YouChoose: A Lightweight Anonymous Proof of Account Ownership

Aarav Varshney[1], **Prashant Agrawal**[2,3], **and Mahabir Prasad Jhanwar**[1,3]

[1]Department of Computer Science, Ashoka University, `{aarav.varshney, mahavir.jhawar}@ashoka.edu.in`
[2]Department of Computer Science and Engineering, IIT Delhi, `prashant@cse.iitd.ac.in`
[3]Centre for Digitalisation, AI and Society, Ashoka University

Abstract

We explore the issue of anonymously proving account ownership (anonymous PAO). Such proofs allow a prover to prove to a verifier that it owns a valid account at a server without being tracked by the server or the verifier, without requiring any changes at the server's end and without even revealing to it that any anonymous PAO is taking place. This concept is useful in sensitive applications like whistleblowing. The first introduction of anonymous PAOs was by Wang et al., who also introduced the secure channel injection (SCI) protocol to realize anonymous PAO in the context of email account ownership. In this paper, we propose YouChoose, an approach that improves upon Wang et al.'s SCI-based anonymous PAO. Unlike SCI, which demands carefully designed multi-party computation (MPC) protocols for efficiency, YouChoose works without MPC, simply relying on the verifier to selectively forward TLS records. It is faster, more efficient, and more adaptable compared to SCI. Further, the simplicity of the YouChoose approach readily enables anonymous PAO in different settings such as various ciphersuites of TLS, account types other than email, etc., while the SCI approach needs specifically designed MPC protocols for each use case. We also provide formal security definitions for a generalized anonymous PAO of which both YouChoose and SCI are concrete instantiations.

1 Introduction

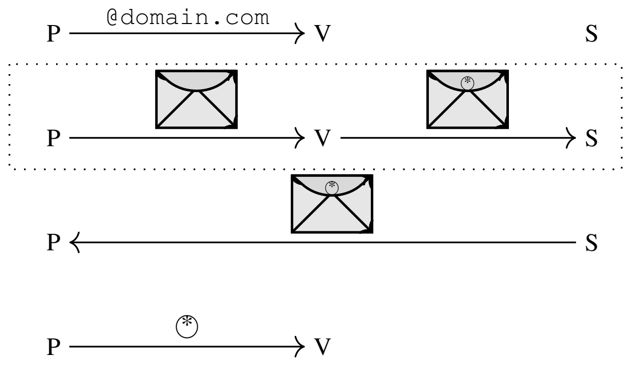
Anonymous credentials provide a powerful tool for making assertions about identity while maintaining privacy. In an anonymous credential scheme, individuals identify themselves by using different pseudonyms with different organizations, thus preventing even colluding credential issuers and verifiers from identifying and tracking users. More precisely, say a person P wants to anonymously prove to a verifier V that they possess a credential from a credential issuer I. P obtains a credential from I against a pseudonym P_I they use with I, and transforms it to a credential against a pseudonym P_V they use 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 obtain such anonymous credentials from I. For instance, consider a whistleblower 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. 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 any self-incriminating anonymous credential. Similarly, when third-party identity providers act as credential issuers then the business interests of these identity providers do not always align with the privacy interests of the users. Thus, infrastructural changes to support anonymous credentials may not be a priority item for them. Hence, a practical security goal is anonymous authentication without requiring any infrastructural changes at the issuer and without even letting the issuer know of any such authentication taking place.

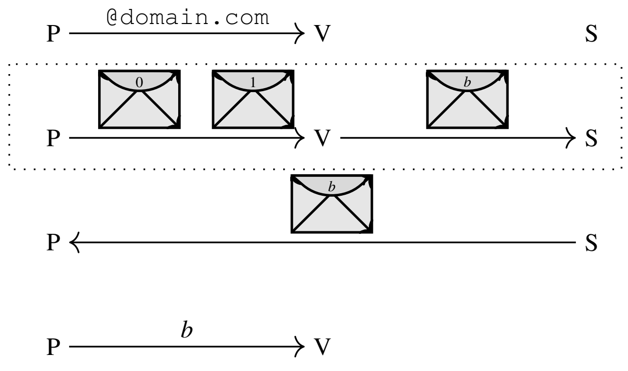
Specifically, we focus on a specific anonymous credential called an email-based anonymous proof of account ownership (PAO). Here, P wants to prove to V that they own an email account with I, without revealing which account. Email is ubiquitous in business, government, education and daily life. As a result, employees routinely have working email addresses within their organizations. Email-based anonymous PAO thus fits perfectly with our goal of avoiding infrastructural changes from I while anonymously proving the insider status of P to V.

An email-based non-anonymous PAO typically relies on P's ability to *retrieve* email sent to the claimed email address (V sends a random challenge to the claimed email address and requires P to produce the challenge), but it reveals P's email address to V. Email-based anonymous PAO was first introduced by Wang et al. [1], where they let P prove ownership of an email account at a domain without revealing their email address, by relying on P's ability to *send* emails from the claimed email address. The verifier V acts as a proxy sitting between P and 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 securely injects a challenge into messages sent over the secure channel such as TLS between P and S. This is done via a 2-party computation protocol called *secure channel injection* (SCI) [1] that allows P and V to jointly compute an authenticated and encrypted message containing the injected challenge, without P learning the challenge bits and V learning the message contents 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 it proves to V that the email was successfully sent by S. This is a proof of email account ownership because S successfully sends emails from P only if it can verify that P holds an account with S.

YouChoose. In this paper, we introduce YouChoose, an alternative approach to anonymous PAO that is considerably lighter-



(a) Anonymous PAO using SCI



(b) Anonymous PAO using YouChoose

Figure 1: SCI based anonymous proof of account ownership (PAO) on the left versus our YouChoose based anonymous PAO on the right. Here, P is the prover, S is the email server of `domain.com`, and V is the verifier. ⊛ represent the injected challenge; 0, 1 represent the order of the email messages and b represents the bit denoting the email forwarded by V.

weight and general than the SCI-based approach of Wang et al. The basic idea behind YouChoose is shown in Figure 1b. P sends two email messages m_0 and m_1 from `alice@domain.com` to an account they own. V ensures that the interaction is with `domain.com` and forwards only one of the emails m_b for a randomly chosen $b \in \{0, 1\}$. P's challenge is to find out whether m_0 or m_1 was forwarded. If P sends the correct bit b to V, it 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 repeating the above process, this would require P to send multiple emails. We optimize this by carefully designing a protocol that achieves a negligible soundness error with just one email while respecting the restrictions imposed by the underlying TLS secure channel.

The YouChoose approach is simpler, more efficient, and more general than the SCI-based approach for anonymous PAO. Since general-purpose 2-party computation (2PC) protocols for SCI are expensive, Wang et al. designed special-purpose SCI protocols by leveraging recent advancements in maliciously secure 2PC for the AES-CBC/HMAC-SHA-256 ciphersuite of TLS 1.2 [1] and in oblivious polynomial evaluation for the AES-GCM ciphersuite. Similar efficient 2PC protocols need to be carefully designed to support other ciphersuites like ChaCha20-Poly1305 in TLS 1.3 [2]. In contrast, the YouChoose approach of selectively forwarding messages is fairly general and can be used not only with any ciphersuite in TLS 1.2 or TLS 1.3, but also for anonymous PAO beyond just email. For example, P could anonymously prove ownership of a Twitter account to V by having V selectively forward tweets submitted by P via the Twitter API. Further, since the most expensive step in YouChoose is a few simple coin tosses, it is more efficient than Wang et al.'s specially designed SCI protocol, which incurs roughly 3 seconds overhead per session versus only a second in case of YouChoose.

It is also interesting to note that Wang et al.'s scheme requires the prover to have exclusive access to the recipient account and is thus restricted to email-type applications. In YouChoose, the challenge is identifying whether the first or the second email was forwarded and not the email contents themselves, so it does not require the prover to have exclusive access to the recipient account. This may be useful in applications such as the Twitter example above, where the finally posted tweets may be read by anybody.

Our main contributions are the following:

- We introduce the YouChoose approach, a simpler and more efficient alternative to SCI-based anonymous PAO.
- We formalize the general paradigm of anonymous PAO common to both YouChoose and SCI, of V acting as a proxy between P and S, securely modifying messages in the secure channel established between P and S and using these modifications as a challenge for P to prove account ownership (Section 3). Our security definitions generalize the definitions given by Wang et al.
- We propose and implement a concrete YouChoose protocol for SMTP-TLS and discuss the implementation aspects for anonymously proving email account ownership using SMTP over the TLS secure channel (Section 4). We analyze the security of the protocol in Section 5. We provide our implementation at this github link[1] and plan to release it as open-source software.

2 RELATED WORK

Whistleblowing. Whistleblowing is crucial to educate the public of misdeeds and to call those in power to account. Therefore, it is desirable to cryptographically protect the identity of a whistleblower to allow a low-risk disclosure of wrongdoing. The use of secure messaging apps [3, 4], mix-nets [5], or onion routing systems such as the Tor network [6] allows whistleblowers to communicate anonymously with journalists. However, these systems do not provide a way for whistleblowers to prove their insider status.

A whistleblower could use group signatures schemes [7] instead which allow a member of a group (such as an organization) to sign a message on behalf of the group without revealing their identity. However, it violates the privacy we seek as a group manager can still identify the signer. Ring signatures [8] do

[1]Link: https://github.com/aarav22/anon-pao

not require such group manager, but nevertheless allow signing on behalf of an ad hoc group of public keys. However, this method requires that all group members possess public keys accessible via a trusted directory, hence requiring cooperation of the organization.

Anonymous credentials allow a user to prove to another party that they have a valid identity certified by some certificate authority (CA), without revealing the identity itself [9, 10, 11]. Nevertheless, the CA learns that the user wishes to use a privacy tool such as an anonymous credential, which in itself is sensitive information in use-cases such as whistleblowing.

Email-based Anonymous PAO. An anonymous PAO allows a prover to prove ownership of an account to a verifier without revealing which account. In case of email identity, instead of proving that the account owner owns a particular email address such as `alice@domain.com`, the verifier will be convinced if the prover proves that it owns some valid email address at `domain.com`.

Secure Channel Injection (SCI) [1] allows a verifier to inject a random challenge into an email sent from a prover's email account to an account accessible only by the prover. The SCI protocol by Wang et al. depends on two 2PC protocols: 2P-HMAC and 2P-CBC, which are used to jointly compute the HMAC tag and the TLS ciphertext, respectively. However, this approach not only requires a 2PC protocol making it computationally expensive but it is currently only optimized for AES in CBC mode that lacks support in TLS 1.3.

Multi-context TLS (mcTLS) [12] is a modified version of TLS that enables middleboxes to read and/or access specific parts of a TLS connection by revealing some specific session keys to them. This modification allows for message injection similar to SCI, but it is not compatible with existing web infrastructure as it requires changes in TLS. Moreover, for mcTLS to work, the server must have full awareness of the modifications made by the middleboxes, which might arouse suspicion of a non-standard protocol execution in the server.

Mailet, as described in [13], is a censorship-circumvention system designed to enable users in censored regions to access social media websites. It employs a similar setup where the client communicates with the website through a distributed set of proxy servers, but the client needs to share its credentials among the proxies. These distributed proxies then engage in a secure computation protocol with each other to collectively compute the TLS record, facilitating authentication with the remote server. Although one might consider utilizing their protocol for anonymous PAO, if the proxy servers collude, it allows them to impersonate as the client and forge messages.

DECO [14] is a protocol that allows the client to prove to a third-party verifier that it received a certain response from the server. This could theoretically be used to anonymously prove ownership of an account if the server sends a distinct success message upon successful authentication, since the client could prove to the verifier that it received a success message. However, the way it is implemented requires the session key to be split between the client and the verifier and thus requires a secure MPC protocol for client authentication. However, this approach does not work for email-based anonymous PAO as the SMTP standard does not require servers to send a distinct success message to the clients upon authentication.

3 FORMALIZATION OF ANONYMOUS PAO

We now formalize a generalized version of SCI-based anonymous PAO [1] where *a*) the modifications made by V could be more general than simply injecting messages in the secure channel between P and S and, in particular for YouChoose, include dropping of messages, and *b*) the proof is for general account ownership as opposed to just email account ownership; we characterize this general account ownership by a credential verification algorithm CVer run by S that takes as input an account credential *cred* and outputs 1 if the credential was valid and 0 otherwise (e.g., for email-based account ownership, CVer would take the email and password as input credential and output 1 if the email was registered at the domain and the supplied password was correct and 0 otherwise).

Formally, we say that an anonymous PAO scheme is a tuple of protocols $(\mathsf{Setup}, \mathsf{GenToken}, \mathsf{Prove})$ between P, V and S defined as follows:

- $(P\langle k\rangle, V\langle\rangle, S\langle k\rangle) \leftarrow \mathsf{Setup}(d, P\langle\rangle, V\langle\rangle, S\langle\rangle)$: This is a protocol to establish a secure channel between P and S with V acting as a proxy. All of P, V and S obtain an identifier d identifying S's domain; neither of them obtain any secret input. P and S output a common session key k for the secure channel; V does not obtain any output.
- $(P\langle st_p\rangle, V\langle st_v\rangle, S\langle m'/\perp\rangle) \leftarrow \mathsf{GenToken}(P\langle k, cred\rangle, V\langle\rangle, S\langle k\rangle)$: This is a protocol to generate an authentication token that P can use to anonymously prove to V that it owns a valid account at S. P supplies the account ownership credential *cred* as input, and both P and S supply the secure channel's session key k. P sends a message m to the secure channel during the protocol and S outputs a message $m' \in \mathcal{M}(m)$ if $\mathsf{CVer}(c) = 1$, where $\mathcal{M}(m)$ denotes the set of modifications that V is allowed to make on m; S outputs $\perp$ if $\mathsf{CVer}(cred) = 0$. The output m' acts as a token that P needs to obtain from S through an out-of-band channel. P and V output their internal states st_p and st_v to be used later.
- $(P\langle\rangle, V\langle 0/1\rangle) \leftarrow \mathsf{Prove}(P\langle st_p, m'\rangle, V\langle st_v\rangle)$: This is a protocol that allows P to prove account ownership to V using the token obtained from S at the end of the $\mathsf{GenToken}$ protocol. P and V supply their internal states st_p and st_v as input and P additionally supplies the token m'. V outputs 1 if the token is valid and 0 otherwise.

We also assume that all protocols implicitly obtain a security parameter n (in unary).

Security Properties. Our security properties are naturally extensions of the security properties proposed by Wang et al. [1] for SCI-based PAO. The soundness requirement (Definition 3.1) captures that the verifier accepts the proof only if the prover indeed owns a valid account with the server. This soundness requirement is more direct than the *injection secrecy* property of [1] that provides soundness in the SCI setting. Injection secrecy requires that P does not learn the challenge injected by V during the $\mathsf{GenToken}$ protocol and hence implies soundness because then P can retrieve the challenge only if it has an account at S. We cannot use this definition directly because the challenge in YouChoose is which of the message pairs selected by P was dropped and hence we opt for the more direct soundness definition.

Transcript privacy (Definition 3.2) is required to prevent leakage of credential information to V and thus impersonation of P by V. Transcript integrity (Definition 3.3) prevents V from modifying the messages sent over the secure channel beyond what is allowed by the modification function $\mathcal{M}$. This is required, e.g., to prevent V from sending a malicious email to a modified recipient address on behalf of P. Finally, server obliviousness (Definition 3.4) captures that S remains unaware of any anonymous PAO taking place between P and V. We require that the transcript that S receives looks indistinguishable from the case when P establishes a secure connection with S and sends through the secure channel a message that follows a distribution $\mathcal{D}$, where $\mathcal{D}$ denotes the distribution of messages routinely sent by account owners through S for purposes other than anonymous PAO. Thus, server obliviousness captures that the server cannot distinguish an execution of anonymous PAO with a routine communication of messages from the account owners via S.

Definition 3.1 (Soundness)**.** For all PPT adversaries P supplying an input credential *cred* such that $\mathsf{CVer}(cred)$ outputs 0, the probability that V outputs 1 in the Prove protocol when S runs honestly is negligible.

Definition 3.2 (Transcript privacy)**.** The $\mathsf{GenToken}$ and Prove protocols do not reveal any information about P's account credential *cred* or the session key k to V.

Definition 3.3 (Transcript integrity with respect to $\mathcal{M}$)**.** For all PPT adversaries V, if P and S are honest and P supplies a message m to the secure channel, S cannot output $m' \notin \mathcal{M}(m)$, except with negligible probability.

Definition 3.4 (Server obliviousness with respect to $\mathcal{D}$)**.** Assuming that P and V are honest, S cannot distinguish between *a*) an execution of Setup and $\mathsf{GenToken}$ where it obtains a key k during Setup and a message m' during $\mathsf{GenToken}$, and *b*) an execution of the setup and communication steps of a secure channel established with P where P sends a message $m' \leftarrow \mathcal{D}$ to the secure channel.

Network Assumptions. We now highlight the network assumptions under which we aim to achieve the above security properties. We do not require any more assunmptions than those made in Wang et al.'s work for SCI-based anonymous PAO [1]:

1. Each party can only view the traffic on their own local network. In particular, P cannot access the transcript of interactions between V and S and S cannot access the transcript of interactions between P and V. Without the first assumption, it is impossible to achieve soundness since P can readily identify the modifications sent to S without requiring an account at S. Without the second assumption, it is impossible to achieve server obliviousness.
2. No party can spoof their identities or re-route messages to nodes other than the intended recipients. Without this assumption, a malicious S could spoof as V and violate server obliviousness (and identify P) or a malicious P can spoof as S and violate soundness.
3. S cannot detect a non-standard secure channel protocol by just looking at V's IP address.

3.1 Preliminaries

In this section, we describe notation and recall some background about TLS, which acts as the secure channel for our protocol, and SMTP, which acts as the application layer protocol for establishing email account ownership.

Notation. Let $x \xleftarrow{\$} S$ denote an element x sampled uniformly at random from a set S. Given a positive integer x, let $[x]$ denote the set $\{1, \ldots, x\}$. We use $\perp$ to indicate an empty message or failure and $||$ to denote concatenation of elements.

3.1.1 Electronic Mail and SMTP.

The Internet e-mail system has three major components: *user agents*, *mail servers*, and the *Simple Mail Transfer Protocol* (SMTP). These components can be described in the context of a sender, Alice, sending an e-mail message to a recipient, Bob. A typical e-mail message starts its journey in the sender's user agent, travels from the sender's user agent to the sender's mail server (this process is called *message submission*), and travels from the sender's mail server to the recipient's mail server (this process is called *message transmission*), where it is deposited in the recipient's mailbox. The recipient could access the messages in their mailbox by logging into its mail server. Microsoft's Outlook, Apple Mail, and Mozilla Thunderbird are among the popular user agents for e-mail. Mail servers are commonly operated by the user's service provider. Webmail solutions such as Gmail blur the distinctions between mail submission and mail transfer somewhat.

SMTP is the principal application-layer protocol for Internet electronic mail. SMTP define both server-to-server and client-to-server communication patterns. While it is primarily used to execute message transmission, i.e., transfer mail from the sender's mail server to the recipient's mail server, it is also used for mail submission, i.e., sending mail from the sender's user agent to the sender's mail server. Once at the destination server, email can be retrieved using protocols such as POP3 and IMAP.

We now briefly review how SMTP submits an e-mail message from a sender's user agent to the sender's mail server. First, the client SMTP (running on the sender's user client) has TCP establish a connection to port 587 at the server SMTP (running on the sender's mail server). Once the connection is established, the client and server perform application layer SMTP handshaking: the SMTP client indicates the e-mail address of the sender and the e-mail address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client sends the messages. In particular, the client issues five commands: `EHLO` (an abbreviation for Extended HELLO), `MAIL FROM`, `RCPT TO`, `DATA`, and `QUIT`. These commands are self-explanatory (the `DATA` command signals the client's readiness to send the email content, including headers and body, after which the client starts transmitting the actual email data). The server issues replies to each command, with each reply having a reply code. SMTP did originally not require authentication for message submission (i.e., user agent to mail server), but this was added later to fight spam. If the client needs to authenticate itself before sending the email, it uses the `AUTH` command.

The server may support various authentication mechanisms like LOGIN, PLAIN, or CRAM-MD5. The `PLAIN` version of the AUTH command allows a client to provide its username and password to the server in plain.

As SMTP runs over TCP, all SMTP messages travel as payload in TCP packets. TLS is a popular mechanism for enhancing TCP communications with confidentiality, data integrity, server authentication, and client authentication. TLS consists of two primary components: **Handshake Protocol** - it authenticates the communicating parties, negotiate cryptographic modes and parameters, and establish shared keying material; **Record Protocol** - it uses the parameters established by the handshake protocol to protect (confidentiality and integrity) traffic between the communicating peers.[2] Therefore, for a comprehensive security, SMTP must run over TLS. There are two ways to negotiate an TLS session. The first is to use TLS directly. This requires a IANA-assigned dedicated port. Application layer protocols that use this method are often indicated by adding a 'S' at the end, e.g., SMTPS. In the case of SMTP, port 465 was initially defined for SMTPS, but was deprecated later (it is nevertheless still used).

The second major way to use TLS is to connect with TCP on the normal port first and then upgrade the connection using a protocol-specific command. This method is commonly referred to as STARTTLS. The specifications in the RFCs commonly require clients to first query a server for STARTTLS support with a specific 'capability' command before trying to upgrade the connection [15]. The server can confirm an upgrade; the TLS handshake follows.

Requirements for the SMTP server. (1) Auth: Only authenticated users with legitimate sender addresses can send emails using a server that is correctly setup using a valid TLS certificate and being configured as a closed relay. (2) NoEcho: For soundness, the server should not echo back received commands to the client.

3.1.2 TLS Ciphersuites

TLS offers multiple ciphersuites to construct TLS records. In this paper, we focus on two ciphersuites: AES in CBC mode with HMAC-SHA256 (MAC-then-encrypt construction) and AES with Galois / Counter Mode (GCM). We briefly describe these two cipher suites below.

Cipher I. In this scheme, CBC mode of operation is used in AES for encryption and HMAC-SHA256 for authentication. This cipher suite is vulnerable to specific padding and timing attacks, as detailed in [16, 17, 18] and is defined in a now-expired RFC draft [19]. Despite its exclusion from TLS 1.3, it remains prevalent in TLS 1.2. In the widely adopted MAC-then-encrypt method, an authentication tag T is computed over a message M, followed by the encryption of both M and T to generate the ciphertext C. Authentication tag T using HMAC is calculated as $T = \mathsf{HMAC}(K, M) = \mathsf{H}((K \oplus opad) \,||\, \mathsf{H}((K \oplus ipad) \,||\, M))$, where K denotes the MAC key, H represents the SHA256 hash function, and *ipad* and *opad* are predefined constants. Let $P = (P_1, \ldots, P_t)$ where P_i, $i \in [t]$, represents a 128-bit block. To encrypt P using the CBC mode, a random 128-bit IV is chosen, such that $C_0 = IV$ and $C_i = \mathsf{AES}_{K'}(C_{i-1} \oplus P_i)$ for $i \in [t]$, where K' is an AES key to ultimately output $C = (C_0, \ldots, C_t)$.

Cipher II. AES-GCM utilizes AES in counter mode and employs Galois field arithmetic for authentication, as specified in [20]. In contrast to **Cipher I**, AES-GCM integrates authentication within the stream cipher, avoiding the need for a separate MAC computation. It is both supported and recommended for TLS 1.3 as it takes advantage of parallel processing and pipelining to achieve high speeds with low cost and low latency. In AES-GCM, a series of 128-bit counters J_i for $i \in [t]$ are encrypted using AES with a key K. The resulting ciphertext C is produced by combining these encrypted counters with 128-bit plaintext blocks $P = (P_1, \ldots, P_t)$ as $C = \mathsf{AES}_K(J_i) \oplus P_i$ for $i \in [t]$. The associated data A and ciphertext blocks are processed together using a multiplication operation with a K-dependent constant H (where $H = \mathsf{AES}_K(0^{128})$) in the Galois field $\mathsf{GF}(2^{128})$. This process, carried out by the GHASH function, generates the authentication tag $T = \mathsf{GHASH}_H(A, C)$. The authentication tag is appended to the ciphertext to produce the final output $C' = (C, T)$.

3.1.3 Secure Channel Injection (SCI)

SCI [1] is a three-party protocol between a client, a proxy, and a server, parameterized by a message $M = M^p || M^* || M^s$ where the client holds as input a message prefix $M^p \in \{0,1\}^*$, message suffix $M^s \in \{0,1\}^*$ and the proxy is interested in injecting a challenge/message $M^* \in \{0,1\}^*$ into the client-server interaction. The protocol is secure if the client learns nothing about the injected message M^*, and the proxy learns nothing about the client's message prefix M^p and message suffix M^s. SCI must be instantiated with a secure channel protocol, such as TLS, and the proxy is required to inject a message into an existing secure channel communication between the client and the server. In the case of SCI for TLS, the client and the proxy collaboratively compute a valid TLS record, leveraging the intricacies of TLS encryption. They describe their implementation for two TLS cipher suites: **Cipher I** and **Cipher II**.

For **Cipher I**, SCI is implemented using two sub-protocols 2P-HMAC and 2P-CBC. Within 2P-HMAC, the client computes a partial tag T_1 over M^p and sends it to the proxy. Subsequently, the proxy calculates a partial tag T_2 over M^* using T_1 and returns it to the client. The client completes the computation of the tag T by using T_2 and the remaining message M_s. Moving to 2P-CBC, when $M = M^p || M^* || M^s$, where $M^p = (P_1, \ldots, P_q)$, $M^* = (P_{q+1}, ..., P_{q+r})$, and $M^s = (P_{q+r+1}, \ldots, P_t)$, then the client transmits $C_0, \ldots, C_q$ to the proxy. It engages in a secure multi-party computation (MPC) for $C_{q+1}, \ldots, C_{q+r}$ (or $\mathsf{AES}_\mathsf{K}(C_{i-1} \oplus P_i)$ for $i = q+1$ to $q+r$) and obtains C_{q+r} from the proxy to compute $C_{q+r+1}, \ldots, C_t$. The proxy forwards a valid TLS record $C = (C_0, \ldots, C_t)$,encapsulating the message M, to the server.

For **Cipher II**, SCI is implemented using 2P-CTR and 2P-GMAC. In 2P-CTR, the client straightforwardly encrypts $M^p = (P_1, \ldots, P_q)$ as $C_p = (\mathsf{AES}_K(J_1) \oplus P_1, \ldots, \mathsf{AES}_K(J_q) \oplus P_1)$ and M^s as $C_s = (\mathsf{AES}_K(J_{q+r+1}) \oplus P_{q+r+1}, \ldots, \mathsf{AES}_K(J_t) \oplus P_t)$ locally, computes a 'key stream,' $\mathsf{AES}_K(J_{q+1}), \ldots, \mathsf{AES}_K(J_{q+r})$ and then transmits it to the proxy, and the proxy uses the key stream to encrypt M^*, resulting in C^*. 2P-GMAC is more intricate. For GMAC, the computation of GHASH over ciphertext blocks $C = (C_1, \ldots, C_t)$ at the point H is required. However,

[2]Handshake and Record Protocol correspond to the setup and communication steps of the TLS secure channel respectively.

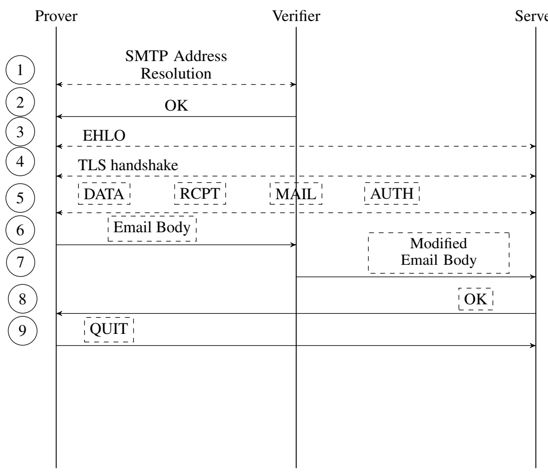


Figure 2: SMTP-TLS based anonymous PAO. Here, the dashed rectangle represents a TLS fragment. Steps 1-2 set up the prover and the verifier. Step 3 queries the server's capabilities. Step 4 performs a TLS handshake. Steps 5-9 send the email.

the client does not possess the entire C. It only holds the point H, $C_p = (C_1, \ldots, C_q)$, and $C_s = (C_{q+r+1}, \ldots, C_t)$, while the proxy holds $C^* = (C_{q+1}, \ldots, C_{q+r})$. This computation is reduced to evaluating a polynomial $p(x)$ at the point H, where $p(x) = \sum_{i=1}^{t} C_i x^{t-i+1}$. Here, $p(x)$ can be further expressed as the sum of polynomials: $p(x) = p^p(x) + p^*(x) + p^s(x)$. The client possesses both $p^p(x)$ and $p^s(x)$ and only needs the evaluation of $p^*(x)$ which depends on C^*. To evaluate $p^*(x)$, an oblivious polynomial evaluation scheme is used. In the process, the verifier acts as a sender, supplying coefficients of the polynomial $p^*(x)$, and the client serves as a receiver, holding the point H and only learning the evaluation of $p^*(H)$.

4 THE YOUCHOOSE PROTOCOL

We now describe the YouChoose protocol for anonymous PAO. The high-level outline of our scheme using TLS as the secure channel and SMTP as the application level protocol is shown in Figure 2. We begin with a basic version of our protocol that works for all authenticated encryption schemes based on the MAC-then-Encrypt or Encrypt-then-MAC paradigm, such as the AES-CBC/HMAC-SHA256 ciphersuite used in TLS 1.2 (Section 4.1). However, this basic protocol does not satisfy transcript privacy under nonce-based authenticated encryption schemes such as AES-GCM used in TLS 1.2 and 1.3. Thus, we also propose a variant of YouChoose for this case in Section 4.2.

4.1 *The Basic Protocol*

We now describe the basic YouChoose protocol (see Figure 3). First, during the Setup protocol, P and S establish a secure channel among them with V acting as a proxy in-between. During this protocol, P and V obtain S's domain name d. V queries the DNS for the IP address of an SMTP server with domain name d, checks if the server supports the STARTTLS extension by making an SMTP query to this IP address, and aborts the protocol if it does not. P then initiates a TLS handshake with S with V acting as a proxy to establish a session key k for the secure channel between P and S.

```
Setup($d, P\langle\rangle, V\langle\rangle, S\langle\rangle$) :
 1   $V$:  query DNS for $d$ and obtain an IP address $Saddr$
 2         send SMTP query to $Saddr$ to check if it supports STARTTLS
 3         **if** yes: establish a TCP connection with $S$ and notify $P$
 4         **else**: abort
 5   $P$:  perform TLS handshake with $S$ via proxy $V$ and establish key $k$
**output:** $P\langle k\rangle, V\langle\rangle, S\langle k\rangle$

GenToken($P\langle k, cred\rangle, V\langle\rangle, S\langle k\rangle$) :
 1   $P$:  send header AUTH containing $cred$ to $S$ via $V$
 2         send header MAIL containing sender email address to $S$ via $V$
 3         send header RCPT containing recipient email address to $S$ via $V$
 4         send header DATA to $S$ via $V$ // begin email body
 5         prepare email body $m = ((p_i)_{i=1}^{n_p}, (c_{i0}, c_{i1})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$
 6         $st_p := (c_{i0}, c_{i1})_{i=1}^{n}$
 7         let $sqn$ = current TLS sequence number
 8         **for** $i \in [n_p + n + n_s]$:
 9           **if** $i \leq n_p$:
10             $ep_i \leftarrow$ TLS.Enc($k, p_i, sqn$) // see Figure 4
11           **else if** $i \leq n_p + n$:
12             $ec_{i0} \leftarrow$ TLS.Enc($k, c_{i0}, sqn$)
13             $ec_{i1} \leftarrow$ TLS.Enc($k, c_{i1}, sqn$)
14           **else if** $i > n_p + n$:
15             $es_i \leftarrow$ TLS.Enc($k, s_i, sqn$)
16           $sqn \leftarrow sqn + 1$
17         update the sequence number in TLS to $sqn$
18         send $(n_p, n, n_s, (ep_i)_{i=1}^{n_p}, (ec_{i0}, ec_{i1})_{i=1}^{n}, (es_i)_{i=1}^{n_s})$ to $V$
19         send header QUIT to $S$ via $V$ // end email transaction

20   $V$:  **for** $i \in [n]$: $b_i \overset{\$}{\leftarrow} \{0, 1\}$
22         send $((ep_i)_{i=1}^{n_p}, (ec_{ib_i})_{i=1}^{n}, (es_i)_{i=1}^{n_s})$ to $S$
23         $st_v := (b_1, \ldots, b_n)$
24   $S$:  decrypt $m' = ((p_i)_{i=1}^{n_p}, (c_{ib_i})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ from the TLS channel
25         **if** CVer($cred$) = 1: $res \leftarrow m'$
26         **else**: $res \leftarrow \perp$
**output:** $P\langle st_p\rangle, V\langle st_v\rangle, S\langle res\rangle$

Prove($P\langle st_p, m'\rangle, V\langle st_v\rangle$) :
 1   $P$:  parse $m'$ as $m' = ((p'_i)_{i=1}^{n_p}, (c'_i)_{i=1}^{n}, (s'_i)_{i=1}^{n_s})$
 2         parse $st_p$ as $st_p = (c_{i0}, c_{i1})_{i=1}^{n}$
 3         **for** $i \in [n]$:
 4           $b'_i = 0$ if $c'_i = c_{i0}$ else $b'_i = 1$
 5         send $(b'_1, \ldots b'_n)$ to $V$
 6   $V$:  parse $st_v$ as $st_v = (b_1, \ldots, b_n)$
 7         $res \leftarrow 1$ if $(b_1, \ldots b_n) = (b'_1, \ldots b'_n)$ else 0
**output:** $P\langle\rangle, V\langle res\rangle$
```

Figure 3: The basic YouChoose protocol. TLS.Enc denotes the TLS encryption algorithm that explicitly takes a sequence number as input (see Figure 4).

During the GenToken protocol, P submits an email to the mail server S using SMTP over the TLS secure channel established above, and V drops parts of the email body such that the email finally forwarded by S allows P to construct an anonymous PAO authentication token. P first sends SMTP headers AUTH, MAIL, RCPT, and DATA to S in order to set up the email transaction, where AUTH contains P's email account credentials (e.g., username and password), MAIL contains the sender email address, RCPT contains the recipient email address, and DATA notifies the beginning of the email body. After this, P constructs an email body containing a sequence of challenge message pairs and V drops a randomly chosen message out of each message pair before forwarding them to S, such that S still accepts the received transcript as a valid SMTP email. As per the SMTP protocol, S forwards the email to the recipient mail server only if the credentials supplied in the AUTH header are valid. P can retrieve the contents of the forwarded email either by accessing the recipient email account or by accessing the sent emails folder of the sender email account.

During the Prove protocol, P compares the sequence of forwarded messages in the retrieved email with the original message pairs and constructs the sequence of challenge bits denoting which of the message pairs were forwarded by V. V verifies these challenge bits, completing the anonymous PAO.

```
TLS.Enc($k = (k_{hmac}, k_{enc}), m, sqn = \perp$)
  if $sqn == \perp$:
    let $sqn$ = current TLS sequence number
  let $type$ = TLS data type, $version$ = TLS version
  $mac \leftarrow$ HMAC($k_{hmac}, sqn \parallel type \parallel version \parallel size(m) \parallel m$)
  $data := m \parallel mac$
  $ciphertext \leftarrow$ AES-CBC($k_{enc}, data$)
  $record := (type \parallel version \parallel$ size$(ciphertext) \parallel ciphertext)$
  return $record$
```

Figure 4: TLS.Enc algorithm for the AES-CBC/HMAC authenticated encryption scheme.

Dropping parts of email body under TLS. Notice that modification or even dropping of parts of the email body by V is not allowed as per the integrity guarantees provided by the TLS via authenticated encryption schemes and strict sequence numbering. We circumvent this restriction as follows. P splits the email message body m into multiple fragments such that $m = ((p_i)_{i=1}^{n_p}, (c_{i0}, c_{i1})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$, where c_{i0}, c_{i1} for each i denote the fragments corresponding to the challenge message pairs and p_i and s_i respectively denote fragments for the prefix and suffix of the challenge message pairs in the email body. P encrypts each of these fragments individually into a TLS record via an Encrypt-then-MAC or MAC-then-Encrypt authenticated encryption scheme such that for each challenge message pair i, both c_{i0} and c_{i1} are assigned the same sequence number and a strictly monotonic sequence numbering is followed otherwise. V then drops one of the encrypted TLS records corresponding to each challenge message pair, while forwarding all other TLS records as-is. In this way, S obtains a valid email body consisting of a sequence of TLS records with strictly monotonic sequence numbers.

Since the only unpredictable decision made by V is whether to drop or forward a TLS record corresponding to a challenge message pair, the number of challenge message pairs n directly determines the level of security provided by our scheme (i.e., with n such pairs, the probability that P incorrectly proves account ownership to V is bounded by 2^{-n}).

Transcript privacy. Note that in Encrypt-then-MAC or MAC-then-Encrypt schemes, V only obtains a sequence of semantically secure ciphertexts and hence transcript privacy is maintained. This is not the case in nonce-based authenticated encryption schemes such as AES-GCM, where V obtains a sequence of ciphertexts encrypted under the same nonce for each challenge message pair. V can exploit this to learn the session key, thus violating transcript privacy. Thus, we propose an alternative protocol for AES-GCM in Section 4.2.

Preparing email body for server obliviousness. To achieve server obliviousness, we require that the final email body $m' = ((p_i)_{i=1}^{n_p}, (c_{ib_i})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ for some $b_i \in \{0, 1\}$ received by S must be indistinguishable from some distribution $\mathcal{D}$ representing routine email communication among the account owners. Towards this end, we suggest a simple strategy that hides the challenge message pairs into innocuous looking images that users routinely share over emails. Consider such an image I. P splits I into n fragments $I = (I_1, \ldots, I_n)$ and creates a noisy version of each fragment I_i by adding a small amount of noise ϵ_i to it, i.e., $I'_i = I_i + \epsilon_i$. P then constructs the email body as before: $m = ((p_i)_{i=1}^{n_p}, (I_i, I'_i)_{i=1}^{n}, (s_i)_{i=1}^{n_s})$. The email body finally obtained by S would be $m' = ((p_i)_{i=1}^{n_p}, (I''_i)_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ where I'' is a composite image such that its ith fragment I''_i equals either I_i or I'_i. Since the noises ϵ_i are small, I'' is visually indistinguishable from I. Thus, I'' can be assumed to be drawn from the distribution $\mathcal{D}$ which achieves server obliviousness. Note, however, that P on obtaining m' can still reliably extract the challenge bits since it exactly knows both I and I'.

Size of email body. In a standard SMTP-TLS connection, the email body is fragmented into TLS records of size 16KB (if the email body is smaller than 16KB then it is not fragmented). Since our challenge is solely dependent on the ability to drop or forward an entire fragment, we require each challenge message to be in a fragment of its own. Thus, for a realistic security parameter $n = 80$, we would require the email body to be split into 160 fragments. This would require communication of size at least $160 \times 16\text{KB} \approx 2.5\text{MB}$ of data from P to V, which results in an email carrying 1.25MB of data due to selective forwarding by V. This is commensurate to the size of a typical email containing an image or a pdf file.

4.2 *Supporting nonce-based authenticated encryption schemes*

In our protocol, we encrypt each challenge message pair (c_{i0}, c_{i1}) using the same sequence number sqn so that V can drop one of the messages. However, this poses a problem in nonce-based authenticated encryption schemes such as AES-GCM used in TLS 1.2 and 1.3. When the same sequence number is used to encrypt two messages it results in the same nonce for both the encryptions (see the TLS.Enc algorithm in Figure 5 for AES-GCM based encryption within TLS). This violates the nonce uniqueness requirement [20] that prohibits using the same nonce to encrypt more than one plaintext while using the same key. Violation of this requirement may lead to key recoverability attacks as shown in [21] which in turn leads to violation of transcript privacy.

To prevent V from obtaining both the ciphertexts (ec_{i0}, ec_{i1}) created using the same nonce, we use an oblivious transfer (OT) protocol where V fetches ec_{ib_i} from P without learning the other ciphertext $ec_{i\bar{b}_i}$ and P does not learn the bit b_i chosen by V. V forwards to S the ciphertext ec_{ib_i} thus obtained. That is, instead of V obtaining ec_{ib_i} as per lines 18-22 of the GenToken protocol, it obtains them via the OT protocol as follows: $(P\langle\rangle, V\langle ec_{ib_i}\rangle) \leftarrow \mathsf{OT}(P\langle ec_{i0}, ec_{i1}\rangle, V\langle b_i\rangle)$. This mechanism ensures that V only learns a single ciphertext for any given nonce, thus respecting the nonce uniqueness requirement mentioned above and hence preserving transcript privacy. Also, P is oblivious to which ciphertext was fetched by V and then forwarded to S. This preserves the soundness of the protocol.

We note that OT is very fast; in fact, it is used as a primitive operation in many MPC protocols. We only need n OTs for the security parameter n, which is a small number in practice. Thus, the overhead of using OT in our protocol is negligible, as we show in Section 6.

```
$\mathsf{TLS.Enc}(k, m, sqn = \perp)$
  if $sqn == \perp$:
    let $sqn$ = current TLS sequence number
  let $type$ = TLS data type, $version$ = TLS version, $taglen$ = TLS tag length
  $data := m \| type$
  $outlen := \mathsf{size}(data) \| taglen$
  $authdata := (type \| version \| outlen)$
  calculate $nonce$ deterministically using $sqn$
  $ciphertext \leftarrow \mathsf{AES\text{-}GCM}(k, nonce, data, authdata)$
  $record := (type \| version \| \mathsf{size}(ciphertext) \| ciphertext)$
  return $record$
```

Figure 5: TLS.Enc algorithm for AES-GCM authenticated encryption scheme.

5 SECURITY ANALYSIS

Theorem 5.1. The protocols presented in Section 4 satisfy soundness as per Definition 3.1.

Proof (Sketch). In the basic YouChoose protocol presented in Section 4.1, P does not learn about which message out of the message pairs were forwarded by V because V sends them directly to S and by the requirements on SMTP (see Section 3.1), S does not echo any message back to P (note that S is honest in the soundness definition). Thus, the probability that P guesses the challenge bits $(b_1, \ldots, b_n)$ correctly and makes V output 1 during the Prove protocol is bounded by 2^{-n}.

In the variant designed for AES-GCM (Section 4.2), P does not learn which message of any message pair was forwarded by V since the ciphertext ec_{b_i} is obtained by V by running the OT protocol with P where P remains oblivious of the bit b_i. The rest of the argument follows similarly as above. □

Theorem 5.2. The protocols presented in Section 4 satisfy transcript privacy as per Definition 3.2.

Proof (Sketch). In the basic YouChoose protocol presented in Section 4.1, V only obtains ciphertexts $(ec_{i0}, ec_{i1})_{i=1}^{n}$ under a MAC-then-Encrypt or Encrypt-then-MAC scheme, which leak no information about the underlying messages, session keys, credentials, etc. In the variant designed for AES-GCM (Section 4.2), V only learns one ciphertext out of (ec_{i0}, ec_{i1}) for any i in 1 to n, by the properties of OT. In particular, V never learns two ciphertexts encrypted with the same nonce (corresponding to the same sequence number) for any i. This is indistinguishable to receiving a fresh ciphertext encrypted under AES-GCM for each i, which does not leak any information about the underlying messages or credentials. □

Theorem 5.3. The protocols presented in Section 4 satisfy transcript integrity with respect to a modification function $\mathcal{M}$ as per Definition 3.3, where $\mathcal{M}$ takes as input a message $m = ((p_i)_{i=1}^{n_p}, (c_{i0}, c_{i1})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ and outputs the set $\{((p_i)_{i=1}^{n_p}, (c_{ib_i})_{i=1}^{n}, (s_i)_{i=1}^{n_s}) \mid b_i \in \{0, 1\}\}$.

Proof (Sketch). As per the GenToken protocol, P authenticates TLS records containing repeated sequence numbers only for challenge message pairs (c_{i0}, c_{i1}) for $i = 1$ to n and otherwise follows strict sequence ordering. This restricts V to drop exactly one ciphertext ec_{ib_i} from each ciphertext pair (ec_{i0}, ec_{i1}) and sequentially forward all other TLS records, otherwise S will abort the connection as per the integrity guarantees of TLS. Thus, if S does not abort, it only obtains a message $m' \in \mathcal{M}(m)$. □

Theorem 5.4. The protocols presented in Section 4 satisfy server obliviousness with respect to $\mathcal{D}$ as per Definition 3.4, where $\mathcal{D}$ is the distribution defined in Section 4.

Proof (Sketch). If both P and V are honest then by the properties of TLS, S obtains a message $m' = ((p_i)_{i=1}^{n_p}, (c_{ib_i})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ for some $b_i \in \{0, 1\}$, where P sent the message $m = ((p_i)_{i=1}^{n_p}, (c_{i0}, c_{i1})_{i=1}^{n}, (s_i)_{i=1}^{n_s})$ to the TLS channel. If m was chosen by adding small noises to the images, then m' is indistinguishable from a message sampled from $\mathcal{D}$ because of the arguments mentioned in Section 4. Hence, we achieve server obliviousness. □

6 IMPLEMENTATION AND EVALUATION

To evaluate the feasibility and efficiency of YouChoose we implement a prototype. We develop our prototype in Python and use tlslite – ng [22] as our TLS library and use smtplib within tlslite – ng to send emails according to the SMTP specifications [23]. However, we customize the TLS library on the client's side to accept sequence numbers as parameters at authenticated encryption interfaces like `_encryptThenSeal` (AES-GCM and ChaCha20/Poly1305) and `_macThenEncrypt` (AES-CBC / HMAC-SHA256). These changes enable YouChoose to selectively discard specific TLS records by V, as detailed in Section 4. It is important to note that these changes are necessary only on the client's side and no changes are made on the server's side. We use Oblivious Transfers (OTs) to support nonce-based ciphersuites, utilizing the OTC library [24], developed by Chou-Orlandi [25]. We make our implementation open source on GitHub at https://github.com/aarav22/anon-pao.

Our prototype mainly consists of P's code for preparing and sending emails and V's code for dropping selected TLS records. We evaluate our prototype for different network configurations. We denote the configuration where P, V, and S are located on separate servers as L1, L2, L3. This configuration mimics real-world scenarios where S might be situated in a different region and/or operated by standard services like Outlook or Gmail. We denote the configuration where P and V are located on separate servers, while S is located on the same server as V as L1, L2, L2. This configuration is useful to minimize the effect of latency between V and S on the performance of YouChoose. In the L1, L1, L1 configuration, P, V, and S are all located on the same server. This configuration is useful for comparing the communication and computation overheads of YouChoose. Using an entirely local setup should minimize network latency and, consequently, the communication overhead between all the parties. To test L1, L2, L3 and L1, L2, L2, we deploy our programs on two separate Linode [26] nanode instances, each powered by a 1-core processor and 1GB of RAM.

We compare our prototype with the existing state of the art anonymous PAO, SCI. As a reminder from Section 3.1.3, SCI requires a unique design and implementation tailored to each TLS ciphersuite. At present, SCI designs implementations for both **Cipher I** and **Cipher II**, as described in Section 3.1.2. We developed SCI for **Cipher I** independently, as we lacked access to the authors' codebase. We implemented 2P-HMAC by adding interfaces in SHA256 module to extract partial states that can be exchanged between P and V. To implement 2P-CBC, we leveraged advancements in maliciously secure two-party computation, adopting a more recent and efficient protocol [27] that significantly reduces the processing time compared to the previously employed protocol [28]. We make this implementation open source as well.

We need an Oblivious Polynomial Evaluation (OPE) protocol to implement SCI for **Cipher II**. Wang et al. reference a now insecure OPE scheme [29]. Therefore, we do not report benchmarks for SCI with **Cipher II**.

We present our findings in Table 6, covering both ciphersuites. These results reflect the median time and standard deviation from five runs. Notably, the size of the email body varies in our tests, depending on the method used. For the baseline, we send a standard email with a minimal body size of 130 bytes. In the case of SCI, the email body extends to 192 bytes, with an additional 32 bytes stemming from M^*, as injected by V. With YouChoose, the email body reaches 2.5MB. However, it is important to note that only 1.25MB of this is actually transmitted to S, as detailed in Section 4.

Our results show that for **Cipher I**, YouChoose outperforms SCI, being at least 2x faster when sending an email to Outlook and 4.9x faster with a local SMTP server. From a broader perspective, when employing YouChoose with **Cipher I**, the time difference compared to baseline emails is minimal as it is only 0.35 seconds slower for a local SMTP server and 0.95 seconds slower for an Outlook server. This discrepancy becomes even less pronounced with **Cipher II**. In this scenario, YouChoose lags behind the baseline by only 0.33 seconds for a local SMTP server and 0.68 seconds for an Outlook server.

We find that the performance of YouChoose is primarily influenced by the communication overhead, which correlates with the size of the email body. However, it shows negligible computation overhead evident from the minimal time differences observed between the L1, L2, L3 configurations and the L1, L2, L2 configurations. Furthermore, our email body size is comparable to the size forced upon by SCI because the recommended way in SCI to hide challenges in the email body and achieve server obliviousness is to embed them in an image or media file, which are typically in MBs [1]. This comparison underscores that the larger email body size, a factor in YouChoose's communication overhead, is not an anomaly but rather a measure to achieve server obliviousness.

7 CONCLUSION AND FUTURE WORK

We formalized the security requirements for anonymous PAO and proposed YouChoose, a lightweight approach to meet these security guarantees. Our approach provides a simple way to achieve anonymous PAO without resorting to specially designed MPC protocols for different proofs of account ownership. We

Cipher	P, V, S	Baseline (s)	SCI (s)	YouChoose (s)
Cipher I	L1, L2, L3	0.37 (±0.03)	3.1 (±0.04)	1.32 (±0.02)
	L1, L2, L2	0.1 (±0)	2.21 (±0.17)	0.45 (±0.01)
	L1, L1, L1	0 (±0)	0.37 (±0.01)	0.24 (±0.01)
Cipher II	L1, L2, L3	0.6 (±0.25)	-	1.28 (±0.07)
	L1, L2, L2	0.1 (±0)	-	0.43 (±0.01)
	L1, L1, L1	0	-	0.26 (±0)

Figure 6: The median time and standard deviation (in parentheses) in seconds taken by SCI and YouChoose to send an email for anonymous PAO across 5 executions. The email-body size in YouChoose is set to 2.5MB, although only 1.25MB is finally sent. For SCI, the total email-body size including the challenge is 192 bytes. L1, L2, and L3 denote three different locations.

demonstrate our approach for email account ownership using SMTP over TLS. Specifically, we implemented the first anonymous PAO that works with all the ciphersuites of TLS 1.2 and 1.3. The overhead introduced in our anonymous PAO is not more than the time required to send a typical email containing an image file as an attachment.

Anonymous PAOs introduced by Wang et al.[1] are an important primitive to provide the necessary security and privacy guarantees for extremely sensitive uses cases such as whistleblowing. However, they present interesting opportunities in other use-cases as well. In this regard, we conjure that the following directions are worth exploring in the future:

1. **Anonymous PAO for other account types**. Although both our work and Wang et al.'s work has focused on email-based anonymous PAO, it appears that anonymous PAO for other account types such as social media accounts may have other interesting use-cases. For example, many Internet based service providers need to verify that the account owners on their website are genuine and for this purpose they trust social-media identity providers such as Google, Facebook and Twitter. However, the current technologies for this purpose (e.g., OAuth [30]) reveal to the identity providers the services availed by the user. Furthermore, tracking users is in the business interest of these identity providers. Anonymous PAOs fit in this setup perfectly as they allow the users to anonymously prove to the service providers that they own a genuine social media account without even requiring any co-operation from the social media account providers. For such a use-case, an MPC-based SCI approach is likely to be expensive and cumbersome to implement. In contrast to the MPC-based SCI protocol in email-based account ownership that exploits the fixed format of the SMTP protocol messages, the format of messages exchanged in these social media account APIs is likely to be more complex and dynamic. The YouChoose approach of selectively forwarding messages does not require any MPC protocol and is thus likely to be a more suitable fit.
2. **Shared email hosting**. We now point out a limitation of YouChoose and SCI's paradigm of email-based anonymous PAO where V sends a challenge to S and considers a reproduction of this challenge by P as a proof that P owns an email account at the domain of S. For example consider a common case where the organization controlling the domain `domain.com` outsources its email

services to third party providers such as Gmail (Google Workspace [31]). In this case, P, an employee at the organization wants to prove to V that it owns an email address of the form ____@domain.com but the outsourced mail server S provides its services to other organizations too, say domain2.com. Now, when V forwards its challenge to S, and this challenge is reproduced by P, V is not convinced that P owns an account at domain.com or domain2.com. Such use-cases are completely beyond the general paradigm of both YouChoose and SCI. For this, one could leverage a modified version of the "parse-and-extract" protocol from recent research on zero-knowledge middleboxes (ZKMBs [32]). This will prove to V in zero-knowledge that P's email address (used in the SMTP AUTH step) ends with domain.com. We leave this thread open for future work.

REFERENCES

[1] Liang Wang, Gilad Asharov, Rafael Pass, Thomas Ristenpart, and Abhi Shelat. Blind certificate authorities. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019.

[2] Adam Langley, Wan-Teh Chang, Nikos Mavrogiannopoulos, Joachim Strombergson, and Simon Josefsson. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905, June 2016. URL https://www.rfc-editor.org/info/rfc7905.

[3] James Dolan Freedom of the Press Foundation, Aaron Swartz. Securedrop. Open-source software. URL https://securedrop.org/.

[4] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol, 2017.

[5] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. In Dimitris Gritzalis, editor, *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, pages 211–219. Springer, 2003. doi: 10.1007/978-1-4615-0239-5_14. URL https://doi.org/10.1007/978-1-4615-0239-5_14.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association. URL https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[7] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pages 257–265, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. ISBN 978-3-540-46416-7.

[8] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 552–565, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45682-7.

[9] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 21–30, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581136129. doi: 10.1145/586110.586114. URL https://doi.org/10.1145/586110.586114.

[10] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby x.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 235–254, 2016. doi: 10.1109/SP.2016.22.

[11] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013. URL https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revis

[12] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. *SIGCOMM Comput. Commun. Rev.*, 45(4):199–212, aug 2015. ISSN 0146-4833. doi: 10.1145/2829988.2787482. URL https://doi.org/10.1145/2829988.2787482.

[13] Shuai Li and Nicholas Hopper. Mailet: Instant social networking under censorship. *Proceedings on Privacy Enhancing Technologies*, 2016(2), April 2016.

[14] Fan Zhang, Sai Krishna Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: liberating web data using decentralized oracles for TLS. *CoRR*, abs/1909.00938, 2019. URL http://arxiv.org/abs/1909.00938.

[15] Paul E. Hoffman. SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207, February 2002. URL https://www.rfc-editor.org/info/rfc3207.

[16] Nadhem J. Al Fardan and Kenneth G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013. doi: 10.1109/SP.2013.42.

[17] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bites: Exploiting the ssl 3.0 fallback. OpenSSL, 2014. URL https://www.openssl.org/~bodo/ssl-poodle.pdf.

[18] Thai Duong and Juliano Rizzo. Beast. *Online]. Tersedia: https://vnhacker. blogspot. co. id/2011/09/beast. html*, 2011.

[19] David McGrew, John Foley, and Kenny Paterson. Authenticated Encryption with AES-CBC and HMAC-SHA. Internet-Draft draft-mcgrew-aead-aes-cbc-hmac-sha2-05, Internet Engineering Task Force, July 2014. URL https://datatracker.ietf.org/doc/draft-mcgrew-aead-aes-cbc-hmac-sha2/05/. Work in Progress.

[20] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites

for TLS. RFC 5288, August 2008. URL https://www.rfc-editor.org/info/rfc5288.

[21] Antoine Joux. Authentication failures in NIST version of GCM. Online, 2012. URL http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf.

[22] Hubert Kario. tlslite-ng. https://github.com/tlsfuzzer/tlslite-ng.

[23] J. Klensin. Simple mail transfer protocol. *RFC*, 10:17487, October 2008. URL https://www.rfc-editor.org/info/rfc5321>.

[24] nthparty. Otc. Open-source software. URL https://github.com/nthparty/otc.

[25] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. Cryptology ePrint Archive, Paper 2015/267, 2015. URL https://eprint.iacr.org/2015/267. https://eprint.iacr.org/2015/267.

[26] LLC Linode. Linode - cloud hosting & linux servers, 2023. URL https://www.linode.com/. Accessed: 2023-12-22.

[27] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. Cryptology ePrint Archive, Paper 2017/030, 2017. URL https://eprint.iacr.org/2017/030. https://eprint.iacr.org/2017/030.

[28] Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 297–314, USA, 2016. USENIX Association. ISBN 9781931971324.

[29] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. Cryptology ePrint Archive, Paper 2017/409, 2017. URL https://eprint.iacr.org/2017/409. https://eprint.iacr.org/2017/409.

[30] OpenID Foundation. Openid connect core 1.0 incorporating errata set 1. OpenID Connect Core 1.0, 2014. URL https://openid.net/specs/openid-connect-core-1_0.html.

[31] Google. Google Workspace. Software suite. URL https://workspace.google.com/.

[32] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. Zero-knowledge middleboxes. Cryptology ePrint Archive, Paper 2021/1022, 2021. URL https://eprint.iacr.org/2021/1022. https://eprint.iacr.org/2021/1022.