YouChoose: A Lightweight Anonymous Proof of Account Ownership

Aarav Varshney
College of Computing and Data Science
Nanyang Technological University
Singapore, Singapore
aarav.varshney@ntu.edu.sg

Prashant Agrawal
Max Planck Institute for Security and Privacy
Bochum, Germany
prashant.agrawal@mpi-sp.org

Mahabir Prasad Jhanwar
Department of Computer Science and Centre for Digitalisation, AI and Society
Ashoka University
Sonepat, India
mahavir.jhawar@ashoka.edu.in

Abstract

Anonymous proofs of account ownership (anonymous PAOs) enable a prover to convince a verifier that the prover owns a valid account at a server without revealing their identity. Importantly, no server-side changes are required - the server does not even learn that such a proof is taking place. This functionality is particularly valuable in sensitive applications such as whistleblowing, or in settings where infrastructural changes at the server are impractical. Anonymous PAOs were first introduced by Wang et al. (IEEE S&P 2019), who realized them in the email setting via secure channel injection (SCI), a technique that requires carefully engineered multi-party computation (MPC) protocols. In this work, we propose *YouChoose*, a new approach to anonymous PAO that avoids MPC entirely, relying instead on the verifier to selectively forward TLS records. Compared to SCI, YouChoose is simpler, faster, and more adaptable across services. We also provide the first formal security definition for anonymous PAOs, establishing a rigorous framework for analysis and future constructions. Finally, we present a prototype implementation of YouChoose, evaluate it in the context of email and other social media accounts, showing its better performance than SCI with minimal latency overhead in practice.

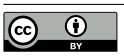
CCS Concepts

• **Security and privacy → Privacy-preserving protocols**; **Authentication**; **Pseudonymity, anonymity and untraceability**.

Keywords

Anonymous Proofs of Account Ownership, TLS, Email, Privacy

ACM Reference Format:
Aarav Varshney, Prashant Agrawal, and Mahabir Prasad Jhanwar. 2026. YouChoose: A Lightweight Anonymous Proof of Account Ownership. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3774904.3792744



This work is licensed under a Creative Commons Attribution 4.0 International License.
WWW '26, Dubai, United Arab Emirates
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792744

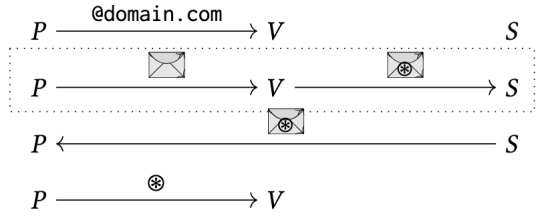
1 Introduction

Anonymous credentials provide a powerful tool for making assertions about one's identity while maintaining privacy. In an anonymous credential scheme, individuals identify themselves via different unlinkable pseudonyms with different organizations, thus preventing even colluding credential issuers and verifiers from identifying and tracking the users. More precisely, say a person P wants to anonymously prove to a verifier V that P possesses a credential from a credential issuer I. P obtains a credential from I against P's pseudonym P_I with I and *transforms* it to a credential against its pseudonym P_V with V. This convinces V that the credential is genuine, but even if I and V actively collude, they cannot link the pseudonyms P_I and P_V together.

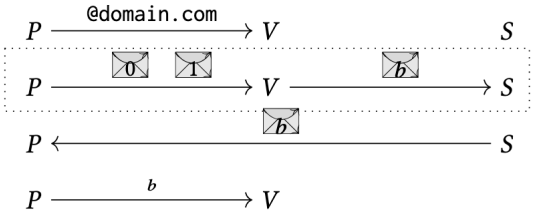
However, in some applications, it may be unrealistic for P to even obtain anonymous credentials from I. For instance, consider a whistleblowing use-case where an employee wants to anonymously report misconduct at their organization to a journalist and the journalist wants assurance that the information is sent by a genuine employee of the organization. In this case, the employee cannot use a standard anonymous credential scheme to prove their employee status, simply because merely using a privacy tool such as an anonymous credential would make I suspicious of P and unwilling to issue such a (potentially self-incriminating) credential.

Also, in some use-cases, issuers' business interests may not align with users' privacy interests and thus infrastructural changes to support anonymous credentials may be deprioritized. For instance, consider OpenID [29], which enables users to seamlessly access multiple websites or applications by allowing them to prove account ownership at identity providers like Google or Twitter. In these proofs, the identity providers necessarily learn the websites the users visit, which serves their business interests of user profiling and targeted advertising. Thus, supporting anonymous proofs that diminish this ability is likely to get deprioritized by them.

Therefore, a practical user-privacy question is whether the user can anonymously authenticate itself to the verifier without necessitating infrastructural changes at the issuer (as in OpenID) and without even revealing to the issuer that such authentication is taking place (as in whistleblowing). We call such protocols *anonymous proofs of account ownership* or *anonymous PAOs*. As an example, in an anonymous PAO for email accounts, a prover proves to a verifier that they own an email account with an email account provider, without revealing the account details to the verifier and that an anonymous PAO is taking place to the email account provider.



(a) Anonymous PAO using SCI



(b) Anonymous PAO using YouChoose

Figure 1: SCI based anonymous PAO (top) versus YouChoose based anonymous PAO (bottom). In the top figure, ⊛ denotes the email forwarded by V containing the injected challenge. In the bottom figure, 0 and 1 denote the two email messages sent by P and bit b denotes the email forwarded by V.

Wang et al. [37] introduced the first anonymous PAO protocol for email accounts, based on the prover's ability to send emails from their claimed email address. The verifier V acts as a network proxy sitting between the prover P and the email server S (see Figure 1a). P sends an email from their claimed email account, say `alice@domain.com`, to themselves or some other account they own. V injects a challenge into the email's contents. This is done via a multiparty party computation (MPC) protocol called *secure channel injection* (SCI) [37] between P and V. SCI allows P and V to jointly compute an authenticated and encrypted message containing the injected challenge, such that P does not learn V's secret challenge and V does not learn P's email contents, credentials and session keys. Later, if P is able to produce the challenge to V (by accessing either the sent emails folder of the claimed email account or the inbox of the recipient account), then they prove to V that they own an account at S. This is a proof of account ownership because S successfully sends emails from P only if it can verify P's account credentials. Note that S does not need to do anything special; it simply sends emails as a regular email server and does not even learn about an anonymous PAO happening (as long as the injected message itself does not look suspicious).

In this paper, we introduce *YouChoose*, an alternative anonymous PAO approach that is considerably lighter-weight and general than the above MPC-based SCI approach. The basic idea behind YouChoose is very simple (see Figure 1b). P sends two email messages m_0 and m_1 from `alice@domain.com` to that same account or another account they have access to. V forwards only one of the emails m_b, for a randomly chosen $b \in \{0, 1\}$, to `domain.com`'s email server S. P's challenge is to find out whether m_0 or m_1 was forwarded. To complete the proof, P accesses the sent email and sends the correct bit b to V. This proves with a soundness error of 1/2 that S had forwarded the email and thus P is an eligible `domain.com` user.

Although the soundness error can be made negligibly small by simply repeating the above process, this would require P to send multiple emails. The YouChoose protocol is carefully designed to achieve a negligible soundness error with just one email instead: P splits its two messages into multiple TLS records as $m_0 := m_{0,1}m_{0,2}\ldots m_{0,k}$ and $m_1 := m_{1,1}m_{1,2}\ldots m_{1,k}$ and V forwards an interleaved message $m := m_{d_0,0}m_{d_1,1}\ldots m_{d_k,k}$ for a randomly chosen k-length bit vector $\vec{d}$. The main challenges lie in carefully navigating around the integrity protections imposed by TLS, ensuring that P does not learn the bits chosen by V, and ensuring that S remains oblivious of the entire anonymous PAO protocol happening.

The YouChoose approach of selectively forwarding messages is actually fairly general and can prove account ownership beyond just the email-over-TLS setting. We show how the YouChoose protocol can be extended to a wide class of *authenticated messaging protocols* (AMPs) — protocols allowing (only) authenticated users to post messages through the server — and stateful *secure channel* (SC) protocols. This generality allows users to anonymously prove to service providers like Quora that they own an account at an identity provider like Twitter, while preventing Twitter from learning the service used by the user (Quora) and without requiring any infrastructural changes at Twitter (in this case, obliviousness of Twitter to anonymous PAO happening is not strictly necessary). Our general approach also readily supports multiple ciphersuites of TLS (or even secure channels like SSH), whereas for SCI, specific MPC-based protocols need to be designed for each secure channel.

Our main contributions are as follows:

- We formalize the *modification-based paradigm* of anonymous PAOs, where V acts as a proxy between P and S, modifies messages in the secure channel between P and S and uses the modifications as a challenge for P (Section 3). We call this notion a Δ-*anonymous PAO*, where Δ is a neighborhood function modelling the desired degree of server obliviousness ($\Delta(m)$ denotes the set of messages that S cannot tell apart from m). Our security definitions generalize those given in [37].
- We present the YouChoose protocol for the email-over-TLS setting, and discuss some optimizations (Section 4). We introduce a novel steganographic technique here, which creates small intensity variations to an image to achieve server obliviousness.
- We identify the core restrictions imposed by the YouChoose protocol on the choice of AMPs and SCs and show that it is Δ-anonymous PAO for an appropriate choice of Δ (Section 5).
- We build a prototype for YouChoose and evaluate it for the email-over-TLS setting (Section 6). As expected, YouChoose achieves significantly better performance over the MPC-based SCI. It is up to 10× faster than SCI while adding only 0.1–0.3s latency over standard email delivery. We also implement YouChoose against the Twitter query API acting as the AMP. These experiments demonstrate the simplicity, efficiency and generality of our approach. Our implementation is available at [36].

2 Related Work

Traditional anonymous communication tools fail to achieve the strict requirements of server obliviousness of anonymous PAOs.

Secure messaging applications like Signal [5] and SecureDrop [8], as well as systems like mix-nets [2] and onion routing (e.g., Tor [7]), allow users to communicate anonymously. However, they do not let users anonymously authenticate themselves to others.

Group signatures [3] allow a group member to sign a message on behalf of the group without revealing their identity. However, it violates the privacy we seek as the group manager can identify the signer. Ring signatures [28] avoid a group manager but require all members' public keys to be published in a trusted directory, requiring cooperation of other members.

Anonymous credentials allow a prover to prove assertions about their identity without revealing the identity itself. These assertions are certified by a trusted Certificate Authority (CA) [1, 6, 23], who learns that the prover is obtaining an anonymous credential.

Anonymous PAOs. In addition to the SCI approach [37], the following anonymous PAO approaches also exist.

Multi-context TLS (mcTLS) [21] extends TLS to support on-path middleboxes that proxy between client and server to inspect/modify exchanged data, similar to the SCI verifier. However, mcTLS requires a new protocol incompatible with existing TLS.

Mailet [17] is a censorship-circumvention system helping users in censored regions access social media websites. It uses distributed proxy servers to facilitate communication between client and website. The client must secret-share its credentials among these proxies, which then use secure computation to collectively compute the TLS record, enabling authentication with the server. While this protocol may be used for anonymous PAO (where proxies act as the SCI verifier), colluding proxies could impersonate the client and forge messages.

3 Formalizing Δ-anonymous PAOs

Notation. We let $x \leftarrow_R \mathcal{S}$ denote an element x sampled uniformly at random from a set $\mathcal{S}$ and $2^{\mathcal{S}}$ denote the power set of $\mathcal{S}$. Given a positive integer x, we let $[x]$ denote the set $\{0, \ldots, x-1\}$. We use $\perp$ to indicate an empty message or failure, $\|$ to denote concatenation of elements, and $\cdot$ to denote a variable whose value we do not care. We use H to denote a hash function modelled as a random oracle. We denote an interactive protocol Prot between parties $P_1, \ldots, P_k$ as $(P_i\langle so_i\rangle)_{i\in[k]} \leftarrow \mathsf{Prot}(ci, (P_i\langle si_i\rangle)_{i\in[k]})$, where ci denotes the common input of each party, si_i denotes the secret input of P_i and so_i denotes the secret output of P_i. We abbreviate *probabilistically polynomial time* as PPT.

3.1 Secure Channels

We rely on a stateful secure channel like TLS or SSH to ensure confidentiality and integrity of messages (see Definition 3.1). It consists of a handshake protocol Handshake for *authenticated key exchange*. After the handshake, the channel provides a *stateful authenticated encryption* interface AEnc/ADec with explicit state updates via NextState.

Definition 3.1. (Secure channel) A *secure channel protocol* $\mathsf{SC} := \langle \mathsf{Handshake}, \mathsf{AEnc}, \mathsf{ADec}, \mathsf{NextState}\rangle$ is an interactive protocol between a client P and a server S, defined as follows:

- $P\langle k, \mathsf{st}_0\rangle, S\langle k, \mathsf{st}_0\rangle \leftarrow \mathsf{Handshake}(1^n, \mathsf{pk}_S, P\langle\rangle, S\langle \mathsf{sk}_S\rangle)$ is a protocol that takes as common input a security parameter n (given in unary), and S's public key pk_S. S inputs its secret key sk_S as its private input. The protocol outputs a session key $k \in \mathcal{K}_{\mathsf{SC}}$, where $\mathcal{K}_{\mathsf{SC}}$ represents the key space of the secure channel, and an initial state st_0 for both P and S.
- $c \leftarrow \mathsf{AEnc}(k, m, \mathsf{st})$ is an authenticated encryption algorithm. It takes as input the session key k, a message $m \in \Sigma_{\mathsf{SC}}$, and an internal state st, and outputs a ciphertext $c \in C_{\mathsf{SC}}$, where Σ_{SC} and C_{SC} respectively represent the message space and ciphertext space of the secure channel. It is assumed that any integrity-preserving tags are embedded within the ciphertext.
- $m/\perp \leftarrow \mathsf{ADec}(k, c, \mathsf{st})$ is an authenticated decryption algorithm. It takes as input the session key k, a ciphertext c and an internal state st and outputs a message m if c was obtained by running $\mathsf{AEnc}(k, m, \mathsf{st})$ and $\perp$ otherwise.
- $\mathsf{st}' \leftarrow \mathsf{NextState}(\mathsf{st})$ is a deterministic algorithm that updates the internal state. Both client and server must maintain synchronized states, updating them identically after each encryption/decryption operation to ensure proper message ordering and replay protection.

We assume that SC satisfies the *Authenticated and Confidential Channel Establishment (ACCE)* model [13], which guarantees confidentiality and ciphertext integrity of transmitted data, and—through statefulness—prevents reordering, duplication, and dropping of ciphertext blocks. In practice, both TLS and SSH instantiate this by maintaining monotonically increasing sequence numbers as part of their internal state, updated after each invocation of AEnc or ADec.

3.2 Authenticated Messaging Protocols

Definition 3.2 models an authenticated messaging protocol. Note that it includes a proxy V between P and S, which naturally captures realistic network scenarios where intermediaries handle message routing (e.g., ISPs, CDNs, or application-layer proxies). In the PAO setting, this abstraction allows V to represent the verifier.

Definition 3.2. (Authenticated messaging protocol) Given a secure channel protocol SC, an *authenticated messaging protocol* AMP is a three-party protocol between an account owner P, an account provider S and a proxy V forwarding messages between P and S:

$$V\langle \mathsf{st}_V\rangle, S\langle m'\rangle \leftarrow \mathsf{AMP}^{\mathsf{SC}}(1^n, \mathsf{pk}_S, P\langle\sigma, m\rangle, V\langle\rangle, S\langle \mathsf{sk}_S\rangle)$$

where n is a security parameter, pk_S is S's public key, $\sigma \in \{0,1\}^*$ is P's account credential, $m \in \{0,1\}^*$ is the message sent by P to S, and sk_S is the secret key of S. S verifies P's credentials; if verification succeeds it outputs $m' = m$, and otherwise outputs $m' = \perp$. V outputs its internal state st_V obtained during the protocol.

We say that P *owns an* AMP *account at* S if there exists some valid credential σ such that the protocol execution yields $m' \neq \perp$. In this case, σ is called a *valid* AMP *credential at* S.

We also assume that P can retrieve the messages posted to S via an out-of-band channel. For example, in the case of email, P may be able to retrieve the email either from their sent emails folder or from the recipient's inbox if it was sent to their own account.

3.3 Modification-based Δ-anonymous PAOs

Assume that S provides an AMP that lets only legitimate account owners P to post a message through it (e.g., the SMTP protocol for email account providers or an authenticated HTTP POST API

exposed by social media platforms). Let m denote an innocuous looking message that a legitimate user would post via the AMP and $\Delta(m)$ denote a set of messages that S (or anyone else) cannot distinguish with m (e.g., m could be a cat image and $\Delta(m)$ could be the set of messages with some small noise added to m). The goal of Δ-anonymous PAO is for P to anonymously prove to V that it can post a message via the AMP (and hence is a legitimate user) while ensuring that S only sees a message within $\Delta(m)$.

We formalize a special-case of Δ-anonymous PAOs that we call *modification-based anonymous PAOs*, where P submits a message to be posted via the AMP, while V, acting as a proxy, introduces controlled modifications and forwards the modified message to S. P later retrieves the posted message from S through an out-of-band channel, extracts the modifications, and submits them back to V. If the response is correct, V accepts P as the legitimate account owner at S. At the end of the protocol, V and P obtain a shared key, which establishes a secure channel between them for communication of actual sensitive information after P is authenticated.

Such a PAO should informally satisfy the following properties:

(1) **(Completeness)** An honest account owner P can recover the modifications from the message retrieved from S.
(2) **(Soundness)** If V successfully establishes a key with a (potentially malicious) prover P', then P' must indeed have the ability to send a message through S (verifier security). This must hold even if P' is in the network path of another anonymous PAO session between an honest prover P, V and S.
In addition, P should only establish a key with the legitimate verifier (owner of V's secret key), else it may be fooled into sharing sensitive information to the adversary (prover security).
(3) **(Transcript privacy)** V should not learn any information about the message contents beyond what the underlying AMP leaks.
(4) **(Transcript integrity)** The modifications must be limited in scope to prevent V from posting arbitrary messages for P.
(5) **(Server obliviousness - optional)** The protocol execution must avoid S detecting any non-standard AMP behavior by P.

We keep server obliviousness as an optional requirement because it appears necessary for the whistleblowing use-case but not for the anonymous OpenID use-case, where it may be reasonable for S to learn that anonymous PAO is happening as long as it does not require any infrastructural changes and does not identify V.

We use the neighborhood function Δ to capture both the modifications that V is allowed to make and the ability of S to detect non-standard messages, by requiring that V may only produce a modified message $m' \in \Delta(m)$ for S.

Definition 3.3. (Δ-anonymous PAO) Given an AMP protocol AMP between an account owner P, a proxy V, and an account provider S, and a *neighborhood function* $\Delta : \{0,1\}^* \to 2^{\{0,1\}^*}$, a Δ-*anonymous PAO* under AMP is a protocol between P, V and S, where P acts as a prover, V acts as a verifier, and S continues acting exactly as the AMP account provider, with the following structure:

$$P\langle \mathsf{st}_P, s_P\rangle, V\langle \mathsf{st}_V, s_V\rangle, S\langle m'\rangle \leftarrow \mathsf{AnonPAO}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, V\langle \mathsf{sk}_V\rangle, S\langle \mathsf{sk}_S\rangle)$$

Here, n is a security parameter, public keys pk_V, pk_S of V and S are common input; P inputs its AMP account credential $\sigma \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$; S and V input their secret keys sk_S and sk_V; S outputs $m' = m$ if authenticated, else $m' = \perp$; P and V output keys s_P and s_V established with each other and their internal states st_P and st_V. The following guarantees must hold:

- **(Completeness)** For all valid AMP credentials σ and messages m, P and V output a shared secret $s_P = s_V$ in the $\mathsf{AnonPAO}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, V\langle \mathsf{sk}_V\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol.
- **(Soundness)** For all PPT adversaries $\mathcal{A}$, security parameter n, keys $(\mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_V, \mathsf{sk}_V)$, valid credentials σ, and messages m, the following conditions must hold for the $\mathsf{AnonPAO}^{\mathcal{A}}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, V\langle \mathsf{sk}_V\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol, where $\mathcal{A}$ controls either the communication between P and V or between V and S (but not both) and obtains S's output m':
  (1) *(Verifier security)* If V outputs $s_V \neq \perp$, then the probability that $\mathcal{A}$ outputs s_V is at most negligibly more than the maximum probability that S outputs a non-$\perp$ message in the $\mathsf{AMP}(1^n, \mathsf{pk}_S, \mathcal{A}\langle\cdot\rangle, V\langle\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol.
  (2) *(Prover security)* If P outputs $s_P \neq \perp$, then the probability that $\mathcal{A}$ outputs s_P must be negligible.
- **(Transcript privacy)** For all PPT adversaries $\mathcal{A}$, security parameter n, keys $(\mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_V)$, valid credentials σ, and messages m, $\mathcal{A}$'s view in an $\mathsf{AnonPAO}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, \mathcal{A}\langle\cdot\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol is simulatable by a PPT simulator $\mathcal{S}$ in the $\mathsf{AMP}^{\mathsf{SC}}(1^n, \mathsf{pk}_S, P\langle\sigma, m\rangle, \mathcal{S}^{\mathcal{A}}\langle\cdot\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol.
- **(Transcript integrity)** For all PPT adversaries $\mathcal{A}$, security parameter n, keys $(\mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_V)$, valid credentials σ, and messages m, the probability that S outputs $m' \notin \Delta(m)$ in the $\mathsf{AnonPAO}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, \mathcal{A}\langle\cdot\rangle, S\langle \mathsf{sk}_S\rangle)$ protocol must be negligible.
- **(Server obliviousness - optional)** For all PPT adversaries $\mathcal{A}$, security parameter n, keys $(\mathsf{pk}_S, \mathsf{sk}_S, \mathsf{pk}_V, \mathsf{sk}_V)$, valid credentials σ, and messages m, $\mathcal{A}$'s view in an $\mathsf{AnonPAO}(1^n, \mathsf{pk}_S, \mathsf{pk}_V, P\langle\sigma, m\rangle, V\langle \mathsf{sk}_V\rangle, \mathcal{A}\langle\cdot\rangle)$ protocol must be indistinguishable from some $\mathsf{AMP}^{\mathsf{SC}}(1^n, \mathsf{pk}_S, P\langle\sigma, m'\rangle, V\langle\rangle, \mathcal{A}\langle\cdot\rangle)$ protocol s.t. $m' \in \Delta(m)$.

Our security properties extend the security properties proposed by Wang et al. [37] for SCI-based anonymous PAO. In particular, our soundness requirement is more general than the *injection secrecy* property of [37] that only requires P to be unaware of the challenge injected by V. We consider a more powerful network adversary than SCI's that not only corrupts P but can also intercept, inject, and drop messages. This adversary may or may not hold a valid account credential at S but aims to either impersonate P or piggyback on P's session to establish a key-pair with V. Further, we consider a more general type of modification rather than just injecting a challenge.

We restrict the adversary from controlling both P–V and V–S links. This restriction is necessary: an adversary controlling both links can drop honest P's connection, initiate its own secure channel with S (using its own keys), directly observe V's modifications in the V–S traffic, decrypt them, and send the correct challenge back to V—all without possessing valid S credentials or retrieving anything from S's storage. Since this attack is inherent to any modification-based anonymous PAO, we explicitly exclude such adversaries.

The server obliviousness property prevents S from distinguishing between an anonymous PAO execution and a standard AMP execution, where the message posted to S lies within the neighborhood $\Delta(m)$ of the message m sent by P. It also prevents S from learning V's identity, thus preventing it from profiling P.

We let P and V output their internal states st_P and st_V at the end of the protocol, so that they can be used in building larger protocols using anonymous PAOs, such as blind CAs [37].

4 YouChoose for email-over-TLS

We now present the YouChoose protocol for the specific and ubiquitous setting of email-over-TLS, where SMTP-based email acts as the AMP and TLS acts as the SC. In Section 5, we extract the core assumptions behind the YouChoose approach for other general settings and analyse its security under these assumptions.

4.1 Background

4.1.1 Transport Layer Security (TLS). TLS is the Internet's standard secure channel, providing confidentiality, integrity, and endpoint authentication. A handshake (TLS.Handshake) authenticates the server and derives the shared session key (e.g., via ECDHE [32]). Application data is split into TLS records of at most 2^{14} bytes, which are encrypted and then reassembled on receipt.

Encryption/decryption of each TLS record proceeds via stateful authenticated encryption algorithms TLS.AEnc/TLS.ADec, where the state is a monotonically increasing sequence number. Each party increments the sequence number after sending/receiving a record to prevent reordering, replay, and truncation of TLS records.

TLS offers various ciphersuites for TLS.AEnc/TLS.ADec. We consider two representative ones: *a*) **Ciphersuite I** employing AES CBC mode for encryption and HMAC-SHA256 for message authentication [19] (commonly used in TLS 1.2); and *b*) **Ciphersuite II** utilizing AES counter mode for encryption and GMAC for message authentication [30] (used in TLS 1.2/1.3).

The two ciphersuites use the sequence number differently. In Ciphersuite I, it is used to compute the HMAC authentication tag, while in Ciphersuite II, it is used to derive the nonce for counter-mode encryption. This distinction is important for YouChoose, as reusing the same sequence number to encrypt different messages is allowed in Ciphersuite I but violates nonce-uniqueness requirements in Ciphersuite II, thus compromising confidentiality [14].

4.1.2 Email and SMTP. Email involves three main components: Mail User Agents (MUAs), Mail Submission Agents (MSAs), and Mail Transfer Agents (MTAs), coordinated by the Simple Mail Transfer Protocol (SMTP). A message originates in the sender's MUA, is submitted to the sender's MSA (*message submission*), and relayed through MTAs until it reaches the recipient's MSA (*message transmission*), where it is stored for later retrieval. Common MUAs include Gmail, Outlook, Apple Mail, and Thunderbird.

For our purposes, the relevant stage is *message submission*, where an MUA posts an email to an MSA using SMTP. An SMTP client (in the MUA) opens a TCP connection to an SMTP server (in the MSA) and sends a greeting commmand `EHLO` (an extended HELLO). Next, the TLS handshake is initiated using the `STARTTLS` extension [11, 20]. Once the TLS secure channel is established, the sender authenticates itself using the `AUTH` command [31], indicates sender and recipient addresses using `MAIL` and `RCPT` commands, and finally indicates its readiness to transmit the email headers and body using the `DATA` command. To encode email content including text and images, a standard MIME encoding is applied to raw data.

$\underline{\text{YouChoose}(1^n, \mathsf{pk}_V, \mathsf{pk}_S, P\langle\sigma, m\rangle, V\langle \mathsf{sk}_V\rangle, S\langle \mathsf{sk}_S\rangle)}$:

$P\langle k_{PS}, \mathsf{st}^0_{PS}\rangle, V\langle \mathsf{st}_V\rangle, S\langle k_{SP}, \mathsf{st}^0_{SP}\rangle \leftarrow$ SMTP.Setup$(1^n, \mathsf{pk}_S, P\langle\sigma, |m|\rangle, V\langle\rangle, S\langle \mathsf{sk}_S\rangle)$ // Setup

P: **for** $s \in \{+,-\}$: $m^s \leftarrow \delta_s(m)$ // Msg Gen
  let $k := \lceil |m|/B \rceil$; $\epsilon \xleftarrow{\$} \{+,-\}^k$
  $m^0 := (m^{\epsilon_i}[i])_{i\in[k]}$; $e^0 := \mathcal{E}(m^0)$
  $m^1 := (m^{\neg\epsilon_i}[i])_{i\in[k]}$; $e^1 := \mathcal{E}(m^1)$
  choose $\mathcal{I} = \{i_0, \ldots, i_{n-1}\}$ s.t. $e^0[i_\ell] \neq e^1[i_\ell]$

  let $\kappa := \lceil |(e^0| = |e^1)|/B' \rceil$; // Msg Encryption
  for $i \in [\kappa]$:
    for $b \in \{0,1\}$: $c^b[i] \leftarrow \mathsf{TLS.AEnc}(k_{PS}, e^b[i], \mathsf{st}^i_{PS})$
    $\mathsf{st}^{i+1}_{PS} \leftarrow \mathsf{TLS.NextState}(\mathsf{st}^i_{PS})$
$P \to V$: $\mathcal{I}, \{(c^0[i], c^1[i])_{i\in[\kappa]}$

V: $\vec{d} \xleftarrow{\$} \{0,1\}^n$ // Challenge Gen + Msg Decryption
$V \to S$: $c := \left(c^{d_\ell}[i] \textbf{ if } i = i_\ell \in \mathcal{I}, \ell \in [n] \textbf{ else } c^0[i]\right)_{i\in[\kappa]}$
$S\langle m'/\perp\rangle \leftarrow \mathsf{SMTP.Finish}(1^n, P\langle k_{PS}, \mathsf{st}^\kappa_{PS}\rangle, V\langle\rangle, S\langle k_{SP}, \mathsf{st}^\kappa_{SP}, c\rangle)$
$S \to P$: m' // out-of-band

// Challenge Verification
$P\langle k_{PV}, \mathsf{st}^0_{PV}\rangle, V\langle k_{PV}, \mathsf{st}^0_{PV}\rangle \leftarrow \mathsf{TLS.Handshake}(\mathsf{pk}_V, P, V)$
P: **if** $m' = \perp$: abort **else:** $e' \leftarrow \mathcal{E}(m')$
  for $\ell \in [n]$: $d'_\ell := 0$ if $e'[i_\ell] = e^0[i_\ell]$; else 1
$P \to V$: $c_h \leftarrow \mathsf{TLS.AEnc}(k_{PV}, \mathsf{H}(d' \parallel \mathsf{pk}_V), \mathsf{st}^0_{PV})$
V: $h' \leftarrow \mathsf{TLS.ADec}(k_{PV}, c_h, \mathsf{st}^0_{PV})$
  $s_V := k_{PV}$ if $h' = \mathsf{H}(d \parallel \mathsf{pk}_V)$ else $\perp$
P: $s_P := k_{PV}$

Output: $P\langle s_P\rangle, V\langle s_V\rangle, S\langle m'/\perp\rangle$

Figure 2: YouChoose for email-over-TLS (Ciphersuite I).

4.2 Protocol

Figure 2 illustrates the YouChoose protocol for email-over-TLS. Note that this scheme is secure only for Ciphersuite I. A minor tweak is required to support Ciphersuite II (see Section 4.3.1). We assume a network topology where P does not observe traffic between V and S and S does not observe traffic between P and V, and that S cannot distinguish between a connection from P and a connection from V. We also assume that before initiating the protocol, V has verified the authenticity of S, say, by verifying its TLS certificate. The protocol proceeds in five main phases:

Setup. In this phase, P and S follow the setup part of the standard SMTP protocol, denoted SMTP.Setup, where P initiates an SMTP connection with S through V by sending `EHLO` and requests the TLS handshake via `STARTTLS`. A TLS session key k_{PS} is established between P and S with V acting as a transparent proxy. Over this secure channel, P transmits its credentials using the `AUTH` SMTP command, followed by the `MAIL`, `RCPT` and `DATA` commands.

Message Generation. P chooses a bitmap image m as the message to be sent with the email. A bitmap image is an array of bytes, where each byte represents a single pixel's R/G/B value. Let $\mathcal{E}$ denote the MIME encoding function that adds necessary email headers and base64-encodes the raw image data to produce SMTP-compatible email content. This encoded data is split into fixed-size TLS records, which are individually encrypted and transmitted.

We define a message block size B such that the encoding of a B-byte message block exactly matches the TLS record size. This

sizing enables a one-to-one mapping between message blocks and their corresponding TLS records. Let $m[i]$ denote the i^{th} message block and $e[i] := \mathcal{E}(m[i])$ denote the encoded message block. P creates two variants m^0 and m^1 of the image m where for each block $m[i]$, the pixel intensities are modified by a small fixed amount: $m^0[i]$ randomly increases or decreases the pixel intensity of each pixel in $m[i]$ and $m^1[i]$ mirrors $m^0[i]$ in the opposite direction (represented by functions δ_+ and δ_-). These modifications produce visually indistinguishable images.

Message Encryption. P creates $e^0 := \mathcal{E}(m^0)$ and $e^1 := \mathcal{E}(m^1)$ and encrypts them through TLS.AEnc to yield sequences c^0 and c^1 of ciphertext records respectively. Importantly, P reuses the *same* TLS state st^i_{PS} (same sequence number) when encrypting both variants. This permits V to later select either ciphertext while maintaining the strictly incrementing sequence numbers expected by S.

Since TLS encrypts each record independently, S can decrypt any individual record regardless of which variant it came from (as long as the sequence number is valid)[1]. This enables V to interleave records from c^0 and c^1 while ensuring valid decryption. P selects n challenge indices $\mathcal{I} = \{i_0, \ldots, i_{n-1}\}$ where $e^0[i] \neq e^1[i]$, representing positions where interleaving is allowed (excluding, e.g., header records). It sends all ciphertext pairs $\{(c^0[i], c^1[i])_{i\in[\kappa]}\}$ along with the set $\mathcal{I}$ to V, who will interleave only at indices in $\mathcal{I}$.

Challenge Generation and Message Decryption. V randomly samples a challenge bit vector $\vec{d} \in \{0,1\}^n$, forwards either record $c^0[i]$ or $c^1[i]$ for each $i \in \mathcal{I}$ to S based on the challenge bits (and $c^0[i]$ for each $i \notin \mathcal{I}$). S then engages in standard SMTP message decryption and processing, denoted SMTP.Finish, where it decrypts the received ciphertext records using TLS.ADec, decodes the resulting MIME content, verifies P's credentials, and outputs either the deceryption m' (accepted) or $\perp$ (rejected). P retrieves m' from S in an out-of-band way by, e.g., accessing their sent folder.

Challenge Verification. P establishes a separate TLS session with V by performing a fresh handshake, deriving a shared session key k_{PV}. This step allows P to verify V's identity via V's TLS certificate. P re-encodes the retrieved message, reconstructs the challenge vector by checking the sign of the noise, and sends a hash of the challenge to V via the TLS channel. V accepts if the hash corresponds to the correct challenge. The key k_{PV} acts as the shared key for future anonymous communication between P and V.

4.3 Tweaks and Optimizations

4.3.1 Tweak to support Ciphersuite II. Recall from Section 4.1.1 that in Ciphersuite II, reusing the same sequence number for multiple ciphertexts violates the ciphersuite's nonce-uniqueness requirements and compromises confidentiality. To adapt YouChoose for such ciphersuites, we use a lightweight *oblivious transfer (OT)* mechanism, where P holds two ciphertexts for each challenge position, while V obtains one of the bits corresponding to its challenge bits. The OT mechanism ensures that P cannot infer V's challenge bits (preserving soundness) and that V can learn only one valid ciphertext per sequence number (preserving confidentiality).

[1]Note: AES CBC-chaining happens within a TLS record encryption, not across records.

Standard OT protocols [4] are efficient enough that this modification introduces negligible overhead. Even for typical instantiations of YouChoose requiring approximately 80 OTs to achieve 80-bit security, the total additional cost remains minimal (see Section 6).

4.3.2 Optimizing Communication Overhead. A standard TLS record size is 16 KB. Since the challenge space in YouChoose is due to V's selective forwarding of TLS records, achieving an 80-bit challenge space would required P to send 160 TLS records to V, so that V could forward 80 records to S. This implies a minimum message size of 160×16 KB $\approx$ 2.5 MB, with ~1.25 MB transmitted by V to S.

Instead, we exploit the dynamic record sizing behavior of modern TLS implementations. Due to TCP slow start, common TLS libraries (e.g., Go's `crypto/tls` [33, 34]) use smaller records, typically around 1500 B, for the first 1 MB of data [10]. Using these smaller records, a 160-record exchange requires only ≈234 KB, comparable to a small image or PDF attachment in everyday email.

Further reduction is also theoretically possible by forcing even smaller record sizes, though this might be considered non-standard behavior and signal to S that an anonymous PAO is in progress.

4.3.3 Steganographic strength. Our server obliviousness guarantees depend upon S's ability to detect the presence of an embedded message in the decrypted image. As per recent studies in image steganalysis [12, 16, 26], detection is effective only when the embedded message density exceeds 0.1 bits per pixel (bpp). We are encoding ≈ 80 bits across a typical 200KB image (≈ 200,000 pixels), which yields approximately 0.0004 bpp—roughly 250 times below the detectable threshold. Further, P can select images with high inherent noise or texture, add challenges near object boundaries, or randomize the magnitude and direction of intensity changes, all of which significantly hamper S's detection ability.

5 Generalising to other AMPs, δ and encodings

Now we present how the YouChoose approach extends to other AMPs and SCs and the assumptions lying herein.

Assumptions on the network. We summarise our network assumptions in Table 1. Broadly, the assumptions affect either soundness or server obliviousness. Without **N1a** and **N2a**, P could trivially break soundness by either observing V's modifications or impersonating S. This requires an unrealistically powerful prover capable of network interception or IP/DNS spoofing. Without **N1b**, S can trivially detect the anonymous PAO protocol by observing the P–V traffic; this assumption holds when S does not have network-level visibility into the path between P and V (e.g., S is not an ISP or state-level adversary conducting surveillance). **N2b** prevents S from fooling P into sharing sensitive information by impersonating V. Finally, **N3** prevents S from detecting anonymous PAO activity by inspecting the IP address of P or V. It also prevents S from identifying V, thus preventing profiling of P in the OpenID use-case. For whistleblowing use-cases (where server obliviousness is most critical), these assumptions are standard and can be achieved through common privacy tools: **N2b** is ensured by TLS authentication, while **N3** can be achieved using VPNs.

Assumptions on the AMP protocol. We require the AMP protocol to work on top of a stateful secure channel. It should consist of

Table 1: Network assumptions and security implications

Assumption	Soundness	Server Obliviousness
N1: Traffic visibility		
N1a: P cannot view V–S traffic	×	
N1b: S cannot view P–V traffic		×
N2: No spoofing		
N2a: P cannot spoof as S	×	
N2b: S cannot spoof as V		×
N3: S cannot distinguish P/V's IP		×

a Setup sub-protocol for establishing the secure channel between the account owner and the account provider, a block-wise encode-and-encrypt mechanism for message submission, and a Finish sub-protocol for decrypting and decoding the received ciphertexts and checking the authencity of the account owner. It should also enable account owners to retrieve the posted messages. We call such AMPs as *standard*. Note that real-world protocols of our concern, like HTTP authenticated POST APIs, are standard.

Assumptions on neighborhood function Δ. Note that the YouChoose approach fundamentally requires that given any two "noisy" versions of an original message created using functions δ_+, δ_-, an arbitrary interleaving of them should not be distinguishable from the original message by S or any external observer. We capture this requirement by considering the (δ_+, δ_-)*-neighborhood*, defined as $\Delta(m) := \{(\delta_{b_i}(m[i]))_{i\in[B]} \mid b \in \{+,-\}^B\}$ where B is the message block size, as the neighborhood function. Note that a (δ_+, δ_-)-neighborhood is closed under interleaving in the following sense.

Definition 5.1. (Closure under interleaving) Let $m \in \{0,1\}^*$ and let $m^0, m^1 \in \Delta(m)$ be two neighbors of m, where $m^0 = (m^0[1], \ldots, m^0[\ell])$ and $m^1 = (m^1[1], \ldots, m^1[\ell])$ have equal number of blocks ℓ. We say that Δ satisfies *closure under interleaving* if for every choice of a bitstring $d \in \{0,1\}^\ell$, the interleaved message

$$m^d[j] := \begin{cases} m^0[j] & \text{if } d_j = 0, \\ m^1[j] & \text{if } d_j = 1 \end{cases} \qquad \text{for all } j \in [\ell]$$

also belongs to $\Delta(m)$.

Assumptions on the encoding scheme. Finally, modifications introduced by V on the encoded message must be such that the resulting message remains valid and decodable by S. This creates well-formedness requirements for the encoding scheme (Definition 5.2) so that V's modifications do not break message integrity.

Definition 5.2. (Well-formed encoding) An encoding scheme $(\mathcal{E}, \mathcal{D})$, where $\mathcal{E} : \{0,1\}^* \to \{0,1\}^*$ and $\mathcal{D} : \{0,1\}^* \to \{0,1\}^*$, is called *well-formed* if:

- **(Losslessness)** For all $m \in \{0,1\}^*$, $\mathcal{D}(\mathcal{E}(m)) = m$.
- **(Uniformity)** For all $m^0, m^1 \in \{0,1\}^*$, $|m^0| = |m^1|(= k$, say) and $|\mathcal{E}(m^0)| = |\mathcal{E}(m^1)|$ $(= \kappa$, say).
- **(Separability)** For all $m \in \{0,1\}^*$, let k, κ be as defined above. For each $i \in [\kappa]$, there exists an index set $J_i \subseteq [k]$ and a function $f_i : \{0,1\}^{|J_i|} \to \{0,1\}^*$ such that *a*) $(J_i)_{i\in[\kappa]}$ are disjoint to each other, and *b*) for all $m' \in \{0,1\}^*$, $e' := \mathcal{E}(m')$ satisfies $e'[i] = f_i((m'[j])_{j\in J_i})$.

Losslessness ensures that no information is lost during encoding, which is essential for P to detect and analyze any modifications made to their message. *Uniformity* guarantees that messages of equal length always produce encoded outputs of equal length (required for server-obliviousness). *Separability* ensures that each encoded block encodes disjoint portions of the message, ensuring that an interleaving of encoded blocks is valid and decodable.

These requirements, particularly separability, unfortunately preclude formats like PNG that use global compression or checksums, as interleaving would break decompression or integrity checks. However, since JPEG uses independent blocks, our approach can be adapted to JPEG by modifying the JPEG DCT coefficients at the block level (see Appendix C and our implementation [36] for details).

Abstracting OT. In our generalised YouChoose protocol, we let V obtain only one of the two ciphertexts via the OT mechanism of Section 4.3.1. We abstractly model this mechanism using an ideal functionality F_{OT} (see Appendix A).

5.1 Security Analysis

Theorem 5.3. *Given a standard AMP, a well-formed encoding scheme* $(\mathcal{E}, \mathcal{D})$*, and the* (δ_+, δ_-)*-neighborhood* $\Delta(m)$*, YouChoose is a* Δ*-anonymous PAO under* F_{OT} *and network assumptions* ***N1****-****N3****.*

We defer the full proof to Appendix B. Below, we provide the main intuitions:

Completeness. SC ensures that S recovers the correct blocks from ciphertexts, while separability of the encoding guarantees that these blocks assemble into a coherent message. Losslessness then ensures that decoding yields the precise interleaving chosen by V. Consecutive sequence numbers in the interleaved records ensures that decryption at S passes. Thus, P can reliably recover V's challenge and establish a shared secret key.

Soundness. By OT, P never learns V's challenge bits and thus cannot forge the challenge. Moreover, since the interleaved message is identically distributed to P's message variants, any outside observer observing just the posted message cannot learn V's choices.

Transcript Privacy. By the confidentiality of SC, all that V ever sees are ciphertexts, and thus does not learn anything more than a standard execution of AMP.

Transcript Integrity. By the stateful integrity of SC, any reordering, duplication, or injection of blocks causes decryption failure. Thus, the only transcripts that pass verification correspond to valid block-wise interleavings of P's two variants. By the separability of the encoding and the closure of Δ under interleaving, such transcripts necessarily lie in $\Delta(m)$.

Server Obliviousness. S receives a lawful stateful encryption of some message $m^* \in \Delta(m)$. A simulator can reproduce this exact view simply by choosing an arbitrary neighbor $m^* \in \Delta(m)$ and encrypting it honestly. Since S cannot distinguish between messages in $\Delta(m)$ by assumption, it remains oblivious to whether an anonymous PAO protocol or a standard AMP was executed.

6 Implementation and Evaluation

To evaluate the feasibility and efficiency of YouChoose, we implemented prototypes for both email-over-TLS and other AMPs. We

Table 2: Time taken by Baseline, SCI and YouChoose to send an email. L1, L2, and L3 denote three different locations. A dash (-) indicates not applicable and 0 indicates negligible time.

Ciphersuite	P, V, S	Baseline(s) Size: 130 B	SCI (s) Size: 192 B	YouChoose over email (s) Size: 237 KB	YouChoose over HTTP POST (s) Size: 168 B
AES-CBC/ HMAC-SHA256	L1, L2, L3	0.37 (±0.03)	3.1 (±0.04)	0.727 (±0.033)	0.69 (±0.06)
	L1, L2, L2	0.1 (-)	2.21 (±0.17)	0.20 (±0.01)	0.14 (-)
	L1, L1, L1	0 (-)	0.37 (±0.01)	0.078 (±0.003)	0.05 (-)
AES-GCM	L1, L2, L3	0.33 (±0.03)	-	0.721 (±0.028)	0.56 (±0.03)
	L1, L2, L2	0.1 (-)	-	0.199 (±0.004)	0.19 (±0.01)
	L1, L1, L1	0 (-)	-	0.103 (±0.003)	0 (-)

provide our implementation at [36]. All experiments were run on Linode [18] nanode instances (1 core, 1 GB RAM).

6.1 YouChoose for Email-over-TLS

In this prototype, we let P use a modified version of tlslite − ng [15], implementing the sequence number reuse mechanism that allows V to selectively forward TLS records (see Section 4). We implemented support for both Ciphersuites I and II. For Ciphersuite II, we used the OTC library [22] (based on [4]) to perform 80 OTs.

We used smtplib [25] to send emails as per the SMTP specification [31]. We used TLS records of size 1500 B commensurate with some standard TLS libraries [33, 34] (see Section 4.3.2).

We evaluated our prototype under three network configurations: (1) L1, L2, L3 places P, V, and S on separate servers, mimicking realistic deployments where S runs on commercial services like Outlook or Gmail; (2) L1, L2, L2 colocates V and S to study the effect of reduced latency between them; (3) L1, L1, L1 runs all parties locally, isolating the intrinsic computation and communication costs without wide-area latency.

We compare YouChoose against a baseline text email and against SCI [37], the state-of-the-art anonymous PAO protocol. SCI requires a distinct MPC-based design for each TLS ciphersuite. Since we lacked access to the authors' code, we reimplemented SCI for Ciphersuite I. Here, we also incorporated recent advances in maliciously secure two-party computation [38], which substantially improves performance over earlier protocols [27].

SCI for Ciphersuite II requires an Oblivious Polynomial Evaluation (OPE) protocol, which has since been shown insecure [9]. We therefore currently do not implement SCI with Ciphersuite II.

Table 2 reports the median runtime and standard deviation over five trials for both ciphersuites. Message sizes vary by protocol: baseline emails use a minimal body of 130 B; SCI adds a 32 B challenge to reach 192 B; YouChoose uses 237 KB for the interleaved variants, though only half of this is transmitted to S by V.

Our results show that for Ciphersuite I, YouChoose substantially outperforms SCI: it is at least 4× faster when sending to Outlook and over 10× faster with a local SMTP server. Compared to baseline email, YouChoose incurs only modest overhead—0.1s on a local server and 0.3s when sending to Outlook. Similar trends hold under Ciphersuite II, despite the additional 80 OTs. The dominant cost in YouChoose is communication, which scales with email body size, whereas the computational overhead is negligible.

Note that larger message sizes are not unique to YouChoose but seem inherent to achieving server obliviousness in practice. Although SCI challenges are only 32 B, these are random strings and must be embedded in large, typically image or media, files to avoid detection by S, leading to similar kilobyte-scale messages [37].

6.2 YouChoose over Authenticated HTTP APIs

To demonstrate YouChoose's generality beyond email, we also evaluated it on authenticated HTTP APIs, specifically Twitter's tweet posting API. The client request (e.g., a JSON payload) contains the message to be posted and is transmitted over TLS following the same encode-and-encrypt pattern, and thus naturally satisfies our requirements on the AMP structure and encoding.

We used httplib [24] to issue authenticated POST requests to publish tweets. To accommodate Twitter's character limits [35], we transmitted tweet content in TLS records of 1 byte each. While this unusual record fragmentation violates server obliviousness (Twitter could detect non-standard TLS behavior), we argue this trade-off is acceptable for credential-less authentication scenarios where the server doesn't need to remain unaware of the anonymous PAO protocol. The resulting protocol performs even better than YouChoose over email (see the last column of Table 2).

7 Conclusion and Future Work

We formalized the security requirements for anonymous PAO and proposed YouChoose, a lightweight and general approach to meet these requirements without resorting to specially designed MPC protocols for different proofs of account ownership. We demonstrated our approach for email account ownership using SMTP over TLS, making it the first anonymous PAO that works with all ciphersuites of TLS 1.2 and 1.3. The overhead is not more than sending a typical email containing a small image file as an attachment.

While we provide preliminary demonstration of anonymous PAOs using a general AMP like the Twitter tweet API, transitioning to a full-fledged anonymous OpenID framework via this approach presents several further challenges. First, our interleaving mechanism and encoding well-formedness requirements preclude common formats like PNG that include a global operation. Second, although YouChoose enables users to anonymously authenticate themselves to service providers, anonymous OpenID systems must also let users maintain a persistent identity with the service providers without forcing users to remember a different identity for each provider (this was one of the core design goals of OpenID). Anonymously supporting other OpenID features like recovery of forgotten passwords also presents interesting challenges. We leave these questions for future work.

References

[1] Jan Camenisch and Els Van Herreweghen. 2002. Design and Implementation of the Idemix Anonymous Credential System. In *CCS*. 21–30.
[2] David Chaum. 2003. Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms. Advances in Information Security, Vol. 7. 211–219.
[3] David Chaum and Eugène van Heyst. 1991. Group Signatures. In *EUROCRYPT*. 257–265.
[4] Tung Chou and Claudio Orlandi. 2015. The Simplest Protocol for Oblivious Transfer. In *LATINCRYPT*. 40–58.
[5] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2017. A Formal Security Analysis of the Signal Messaging Protocol. In *EuroS&P*. 451–466.
[6] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. 2016. Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Computation. In *IEEE S&P*. 235–254.
[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *Usenix Security*.
[8] James Dolan Freedom of the Press Foundation, Aaron Swartz. [n. d.]. *SecureDrop*. https://securedrop.org/
[9] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. 2017. Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead. In *Cryptology ePrint Archive*.
[10] John Graham-Cumming. 2016. Optimizing TLS over TCP to Reduce Latency. https://blog.cloudflare.com/optimizing-tls-over-tcp-to-reduce-latency/. Accessed: 2024-06-26.
[11] Paul E. Hoffman. 2002. SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207.
[12] E. Hong, K. Lim, T. W. Oh, and H. Jang. 2023. Lightweight image steganalysis with block-wise pruning. *Scientific Reports* 13, 1 (Sept. 2023), 16148. doi:10.1038/s41598-023-43386-2
[13] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2017. Authenticated Confidential Channel Establishment and the Security of TLS-DHE. *J. Cryptology* 30 (2017), 1276–1324. doi:10.1007/s00145-016-9248-2
[14] Antoine Joux. 2012. Authentication Failures in NIST Version of GCM.
[15] Hubert Kario. [n. d.]. tlslite-ng. https://github.com/tlsfuzzer/tlslite-ng.
[16] Meike Helena Kombrink, Zeno Jean Marius Hubert Geradts, and Marcel Worring. 2024. Image Steganography Approaches and Their Detection Strategies: A Survey. *Comput. Surveys* 57, 2, Article 33 (Oct. 2024). doi:10.1145/3694965
[17] Shuai Li and Nicholas Hopper. 2016. Mailet: Instant Social Networking under Censorship. In *PETS*.
[18] LLC Linode. 2023. Linode - Cloud Hosting & Linux Servers. https://www.linode.com/
[19] David McGrew, John Foley, and Kenny Paterson. 2014. *Authenticated Encryption with AES-CBC and HMAC-SHA*. Technical Report. Internet Engineering Task Force.
[20] Keith Moore and Chris Newman. 2018. Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access. RFC 8314.
[21] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. 2015. Multi-Context TLS (McTLS): Enabling Secure In-Network Functionality in TLS. *SIGCOMM Comput. Commun. Rev.* 45, 4 (2015), 199–212.
[22] nthparty. [n. d.]. *OTC*. https://github.com/nthparty/otc
[23] Christian Paquin and Greg Zaverucha. 2013. U-Prove Cryptographic Specification V1.1 (Revision 3).
[24] Python Software Foundation. 2023. http.client — HTTP protocol client. https://docs.python.org/3/library/http.client.html
[25] Python Software Foundation. 2023. smtplib — SMTP protocol client. https://docs.python.org/3/library/smtplib.html
[26] T.-S. Reinel et al. 2021. GBRAS-Net: A Convolutional Neural Network Architecture for Spatial Image Steganalysis. *IEEE Access* 9 (2021), 14340–14350. doi:10.1109/ACCESS.2021.3052494
[27] Peter Rindal and Mike Rosulek. 2016. Faster Malicious 2-Party Secure Computation with Online/Offline Dual Execution. In *Usenix Security*. 297–314.
[28] Ronald L. Rivest, Adi Shamir, and Yael Tauman. 2001. How to Leak a Secret. In *ASIACRYPT*. 552–565.
[29] Nat Sakimura, John Bradley, Michael Jones, Breno de Medeiros, and Mortimore Chuck. 2014. OpenID Connect Core 1.0. (2014). http://openid.net/specs/openid-connect-core-1_0.html
[30] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. 2008. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288.
[31] Rob Siemborski and Alexey Melnikov. 2007. SMTP Service Extension for Authentication. RFC 4954.
[32] Douglas Stebila and Jonathan Green. 2009. Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer. RFC 5656.
[33] The Go Authors. [n. d.]. go. https://github.com/golang/go Accessed: 2024-07-11.
[34] tombergan. 2016. crypto/tls: TLS should dynamically tune record size for better HTTP performance 14376 . https://github.com/golang/go/issues/14376. Accessed: 2024-06-26.
[35] Twitter Developer. 2023. *Counting Characters*. https://developer.twitter.com/en/docs/counting-characters
[36] Aarav Varshney, Prashant Agrawal, and Mahabir Prasad Jhanwar. 2025. YouChoose source-code and benchmarks. https://github.com/aarav22/anon-pao.
[37] Liang Wang, Gilad Asharov, Rafael Pass, Thomas Ristenpart, and Abhi Shelat. 2019. Blind Certificate Authorities. In *IEEE S&P*.
[38] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *Cryptology ePrint Archive*.

A Ideal functionality F_{OT} for Oblivious Transfer

The OT functionality F_{OT} in Definition A.1 abstracts oblivious transfer of ciphertexts from P to V in the YouChoose protocol (Section 4.3.1). F_{OT} is not strictly required for Ciphersuite I as we can let P send both ciphertexts to V directly without violating confidentiality. However, it is required to maintain confidentiality in Ciphersuite II and potentially other ciphersuites as well. We use the OT protocol of [4] to realize F_{OT}.

Definition A.1. (Idealized OT Functionality F_{OT}) F_{OT} models a two-party oblivious transfer between a sender P_1 and a receiver P_2: $\mathsf{F}_{\mathsf{OT}}(P_1\langle c^0, c^1\rangle, P_2\langle d\rangle)$ is defined as follows:

(1) P_1 inputs two messages (c^0, c^1); P_2 inputs a choice bit $d \in 0, 1$.
(2) F_{OT} sends c^d to P_2 and reveals nothing to P_1 about d.

B Proof for Theorem 5.3

Proof. Let Π_{YC} denote the YouChoose protocol of Figure 2 but instantiated for the given AMP and SC.

Completeness. First, by the uniformity property of the encoding $\mathcal{E}$ (Definition 5.2), $|e^0| = |e^1|$ holds.

By the correctness of F_{OT}, it holds that

$$c^i = \begin{cases} c^{d_\ell}[i_\ell] \leftarrow \mathsf{SC.AEnc}(k_{PS}, e^{d_\ell}[i_\ell], \mathsf{st}_{PS}^{i_\ell}) & \text{if } \exists \ell \in [n] : i = i_\ell \\ c^0[i] \leftarrow \mathsf{SC.AEnc}(k_{PS}, e^0[i], \mathsf{st}_{PS}^{i}) & \text{otherwise} \end{cases}$$

Note that the state used for constructing each c^i is sequentially increasing. Thus, by the correctness of SC.ADec, it holds that

$$e^i = \begin{cases} e^{d_\ell}[i_\ell] & \text{if } \exists \ell \in [n] : i = i_\ell \\ e^0[i] & \text{otherwise} \end{cases}$$

Note first that since $m^0, m^1 \in \Delta(m)$, by the separability of encoding $\mathcal{E}$, the following holds:

$$e^i = \begin{cases} f_{i_\ell}((m_j^{d_\ell})_{j \in J_{i_\ell}}) & \text{if } \exists \ell \in [n] : i = i_\ell \\ f_i((m_j^0)_{j \in J_i}) & \text{otherwise} \end{cases} \tag{1}$$

where all J_i's are disjoint to each other.

To show that the decrypted message corresponds to a valid interleaving, we construct the following message m^*.

$$m_j^* := \begin{cases} m_j^{d_\ell} & \text{if } \exists \ell \in [n] : j \in J_{i_\ell} \\ m_j^0 & \text{otherwise} \end{cases}$$

Note that by the closure property of Δ under interleaving, $m^* \in \Delta(m)$. Thus, by the uniformity property, $|\mathcal{E}(m^*)| = \kappa$, and by the separability property, we have:

$$(\mathcal{E}(m^*))_i = \begin{cases} f_i((m_j^{d_\ell})_{j \in J_{i_\ell}}) & \text{if } \exists \ell \in [n] : i = i_\ell \\ f_i((m_j^0)_{j \in J_i}) & \text{otherwise} \end{cases} \tag{2}$$

Simulator $\mathcal{S}^{\mathcal{A}}$

$\mathcal{S}$ forwards all messages obtained in AMP.Setup to $\mathcal{A}$
$P \to \mathcal{S}$: c // encryption of a message $m \in \{0,1\}^*$
$\mathcal{S} \to \mathcal{A}$: $\mathcal{I}' := (i'_0, \ldots, i'_{n-1}) \leftarrow_\$ [\kappa]$ and c
$\mathcal{A} \to \mathcal{S}$: d
forward all messages obtained in AMP.Finish to $\mathcal{A}$
$\mathcal{S}\langle k_{PV}, \mathrm{st}^0_{PV}\rangle, \mathcal{A}\langle\cdot\rangle \leftarrow \mathsf{SC.Handshake}(\mathsf{pk}_{\mathcal{A}}, \mathcal{S}\langle\rangle, \mathcal{A}\langle\cdot\rangle)$
$\mathcal{S} \to \mathcal{A}$: $c_h \leftarrow \mathsf{SC.AEnc}(k_{PV}, \mathsf{H}(d \parallel \mathsf{pk}_{\mathcal{A}}), \mathrm{st}^0_{PV})$

Figure 3: Description of the simulator $\mathcal{S}^{\mathcal{A}}$.

By Equations 1 and 2, $e[i] = (\mathcal{E}(m^*))_i$ for each $i \in [\kappa]$, i.e., $e = \mathcal{E}(m^*)$. Thus, $m' := \mathcal{D}(e) = \mathcal{D}(\mathcal{E}(m^*))$. By the losslessness of encoding, $m' = m^*$.

Since P's credential σ is valid, P obtains $m' = m^*$ and not $\perp$ at the end message decryption phase in Π_{YC}. Further, $e' := \mathcal{E}(m') = e$ since $\mathcal{E}$ is a deterministic algorithm. Thus, $e'[i_\ell] = e^{d_\ell}[i_\ell]$ for all $\ell \in [n]$. We established that $e^0[i_\ell] \neq e^1[i_\ell]$ for all challenge indices, allowing P to uniquely determine that $d'_\ell = d_\ell$ for each $\ell \in [n]$. Thus, V and P successfully establish a shared secret key $s_P = s_V \neq \perp$.

Soundness. Part 1 (Verifier security): We first simplify the original game as follows:

(1) Let E_0 denote the original $\Pi^{\mathcal{A}}_{YC}$ protocol.
(2) E_1: All encryptions/decryptions happen against keys k_{PS}, k_{PV}, k_{VP}, k_{SP} where k_{PS}, k_{PV} are sampled uniformly randomly from $\mathcal{K}_{\mathsf{SC}}$, $k_{SP} := k_{PS}$ and $k_{VP} := k_{PV}$. E_1 is indistinguishable from E_0 because by the authentication property of SC.Handshake, P does not share keys k_{PS} and k_{PV} with $\mathcal{A}$, and by the key secrecy property of SC.Handshake, keys k_{PS}, k_{PV} are indistinguishable from random elements drawn from $\mathcal{K}_{\mathsf{SC}}$. By the consistency property of SC.Handshake, peers V and S obtain symmetric keys $k_{SP} = k_{PS}$ and $k_{VP} = k_{PV}$.
(3) E_2: P sends $c^{d_\ell}[d'_\ell]$ as an encryption of a fixed dummy message from Σ_{SC} instead of an encryption of d'_ℓ. E_2 is indistinguishable from E_1 due to the IND-CPA security of SC.AEnc.
(4) E_3: V chooses $d := 0^n$ instead of choosing it as $d \leftarrow_R \{0,1\}^n$. We consider the following two sub-cases:
   - *Case 1:* $k_{SP} \neq k_{PS} \vee \hat{\mathcal{I}} \neq \mathcal{I} \vee \hat{c}^0 \neq c^0 \vee \hat{c}^1 \neq c^1$ where $\hat{\mathcal{I}}, \hat{c}^0$, and $\hat{c}^1$ denote potential adversary modifications to $\mathcal{I}, c^0$, and c^1 respectively. In this case $m' = \perp$. Hence, $\mathcal{A}$'s view is identical for any choice of d. Thus, $\mathcal{A}$ cannot distinguish between E_3 and E_2 in this case.
   - *Case 2:* $k_{SP} = k_{PS} \wedge \hat{\mathcal{I}} = \mathcal{I} \wedge \hat{c}^0 = c^0 \wedge \hat{c}^1 = c^1$. In this case $m' \neq \perp$. We show that m^0 (obtained in case V chooses $d = 0^n$ as in E_3) is indistinguishable from m' (obtained in E_2). Note that m' is defined as follows:

$$m'_j := \begin{cases} m_j^{d_\ell} & \text{if } \exists \ell \in [n] : j \in J_{i_\ell} \\ m_j^0 & \text{otherwise} \end{cases}$$

   By the separability of encoding, m^0 and m' differ at the indices i_ℓ for $\ell \in [n]$, where $d_\ell = 1$. However, $m_j^{\epsilon_j}$ and $m_j^{\neg\epsilon_j}$ are indistinguishable for all $j \in [|m^0|]$ since ϵ_j is independent and identically distributed. Thus, m^0 and m' are indistinguishable and thus $\mathcal{A}$ cannot distinguish between E_3 and E_2 in this case too.
(5) E_4: P sends $\hat{c}^0, \hat{c}^1$ as encryptions of a fixed dummy message from Σ_{SC} instead of encryptions of $\hat{e}^0, \hat{e}^1$ respectively. E_4 is indistinguishable from E_3 because of the IND-CPA security of SC.AEnc. Note that in this game, $\mathcal{A}$ does not obtain decryption of any message using key k_{PS}, which allows reduction to the IND-CPA game.

We now construct an adversary $\mathcal{B}$ that participates in the $\mathsf{AMP}(1^n, \mathsf{pk}_S, \mathcal{B}\langle\rangle, V\langle\rangle, S\langle\mathsf{sk}_S\rangle)$ protocol, and tries to make S output $m' \neq \perp$, while using $\mathcal{A}$ and presenting $\mathcal{A}$ a view indistinguishable to its view in E_4 to extract s_V.

The high-level approach for constructing $\mathcal{B}$ goes as follows. We split between the cases when $\mathcal{A}$ sits between P and V and when it sits between V and S.

If $\mathcal{A}$ sits between P and V: $\mathcal{B}$ relays messages between $\mathcal{A}$ and S during AMP.Setup. Next, $\mathcal{B}$ provides dummy encryptions from P to $\mathcal{A}$ just as in E_4, and sends encryptions of zeros to the server S played by the AMP challenger (just as in E_4). Then $\mathcal{B}$ sends S's output m' to $\mathcal{A}$. Finally, $\mathcal{B}$ supplies dummy encryptions from P to $\mathcal{A}$ in the challenge verification phase as in E_4. Now, if $m' = \perp$ then the probability that $s_V \neq \perp$ is negligible. Thus, it must be that $m' \neq \perp$.

If $\mathcal{A}$ sits between V and S: In this case, $\mathcal{B}$ simply relays the ciphertexts obtained from V played by the AMP challenger to $\mathcal{A}$. It then relays the message m' obtained from S played by the AMP challenger to $\mathcal{A}$. The case reduces to the previous case when $\mathcal{A}$ only obtains m' by the IND-CPA security of SC.AEnc.

Part 2 (Prover security): If $s_P \neq \perp$, the probability that $\mathcal{A}$ outputs s_P is negligible as this probability is exactly the probability of $\mathcal{A}$ breaking the authentication property of SC.Handshake.

Transcript Privacy. We proceed with the description of $\mathcal{S}$ in Figure 3. We argue that $\mathcal{A}$'s view in the simulation is indistinguishable from its view in Π_{YC}. Firstly, $\mathcal{S}$ forwards all messages obtained during AMP.Setup and AMP.Finish to $\mathcal{A}$, and $\mathcal{A}$'s view remains unchanged. Further, $\mathcal{S}$ samples $\mathcal{I}'$ randomly from κ similar to P randomly sampling $\mathcal{I}$ from $|\kappa|$ in Π_{YC}. Hence, $\mathcal{I}$ and $\mathcal{I}'$ are indistinguishable to $\mathcal{A}$.

Note that because of F_{OT}, $\mathcal{A}$ only obtains a single ciphertext for each $i \in \mathcal{I}$ in Π_{YC}. $\mathcal{A}$ cannot distinguish between this ciphertext and the ciphertext sent by $\mathcal{S}$ (for any $i \in I$) due to the IND-CPA property of SC.AEnc.

Finally, $\mathcal{S}$ engages in SC.Handshake with $\mathcal{A}$, obtains k_{PV} and st^0_{PV}, and sends c_h to $\mathcal{A}$, which is identical to steps in Π_{YC}. Hence, $\mathcal{A}$'s view in the simulation is indistinguishable from its view in Π_{YC}.

Transcript Integrity. Let $\hat{c}$ be the ciphertext sequence that $\mathcal{A}$ delivers to S. If S rejects $\hat{c}$ (outputs $\perp$), there is no violation. We consider the case where S accepts and outputs $m' \neq \perp$.

By the integrity and statefulness properties of SC.AEnc, $\mathcal{A}$ cannot forge new ciphertext blocks or reorder existing ones without causing decryption failure. Specifically, any $\hat{c}[i] \notin \{c^0[j], c^1[j]\}_{j \in [\kappa]}$ will cause $\mathsf{SC.ADec}(k_{SP}, \hat{c}[i], \mathrm{st}_i)$ to output $\perp$ with all but negligible probability. Similarly, any reordering where $\hat{c}[i] \in \{c^0[i'], c^1[i']\}$ for $i' \neq i$ will fail due to state mismatch. Therefore, except with negligible probability, we have:

$$\hat{c}[i] = \begin{cases} \hat{c}^{\hat{d}_\ell}[i_\ell] & \text{if } \exists \ell \in [n] : i = i_\ell \text{ where } \hat{d}_\ell \in \{0,1\} \\ c^0[i] & \text{otherwise} \end{cases}$$

By correctness of SC.ADec, S recovers encoded blocks:

$$e[i] = \begin{cases} e^{\hat{d}_\ell}[i_\ell] & \text{if } \exists \ell \in [n] : i = i_\ell \text{ where } \hat{d}_\ell \in \{0,1\} \\ e^0[i] & \text{otherwise} \end{cases}$$

By the separability property of $\mathcal{E}$, there exist disjoint sets J_i such that $e^b[i] = f_i((m_j^b)_{j \in J_i})$ for $b \in \{0,1\}$. This means we can construct a message m^* by symbol-wise interleaving:

$$m_j^* = \begin{cases} m_j^{\hat{d}_\ell} & \text{if } \exists \ell \in [n] : j \in J_{i_\ell} \text{ where } \hat{d}_\ell \in \{0,1\} \\ m_j^0 & \text{otherwise} \end{cases}$$

Since $m^0, m^1 \in \Delta(m)$ and $\Delta(m)$ is closed under interleaving, we obtain $m^* \in \Delta(m)$. By the uniformity and losslessness properties, $e = \mathcal{E}(m^*)$ and thus $m' = \mathcal{D}(e) = \mathcal{D}(\mathcal{E}(m^*)) = m^* \in \Delta(m)$.

Therefore, $\Pr[m' \notin \Delta(m)] \leq \mathrm{negl}(n)$.

Server Obliviousness. We show that $\mathcal{A}$'s view in YouChoose is indistinguishable from its view in AMP with some $m' \in \Delta(m)$.

We first establish what happens in the real YouChoose execution. $\mathcal{A}$ receives a ciphertext sequence $\hat{c}$ from V. By the integrity and statefulness properties of SC, any accepted $\hat{c}$ must, except with negligible probability, be position-wise taken from the honestly generated pairs $(c^0[i], c^1[i])$ at challenge indices and equal to $c^0[i]$ at non-challenge indices. Therefore, the decrypted blocks e correspond to some selection from $\{e^0[i], e^1[i]\}$ at challenge positions and $e^0[i]$ elsewhere.

By the separability property of $\mathcal{E}$, this selection induces a symbol-wise interleaving m^* of m^0 and m^1 such that $e = \mathcal{E}(m^*)$. By the closure property of $\Delta(m)$, we obtain $m^* \in \Delta(m)$, and by losslessness, $m' = \mathcal{D}(e) = m^*$.

We now construct a simulator $\mathcal{S}$ that simulates $\mathcal{A}$'s view in the AMP protocol. $\mathcal{S}$ operates as follows:

Setup phase: $\mathcal{S}$ runs SC.Handshake with $\mathcal{A}$ to obtain the same key and state $(k_{SP}, \mathrm{st}_{SP}^0)$ as in the real protocol.

Message transmission: $\mathcal{S}$ samples challenge indices $\mathcal{I}$ and selection bits d according to the same distribution as in Π_{YC}, and defines the interleaving m^* that these choices induce. $\mathcal{S}$ then computes $e^* = \mathcal{E}(m^*)$ and sends the stateful encryption $c^* = (\mathsf{SC.AEnc}(k_{SP}, e_i^*, \mathrm{st}_{SP}^i))_i$ to $\mathcal{A}$, exactly as a standard AMP sender would do for message m^*.

Remaining phases: $\mathcal{S}$ handles any network effects and the AMP.Finish phase identically to how $\mathcal{A}$ experiences them in the real protocol.

The views are indistinguishable because: (1) The handshake transcripts are identical by construction; (2) In the real execution, conditioned on acceptance, $\mathcal{A}$ receives the stateful encryption of $\mathcal{E}(m^*)$ for some $m^* \in \Delta(m)$, which is exactly what $\mathcal{S}$ sends in the simulated execution; (3) Any tampering or failure cases are handled identically since $\mathcal{A}$ controls the same network behavior in both cases; (4) The Finish phase proceeds identically since the delivered ciphertexts have the same distribution.

Therefore, $\mathcal{A}$'s view in YouChoose is indistinguishable from its view in AMP with message $m^* \in \Delta(m)$, establishing server obliviousness. □

C JPEG Compatibility

JPEG compresses images by partitioning them into 8x8 pixel blocks and applying a discrete cosine transform (DCT) to each block, yielding 64 frequency coefficients per block. These coefficients are quantized and then entropy-coded using Huffman or arithmetic coding. The first coefficient known as the DC coefficient (position (0,0)) represents the block's average intensity, while the remaining 63 coefficients known as AC coefficients encode higher-frequency details. Critically, JPEG's block independence after quantization means that modifications to one block's DCT coefficients do not affect the decodability of other blocks, directly satisfying our separability requirement. Although entropy coding operates across the entire image, JPEG uses standardized Huffman tables that are common across images, ensuring that blocks from separately encoded JPEGs can be interleaved without introducing decoding errors.

We create two image variants m^0 and m^1 by applying small modifications to DCT coefficients in the luminance (Y) channel for a randomly selected set of n challenge blocks. Within each challenge block, we randomly select a single non-zero AC coefficient and apply opposite perturbations: variant m^0 decreases its magnitude by 1 (moving toward zero), while m^1 increases its magnitude by 1 (moving away from zero), producing visually indistinguishable images that differ in exactly n blocks. The verifier can then interleave blocks from these variants to create a valid JPEG that the server decodes as a standard image. Our JPEG implementation is available at [36].