modified one that had a different tag, the intended recipient would not recognize the modified tag, and the adversary would gain no useful information. Moreover, since messages are encrypted using an encryption algorithm that provides ciphertext integrity, the adversary cannot alter the messages without being detected.

6.2.4 Intersection attacks

AOT resists intersection attacks through dummy requests. In an intersection attack [22], the adversary attempts to break the anonymity of communications by correlating the behaviors of users. For example, if a receiver B always receives a message whenever the sender A sends one, the adversary might conclude that A and B are communicating. Mitigating intersection attacks is challenging and many anonymity systems are vulnerable to such attacks [15, 38, 43, 67]. In AOT, dummy requests (including by actual receivers) help hide the correlations of user behaviors from the adversary. As explained in Sections 6.1 and 7.2, the recipient of any published message is indistinguishable from the users who send dummy requests.

6.3 Protocol Integrity

We explain how entities detect certain possible integrity errors and what they do about such errors. Such error detection is possible because users encrypt their payloads using an encryption algorithm that provides ciphertext integrity, and nodes protect their communications with other nodes using TLS.

As in cMix [14], we define the protocol integrity as follows:

Definition 1. *A protocol maintains integrity if at the end of each run involving honest users:*
1. *either, all the messages are delivered unaltered to the intended recipients;*
2. *or, a malicious mixnode is detected with a non-negligible probability, and no honest party is proven malicious.*

If the users and the mixnodes in AOT follow the protocol correctly, AOT delivers messages anonymously to recipients without modification.

Whenever users or mixnodes detect an altered message, AOT will begin an *audit process*. During the audit process AOT traces the message from the step during which the alteration is detected, back to the sender of the message. Tracing a message to the sender is possible because the Level-1 and Level-2 nodes commit to the shuffles that they apply to messages in containers and batches. Because AOT delivers messages using OT, tracing an altered message to its sender does not violate anonymity. The adversary cannot detect the receiver of a message, nor can it link the sender and the receiver of a message by incorrectly claiming that the message has been altered.

We consider three scenarios where users and/or AOT nodes act maliciously.

Level-2 nodes detect an altered message. Each Level-2 node checks each in-coming user message for integrity. If the integrity check fails, the Level-2 node asks the forwarding Level-1 node to prove that it has not modified the message. If the Level-1 node can prove that it did not modify the message, then AOT assumes that the input received by the Level-1 node was in error (e.g., by the sender or in transit).

AOT publishes an altered message. Each receiver should check the received message for integrity. If a receiver detects an integrity error caused by a malicious node, the receiver can notify the sender using AOT. The sender can ask for the message and notify the Level-1 node that received the message. When users notify AOT that an altered message has been published, nodes will then trace the message and detect the malicious node.

Moreover, after a sender sends a message, they can ask for the message and verify if the message has been altered. Senders also send dummy messages to verify the integrity of the system.

Level-3 nodes send a signed hash of the encrypted (payload, tag)-pair, along with the tuple in the delivery step. Therefore, Level-3 nodes cannot alter the message during the publication and delivery steps, or use the unaltered message during an audit, without being detected.

Honest parties cannot be proven malicious. Senders encrypt messages with a public-key encryption system that provides ciphertext integrity. Therefore, malicious nodes cannot alter a message and incorrectly claim that some user modified the message. Moreover, all of the communications between mixnodes and users, and among the mixnodes, are signed. If any party is accused of altering a message, they can use the signed messages that they receive from other parties as proof of their correct behavior.

7 Anonymity Analysis

We analyze the anonymity of AOT in two ways. First, we bound the size of the anonymity set. Second, we show that AOT has receiver anonymity even if all nodes are compromised.

7.1 Receiver and Sender Anonymity Sets

We bound the size of the sender anonymity set and receiver anonymity set in AOT. We do so without considering that the adversary may be able to reduce the size of the anonymity sets—though we are not aware of any such attacks. For example, the adversary may be able to reduce the size of the anonymity sets if they have additional information about the communication (e.g., time or location), or if the protocol has vulnerabilities. As such, our calculations should be interpreted as upper bounds on the sizes of the anonymity sets. With similar caveats, we also show that members within the anonymity sets are equally like to be the sender or the receiver.

In Section A.3, we also analyze the anonymity of a version of AOT without the latency limit τ using an information-theoretic definition of anonymity.

Chaum [12] introduced the concept of anonymity set in 1988. As defined by Pfitzmann and Kohntopp [53], *anonymity* "is the state of being not identifiable within a set of subjects, the anonymity set." Kesdogan et al. [39] define anonymity set as the set of users that have a non-zero probability of being the sender or receiver of a particular message. Pfitzmann and Kohntopp argue that stronger anonymity is associated with a larger anonymity set and an even chance of being the sender or receiver among members of the set.

We assume that messages arrive into AOT at the rate of $\beta_2\tau/\lambda$ messages per second, and that on average, Level-2 nodes receive an equal number of messages per second from Level-1 nodes. We also assume that, at the time when Level-2 nodes first process and send real messages to Level-3 nodes, each of the Level-3 nodes has dummy messages in their message repository.

As explained in Figure 4a, during the publication step, each Level-3 node publishes β_2/α tags of messages that they randomly select from the λ most recent buckets—$\beta_2/(\alpha\lambda)$ from each bucket—for publication from the node's message repository. Altogether, Level-3 nodes publish $\beta_2+\rho\beta_2/\alpha$ messages. Assuming receivers select active and passive nodes uniformly at random for each round, on average $\rho\beta_2/\alpha$ of the selected messages are dummy messages.

7.1.1 Sender Anonymity Set

Each published tag could have been selected from λ recent batches of size β_2 that Level-2 nodes processed. We explain: for $1 \le i \le \lambda$, let $\Phi_{i,j}$ denote the λ buckets in a Level-3 node $N_{3,j}$, where $\Phi_{\lambda,j}$ denotes the most recent bucket. As explained in Figure 4b, before any publication, the message repository of active node $N_{3,j}$ contains $i\beta_2/(\alpha\lambda)$ messages from buckets $\Phi_{i,j}$, for all $1 \le i \le \lambda$. During each publication step, node $N_{3,j}$ chooses $\beta_2/(\alpha\lambda)$ messages from each $\Phi_{i,j}$. Therefore, $N_{3,j}$ selects messages from λ recent batches received from Level-2 nodes.

Level-1 nodes accumulate incoming messages in the containers C_j before sending them to the Level-2 nodes $N_{2,j}$, for $1 \le j \le Q_2$. Assuming that senders choose Level-1 nodes and Level-2 nodes uniformly at random, an incoming message can go to any of the Q_1Q_2 containers with approximately equal probability of $\approx 1/(Q_1Q_2)$. Containers will accumulate messages at a similar rate. Each container should have β_1 messages before its messages are sent to a Level-2 node.

Therefore, on average, Level-1 nodes will receive $\beta_1Q_1Q_2/2$ incoming messages before sending β_2 messages to a Level-2 node in batches of size β_1. Whenever any Level-3 node publishes a message, it came from λ recent Level-2 batches. Therefore, the size of the sender anonymity set in AOT is $\lambda\beta_1Q_1Q_2/2$.

7.1.2 Receiver Anonymity Set

Because OT hides the messages that receivers request, for any Level-3 node and for any message in its publication list, any receiver that initiates an OT session with the node while the message is in the list could be the receiver of the message. Assuming that receivers ask for all messages that AOT publishes, for each Level-3 node, the receiver anonymity set is at most γ (the maximum number of messages in the publication lists) plus the number of dummy requests to the node during the same time period (minus the number of users who make real requests to receive messages as well as dummy requests). In Section 7.2, we show that the adversary cannot reduce the receiver anonymity set of any message below the number of dummy requests made to the publishing node.

7.2 Receiver Anonymity with Corrupted Nodes

Adapting definitions from Naor and Pinkas [46], we state definitions of receiver and sender security for OT.

Definition 2. *Receiver security in OT. An* OT_1^m *provides receiver security if and only if, for any probabilistic polynomial-time sender* $\mathcal{A}$ *with* m *strings* $s_1, s_2, \ldots s_m$, *given any* $1 \le i < j \le m$ *where the receiver chose either* s_i *or* s_j, $\mathcal{A}$ *cannot distinguish whether the receiver chooses* s_i *or* s_j.

Sender security is defined in terms of a comparison between the information the receiver learns in the ideal implementation of OT and the information the receiver learns in the real implementation.

Definition 3. *Sender security in OT. An* $OT_1^{\lambda m}$ *provides sender security if and only if, for every probabilistic polynomial-time receiver* $\mathcal{B}$, *substituting* $\mathcal{B}$ *in the real implementation of the protocol, there exists a probabilistic polynomial-time machine* $\mathcal{B}'$ *for the receiver's role in the ideal implementation such that, for every sequence of strings* $s_1, s_2, \ldots s_{\lambda m}$ *of the sender, the outputs of* $\mathcal{B}$ *and* $\mathcal{B}'$ *are computationally indistinguishable.*

OT protocols introduced in [4, 19, 52] satisfy Definitions 2 and 3 against malicious senders and receivers.

A strength of AOT's design is that it preserves receiver anonymity, even if a covert adversary corrupts all of the nodes. This property follows from three sources: receivers retrieve messages using OT; receivers issue dummy message requests; and Level-1 nodes verify that their messages have been posted.

For example, a malicious Level-3 node might attempt to find the recipient of a targeted message by posting only the targeted message, along with a set of $\gamma - 1$ fake messages. This attack would fail for two reasons: First, some receivers would issue dummy requests. Therefore, the Level-3 node could not uniquely identify the receiver. Second, because senders always verify that their messages have been posted, senders would notice that their messages were not posted.

These observations lead to the following theorem.

Theorem 1. *If the oblivious transfer protocol used in AOT provides receiver security according to Definition 2, AOT provides receiver anonymity as defined in Definition 4 (below), even if a covert adversary controls all the nodes in AOT.*

One way to reason about the anonymity properties of AOT is in terms of a game between a *challenger* $\mathcal{C}$ and an *adversary* $\mathcal{A}$ [9, 14, 63]. The challenger $\mathcal{C}$ performs the protocol on behalf of honest mixnodes and users, while adversary $\mathcal{A}$—a *probabilistic polynomial time (PPT)* Turing machine—performs the protocol on behalf of compromised mixnodes and users.

The covert adversary can compromise all of the mixnodes and a fraction of the users. The adversary can observe the internal states of all compromised mixnodes, and the adversary may inject, drop, or delay messages. The adversary can also eavesdrop on communications between mixnodes and users.

The game between the challenger and the adversary works as follows:

Step 1. The challenger $\mathcal{C}$ runs AOT for all of the honest users, and shares all public information with the adversary.

Step 2. For as many times as the adversary $\mathcal{A}$ wants, $\mathcal{C}$ simulates the honest users, and $\mathcal{A}$ runs the protocol with β_2 messages with inputs from $\mathcal{C}$.

Step 3. The adversary chooses two honest users c_0 and c_1 and a message m. The challenger chooses a bit b at random by tossing a uniform random coin. The challenger sets c_b to be the receiver of the message m, sets c_{1-b} to be a user who does not have a message in the publication list, and initiates a dummy request with AOT. Let m' be the message in the publication list that c_{1-b} pretends to ask for. $\mathcal{C}$ runs the protocol with c_b and c_{1-b} and sends m' to $\mathcal{A}$.

Step 4. After the challenge phase, the adversary runs the AOT protocol with β_2 messages as many times as it desires to.

Step 5. The adversary outputs its guess for b.

Let $\langle \mathcal{A} | \mathcal{C} \rangle$ denote the adversary's output in the game. The adversary's advantage in the anonymity game is $|Pr[\langle \mathcal{A} | \mathcal{C} \rangle = b] - 1/2|$.

Definition 4. *A protocol maintains receiver anonymity if the adversary's advantage in the anonymity game is negligible.*

Section A.2 presents a proof of Theorem 1.

8 Comparison with Other Anonymity Systems

We compare design features and security and efficiency properties of AOT with those of selected other ACSs: original mixnet (OM) [11], cMix [15], Riffle [43], Stadium [65], Vuvuzela [66], and Loopix [55]. Table 1 summarizes the comparison.

AOT has two novel design features, which contribute to its security and performance. First, each level in the cascade of nodes performs a different function. Second, during each round, AOT assigns each Level-3 node an active or passive role, where AOT sends batches of dummy messages to passive nodes.

Systems that resist active attacks use dummy messages and dummy requests in a variety of ways. To hide communication patterns, many systems that resist active attacks require users to engage in every round of communication, even if the users do not send or receive genuine messages in every round. In AOT and Loopix, however, users do not have to participate in every round.

Most of the systems in our comparison group are scalable to support a large numbers of users (e.g., Loopix). Riffle is designed for communications of users within groups. AOT (see Section 9.1.3), Stadium, and Loopix scale horizontally.

All of the systems in our comparison group resist *global passive adversaries (GPA)*, who observe all traffic between nodes of the system and between users and the system. We now describe how each of the systems mitigates blending attacks.

Serjantov *et al.* [61] define two properties of blending attacks: a blending attack is *exact* if the attacker can determine the receiver of a message with probability 1; an attack is *certain* if the adversary can always isolate and trace the target message.

cMix and OM follow the threshold mixing strategy; therefore, as noted by Serjantov *et al.* [61], an adversary can perform an exact and certain blending attack on both systems. Through sending loop messages, Loopix renders blending attacks uncertain and inexact. With loop messages Loopix determines if the adversary is blocking incoming traffic, enabling its nodes to change their behavior when under such attack. Since loop messages are indistinguishable from normal messages, the most an adversary can do is to guess which messages are loop messages.

Vuvuzela and Stadium resist blending attacks by having all users make dummy requests in every round. Because sending messages is indistinguishable from receiving messages, to perform a blending attack in these two systems, the adversary's attempt to block incoming traffic also blocks the targeted message from reaching its intended recipient. In both systems, however, the adversary is able to reduce the anonymity set to a smaller group of communicants as follows. To begin, the adversary hypothesizes the composition of some subgroup of users who are communicating frequently. The adversary blocks all incoming messages, except for those that originate from within this subgroup. If the majority of the message locations in the memory of the last mixnode in the cascade are accessed twice—once for writing and once for reading a message—the adversary can conclude that users within the subgroup are frequently communicating with each other.

AOT and Loopix are the only systems that provide offline message delivery and that do not rely on synchronous rounds for security. Synchronous rounds require mandatory user participation in every round.

As shown in Section 7.2, AOT preserves receiver anonymity even if a covert adversary controls all nodes; none of the other systems has this property.

9 Discussion

We explain our key design decisions, calculate the storage requirement for Level-3 nodes, comment on the choice of the number of messages in each OT, and state some open problems. Section A.3 analyzes how converting AOT into a pool mix would affect anonymity and latency,

9.1 Key Design Decisions

We discuss several important design decisions.

9.1.1 Using Oblivious Transfer

The central design decision was to use OT in message delivery. At the cost of executing an OT protocol, OT contributes significantly to receiver anonymity: OT receiver security ensures that neither AOT nor the adversary learns which message the receiver retrieves.

An alternative design would be to post each message tag and its encrypted payload $M_{A,B}$ on the bulletin board. In this

Table 1. Comparison of AOT with selected ACSs with respect to design characteristics, security, and efficiency. The comparison set comprises original mixnet (OM) [11], cMix [15], Riffle [43], Stadium [65], Vuvuzela [66], and Loopix [55].

(a) Design characteristics and features of selected ACSs. Two unique properties of AOT are nodes with different functionality, and grouping of nodes for each batch into active and passive nodes.

		Different Functionality per Level	Includes Passive Nodes	Dummy Requests	Dummy Messages	Fixed Routes	Horizontal Scalability	Participation per Round
OM	1981	×	×	×	×	✓	×	×
Vuvuzela	2015	×	×	✓	✓	✓	×	✓
Riffle	2016	×	×	✓	✓	×	×	✓
cMix	2017	×	×	×	×	✓	×	×
Stadium	2017	×	×	✓	✓	×	✓	✓
Loopix	2017	×	×	✓	✓	×	✓	×
AOT	2021	✓	✓	✓	✓	×	✓	×

(b) Security and efficiency properties of selected ACSs. GPA denotes Global Passive Adversary, and RAC denotes Receiver Anonymity with Compromised network. AOT provides receiver anonymity, even if a covert active adversary subverts all nodes.

		GPA Resistance	Active Attack Resistance	Scalable Deployment	Low Latency	Asynchronous Messaging	Offline Storage	RAC
OM	1981	✓	×	×	×	✓	×	×
Vuvuzela	2015	✓	✓	✓	×	×	×	×
Riffle	2016	✓	✓	×	✓	×	×	×
cMix	2017	✓	×	✓	✓	✓	×	×
Stadium	2017	✓	✓	✓	×	×	×	×
Loopix	2017	✓	✓	✓	✓	✓	✓	×
AOT	2021	✓	✓	✓	✓	✓	✓	✓

alternative design, the receiver could request messages with PIR [18] rather than with OT. We chose to use OT rather than PIR because OT provides both sender and receiver security, while PIR provides only receiver security.

Receiver security of OT increases the cost of the following two attacks:

a) The adversary downloads a (message, tag)-pair, re-encrypts it with the public key of a Level-2 node (adding a new timestamp), and sends the ciphertext to the Level-2 node, while also blocking new messages from reaching AOT with a blending attack. The adversary observes which receivers request messages, in the hope of associating a receiver with the replayed message.
b) The adversary monitors the frequency by which users initiate OT sessions with AOT. When a sender resends a message (when they do not receive an acknowledgment of delivery or a response to their message), the adversary notices a change in the number of OT sessions.

With sender security of OT, an adversary must perform one OT for each (message, tag)-pair retrieved in this fashion. With PIR, the adversary could download all (message, tag)-pairs at once, monitoring all published (message, tag)-pairs with, say, two OT sessions per day.

9.1.2 Shared Secrets

A key enabling design decision was to assume that each pair of communicants shared a secret, and to leverage this shared secret to enable each pair of communicants to recognize the tag of messages sent between them. The cost is requiring each pair of communicants to establish a shared key, and we show how AOT can be applied to do so.

9.1.3 Multiple Nodes Per Level

We designed AOT to scale horizontally for each level, permitting multiple nodes per level. This decision enhances load balancing, reliability, and security. The work of each level is spread over several nodes. In case a node fails, other nodes on that level could process the messages. To control a level, the adversary would have to control several nodes on that level.

9.1.4 Message Batches at Level 1 and Level 2

Following a fundamental technique of mixnets, AOT processes messages in batches, where AOT permutes the order of the messages within each batch. Doing so enhances anonymity but can introduce delays. Unlike most mixnets, AOT uses different batch sizes at the first two levels; this property enables AOT to distribute the processing of messages between Level-1 and Level-2 nodes without introducing delays.

9.1.5 Number of Handshake Tags

Section 5 proposes to use a single global handshake tag. A consequence of this choice is that every user who initiates a handshake must request all messages with this tag. To reduce this overhead, one could use multiple handshake tags—for example, one per geographic region. This alternative choice would come at the cost of revealing more information about communication patterns.

9.1.6 Active and Passive Nodes

We chose, for each round, to partition Level-3 nodes into active and passive nodes, where active nodes handle real messages, and passive nodes handle dummy messages. An alternative choice would be for each Level-3 node to handle real messages and to generate some additional dummy messages. Our choice aims to obfuscate the internal flow of messages.

9.2 Storage Requirements for Level-3 Nodes

We calculate the amount of storage required by each Level-3 node, for one choice of parameters. Let v denote the rate of incoming messages to the AOT (in Procedure $\mathcal{P}$, Figure 4a, we assume $v = \beta_2\tau/\lambda$ messages per second). AOT publishes $v + (\rho v/\alpha)$ real and dummy messages per second, where α is the number of active nodes and ρ is the number of passive nodes. The rate of the incoming messages and published real messages are equal—regardless of number of previous batches λ in the publication repository and the maximum delay τ for messages in the publication repository.

For example, if AOT keeps the messages in each publication list for 12 hours before removing them, with $v = 10,000$ messages per second, 300 bytes message size, and $Q_3 = 5$, each Level-3 node will store approximately 260 gigabytes of data.

9.3 Number of Messages in Each Oblivious Transfer

One of the parameter choices in AOT is the number of messages ζ from which users select a message in each OT. This choice affects the running time of OT and receiver anonymity. Currently, we use $\zeta = \gamma$, the length of the publication list. An alternative decision would be to choose $\zeta < \gamma$ and to allow each user to select from which tags to choose in OT (e.g, set $\zeta = \gamma/10$). This alternative decision would permit a larger γ, when messages in the publication list could be maintained for a longer time, allowing users to be offline longer. This choice, with its shorter list of tags, might also reduce the size of the receiver anonymity set.

9.4 Open Problems

Open problems include: (1) Formally state and prove the security properties of AOT, and (2) perform a formal-methods analysis of the AOT protocol using a protocol-analysis tool such as CPSA [62].

10 Conclusion

We introduced AOT, an anonymous communication system with mixnet architecture that provides offline message delivery and horizontal scalability. Through using oblivious transfer for message delivery, AOT resists blending attacks, and provides receiver anonymity even if a covert adversary controls the entire network. The handshake protocol, which applies AOT to enable two communicants to establish a shared key anonymously, is of independent interest. AOT illustrates the power and flexibility of oblivious transfer as a building block in protocol design to enhance security properties.

Acknowledgments

We thank David Chaum and Jonathan Katz for helpful comments. Sherman was supported in part by the National Science Foundation under SFS grant DGE-1753681, and by the U.S. Department of Defense under CySP grants H98230-19-1-0308 and H98230-20-1-0384.

References

[1] Adida, B., and Wikström, D. Offline/online mixing. In

ICALP 2007 (2007), pp. 484–495.

[2] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), pp. 535–548.

[3] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer extensions with security for malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), Springer, pp. 673–701.

[4] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer extensions. *Journal of Cryptology 30*, 3 (2017), 805–858.

[5] Aumann, Y., and Lindell, Y. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference* (2007), Springer, pp. 137–156.

[6] Beaver, D. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (1996), pp. 479–488.

[7] Bellare, M., and Micali, S. Non-Interactive oblivious transfer and applications. In *Conference on the Theory and Application of Cryptology* (1989), pp. 547–557.

[8] Bernstein, D. J. Cryptography in NaCl.

[9] Brickell, J., and Shmatikov, V. Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), pp. 76–85.

[10] Camenisch, J., and Lysyanskaya, A. A formal treatment of onion routing. In *Annual International Cryptology Conference* (2005), Springer, pp. 169–187.

[11] Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 4*, 2 (1981), 84–88.

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

[13] Chaum, D. Precomputed and transactional mixing. United States Patent, No. 10375042, Aug. 6, 2019.

[14] Chaum, D., Das, D., Javani, F., Kate, A., Krasnova, A., De Ruiter, J., and Sherman, A. T. cMix: Anonymization by high-performance scalable mixing. In *Cryptology ePrint Archive, Report 2016/008 (2016).* (2016). https://eprint.iacr.org/2016/008.pdf.

[15] Chaum, D., Das, D., Javani, F., Kate, A., Krasnova, A., De Ruiter, J., and Sherman, A. T. cMix: Mixing with minimal real-time asymmetric cryptographic operations. In *International Conference on Applied Cryptography and Network Security* (2017), Springer, pp. 557–578.

[16] Chaum, D., Mario, Sherman, A., and Joeri. Udm: User discovery with minimal information disclosure. Arxiv, accepted to *Cryptologia*, 2020.

[17] Chen, C., Asoni, D. E., Barrera, D., Danezis, G., and Perrig, A. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 1441–1454.

[18] Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science* (1995), IEEE, pp. 41–50.

[19] Chou, T., and Orlandi, C. The simplest protocol for oblivious transfer. In *International Conference on Cryptology and Information Security in Latin America* (2015), Springer, pp. 40–58.

[20] Crépeau, C. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In *Conference on the Theory and Application of Cryptographic Techniques* (1986), Springer, pp. 239–247.

[21] Crépeau, C., van de Graaf, J., and Tapp, A. Committed oblivious transfer and private multi-party computation. In *Annual International Cryptology Conference* (1995), Springer, pp. 110–123.

[22] Danezis, G., and Serjantov, A. Statistical disclosure or intersection attacks on anonymity systems. In *International Workshop on Information Hiding* (2004), Springer, pp. 293–308.

[23] DeSantis, A., DiCrescenzo, G., and Persiano, G. Zero-knowledge arguments and public-key cryptography. *Information and Computation 121*, 1 (1995), 23–40.

[24] Dolev, D., and Yao, A. On the security of public key protocols. *IEEE Trans. Inf. Theor. 29*, 2 (Sept. 2006), 198–208.

[25] Dwork, C., and Roth, A. The algorithmic foundations of differential privacy. *Theoretical Computer Science 9*, 3-4 (2014), 211–407.

[26] Even, S., Goldreich, O., and Lempel, A. A randomized protocol for signing contracts. *Communications of the ACM 28*, 6 (1985), 637–647.

[27] Fagin, R., Naor, M., and Winkler, P. Comparing information without leaking it. *Communications of the ACM 39*, 5 (1996), 77–85.

[28] Furukawa, J., and Sako, K. An efficient scheme for proving a shuffle. In *Annual International Cryptology Conference* (2001), Springer, pp. 368–387.

[29] Goldwasser, S., and Levin, L. Fair computation of general functions in presence of immoral majority. In *Conference on the Theory and Application of Cryptography* (1990), Springer, pp. 77–93.

[30] Golle, P., Jakobsson, M., Juels, A., and Syverson, P. Universal re-encryption for mixnets. In *Cryptographers' Track at the RSA Conference* (2004), Springer, pp. 163–178.

[31] Halevi, S., and Micali, S. Practical and provably-secure commitment schemes from collision-free hashing. In *Annual International Cryptology Conference* (1996), Springer, pp. 201–215.

[32] Impagliazzo, R., and Rudich, S. Limits on the provable consequences of one-way permutations. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1989), STOC '89, Association for Computing Machinery, p. 44–61.

[33] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference* (2003), Springer, pp. 145–161.

[34] Jakobsson, M. Flash mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing* (1999), pp. 83–89.

[35] Jakobsson, M., and Juels, A. An optimally robust hybrid mix network. In *Proceedings of the Twentieth Annual ACM*

Symposium on Principles of Distributed Computing (2001), pp. 284–292.

[36] JAVANI, F. *Privacy-Preserving Protocols with Oblivious Transfer and Mix Networks*. PhD thesis, University of Maryland, Baltimore County, April 2021.

[37] JAVANI, F., AND SHERMAN, A. T. BVOT: Self-tallying boardroom voting with oblivious transfer. *arXiv preprint arXiv:2010.02421* (2020). https://arxiv.org/abs/2010.02421.

[38] KEDOGAN, D., AGRAWAL, D., AND PENZ, S. Limits of anonymity in open environments. In *International Workshop on Information Hiding* (2002), Springer, pp. 53–69.

[39] KESDOGAN, D., EGNER, J., AND BÜSCHKES, R. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *International Workshop on Information Hiding* (1998), Springer, pp. 83–98.

[40] KILIAN, J. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (1988), pp. 20–31.

[41] KRAWCZYK, H. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference* (2010), Springer, pp. 631–648.

[42] KRAWCZYK, H., AND ERONEN, P. MAC-based extract-and-expand key derivation function (HKDF). Internet Engineering Task Force (IETF), Request for Comments: 5869, May 2010. https://tools.ietf.org/html/rfc5869.

[43] KWON, A., LAZAR, D., DEVADAS, S., AND FORD, B. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies 2016*, 2 (2016), 115–134.

[44] MÖLLER, B. Provably secure public-key encryption for length-preserving chaumian mixes. In *Cryptographers' Track at the RSA Conference* (2003), Springer, pp. 244–262.

[45] NAOR, M., AND PINKAS, B. Efficient oblivious transfer protocols. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms* (2001), pp. 448–457.

[46] NAOR, M., AND PINKAS, B. Computationally secure oblivious transfer. *Journal of Cryptology 18*, 1 (2005), 1–35.

[47] NIELSEN, J. B., NORDHOLT, P. S., ORLANDI, C., AND BURRA, S. S. A new approach to practical active-secure two-party computation. In *Annual Cryptology Conference* (2012), Springer, pp. 681–700.

[48] NURMI, H., SALOMAA, A., AND SANTEAN, L. Secret ballot elections in computer networks. *Computers & Security 10*, 6 (1991), 553–560.

[49] OHKUBO, M., AND ABE, M. A length-invariant hybrid mix. In *International Conference on the Theory and Application of Cryptology and Information Security* (2000), Springer, pp. 178–191.

[50] ØVERLIER, L., AND SYVERSON, P. Improving efficiency and simplicity of tor circuit establishment and hidden services. In *International Workshop on Privacy Enhancing Technologies* (2007), Springer, pp. 134–152.

[51] PARK, C., ITOH, K., AND KUROSAWA, K. Efficient anonymous channel and all/nothing election scheme. In *Workshop on the Theory and Application of of Cryptographic Techniques* (1993), Springer, pp. 248–259.

[52] PEIKERT, C., VAIKUNTANATHAN, V., AND WATERS, B. A framework for efficient and composable oblivious transfer. In *Annual International Cryptology Conference* (2008), Springer, pp. 554–571.

[53] PFITZMANN, A., AND KÖHNTOPP, M. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing Privacy Enhancing Technologies* (2001), Springer, pp. 1–9.

[54] PFITZMANN, A., AND WAIDNER, M. Networks without user observability—design options. In *Workshop on the Theory and Application of of Cryptographic Techniques* (1985), Springer, pp. 245–253.

[55] PIOTROWSKA, A. M., HAYES, J., ELAHI, T., MEISER, S., AND DANEZIS, G. The Loopix anonymity system. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), pp. 1199–1216.

[56] RABIN, M. O. How to exchange secrets with oblivious transfer. *Technical Report TR-81, Aiken Computation Lab, Harvard University* (1981).

[57] RACKOFF, C., AND SIMON, D. R. Cryptographic defense against traffic analysis. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), pp. 672–681.

[58] RAYMOND, J.-F. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies* (2001), Springer, pp. 10–29.

[59] REED, M., SYVERSON, P., AND GOLDSCHLAG, D. Anonymous Connections and Onion Routing. *IEEE J-SAC 16*, 4 (1998), 482–494.

[60] SERJANTOV, A., AND DANEZIS, G. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies* (2002), Springer, pp. 41–53.

[61] SERJANTOV, A., DINGLEDINE, R., AND SYVERSON, P. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding* (2002), Springer, pp. 36–52.

[62] SHERMAN, A. T., LANUS, E., LISKOV, M., ZIEGLAR, E., CHANG, R., GOLASZEWSKI, E., WNUK-FINK, R., BONYADI, C. J., YAKSETIG, M., AND BLUMENFELD, I. Formal methods analysis of the secure remote password protocol. In *Logic, Language, and Security: Essays dedicated to Andre Scedrov on the occassion of his 65th birthday* (February 2020), e. a. Nigam, Ed., vol. 12300 of *LNCS Festscrift*, Springer, pp. 103–126. Available as https://arxiv.org/pdf/2003.07421.pdf.

[63] SYTA, E., CORRIGAN-GIBBS, H., WENG, S.-C., WOLINSKY, D., FORD, B., AND JOHNSON, A. Security analysis of accountable anonymity in dissent. *ACM Transactions on Information and System Security (TISSEC) 17*, 1 (2014), 1–35.

[64] SYVERSON, P., DINGLEDINE, R., AND MATHEWSON, N. Tor: The secondgeneration onion router. In *USENIX Security* (2004), pp. 303–320.

[65] TYAGI, N., GILAD, Y., LEUNG, D., ZAHARIA, M., AND ZELDOVICH, N. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), pp. 423–440.

[66] VAN DEN HOOFF, J., LAZAR, D., ZAHARIA, M., AND ZELDOVICH, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), pp. 137–152.

[67] WOLINSKY, D. I., CORRIGAN-GIBBS, H., FORD, B., AND JOHNSON, A. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems*

Design and Implementation (OSDI 12) (2012), pp. 179–182.

A Appendix

A.1 Notation

Table 2. Notation.

C	container of nodes at Level 1
Ψ	batch of messages at Level-2 nodes
Φ	bucket of messages at Level-3 nodes
β_1	batch size at Level-1 nodes
β_2	batch size of real messages at Level-2 nodes
γ	number of messages in pub. lists of Level-3 nodes
ζ	number of messages from which to choose in OT
ρ	number of passive Level-3 nodes
α	number of active Level-3 nodes
$\sigma_{A,B}$	shared secret between sender A and receiver B
k_{A_j,B_j}	tag of message j from sender A to receiver B
P	partition of the set of β_2 indexes
x_j	payload of message j
$E[k, M]$	encryption of message M under key k
p_b	public key of B
s_b	secret key of B
h	hash function
λ	number of batches in each message repository
$\mathcal{R}_i$	round i
ω	residual pool size
Ω	number of incoming messages into pool
T_1	maximum interval for verifying messages
T_2	maximum interval for dummy requests
n	nonce
ts	timestamp
ξ	counter deviation bound

A.2 Proof of Theorem 1

Proof. (of Theorem 1)

We reduce the receiver security of OT to the receiver anonymity of AOT.

Define the *OTRS Problem* as follows: in an OT_1^η oblivious transfer session with the sender $\mathcal{A}_{OT}$ with η strings $s_1, s_2, \dots s_\eta$, given $1 \le i < j \le \eta$ where the receiver chooses either s_i or s_j, distinguish whether the receiver chose s_i or s_j.

We define the OTRS game between the challenger $\mathcal{C}_{OT}$ and the adversary $\mathcal{A}_{OT}$ as follows:

Step 1. $\mathcal{A}_{OT}$ runs the protocol for the sender, and $\mathcal{C}_{OT}$ runs the protocol for the receiver.

Step 2. For as many times as $\mathcal{A}_{OT}$ wants to, $\mathcal{C}_{OT}$ engages in OT_1^η with $\mathcal{A}_{OT}$.

Step 3. $\mathcal{A}_{OT}$ generates the strings $s_1, s_2, \dots s_\eta$. $\mathcal{C}_{OT}$ chooses a string with OT_1^η; let s_a denote $\mathcal{C}_{OT}$'s choice. $\mathcal{C}_{OT}$ sends the indexes i and j, for all $1 \le i < j \le \eta$, to $\mathcal{A}_{OT}$ where $a \in \{i, j\}$.

Step 4. For as many times as $\mathcal{A}_{OT}$ wants to, $\mathcal{C}_{OT}$ engages in OT_1^η with $\mathcal{A}_{OT}$.

Step 5. $\mathcal{A}_{OT}$ outputs its guess of a.

Any instance of the OTRS game can be transformed into an instance of the anonymity game as follows.

In Step 1 of the anonymity game, $\mathcal{C}$ will initiate an OTRS game with the challenger $\mathcal{C}_{OT}$. In Step 2 of the anonymity game, $\mathcal{C}$ simulates the honest users as many times as $\mathcal{A}$ desires.

In Step 3 of the anonymity game, after $\mathcal{C}$ receives the target message m and two honest users c_0 and c_1 from $\mathcal{A}$, it chooses c_b (by tossing a coin) to be the recipient of m, and c_{1-b} to be user that initiates a dummy request to the publication list. $\mathcal{C}$ will set the message that $\mathrm{OT}_1^{\beta_2}$ will pretend to ask for in the dummy request to be the choice of $\mathcal{C}_{OT}$ in the oblivious transfer in the OTRS game. $\mathcal{C}$ will run the protocol for c_b and c_{1-b} as follows.

In $\mathcal{C}$'s simulation of c_b, user c_b will ask for m with $\mathrm{OT}_1^{\beta_2}$ from $\mathcal{A}$. To simulate c_{1-b}, challenger $\mathcal{C}$ starts Step 3 of the OTRS game with $\mathcal{C}_{OT}$, where $\mathcal{C}$'s strings are the messages in the publication list of the Level-3 node. During the $\mathrm{OT}_1^{\beta_2}$ oblivious transfer session of the OTRC game, when $\mathcal{C}$ receives a message from $\mathcal{C}_{OT}$, it will send the message to $\mathcal{A}$, pretending that the message originated from c_{1-b}. When c_{1-b} receives a message from $\mathcal{A}$, $\mathcal{C}$ will forward the message to $\mathcal{C}_{OT}$, pretending that the message originated from $\mathcal{C}$.

At the end of Step 3 of the OTRS game, $\mathcal{C}_{OT}$ sends the indexes i and j to $\mathcal{C}$. Let m_i and m_j denote the messages corresponding to i and j, respectively, from the publication repository, and let m_a denote $\mathcal{C}_{OT}$'s choice. At the end of Step 3 of the anonymity game—as its guess for m_a—$\mathcal{C}$ sends m_i to $\mathcal{A}$. If $\mathcal{A}$ guesses the value b correctly, the challenger $\mathcal{C}$ knows that its guess for m_a is correct and $\mathcal{A}$ has concluded that c_{1-b} has asked for m_i in a dummy request. Therefore, in this case, $\mathcal{C}$ outputs i for its guess of a to $\mathcal{C}_{OT}$. If $\mathcal{A}$ guesses the value b incorrectly, then $\mathcal{C}$ outputs j for its guess of a.

If $\mathcal{A}$'s advantage in the anonymity game is non-negligible, then $\mathcal{A}_{OT}$ can distinguish between s_i or s_j, contradicting the fact that AOT uses an OT protocol that provides receiver security.

□

A.3 Anonymity of a Pool-Mix AOT

Using entropy, we analyze the anonymity of a pool-mix version of AOT. Pool-mixes offer additional defenses to blending attacks.

A *pool mix* [61] stores a residual pool of ω messages (similar to AOT's message repository); whenever it receives Ω messages, it adds them into the pool, selects Ω messages randomly from the $\omega + \Omega$ messages in the pool and outputs them. The random selection of messages causes some of the messages to be delayed for a long time—potentially infinitely. By contrast, in AOT, Level-3 nodes store messages before publication for less than τ seconds; that is, messages are chosen from the message repository from the last λ batches such that each message is published in less than τ time after it enters a Level-3 node.

If this maximum delay is removed from the system, AOT can be converted into a pool mix as follows: AOT selects β_2/α messages randomly from $\beta_2(\lambda+1)/(2\alpha)$ messages in the message repository—as opposed to selecting $\beta_2/(\alpha\lambda)$ from each bucket—for each publication (see Figure 4b).

When AOT selects messages randomly from each bucket in the message repository, AOT publishes messages from each of the last λ rounds with the same probability. Therefore, the size of the anonymity set is an appropriate measure of anonymity. Serjantov and Danezis [60], however, show that anonymity set is not an appropriate measure to analyze the anonymity of pool mixes, because size of the anonymity set does not reflect that messages from different rounds have different probabilities of exiting the mix at each round. Instead, Serjantov and Danezis propose using entropy to measure the anonymity of pool mixes.

Serjantov and Danezis define the effective size of the sender anonymity set as 2^E, where E is the entropy of the probability distribution $\mathcal{U}$, and $\mathcal{U}$ is "the attacker's a-posteriori probability distribution" of all the users being the sender of a specific message. The same definition applies for the receiver anonymity set.

Applying the analysis of Serjantov and Danezis, for a pool mix with pool size ω, where Ω messages are output at each round, when the number of rounds k approaches infinity, we have:

$$\lim_{k\to\infty} E = \left(1 + \frac{\omega}{\Omega}\right) \log\left(\omega + \Omega\right) - \frac{\omega}{\Omega} \log \omega. \tag{7}$$

Considering the message repository of each Level-3 node in AOT, to compare the anonymity sets of AOT with those of a pool-mix version of AOT, we set $\omega = \beta_2(\lambda - 1)/(2\alpha)$ and $\Omega = \beta_2/\alpha$. For each Level-3 node we have:

$$\lim_{k\to\infty} E = \log \frac{\lambda\beta_2}{\alpha} + \log \frac{(\lambda+1)^{\frac{\lambda+1}{2}}}{2\lambda(\lambda-1)^{\frac{\lambda-1}{2}}}, \tag{8}$$

where $\lambda\beta_2/\alpha$ is the size of the anonymity set of each Level-3 node in AOT.

Therefore, removing the maximum delay from AOT and converting AOT to a pool mix will add

$$\log \frac{(\lambda+1)^{\frac{\lambda+1}{2}}}{2\lambda(\lambda-1)^{\frac{\lambda-1}{2}}} \tag{9}$$

to the entropy of the probability distribution $\mathcal{U}$. That is, in a pool-mix version of AOT, the sender anonymity set size will increase by a factor of

$$2^{\left(\log \frac{(\lambda+1)^{\frac{\lambda+1}{2}}}{2\lambda(\lambda-1)^{\frac{\lambda-1}{2}}}\right)}. \tag{10}$$

A poll-mix version of AOT will result in the following latency: messages will be published after an average of $(\lambda + 1)/2$ rounds, with a variance of $(\lambda+1)^2(\lambda-1)/8$ rounds. By contrast, regular AOT publishes messages after an average of $\lambda/2$ rounds, with a variance of $(\lambda^2-1)/12$ rounds.

For example, if we set $\lambda = 9$, for each Level-3 node, the size of the effective anonymity set of a pool-mix version of AOT will be larger than the size of anonymity set of AOT by a factor of $2^{0.44} \approx 1.35$—at the expense of potentially delaying some messages infinitely. It will take the pool version of AOT 5 rounds on average to deliver a message with a variance of 100 rounds.

A.4 Public-Key Encryption with Ciphertext Integrity

AOT requires the encryption function E (Section 4.2) to provide confidentiality and integrity. For example, AOT could use Bernstein's [8] crypto_box function, which performs public-key encryption as follows.

1. Generate a symmetric key at random.
2. Encrypt the packet using the symmetric key.
3. Hash the encrypted packet using a cryptographic hash function.
4. Sign the hash value using Alice's secret key.
5. Encrypt the symmetric key, hash, and signature using Bob's public key.
6. Concatenate the ciphertext (from Step 5) with the encrypted packet (Step 2).

A.5 Acronyms and Abbreviations

ACS	Anonymous Comumnication System
AOT	Anonymity by Oblivious Transfer
GPA	Global Passive Adversary
OT	Oblivious Transfer
MAC	Message Authentication Code
PIR	Private Information Retrieval
PPT	Probabilistic Polynomial-Time Turning Machine
RAC	Receiver Anonymity with Compromised network