Cryptographic Personas: Responsible Pseudonyms Without De-Anonymization

Rachel Thomas*
University of Maryland
Email: rthomase@umd.edu

Oliwia Kempinski*
University of Maryland
Email: okempins@umd.edu

Hari Kailad
University of Maryland
Email: harikeshkailad@gmail.com

Emma Margaret Shroyer
University of Maryland
Email: eshroyer@umd.edu

Ian Miers†
University of Maryland
Email: imiers@umd.edu

Gabriel Kaptchuk†
University of Maryland
Email: kaptchuk@umd.edu

***Abstract*—We present cryptographic personas, an approach for facilitating access to pseudonymous speech within a community without enabling abuse. In systems equipped with cryptographic personas, users are able to authenticate to the service provider under new, unlinkable personas at will and post messages under those personas. When users violate community norms, their ability to post anonymously can be revoked.**

Cryptographic personas draws on techniques from recent work, zk-promises. We develop two significant improvements over existing work that make it possible to integrate cryptographic personas into real-time applications like group messaging: we show how to push expensive proof generation into an offline phase and find a way to optimize server-side overhead using recent proof folding techniques. We implement cryptographic personas, integrating them into Signal and Slack, and show that they are concretely efficient enough for deployment.

I. Introduction

Balancing anonymity and moderation is a common research challenge. But digital anonymity is often discussed in extremes. Images of mask-clad political activists are conjured as justifications for free speech, and stories of malicious trolls are used to illustrate the specter of unmoderated systems. These characterizations are for *completely anonymous* systems, in which an anonymous poster could truly be anyone and the audience is everyone. However, anonymity is often wielded in the subtler contexts of an established community: a whistle blower releases a series of posts under a pseudonym, anonymous Wikipedia editors provide high quality edits on sensitive topics [1], an anonymous poster deliberately reveals parts of their identity in order to build credibility, and founders of a nascent democracy discuss under pseudonyms [2], [3]. Recent work, zk-promises [4], offers a programmable framework that could realize more nuance. In this paper, we consider this approach as applied to group chats, considering both the additional cryptographic machinery needed and the significant performance improvements needed to make it practical.

Personas for group chats. Even in small group chats, anonymity enables freer expression. While participants may have contextual clues as to who wrote "the emperor has no clothes," the lack of clear attribution creates space for productive dissent in two distinct ways: (1) there may still be doubt among participants as to who authored a message, and (2) if chat logs are later presented to a third party, these logs will not identify which member of the group authored the message. Of course, this anonymity can be abused or simply misused in ways that make an anonymous channel unproductive, robbing others of the benefits of participating.

Balancing the dual nature of anonymity—it facilitates honest, productive communication and also invites abuse—requires pushing users to treat the power to post anonymously responsibly. The most natural route to avoiding disruptive speech is to simply unmask abusive users, i.e., implement a system with group signatures [5], [6] rather than ring signatures [7] (or linkable ring signatures for pseudonymity [8]). But this approach is still one of extremes—jumping from anonymous to non-anonymous under revocation—and it undermines the social utility of anonymity or pseudonymity: if users are nervous that their anonymity might be retroactively revoked, they may not be able to fully take advantage of its pro-social effects, even if they have nothing to hide. Users could be de-anonymized for reasons beyond the content that they post, e.g., changes in business priorities, compulsion by law enforcement, or as a result of a data breach. When anonymity measures fail, consequences can be dire—or even lethal [9]. Finally, anonymity and the ability to speak truth to power are especially socially valuable in settings where there is not significant trust in platforms.

One practical approach has emerged, originally in non-private systems like Reddit: users generate throw-away accounts on open platforms in order to speak pseudonymously (e.g., a corporate executive discussing the impact of policy changes [10]) and moderation teams ban these accounts when they transgress communal norms. But precisely because these accounts are brand new and not linked to long-term identity, abusive users can return to the platform under new pseudonyms—and it is difficult for the platform to distinguish between a credible new user, a previously banned user appearing under a new name, or a spammer.

In this paper, we explore a different approach: *selectively* revoking the ability of abusive users to post anonymously in the *future* (or, possibly, decrementing their reputation to limit further posts) and deleting the content deemed abusive,

with the goal of eventually creating a culture in which users treat anonymity as a privilege. Such an approach assumes users have an immutable, long-term identity, preventing them from sidestepping revocation by making a new account. The challenge, therefore, is to build a system that simultaneously allows for pseudonymous posting, enables revoking abusive users' ability to act pseudonymously, manages dynamic reputation scores, and holds long-term identity in order to prevent circumventing revocation—all without relying on a server that has the technical capability to unmask the identities of pseudonymous posters. To accomplish this, we build on cryptographic primitives for anonymous banning systems [11], [12], [13], [4] and explore how to leverage these techniques within the context of realistic deployment infrastructure in a performant, scalable way.

Cryptographic personas. We propose *cryptographic personas*, a trustless approach to balancing anonymity/pseudonymity and accountability. In systems equipped with cryptographic personas, users are able to authenticate to the server under new, unlinkable personas at will and post messages under those personas, providing the user with anonymity. A user's personas are cryptographically connected to one another (although not in a way that can be traced by the server or other parties) such that sanctions imposed on any persona will also be applied to *all* of that user's personas.[1]

We anticipate that cryptographic personas will be most valuable when used *within community*, where there is a reasonable expectation of good-faith engagement (i.e., revocation is expected to be a somewhat rare event). Giving members of a community access to cryptographic personas can help cut through internal power dynamics that might otherwise hamper earnest communication. Consider, for example, a junior faculty member who wants to share frank and genuine suggestions on ways to reform and improve a department with colleagues, but fears that voicing contradictory opinions might negatively impact their chances of promotion and/or tenure. Giving this faculty member access to a cryptographic persona would allow them to speak their mind, but it would still give group members a tool to prevent disruptive posters.

Technical barriers to concretely deployable cryptographic personas. Recent work on anonymous banning systems [4] offers a general-purpose, programmable tool that serves as a feasibility result for cryptographic personas. There are two main technical barriers that must be overcome in order to build cryptographic personas as a real-world deployable primitive: (1) we lack a concrete notion of what pseudonymous but revocable speech could look like in practice, i.e., what features are needed in order to make pseudonymous interactions productive. Technically, this means we require definitions that surface these features, making cryptographic personas socially valuable. The definitions in prior work prioritize generality at the cost of specificity and clarity. An added benefit of formalisms for cryptographic personas is that they can function as a drop-in primitive into new contexts; and (2) new techniques are required to lift existing feasibility results into concretely performant systems that meet deployment constraints—including supporting users working on legacy hardware. Prior work has two primary performance limitations: (a) they require computing non-trivial zkSNARKs in-line with user interaction, resulting in noticeable, disruptive delay; and (b) the computational overhead of processing bans in prior work [4] was prohibitive, introducing many client-server round-trips and high concrete costs. Under some configurations, if a user had previously sent n messages on the system, the server would need to do $O(n^2)$ work to help the client demonstrate that it was up to date with all issued bans. We overcome each of these barriers in this work, resulting in deployable cryptographic personas.

Cryptographic personas and identity. Cryptographic personas are only valuable when integrated into a system with some underlying notion of identity that at least prevents the creation of Sybil accounts, since revocation is worthless without such limitations. One obvious solution is to compose a cryptographic persona with an anonymous credential derived from legal identity documents or some other Sybil resistance mechanism [15], [16], [17]. We leave the selection of the appropriate Sybil-resistant mechanism to future work or deployers.

Contributions. In this work, our main contributions are:

- We formally define *cryptographic personas*, a novel approach to enabling anonymous and pseudonymous speech on digital platforms without letting abusive behavior flourish. Cryptographic personas allow users to generate fresh, unlinkable personas for messaging, but these personas are cryptographically connected such that punishments applied to any one persona automatically propagate to all of a user's personas. We also develop a set of features on top of this base functionality to support usable, pseudonymous interaction. We show that cryptographic personas can be instantiated using zk-promises [4], including modeling the registration process missing from [4].
- We develop a way to push the expensive proof generation step required during the authentication in Shih et al.'s [4] construction into a pre-processing step without compromising user privacy. Our technique builds on Privacy Pass [18] to issue users' authentication tickets that can be redeemed to send messages. Concretely, this reduces the client work it takes to send pseudonymous messages by more than 99%, making it possible for all users—even those using older hardware—to use cryptographic personas.
- We show how to integrate modern proof folding techniques [19], [20], [21], [22], [23] into the construction of Shih et al. [4] in order to significantly reduce the server-side overhead of processing revocations. Concretely, if users have 100 messages that could be used to revoke their access to personas, we find that our approach reduces server-side proof verification by up to $\approx 95\%$ relative to [4].

[1] To our knowledge, [14] was the first project to propose combining pseudonyms with anonymous credentials. Our work takes this observation and explores the rich set of features that are necessary in order to make pseudonyms (which we call personas) valuable in real-world contexts.

– We implement cryptographic personas and integrate our implementation with both Slack and Signal, facilitating the generation of cryptographic personas within group chats. We provide detailed benchmarks demonstrating that cryptographic personas are usably efficient.

II. Overview and Use Cases

When cryptographic personas are used in their basic form, we imagine that users can simply spin up—and post under—new personas for each message. For example, a new PhD student joining a department might want to ask an embarrassing question in a large PhD student group chat under a one-time-use persona. Alternatively, personas might have long-term utility. For example, a user might ask for advice (e.g., imagine a closed system that functions like Reddit's "r/AmItheAsshole" [24]) and want to add clarifications in response to questions. When a user demonstrates that they cannot be trusted to use the power of anonymity responsibly, their ability to use cryptographic personas can be *selectively* revoked. Importantly, this is subtly different than unmasking the author of the abusive content—here, we suggest that the author remains anonymous, but simply loses the ability to post messages in the future anonymously (they may still post messages under their long-term identity). Determining how cryptographic personas can be used within a community—and when to revoke a user's access to cryptographic personas—is ultimately a question of community norms and governance.

A. Technical Sketch

Implementing personas as a user-interface feature (i.e., with no cryptographic guarantees) is trivial as long as the platform is capable of maintaining a map from personas to long-term identities. This approach, however, comes with two downsides: (1) users must trust that the platform will not reveal the long-term identities associated with personas or experience a data breach that reveals the mapping. When the stakes of maintaining anonymity are high, requiring this level of trust may fundamentally undermine the social benefits of cryptographic personas altogether; and (2) some systems are intentionally designed such that no party can maintain such a mapping, making this approach impossible. For example, consider an academic department in which discussions of department affairs regularly occur on Signal (either formally or through informal group chats). Signal uses Sealed Sender [25] to hide the message sender's identity from the server and implements a private group protocol that prevents the server from learning the members of groups. To work in these contexts, we require a technical approach that cryptographically ensures anonymity.

We build cryptographic personas from the powerful zk-promises primitive introduced by Shih et al. [4], as existing anonymous signature variants [7], [8], [5], [6] do not surface the right type of revocation.[2] zk-promises can be seen as a *stateful* anonymous credential [27] that allows for callbacks—that is, the server can force the holder of the anonymous credential to update some private state before they authenticate (anonymously) to the server again in the future. Because the user cannot roll their state back, it acts as oblivious storage for the server about which the user can prove predicates each time they authenticate. We summarize zk-promises in Section III.

Naïvely constructing cryptographic personas from this primitive is relatively direct: the user's oblivious state holds a Boolean value that indicates if the user has had their power revoked. Whenever the user wants to post a message anonymously, they authenticate to the server and prove that their state has this Boolean set to 0. If a user posts a message that results in revocation, the server forces the user to update this Boolean to 1, preventing future authentications. Generating reusable pseudonyms simply requires having each user keep a secret key in their state and proving that the cryptographic hash of the secret key and some chosen nonce is the desired pseudonym. Making this construction practical enough to work at scale, however, requires non-trivial modifications.

System Architecture. Thus far, we have described cryptographic personas in the context of a client–server model. In practice, however, there are three distinct deployment modes. A key design goal across all modes is that platform code need not be explicitly aware of cryptographic personas: persona validation can be layered on top of existing services with minimal changes, enabling easy integration into deployed systems. When a platform does wish to reason explicitly about personas (e.g., for moderation or reputation), this functionality can be factored out into an external validation or reputation manager, minimizing the required refactoring. With this goal in mind, we consider the following deployment models:

(1) *Cryptographic personas as infrastructure:* When services want to integrate personas directly into their own infrastructure, they can create a validation server that checks if a pseudonymous client request (e.g., add a message to a database via an HTTP POST) is allowed using the cryptographic personas protocol. The validation server can then strip off the relevant cryptographic material and forward the request to the service's main request endpoints. Importantly, the services' main API endpoints can remain largely unchanged, making deployment simple. We illustrate a messaging platform using this mode on the left-hand side of Figure 1.

(2) *Cryptographic personas as a service:* The validation described above could instead be run by a third party, copying the widely-deployed Single Sign On paradigm. In this paradigm, the third party is trusted to correctly verify cryptographic proofs and maintain state, but need not be trusted by clients to maintain privacy (as privacy is guaranteed by the cryptographic personas protocol). One upshot of this approach is that pseudonyms can, in principle, be reused across multiple platforms when appropriate. We do not directly illustrate this mode, as its architecture is largely captured by the "Cryptographic

[2]"Revocable" ring signatures in the literature (e.g., [26]) consider a type of revocation that is fundamentally different than the one considered here. Namely, these works allow for the anonymity of the signer to be revoked, rather than that signer's ability to produce more valid ring signatures.

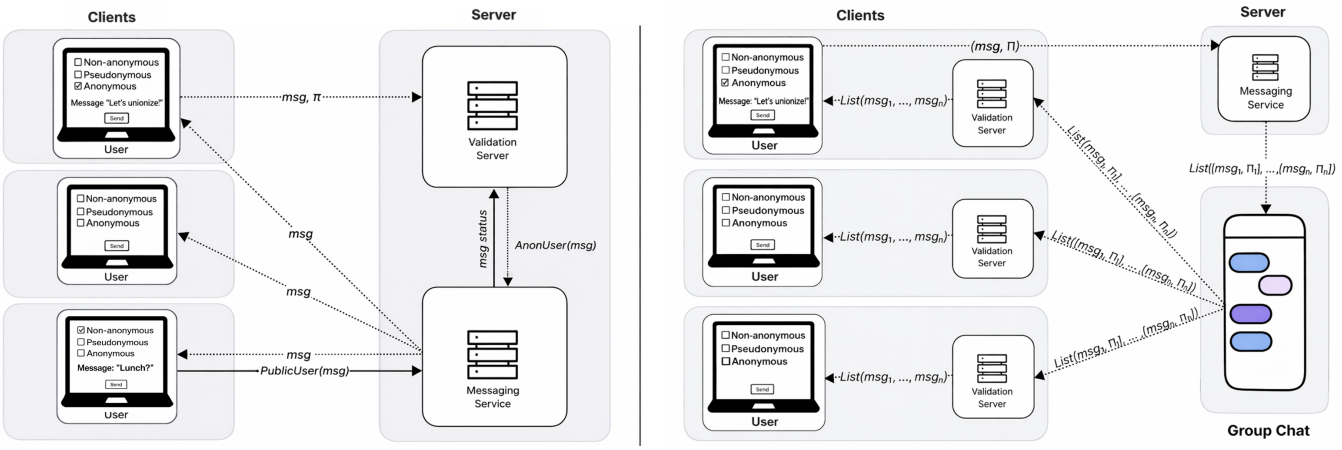


Fig. 1: System architectures for deploying cryptographic personas in a group messaging platform. Left: *cryptographic personas as infrastructure*, where users prove access to a pseudonym to the validation server using π, which then forwards the message msg to the messaging service; the depicted server may also be run by a third party (*cryptographic personas as a service*). Right: *serverless cryptographic personas*, where users run the validation logic locally and the group chat stores message–proof tuples $(msg_1, \pi_1), \ldots, (msg_n, \pi_n)$; clients download this state, verify proofs, and display messages msg_i with valid π_i. Solid arrows denote standard message delivery, and dotted arrows denote additional anonymous/pseudonymous protocol steps.

personas as infrastructure" schematic.

(3) *Serverless cryptographic personas:* Finally, cryptographic personas can be run in a "serverless" mode, wherein each client performs the duties of the validation server locally. While the server can be designed to be completely agnostic to cryptographic personas, this requires modifying clients to generate proofs for messages they send, validate those they receive, and maintain a consistent shared view of state across all users. We illustrate this deployment architecture on the right-hand side of Figure 1.

In all three of these modes, the server must provide an interface for sending messages that are not tied directly to an account. The most obvious way to do this is to add such an API but limit access to specific callers (e.g., the validation server). Alternatively, all cryptographic personas can simply share a single account from which all persona messages will be sent. The authentication material for this account can be either managed by the validation server (who would be responsible for annotating each message with the appropriate pseudonym) or simply held by all users when cryptographic personas is deployed in the serverless mode.

New constructions for real-time performance. Realizing deployable cryptographic personas requires overcoming two barriers: (1) We must streamline the authentication interaction between users and the service provider when sending a message. In our experiments (see Section VI), we find that simply generating the zero-knowledge proof necessary for authentication can take nearly half a second on lower-end consumer hardware. This is an unacceptable delay to introduce in-line with each client-server interaction; and (2) In zk-promises, clients need to demonstrate to the validation server that their local state is "up-to-date" each time they want to send a message. This is done by scanning through all potential open callbacks (e.g., previous interactions that could result in banning) and proving that their local state has properly ingested any invocations (e.g., bans). In our experiments (see Section VI), we find that this process is prohibitively expensive (both for clients and servers). We explore the following two protocol improvements to make zk-promises, each of which we show is critical to making real-world deployment possible in Section VI:

Pushing authentication into pre-processing. The natural way to streamline authentication is to push this work into a pre-processing phase, which can then be efficiently consumed during online authentication. We note, however, that naïve approaches to pre-processing, in which clients locally compute batches of zero-knowledge proofs fails to provide the desired functionality, as this would require the client to pre-commit to the order list of interactions they want to later execute (recall that zk-promises [4] enforces a set of sequential state updates and intentionally does not permit forking). Similarly, performing a naïve interactive authentication for a batch of authentication tickets that can later be redeemed fails to provide the necessary privacy requirements, as all these tickets would be linkable to one another.

We observe that the authentication for zk-promises [4] can be pushed to a pre-processing phase in which a batch of tickets is issued unlinkably using techniques developed in Privacy Pass [18]. Privacy Pass allows clients to authenticate to a server during downtime and receive a set of unlinkable one-time tickets that can be later redeemed in lieu of authentication. In our setting, users generate a single zero-knowledge proof for a batch of tickets (e.g., at the beginning of each day) that can later be redeemed in any order to enable the client to post messages later, making sending many messages or making

many posts sequentially feasible. To ensure that tickets are unlinkable, the client proves that their local state is correctly formatted with respect to *a set of* callback identifiers, each of which is only revealed during redemption.

Folding update proofs. In zk-promises, clients loop through the callback handles associated with prior messages (stored in their local state), and prove that all invoked callbacks have been appropriately applied to their local state. To ensure that this process does not leak the identity of the client, Shih et al. consider: (a) padding out the circuit to have some maximum number of allowed callbacks, resulting in worst-case statements; and (b) users incrementally iterate through the list of prior callback handles, interacting with the server for each element in this list (or fixed-size batches). The latter approach results in quadratic load on the server (each message requires an interaction *for each previous message*) and requires an always online server (which is not available in *serverless* deployments); supporting such a deployment scenario demands a new approach with *one-shot* verification.

The structural problem here is that the number of callbacks is unknown to the server and must stay secret to preserve anonymity. Thus, the "right" tool is a proof system that can accommodate variable-length statements without requiring worst-case complexity, i.e., recently developed proof folding techniques [19], [20], [21], [22], [23]. Folding allows users to prove a statement about each element in the list separately and fold these proofs together. The result is that the server only needs to do constant work per message—just like in standard applications. Interestingly, we also observe that this setting may be appropriate for the "folklore" approach for making folding proofs zero-knowledge: folding in a randomized, satisfied instance. This saves the expensive generation of a succinct, zero-knowledge proof (proving that the folding proof verifies) at the expense of true succinctness. To our knowledge, our work is the first time a suitable application setting has been identified for this folklore approach and concretely evaluated.

B. Use Cases

Departmental communication. Academic departments rely on collaboration and deliberation among faculty members with differing perspectives and priorities to make critical decisions on curricula, hiring, promotion, and resource allocation. At the same time, academic departments are highly hierarchical, with faculty occupying different levels of formal seniority (some of whom are responsible for making promotion recommendations for others) and informal social influence. This can be a significant barrier to honest collaboration and deliberation, as junior faculty might perceive expressing disagreement with the status quo as risky for their reputation among their colleagues. If departments use digital communication platforms to collaborate (e.g., Slack or WhatsApp groups), cryptographic personas can provide faculty with an opportunity to express honest opinions—even on contentious issues—without risking reputational damage. As we explore below, cryptographic personas can also be used to run anonymous, one-ballot-per-user polls, allowing users to quickly gauge faculty opinions. The accountability features of cryptographic personas—possibly managed by the department chair or democratically—ensure that posts stay professional.

Q&A platforms. As we discuss in Section VII, anonymous posting results in more honest and eager engagement by users on Q&A platforms [28], [29], [30], [31], [32], but comes with the risk of trolling and harassment [33], [34], [35]. One example may be a Q&A Slack channel for all PhD students in a department, where junior students may hesitate to ask valuable questions for fear of appearing foolish. Cryptographic personas allow question-askers and answerers to post anonymously. Both can be limited to one persona per question to prevent a single answerer from giving the illusion of consensus. Cryptographic personas could also enable "meronymous" communication [32], wherein posters reveal they belong to a subset of users (e.g., a topic expert). Access to cryptographic personas could be governed by administrators or topic-specific moderators, a strategy successful on many discussion platforms [36], [37], [38], [39].

Support and activism communities. Many communities involve sensitive participation where users face social, professional, or physical risks for speaking openly. Examples range from support groups to activist coordination, including large, semi-open messaging groups used in recent protest movements [40]. In these settings, users benefit from adopting multiple pseudonyms across contexts while maintaining a unified reputation that enables trust-building and moderation. Cryptographic personas allow participants to assume different personas for sharing experiences, offering support, or coordinating actions, while sanctions propagate across all personas when abuse occurs. These communities often require role differentiation, without revealing identities. Cryptographic personas support role-based disclosure through badge-like annotations that prove membership in a role (e.g., leadership) without identifying the speaker. This minimizes participants' digital footprints while preserving coordination capacity, even under adversarial conditions.

III. Preliminaries

Notation. Our construction makes use of a collision-resistant and pre-image resistant hash function $\mathcal{H}$. We denote an algorithm Algo that must read from the bulletin board as $\mathsf{Algo}^{\mathsf{BB}}$. Note that in serverless deployments of cryptographic personas, the bulletin board is implicit, as all participants share a synchronized view of the state. We apply mild semantic updates to the interfaces to differentiate clearly between the local state held by clients (obj) and the commitments to them that are made public $\mathsf{com}_{\mathsf{obj}}$, which are not disambiguated clearly in [4].

zk-promises [4]. zk-promises is an anonymous callback system constituting the following tuple of algorithms:

- $\textsc{Setup}(\Phi) \to \mathsf{pp}$ takes in a set of state transition functions Φ and produces public parameters pp.

– EXECMETHODANDCREATECALL$(\mathsf{pp}, (\mathsf{com}_{\mathsf{obj}}, \mathsf{obj}), \mathsf{pk}_{\mathsf{sp}}, \mathsf{meth}, x, w) \rightarrow ((\mathsf{com}_{\mathsf{obj}'}, \mathsf{obj}'), \pi, \mathsf{cbData}, \mathsf{aux})$[3] takes in the public parameters pp, the user's current private state obj and the corresponding commitment to that state $\mathsf{com}_{\mathsf{obj}}$, the service provider's public key $\mathsf{pk}_{\mathsf{sp}}$, a function meth to be applied to the user's private state, public input x to that function, and auxiliary witness data w and outputs an updated private state obj' and corresponding commitment $\mathsf{com}_{\mathsf{obj}'}$, a proof of validity π, callback handle $\mathsf{cbData} = \{(\mathsf{tik}, \mathsf{meth}), \mathsf{obj.sn}\}$ containing at minimum a callback ticket tik, the method associated with the generated callback, and the serial number obj.sn, and auxiliary data aux. After running this function, the client must post cbData onto the bulletin board BB.
– VERIFYCREATE$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{sk}_{\mathsf{sp}}, \mathsf{com}_{\mathsf{obj}'}, \pi, \mathsf{cbData}, \mathsf{aux}) \rightarrow \{0, 1\}$ has oracle access to the bulletin board BB, takes in public parameters pp, the service provider's secret key $\mathsf{sk}_{\mathsf{sp}}$, a commitment to the users updated private state $\mathsf{com}_{\mathsf{obj}'}$, a proof of validity π, callback handle cbData, and auxiliary data aux and outputs a binary value.
– CALL$(\mathsf{pp}, \mathsf{tik}, \mathsf{args}) \rightarrow (\mathsf{tik}, \mathsf{args})$ takes in the public parameters pp, a ticket tik, and arguments to be passed into the callback args and outputs these last two inputs. To complete the act of calling, the outputs should be posted onto the bulletin board BB.
– VERIFYCALL$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{tik}, \mathsf{args}, \mathsf{aux}) \rightarrow \{0, 1\}$ has oracle access to the bulletin board BB, takes in the public parameters pp, a ticket tik, and arguments to be passed into the callback args and auxiliary data aux and outputs a binary value.
– SCANONE$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{obj}, x) \rightarrow ((\mathsf{com}_{\mathsf{obj}'}, \mathsf{obj}'), \pi, \mathsf{cbData})$ has oracle access to the bulletin board BB, takes in the public parameters pp, the user's current private state obj, public input x and outputs updated user private state obj' and a corresponding commitment $\mathsf{com}_{\mathsf{obj}'}$, update proof π, and callback handle $\mathsf{cbData} = \{(\mathsf{tik}, \mathsf{meth}), \mathsf{obj.sn}\}$.
– VERIFYMETHODEXEC$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{com}_{\mathsf{obj}'}, , \pi, \mathsf{cbData}) \rightarrow \{0, 1\}$ has oracle access to the bulletin board BB, takes in the public parameters pp, a commitment to the updated user private state $\mathsf{com}_{\mathsf{obj}'}$,, update proof π, and callback handle $\mathsf{cbData} = \{(\mathsf{tik}, \mathsf{meth}), \mathsf{obj.sn}\}$ and outputs a binary value.

At a high level, these algorithms are interleaved as follows: After SETUP has been called, users can run EXECMETHODANDCREATECALL to generate a proof π and the callback data cbData, which they post to the bulletin board (which serves as shared, synchronized storage). π can reason over obj, depending on the statement proved, and can demonstrate that they are authorized to make a request. This data is then sent to the server provider along with the request. The service provider verifies requests using VERIFYCREATE and may invoke callbacks by posting CALL records to the bulletin board. The algorithm VERIFYCALL checks if the invocation of CALL is authorized. The user ingests the calls on the bulletin board by calling SCANONE on each callback handle stored in obj. This produces a proof which can be verified using VERIFYMETHODEXEC. In practice, callbacks are batched into *epochs*, allowing users to demonstrate that they are up to date with respect to all active callbacks without remaining in one-to-one lockstep with the server. As we discuss below, this provides an efficiency-liveness trade-off opportunity. Specifically, users can process each epoch together (potentially resulting in better performance), but callbacks will only be processed once the epoch has been updated. These algorithms enable third-party modification of private client state while preserving integrity, confidentiality, and unlinkability via zero-knowledge proofs. Shih et al. formalize the properties provided by these algorithms in the real/ideal world paradigm; we refer readers to the original work [4].

We note that Shih et al. don't specify a registration process. Registration actually requires care to ensure that two adversarial clients cannot clone each other's private state (e.g., after it has been revoked). We show how to do this in Section IV-C.

Privacy Pass [18]. Privacy Pass is a protocol initially designed to help users prove their humanity on the internet in a user-friendly way without compromising on privacy. Privacy pass has two phases: (1) ticket issuance and (2) ticket redemption. During issuance, the client obliviously queries a pseudorandom function at chosen points and during redemption demonstrates knowledge of proper evaluations.

IV. CRYPTOGRAPHIC PERSONAS

A. Threat Model

We consider a set of clients sending pseudonymous messages to one another. To simplify this communication process (in the *as infrastructure* and *as a service* settings), we also consider a service provider that locally runs a validation server or relies on a third-party validation server. The adversary is capable of (statically) corrupting an arbitrary subset of these clients and both the service provider and the validation server, and having them arbitrarily deviate from the protocol. We implicitly assume that all network communication is anonymous, but allow the adversary to control if and when messages are delivered in the system; because the adversary can corrupt the server, we do not assume that communication channels are private. Finally, we assume that there is some external mechanism to prevent Sybil attacks during registration; this can be done with any robust authentication mechanism or, for smaller-scale applications like departmental communication, simply by assuming the clients all know each other.

Our primary security goals are *anonymity*, *unlinkability*, and *revocability*. Informally, anonymity requires that the adversary cannot determine which honest user generated a given persona or message beyond the contents of the message or timing. Unlinkability requires that multiple personas or actions generated by the same honest user cannot be linked to each other, even across sessions or protocol executions (again, up to message contents or timing). Revocability ensures that once a user is revoked, they lose access to all pseudonymous

[3] Shih et al.'s [4] initial algorithmic interfaces do not expose auxiliary witness data w that is not held in obj. We require this interface, and making this change to their construction is trivial.

$\mathcal{F}_{\text{CryptographicPersona}}$

$\mathcal{F}_{\text{CryptographicPersona}}$ is parameterized by two *stateful* functions, AllowedToCreateNewPersona and AllowedToRevoke, both of which have output space $\{\texttt{true}, \texttt{false}\}$; we allow the state to be passed between invocations of the functions implicitly and reason over the internal state of the ideal functionality without passing it as input. For each of the n parties $P_1, \ldots, P_n$ initialize $P_i.\texttt{revoked} = \texttt{false}$. Set $\texttt{time} = 0$. Let RegNotificationSet be a subset of parties.

Register: Upon input (Register) from P:

1. Send (RegisterRequest, P) to Sim. When (RegisterRequestApprove, P) is received from Sim, continue.
2. Add pending messages of the form (Registered, P) for all parties in RegNotificationSet.
3. Send (Registered) to P.

Send Message with New Persona: Upon input (SendMessageNewPersona, m, context, recipients) from P:

1. If $P.\texttt{revoked} = \texttt{true}$, return.
2. If AllowedToCreateNewPersona(P, context) $= \texttt{false}$, return.
3. If P is corrupt, send (PersonaHandleRequest, m, recipients) to Sim and receive (PersonaHandleResponse, persona) in response. Otherwise, sample a new unique persona handle persona. If persona already exists in record, abort. Else, record that (persona, context) is associated with P and the current time time.
4. For each party $P_i \in$ recipients, create a pending message of the form (persona, m, context, time) and sends this message to Sim.

Send Message with Existing Persona: Upon input (SendMessageExistingPersona, m, context, recipients, persona) from P:

1. If $P.\texttt{revoked} = \texttt{true}$, return. If (persona, context) is not associated with P, return.
2. Update the time associated with (persona, context) to be time.
3. For each party $P_i \in$ recipients, create a pending message of the form (persona, m, context, time) and sends this message to Sim.

Receive Messages: Upon input (ReceiveMessages) from P:

1. Send (ReceiveMessagesRequest, P) to Sim. If (ReceiveMessagesApprove, P) received in response, send (ReceiveMessagesResponse, messages) to P, where messages contains all pending messages for P, and clear all pending messages for P.

Revoke: Upon input (Revoke, persona, context, evidence) from P:

1. If AllowedToRevoke(P, persona, context, evidence, time) $=$ true, find the party P' associated with (persona, context). Then set $P'.\texttt{revoked} = \texttt{true}$. If no such P' exists, return.
2. If P' is corrupt, send (Revoked, P', persona, context, evidence, time) to Sim. Otherwise, send (Revoked, persona, context, evidence, time) to Sim.

Increment Time: Upon input (IncrementTime) from Sim:

1. Set $\texttt{time} = \texttt{time} + 1$

Fig. 2: An Ideal Functionality that captures the ideal properties of cryptographic personas.

messaging. The adversary may, however, arbitrarily suppress messages (e.g., refuse to deliver messages). Rather than give a separate definition for each of these, we capture them with a single ideal functionality that provides all of these properties.

B. Definition

We give a formal definition of cryptographic personas as $\mathcal{F}_{\text{CryptographicPersona}}$ in the stand-alone simulation model (see Figure 2). This formalism assumes a static set of participants $P_1, \ldots, P_n$ and two parameterizing functions, AllowedToCreateNewPersona and AllowedToRevoke (discussed below). When the system is deployed in infrastructure mode or "as a service" mode (see **System Architecture** in Section II-A), then one of the parties (e.g., P_1) can be the server, and the exchanged messages can correspond to requests. When the system is deployed without a server, all parties are users. We also make use of a pre-defined set of parties RegNotificationSet that learn when a party registers for the system; in serverless mode, this is likely to be all the parties, and include the server otherwise.

The ideal functionality exposes five main interfaces in addition to registration: (1) sending a message with a new persona, or (2) with an existing persona, (3) receiving messages, (4) merging personas, and (5) revocation of abusive users. When sending a message using either interface (1) or (2), the calling party provides a message m, a context context, a set of recipients recipients, and a persona persona when using an existing persona. Messages result in a pending delivery, eventually retrieved via the ReceiveMessages interface. Because it is not inherent that cryptographic personas must provide the confidentiality of end-to-end encryption, all messages are leaked to the ideal adversary Sim. If end-to-end encryption is important for the target application, then the m's passed into the interfaces should be ciphertexts. We also expose adversarial-controlled time, discussed more below. The parameterizing functions should be understood as follows:

AllowedToCreateNewPersona: In many of the use cases we explored above, it can make sense to limit the number of personas a user can create within a given context. This is handled by AllowedToCreateNewPersona, which reasons over the caller's identity and the chosen context. For the basic construction we discuss in Section IV-C, we can imagine this function simply returning true; we explore another concrete instantiation of this function in Section B.

AllowedToRevoke: For revocation, there might be a set of users who are administrators who can unilaterally decide that the user associated with a persona is acting abusively (e.g., a department chair or thread moderator). Alternatively, members of the community could decide democratically. AllowedToRevoke takes the identity of the caller and auxiliary evidence, and determines whether a revocation is authorized, supporting both unilateral and democratic policies. For efficiency, AllowedToRevoke may restrict revocation to recently used personas, limiting the callbacks that must be considered. Due to space constraints,

our proof sketch is in Section A.

C. Construction

Figure 3 describes our basic construction. The protocol assumes a validation server keypair (pk_{sp}, sk_{sp}), parameters $\mathsf{pp} \coloneqq \mathsf{SETUP}(\Phi)$ for zk-promises, and a bulletin board BB. When deployed in *as infrastructure* or *as a service* settings, the bulletin board is simply maintained by the validation server; in the *serverless* setting, clients locally maintain replicas of the bulletin board state. Messages are posted in an application-defined $\mathsf{context} \in \mathsf{AllowedContexts}$, and the protocol uses a commitment scheme Com and a collision- and second-preimage-resistant hash function H. Enforcement of policies governed by the public functions functions AllowedToCreateNewPersona and AllowedToRevoke.

Each client maintains a zk-object obj containing a secret key sk, a Boolean `revoked` (initialized to `false`), and a field element `rep` that captures an application-specific reputation score associated with the user. Rather than encode the baseline reputation `rep` as 0 and allow it to decrement to negative values (resulting in wrap-around), we encode the initial baseline to be half the maximum field value. In Section B we discuss some other application-specific values that can be stored in the object, including "badges." Additionally, the object contains the machinery associated with zk-promises: a random serial number sn and a list of callbacks cbList that the object must process during scanning.

During registration, the client commits to their initial object zk-object obj as $\mathsf{com}_{\mathsf{obj}} = \mathsf{Com}(\mathsf{obj}; r)$ and sends this, along with their real, long-term, Sybil-resistant identity, to the bulletin-board. This approach alone, however, is insufficient, as an adversarial client could copy the secret key in some existing client's zk-object into a new object during registration. By having two different zk-objects with the same secret key, a revocation will only apply to *one* of the zk-objects (the one used to send the message), leaving the other untouched. As a result, the adversary could continue sending messages using pseudonyms that should have been banned.

To rectify this problem, we have the client produce a deterministic "public key" by hashing their secret key and posting this along with their initial state. The bulletin board can then enforce that all public keys must be unique. To prove that this public key is well-formed, the client also produces a zero-knowledge proof of knowledge π_{init} for the following relation:

$$\mathsf{PoK}\{(\mathsf{sk}, \mathsf{sn}, r) : \mathsf{com}_{\mathsf{obj}} = \mathsf{Com}(\mathsf{obj}; r) \ \wedge \ \mathsf{pk} = H(\mathsf{sk}) \\ \mathsf{revoked} = 0 \wedge \mathsf{rep} = \mathbb{F}/2 \wedge \mathsf{cbList} = \{\}\}$$

and sends it along with the public key. We note that this technique is not discussed by Shih et al. as their work does not work through the details of an end-to-end application.

Algorithm 1 Pseudonymous Posting

```
1: procedure Persona(obj, x = (persona, nonce))
2:     assert (obj.revoked = false)
3:     assert (persona = H(obj.sk||nonce))
4:     return
```

Next, to send a message under a new persona, a client samples a new nonce `nonce` and invokes the EXECMETHODANDCREATECALL interface of zk-promises with meth = Persona, as defined in Algorithm 1. Namely, the proof generated by zk-promises certifies that the user currently is not revoked and that they "own" a persona because they know a pre-image for that persona under a collision-resistant hash function $\mathcal{H}$. Alternatively, if a user wants to reuse a persona for multiple messages, they can simply store and reuse the `nonce` multiple times. This de-randomizes the persona generation function, allowing the party to repeatedly demonstrate that they are the owner of a particular persona.

Algorithm 2 Revocation Callback

```
1: procedure Revoke(obj)
2:     obj.revoked ← true
3:     return obj
```

Finally, in order to allow the system to revoke the power to post anonymously (and pseudonymously), each message sent needs to expose the Revoke callback, defined in Algorithm 2. When authorized, anyone can invoke the CALL interface of zk-promises. Once a client scans the bulletin board and confirms the valid callback, their local system updates their zk-object, setting `revoked` to be `true` (i.e., 1 as a field element), effectively preventing the user from sending further anonymous or pseudonymous messages. Checking AllowedToRevoke is implemented *around* this invocation. Namely, the party that runs the bulletin board (e.g., the service provider) can run AllowedToRevoke and only include the result of CALL if the result is `true`. In serverless deployments, each client locally checks AllowedToRevoke.

We prove the following theorem in Section A:

Theorem 1. *Assuming the existence of a weakly simulation-extractable zkSNARK (e.g., Groth16) and a collision-resistant and pre-image resistant hash function H, and a hiding and binding commitment scheme* Com*, the protocol in Figure 3 securely realizes the functionality $\mathcal{F}_{\mathit{CryptographicPersona}}$ in the stand-alone simulation model against adversaries that statically corrupt any number of parties.*

More expressive behaviors, such as rate-limited personas, scope-restricted identities, retroactive persona merging, and anonymous voting, are implemented as extensions of this mechanism. Each feature corresponds to adding more constraints to a similar proof. For example, rate-limited and scope-restricted personas can be realized by constraining the nonce and binding it to the message context (thus, we include the message context in Figure 3, although it is not directly used in the Persona function in Algorithm 1), respectively. User properties such

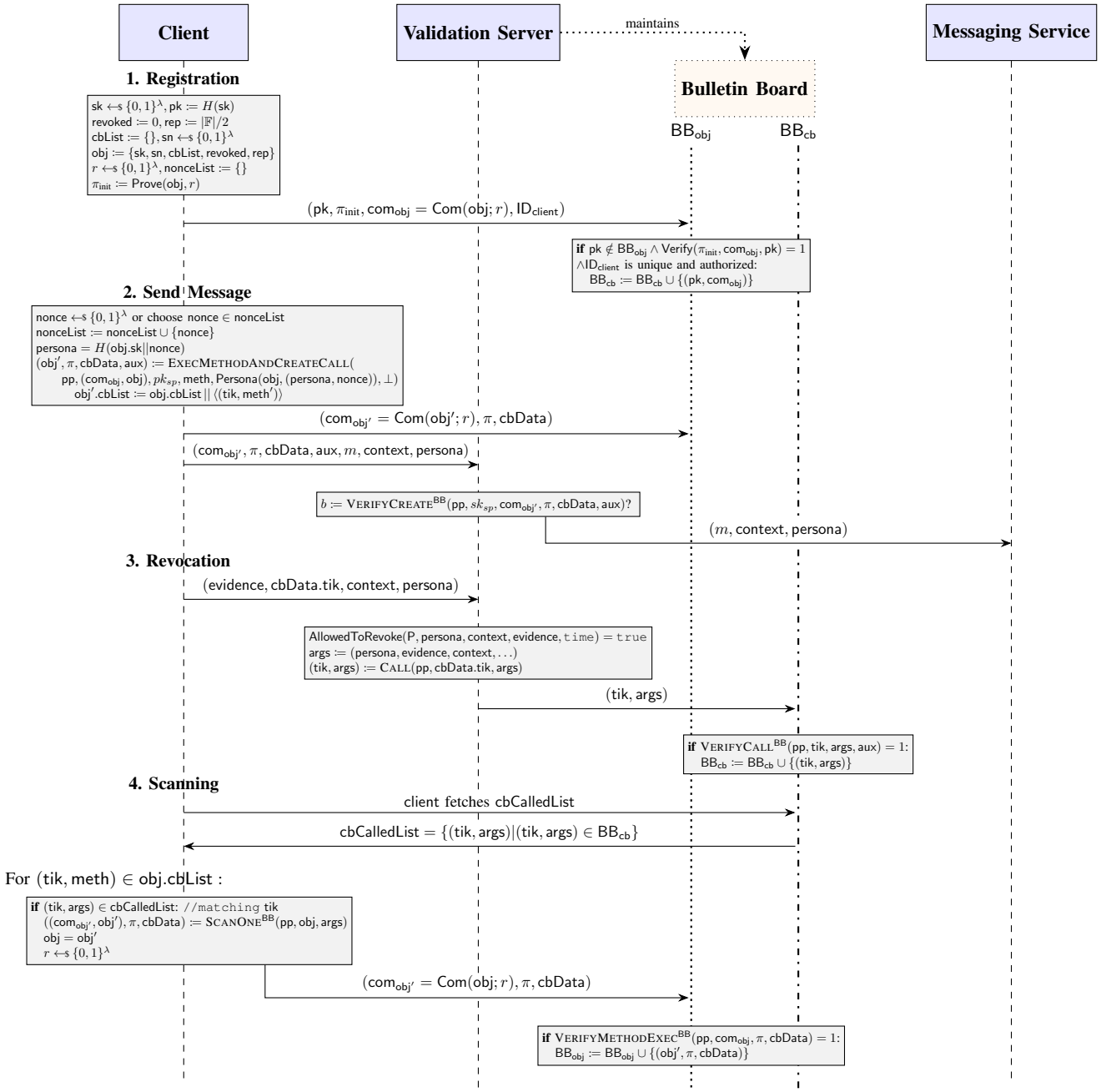


Fig. 3: **Cryptographic Persona Client-Server Protocol.** The bulletin board is logically maintained by the validation server (shown via an orthogonal dotted edge) but depicted separately for clarity. The flow illustrates (1) registration, (2) message dispatch, and (3) revocation/scanning across deployment models.

as reputation scores and badge-based attribute disclosure are handled by adding fields to the user's private state and proving predicates over them. Due to space constraints, we describe these extensions in detail in Section B.

D. Pre-processing Proof Generation

In the protocol described above, each message transmission requires the client to generate a fresh callback and an accompanying zero-knowledge proof, placing cryptographic computation on the critical path of message delivery and introducing nontrivial latency on resource-constrained or legacy devices (see Table I).

In order to move proof generation into a pre-processing phase, one might consider pre-computing a batch of Groth16 proofs locally. However, we observe that such naïve batching approaches fail for two primary reasons: (1) they require the client to pre-commit to both the specific personas and the sequence in which they will be used (as each proof would need to build off the previous one and state-transitions in

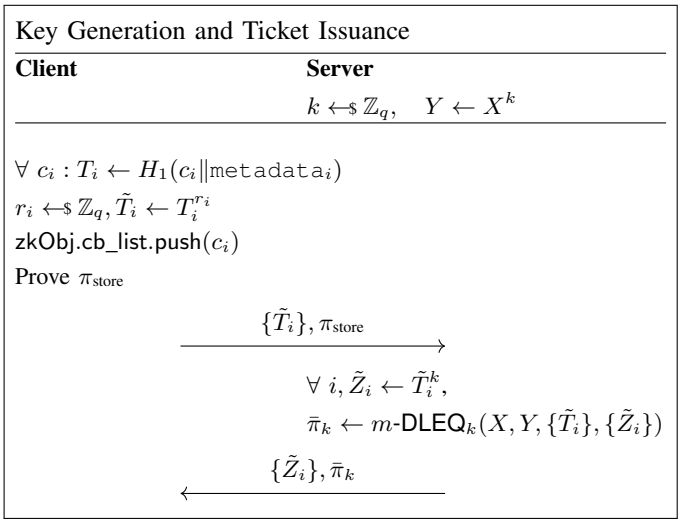


Fig. 4: The server's key generation phase (run during setup) is shown above the line. Ticket issuance using a VOPRF with batch DLEQ proofs is below the line.

zk-personas are inherently sequential), regardless of whether the proofs are computed individually or as a batch; and (2) generating proofs in a batch inherently links the resulting interactions to one another, as the issuance reveals a set of callbacks or nullifiers that are tied to a single point in time. Thus, we require a more intricate approach to pre-processing.

We design our pre-processing technique based on Privacy Pass [18], which enables proof generation and message transmission to be fully unlinkable. The callbacks authorized during pre-processing are independent and unordered: each ticket corresponds to a distinct callback and may be redeemed at any time, in any order, and without reference to other tickets in the batch. This eliminates any requirement for the client to pre-commit to a sequence of interactions while confining zero-knowledge proof generation to a pre-processing phase, providing clients with the flexibility in how and when they consume callbacks. Our construction follows a two-phase model: a *ticket issuance* phase, in which the client performs an expensive interaction with the server to obtain anonymous credentials, and a *ticket redemption* phase, in which the client uses these credentials to post messages.

Ticket issuance. During ticket issuance (see Figure 4), the client prepares a batch of callbacks $\{c_i\}$ and samples a corresponding set of random scalars $\{r_i \in \mathbb{Z}_q\}$. For each i, it computes a blinded input $\tilde{T}_i = H_1(c_i \| \texttt{metadata}_i)^{r_i}$, where $H_1 : \{0,1\}^* \to \mathbb{G}^*$ is a hash-to-curve function into a prime-order group $\mathbb{G}$ with generator X and $\texttt{metadata}_i$ is information that should be associated with c_i. For instance, if c_i will later be used when posting under a known persona, $\texttt{metadata}_i$ would be that persona. The client constructs a zero-knowledge proof π_{store} that demonstrates (1) knowledge of well-formed pre-images $\{c_i \| \texttt{metadata}_i, r_i\}$ with respect to $\tilde{T}_i$, (2) the callbacks $\{c_i\}$ are stored in the user's zk-object, and (3) $\texttt{metadata}_i$ is well formed with respect to the user's zk-object. Additionally, the user proves any application-specific predicate. This proof is submitted once to the server, which holds a VOPRF secret key $k \in \mathbb{Z}_q$ and computes $\tilde{Z}_i = \tilde{T}_i^k$ for each blinded input. To ensure correctness, the server returns the outputs along with a batched discrete log equality (DLEQ) proof over all $(\tilde{T}_i, \tilde{Z}_i)$ pairs, which the client verifies using a challenge hash $H_3 : \mathbb{G}^6 \to \mathbb{Z}_q$ as per the standard batch DLEQ protocol. This step follows the base Privacy Pass protocol; interested readers can refer to [18] for details. After verification, the client unblinds each response to obtain $Z_i = H_1(c_i \| \texttt{metadata}_i)^k$, and derives a keyed MAC with key $K_i = H_2(c_i, Z_i)$, where $H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\kappa$.

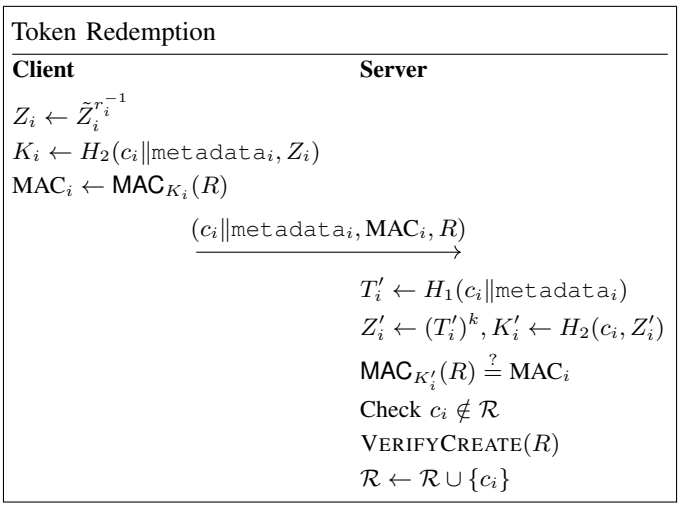


Fig. 5: Redemption of a callback ticket for method execution using derived MAC. R is the client's request.

Ticket redemption. In the ticket redemption phase (see Figure 5), the client selects a callback and its associated PrivacyPass data $(c_i \| \texttt{metadata}_i, K_i)$ and computes a MAC over a fresh application request R using key K_i. This MAC, along with the callback c_i and request R, is sent to the server. Upon receiving this message, the server recomputes the ticket's PRF output $Z_i' = H_1(c_i \| \texttt{metadata}_i)^k$ and derives the corresponding key $K_i' = H_2(c_i, Z_i')$ to verify the MAC's validity. If valid, it checks that c_i has not been previously redeemed by consulting a set $\mathcal{R}$ of used callbacks. The request R is then processed using the pre-authorized commitment structure via VERIFYCREATE and, if accepted, c_i is added to $\mathcal{R}$. No additional zero-knowledge proofs are required at this stage as the callback $\{c_i\}$ is already stored in users' zk-objects.

Ticket expiration and key rotation. This process, in essence, creates a window of valid, unlinkable callbacks that can be redeemed at will. Importantly, unused tickets remain redeemable *indefinitely*, which allows users to circumvent bans. This can be addressed by limiting the validity of unredeemed callbacks by regularly rotating the VOPRF keys each epoch, essentially expiring old tickets. This ensures moderation will eventually have the desired effect; in emergencies, the server can proactively rotate the key.

E. Folding

In zk-promises, callback scans are done one callback handle at a time (or in fixed batches),[4] i.e., a user loops through all previous messages they have sent in the system that *could* have had their callbacks invoked.[5] Users are faced with a choice when it comes to demonstrating that all callbacks associated with them have been properly processed: (1) provide a single proof that encompasses all of the callback handles, or (2) for each callback handle, provide a separate proof. In the former case, note that the statement itself would leak the number of previously sent messages—a privacy leakage—so all users must prove a padded, worst-case circuit, which is extremely inefficient. In the latter case, each of these proofs must be handled and verified unlinkably, meaning that the number of interactions between the user and server *must* be linear in the number of open callbacks. The result is that a client who has previously sent n must interact with the server $O(n)$ separate times in order to send a new message, resulting in a quadratic blowup for the server.

To reduce this load, we first note that scanning is inherently a repeated operation applied to the user state. Thus, we utilize the power of modern folding and accumulation schemes [19], [20], [21], [22], [23] to streamline the scan computation. Specifically, each step of the incrementally verifiable computation reasons about one of the open callbacks and then is folded into a running state. Once this local computation has completed, the user can send the folded proof to the server. Note that this approach doesn't save asymptotic computational complexity for the client (they must still iterate over all callbacks that *could* be invoked), but it means that update proofs have constant complexity for the server—no matter the number of open callbacks. In practice, as we show in Section VI-B, folding also concretely improves the proof generation time on the client by $16 - 75\%$ (for the most basic configurations). Thus, the folding serves a dual purpose: (1) it concretely (although not asymptotically) speeds up the client's computation; and (2) the client can provide a single proof to demonstrate that its state is up to date, even with a variable number of previous callbacks.

Most modern accumulation-based proof systems do not hide the depth of the folding. This can clearly result in deanonymization, as users might have a unique number of callbacks to which they must attend. We can avoid this leakage using a simple trick, described in [13]. Specifically, we can commit to the depth, add the commitment to the statement, and add the depth itself to the witness. The circuit can verify that the witness is consistent with the commitment directly. This approach ensures that the proof reveals no information about how many rounds of folding were performed, thereby preserving user privacy without sacrificing verifiability.

[4]It is also possible to configure the system such that users are responsible for scanning an explicit list of banned personas by simply proving non-membership, as is done in other anonymous banning systems, e.g., [41], [12], [13]. This profile may be favorable in cases where very few callbacks are expected to be invoked, and many messages are sent. We describe this work in the more complex model, as this makes it easier to maintain complex state (e.g., reputation) efficiently.

[5]In principle, this list of callbacks could be *all* messages ever sent with cryptographic personas. In practice, it is prudent to configure the system such that callbacks eventually expire, helping to bound the complexity of this task.

V. Implementation

Our implementation [6] comprises two components: (i) an instantiation in the *as-a-service* paradigm (see Section II) with integrations for Slack and Signal, and (ii) cryptographic primitives and tooling used for benchmarking in Section VI.

We implemented our construction using Groth16 [42] for zero-knowledge proofs, Poseidon [43] for hashing, Nova [19], [23] as our folding scheme as implemented in [44], and Schnorr signatures over BN254 [45] for zk-promises [4]. We use hash-based commitments for the zk-promises scheme.

Deployment model. Our implementation follows the *cryptographic personas as a service* model (see Section II-A), in which a third-party validation service runs cryptographic personas, operating as a proxy that verifies an anonymous message and forwards the message and metadata (e.g., pseudonym, badge, etc.) to an existing service. Rather than reporting isolated micro-benchmarks, we evaluate the additional overhead introduced by cryptographic personas in a platform-agnostic manner, yielding performance results representative across different platform integrations and deployment models (i.e., *infrastructure*, *as-a-service*, or *serverless*).

Integrations. We develop integrations for both Slack and Signal. Our Slack integration sends messages into channels using the APIs exposed for Slackbot integrations, but cannot take in-channel commands, as these commands can not be anonymous. For Signal, messages are sent using a centrally managed "phantom" account using `signal-cli-client` [46] that relays pseudonymous messages after proof verification. This approach necessarily breaks end-to-end encryption and is intended solely for evaluation, as the phantom account has key material that can both send and receive messages. A production Signal deployment could instead realize *cryptographic personas as infrastructure* by allowing all group members to share a *sender key*, keeping end-to-end encryption intact; this requires non-trivial modification of Signal clients.

VI. Evaluation

Our evaluation focuses primarily on the two core operations underlying cryptographic personas: *posting* and *scanning for revocation*. These operations capture the dominant costs associated with anonymous message transmission and moderation enforcement. For simplicity of exposition, our experiments focus solely on a simple pseudonymous post and on revocation as the moderation action. We detail the performance of our more advanced features in Section B-B, which concretely takes at most 80ms more than a pseudonymous post.

We implement both our optimized construction, which uses folding for scanning and precomputation for posting, and a baseline using the techniques from [4]. For each operation, we first show the baseline construction and then measure the

[6]The source code and scripts used in our evaluation are openly available at https://anonymous.4open.science/r/crypto_personas-6184.

TABLE I: Pseudonymous posting benchmarks with regular proof generation as baseline, and authentication as proof replacement as optimization.

Operation Executer	**Proof Generation** Client		**Verification** Server	**End-to-End** Client and Server	
Hardware Unit	*Modern* ms	*Legacy* ms	*Modern* ms	*Modern* ms	*Legacy* ms
Baseline: Pseudonymous Posting	141.4	428.3	5.3	261.2	560.4
Optimized: Precomputed Authentication	1.2	2.9	2.0	121.0	135.0
+ Offline Precomputation Cost	75.2	327.7	1.6		

impact of our corresponding optimizations, namely offline pre-processing for pseudonymous posting and folded scanning for revocation processing.

We report two sets of results to capture performance on modern and legacy hardware. The *Modern* profile uses a 2023 Apple MacBook Pro with an Apple M3 Max (14-core CPU, 30-core GPU), 36GB unified memory, running macOS Sequoia 15.6. The *Legacy* profile uses a 2018 13-inch Apple MacBook Pro with a 2.3GHz quad-core Intel i5, 8GB RAM, running macOS Sequoia 15.7. While older, the 2018 MacBook reasonably approximates hardware on which users may deploy cryptographic personas and serves as a proxy for common mobile devices. We assume the validation server runs on modern hardware and therefore report only *Modern* results.

A. Pseudonymous Posting

Table I gives benchmarks for pseudonymous posting, comparing our optimized version (described in Section IV-D) to a baseline re-implementation (using the state-of-the-art baseline from [4]). It shows the time it takes for the client to generate a proof to post a pseudonymous message, the time it takes for the server to verify the operations cryptographically, and the end-to-end server handling time for a complete request from the point the client starts generating cryptographic material to when our server verifies it (and forwards it to the messaging service).

Baseline. The end-to-end costs are dominated primarily by local API-processing overheads such as serialization, deserialization, and context switching. Recall that for simplicity, all end-to-end measurements were performed on localhost and therefore exclude network latency. Since the communication flow for posting is a single client-to-server message, moving to a WAN setting, to a first-order approximation, simply adds the additional latency to reach the validation server. Client messages here are concretely small at $|\mathsf{msg}| + \sim 500$ bytes for a Groth16 proof and other cryptographic material.

Our benchmarks suggest that the performance of the baseline may be workable as long as clients have modern hardware. By contrast, the legacy hardware results show that proof generation is roughly three times slower, raising client-side latency from ≈ 140ms into the ≈ 430ms range. Our legacy hardware is likely also a reasonable proxy for mobile devices—the devices on which many clients might want to use cryptographic personas. Unfortunately, ≈ 430ms is a user-noticeable delay [47], which could interfere with the usability of cryptographic personas. Thus, we require a way to remove expensive proof generation from the critical path for sending a message, motivating our pre-processing optimization.

Optimized (generating callbacks during pre-processing). Our pre-processing optimization (described in Section IV-D) replaces the expensive online proof-generation costs of approximately 140ms and 430ms with lightweight ticket redemption operations requiring only approximately 1ms and 3ms on modern and legacy hardware, respectively. The tradeoff is that the client must still perform the corresponding proof-generation work during an offline pre-processing phase, which in Table I is shown in gray. Conceptually, the optimization does not eliminate proof generation, but instead shifts it entirely offline, removing it from the critical path of message transmission.

As a result, pre-processing reduces client-side work during online message sending by over 99% on both hardware profiles. On modern hardware, latency decreases from approximately 140ms to 1ms, while the offline pre-processing phase incurs only about 75ms of amortized overhead per ticket. On legacy hardware, the online posting cost similarly decreases from roughly 430ms to 3ms, making cryptographic personas usable even on older systems.

B. Scanning for Revocation

The other key requirement of our system is enforcing moderation through *scanning*, i.e., proving that the client has processed all relevant revocation callbacks. The problem here is the client's costs scale in the number of previous posts they have made, since, for each post they made, they must check if the corresponding callback has been called to revoke their ability to post.[7] Figure 6 gives the costs for a client to scan, assuming there are 100 previous posts.

Baseline. In the baseline, the client scans for each callback individually, creating a new proof each time. At 210ms on modern hardware and approximately 740ms on legacy hardware, this gives us a total processing time for 100 posts of about 21,000ms for modern and 74,000ms for legacy, which are plotted as the blue square and red triangle on Figure 6.

Optimized (folding for scanning). Folding (see Section IV-E) aggregates the sequential processing of multiple callbacks into

[7]Times to post revocation callbacks such that they can be scanned are displayed in Section C.

a single succinct proof. This reduces the amortized proof-generation cost for the client while asymptotically reducing the verification burden on the server, which no longer needs to verify a separate proof for every callback processed by the client.

a) Client-side performance: In our experiments, we fix the total number of callbacks scanned to $x = 100$ and vary the folding batch size $n \in \{1, 2, 4, 5, 10, 20, 25, 50\}$, yielding $t = x/n$ folding steps. Each run concludes with a randomized blinding step to preserve zero-knowledge.

Figure 6 shows the total client-side proving time required to process 100 callbacks across different folding batch sizes on both modern and legacy hardware. Clearly, folded scanning substantially reduces the client-side proving cost across all evaluated configurations. As the batch size n increases, the per-callback proving cost decreases because the client performs fewer folding rounds, although this comes at the cost of larger final proofs and increased blinding overhead.

Across the evaluated configurations, we observe concrete client-side savings ranging from $16\% - 75\%$ at 100 callbacks, depending on the choice of n. The largest gains occur on legacy hardware, where proof generation is substantially more expensive. The best overall performance occurs around $n \in \{20, 25\}$, yielding approximately $4.1\times$ speedup on modern hardware and $4.7\times$ speedup on legacy hardware relative to baseline iterative scanning. We defer a more detailed tradeoff analysis of folding performance to Section D, including computing the number of callbacks a client must have open in order for the folding amortization strategy to be valuable; we find that even with very few open callbacks, folding is more performant.

b) Server-side performance: On the server, the cost of verifying folded scans becomes constant with respect to the total number of callbacks processed and depends only on the batch size n. Concretely, verifying a folded proof for an arbitrary number of callbacks costs only approximately $2\times$ to $4\times$ more than verifying a single Groth16 proof, which itself is limited to reasoning about a single batch of n callbacks. We provide a detailed breakdown of the server performance with folding in Section E.

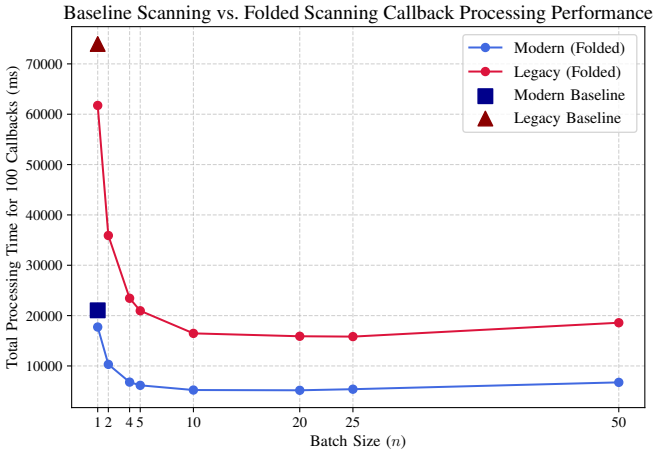


Fig. 6: Client-side proving time to process 100 callbacks under baseline iterative scanning vs our optimized folded scanning (for varying sizes of batches).

VII. Related Work

Social benefits and risks of anonymity. Human-computer interaction research shows that anonymity and pseudonymity—a longstanding online tradition [48]—can mitigate social and professional risks across many digital contexts [49]. For example, anonymous bloggers are more disclosive and honest [50], [51], anonymity aids idea generation [52], and anonymous comment sections are often considered essential features of online discourse [36], [34], [53]. On social media platforms, anonymity can facilitate support-seeking [54], [55] and community formation [56], [57].

Anonymity has been studied most extensively on question-and-answer (Q&A) platforms [28], [29], [30], [31], [32], where it reduces the social costs of participation [30] and leads to more frequent [28] and more sensitive questions [31]. Soliman et al. extended these results by designing an academic Q&A platform supporting partial ("meronymous") identity disclosure to lower barriers to participation [32].

Despite these benefits, anonymity and pseudonymity can also produce disinhibition effects [58]. Anonymous comments are often lower quality and more likely to be anti-social [33], [34], [35]. For instance, the Economics Job Market Rumor Forum has been widely criticized as abusive and toxic [59], [60], and anonymity has been identified as a contributing factor in cyberbullying [61], [62]. These effects are not universal; Tran et al. showed that Wikipedia editors using Tor produced edits of comparable quality to others [1].

Anonymous credentials and anonymous blocklisting. Our work instantiates the anonymous credential paradigm [63], [64], in which users prove authorization without revealing their identity. Since their introduction, anonymous credentials have seen practical deployment [65], [14], [66], [67], including stateful variants [27].

Motivated by the risks of irrevocable anonymous posting, prior work has explored anonymous blocklisting; we refer readers to the overview by Henry et al. [68]. BLAC [11] and subsequent systems avoid trusted third parties by issuing users a secret key and requiring them to prove blocklist non-membership under pseudonymous tags derived from that key.

More recent systems, such as SNARKBlock [12] and Alpaca [13], use modern zero-knowledge techniques to improve blocklist performance, with Alpaca addressing cold-start inefficiencies via folding. Shih et al.'s zk-promises system [4] combines stateful anonymous credentials [27] with a callback-based blocklisting approach [69], enabling programmable, unlinkable credentials whose state can be updated through anonymous callbacks. This design supports richer functionality, such as reputation systems, while offering better performance than earlier callback-based schemes [69] and greater expressiveness than SNARKBlock [12] and Alpaca [13]. The scanning-based revocation approach used in zk-promises and our work is most effective when the number of user actions to be checked is

smaller than the global ban list; in settings where bans are rare but messages are frequent, BLAC-style techniques [11] may be preferable.

VIII. Conclusion

Anonymity and pseudonymity can enable honest participation, but they are also prone to abuse. We introduce cryptographic personas as a primitive that enables pseudonymous communication while selectively revoking anonymity privileges without identity de-anonymization. We build on recent work on anonymous blocklists, extending it to support real-time systems such as group chats.

Acknowledgments. This research was supported by NSF under award 2504578.

References

[1] C. Tran, K. Champion, A. Forte, B. M. Hill, and R. Greenstadt, "Are anonymity-seekers just like everybody else? An analysis of contributions to wikipedia from tor," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 186–202.

[2] A. E. Marwick and danah boyd, "I tweet honestly, i tweet passionately: Twitter users, context collapse, and the imagined audience," *New Media & Society*, vol. 13, no. 1, pp. 114–133, 2011. [Online]. Available: https://doi.org/10.1177/1461444810365313

[3] A. Hamilton, J. Madison, and J. Jay, "The federalist papers," 1787.

[4] M. Shih, M. Rosenberg, H. Kailad, and I. Miers, "zk-promises: Anonymous Moderation, Reputation, and Blocking from Anonymous Credentials with Callbacks," 2025, pp. 4995–5014. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/shih

[5] D. Chaum and E. van Heyst, "Group signatures," in *EUROCRYPT'91*, ser. LNCS, D. W. Davies, Ed., vol. 547. Springer, Berlin, Heidelberg, Apr. 1991, pp. 257–265.

[6] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," in *EUROCRYPT 2003*, ser. LNCS, E. Biham, Ed., vol. 2656. Springer, Berlin, Heidelberg, May 2003, pp. 614–629.

[7] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *ASIACRYPT 2001*, ser. LNCS, C. Boyd, Ed., vol. 2248. Springer, Berlin, Heidelberg, Dec. 2001, pp. 552–565.

[8] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract)," in *ACISP 04*, ser. LNCS, H. Wang, J. Pieprzyk, and V. Varadharajan, Eds., vol. 3108. Springer, Berlin, Heidelberg, Jul. 2004, pp. 325–335.

[9] A. Su, "How One Syrian Fought to the Death for a Free Internet," https://www.wired.com/story/how-one-syrian-fought-to-the-death-for-a-free-internet/, Sep. 2017, accessed on 4 June, 2025.

[10] aiiightb, "Food industry executive here. Tariffs are about to change everything, AMA." https://www.reddit.com/r/AMA/comments/1kiwvme/food_industry_executive_here_tariffs_are_about_to/, May 2025, accessed on 4 June, 2025.

[11] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "Blacklistable anonymous credentials: blocking misbehaving users without ttps," in *ACM CCS 2007*, P. Ning, S. De Capitani di Vimercati, and P. F. Syverson, Eds. ACM Press, Oct. 2007, pp. 72–81.

[12] M. Rosenberg, M. Maller, and I. Miers, "SNARKBlock: Federated anonymous blocklisting from hidden common input aggregate proofs," in *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2022, pp. 948–965.

[13] J. Kim, A. Kothapalli, O. Chardouvelis, R. S. Wahby, and P. Grubbs, "ALPACA: Anonymous blocklisting with constant-sized updatable proofs," Cryptology ePrint Archive, Paper 2025/767, 2025. [Online]. Available: https://eprint.iacr.org/2025/767

[14] M. Rosenberg, J. D. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2023, pp. 790–808.

[15] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 790–808.

[16] "It's now easier to prove age and identity with Google Wallet," https://blog.google/products/google-pay/google-wallet-age-identity-verifications/, Apr. 2025.

[17] M. Frigo and a. shelat, "Anonymous credentials from ECDSA," Cryptology ePrint Archive, Report 2024/2010, 2024. [Online]. Available: https://eprint.iacr.org/2024/2010

[18] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *PoPETs*, vol. 2018, no. 3, pp. 164–180, Jul. 2018.

[19] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *CRYPTO 2022, Part IV*, ser. LNCS, Y. Dodis and T. Shrimpton, Eds., vol. 13510. Springer, Cham, Aug. 2022, pp. 359–388.

[20] A. Kothapalli and S. Setty, "SuperNova: Proving universal machine executions without universal circuits," Cryptology ePrint Archive, Report 2022/1758, 2022. [Online]. Available: https://eprint.iacr.org/2022/1758

[21] ——, "CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves," Cryptology ePrint Archive, Report 2023/1192, 2023. [Online]. Available: https://eprint.iacr.org/2023/1192

[22] L. Eagen and A. Gabizon, "ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances," Cryptology ePrint Archive, Report 2023/1106, 2023. [Online]. Available: https://eprint.iacr.org/2023/1106

[23] A. Kothapalli and S. T. V. Setty, "HyperNova: Recursive arguments for customizable constraint systems," in *CRYPTO 2024, Part X*, ser. LNCS, L. Reyzin and D. Stebila, Eds., vol. 14929. Springer, Cham, Aug. 2024, pp. 345–379.

[24] "Am I the Asshole?" https://www.reddit.com/r/AmItheAsshole/, accessed on 4 June, 2025.

[25] J. Lund, "Technology Preview: Sealed sender for Signal," https://signal.org/blog/sealed-sender/, October 2018, accessed on 4 June, 2025.

[26] D. Y. Liu, J. K. Liu, Y. Mu, W. Susilo, and D. S. Wong, "Revocable ring signature," *Journal of Computer Science and Technology*, vol. 22, pp. 785–794, 2007.

[27] S. E. Coull, M. Green, and S. Hohenberger, "Controlling access to an oblivious database using stateful anonymous credentials," in *PKC 2009*, ser. LNCS, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Berlin, Heidelberg, Mar. 2009, pp. 501–520.

[28] F. Lee, "When the going gets tough, do the tough ask for help? help seeking and power motivation in organizations," *Organizational behavior and human decision processes*, vol. 72, no. 3, pp. 336–363, 1997.

[29] X. Ma, N. Andalibi, L. Barkhuus, and M. Naaman, ""people are either too fake or too real" opportunities and challenges in tie-based anonymity," in *Proceedings of the 2017 CHI conference on human factors in computing systems*, 2017, pp. 1781–1793.

[30] H. Ma, H.-F. Cheng, B. Yu, and H. Zhu, "Effects of anonymity, ephemerality, and system routing on cost in social question asking," *Proceedings of the ACM on human-computer interaction*, vol. 3, no. GROUP, pp. 1–21, 2019.

[31] C. Guo and K. Caine, "Anonymity, user engagement, quality, and trolling on q&a sites," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–27, 2021.

[32] N. Soliman, H. B. Kang, M. Latzke, J. Bragg, J. C. Chang, A. X. Zhang, and D. R. Karger, "Mitigating barriers to public social interaction with meronymous communication," in *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, F. F. Mueller, P. Kyburz, J. R. Williamson, C. Sas, M. L. Wilson, P. O. T. Dugas, and I. Shklovski, Eds. ACM, 2024, pp. 151:1–151:26. [Online]. Available: https://doi.org/10.1145/3613904.3642241

[33] L. Terveen and W. Hill, "Beyond recommender systems: Helping people help each other," *HCI in the New Millennium*, vol. 1, no. 2001, pp. 487–509, 2001.

[34] N. Diakopoulos and M. Naaman, "Towards quality discourse in online news comments," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, 2011, pp. 133–142.

[35] P. G. Kilner and C. M. Hoadley, "Anonymity options and professional participation in an online community of practice," in *Computer Supported Collaborative Learning 2005*. Routledge, 2017, pp. 272–280.

[36] C. Lampe and P. Resnick, "Slash(dot) and burn: distributed moderation in a large online conversation space," in *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, ser. CHI '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 543–550. [Online]. Available: https://doi.org/10.1145/985692.985761

[37] J. Seering, R. Kraut, and L. Dabbish, "Shaping pro and anti-social behavior on twitch through moderation and example-setting," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, ser. CSCW '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 111–125. [Online]. Available: https://doi.org/10.1145/2998181.2998277

[38] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Steering user behavior with badges," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 95–106. [Online]. Available: https://doi.org/10.1145/2488388.2488398

[39] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 2857–2866. [Online]. Available: https://doi.org/10.1145/1978942.1979366

[40] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková, "Collective information security in large-scale urban protests: the case of hong kong," in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 2021, pp. 3363–3380. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/albrecht

[41] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "BLAC: Revoking repeatedly misbehaving anonymous users without relying on TTPs," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 4, pp. 1–33, 2010.

[42] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Berlin, Heidelberg, May 2016, pp. 305–326.

[43] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, Aug. 2021, pp. 519–535. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi

[44] P. S. Explorations, "Sonobe: Succinct proof composition and verification framework," https://privacy-scaling-explorations.github.io/sonobe-docs/, 2024, accessed: 2025-06-04. [Online]. Available: https://github.com/privacy-scaling-explorations/sonobe

[45] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *USENIX Security 2014*, K. Fu and J. Jung, Eds. USENIX Association, Aug. 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[46] S. Scheibner, "AsamK/signal-cli," May 2025, original-date: 2015-05-11T10:49:42Z. [Online]. Available: https://github.com/AsamK/signal-cli

[47] J. Nielsen, "Response Times: The 3 Important Limits," https://www.nngroup.com/articles/response-times-3-important-limits/, Jan 1993, accessed on 4 June, 2025.

[48] J. S. Donath, "Identity and deception in the virtual community," in *Communities in cyberspace*. Routledge, 2002, pp. 37–68.

[49] R. Kang, S. Brown, and S. Kiesler, "Why do people seek anonymity on the internet? informing policy and design," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2013, pp. 2657–2666.

[50] E. E. Hollenbaugh and M. K. Everett, "The effects of anonymity on self-disclosure in blogs: An application of the online disinhibition effect," *Journal of Computer-Mediated Communication*, vol. 18, no. 3, pp. 283–302, 04 2013. [Online]. Available: https://doi.org/10.1111/jcc4.12008

[51] Q. Liao, Y. Pan, M. X. Zhou, and T. Gan, "Your space or mine? community management and user participation in a chinese corporate blogging community," in *Proceedings of the ACM 2012 conference on computer supported cooperative work*, 2012, pp. 315–324.

[52] T. Connolly, L. M. Jessup, and J. S. Valacich, "Effects of anonymity and evaluative tone on idea generation in computer-mediated groups," *Management science*, vol. 36, no. 6, pp. 689–703, 1990.

[53] R. Fredheim, A. Moore, and J. Naughton, "Anonymity and online commenting: The broken windows effect and the end of drive-by commenting," in *Proceedings of the ACM web science conference*, 2015, pp. 1–8.

[54] M. De Choudhury and S. De, "Mental health discourse on reddit: Self-disclosure, social support, and anonymity," in *Proceedings of the international AAAI conference on web and social media*, vol. 8, no. 1, 2014, pp. 71–80.

[55] N. Andalibi, O. L. Haimson, M. De Choudhury, and A. Forte, "Understanding social media disclosures of sexual abuse through the lenses of support seeking and anonymity," in *Proceedings of the 2016 CHI conference on human factors in computing systems*, 2016, pp. 3906–3918.

[56] M. Bernstein, A. Monroy-Hernández, D. Harry, P. André, K. Panovich, and G. Vargas, "4chan and/b: An analysis of anonymity and ephemerality in a large online community," in *Proceedings of the international AAAI conference on web and social media*, vol. 5, no. 1, 2011, pp. 50–57.

[57] N. B. Ellison, L. Blackwell, C. Lampe, and P. Trieu, ""the question exists, but you don't exist with it": Strategic anonymity in the social lives of adolescents," *Social Media+ Society*, vol. 2, no. 4, p. 2056305116670673, 2016.

[58] J. R. Suler and W. L. Phillips, "The bad boys of cyberspace: Deviant behavior in a multimedia chat community," *CyberPsychology & Behavior*, vol. 1, no. 3, pp. 275–294, 1998.

[59] J. Wolfers, "Evidence of a toxic environment for women in economics," *New York Times*, vol. 18, 2017.

[60] F. Ederer, P. Goldsmith-Pinkham, and K. Jensen, "Anonymity and identity online," *arXiv preprint arXiv:2409.15948*, 2024.

[61] C. P. Barlett, D. A. Gentile, and C. Chew, "Predicting cyberbullying from anonymity." *Psychology of Popular Media Culture*, vol. 5, no. 2, p. 171, 2016.

[62] R. Young, S. Miles, and S. Alhabash, "Attacks by anons: A content analysis of aggressive posts, victim responses, and bystander interventions on a social media site," *Social Media + Society*, vol. 4, no. 1, p. 2056305118762444, 2018. [Online]. Available: https://doi.org/10.1177/2056305118762444

[63] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO'82*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Plenum Press, New York, USA, 1982, pp. 199–203.

[64] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT 2001*, ser. LNCS, B. Pfitzmann, Ed., vol. 2045. Springer, Berlin, Heidelberg, May 2001, pp. 93–118.

[65] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *CRYPTO 2004*, ser. LNCS, M. Franklin, Ed., vol. 3152. Springer, Berlin, Heidelberg, Aug. 2004, pp. 41–55.

[66] S. Tessaro and C. Zhu, "Revisiting BBS signatures," in *EUROCRYPT 2023, Part V*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14008. Springer, Cham, Apr. 2023, pp. 691–721.

[67] C. Baum, O. Blazy, J.-H. Hoepman, A. Lehmann, A. Lysyanskaya, R. Mayrhofer, H. Montgomery, N. K. Nguyen, abhi shelat, D. Slamanig, and S. E. Thomsen, "Cryptographers' feedback on the eu digital identity's arf," https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf, Jun 2024.

[68] R. Henry and I. Goldberg, "Formalizing anonymous blacklisting systems," in *2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2011, pp. 81–95.

[69] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "PEREA: towards practical TTP-free revocation in anonymous authentication," in *ACM CCS 2008*, P. Ning, P. F. Syverson, and S. Jha, Eds. ACM Press, Oct. 2008, pp. 333–344.

[70] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs," in *CRYPTO 2017, Part II*, ser. LNCS, J. Katz and H. Shacham, Eds., vol. 10402. Springer, Cham, Aug. 2017, pp. 581–612.

[71] K. Baghery, M. Kohlweiss, J. Siim, and M. Volkhov, "Another look at extraction and randomization of groth's zk-SNARK," in *FC 2021, Part I*, ser. LNCS, N. Borisov and C. Díaz, Eds., vol. 12674. Springer, Berlin, Heidelberg, Mar. 2021, pp. 457–475.

[72] A. Rosenfeld, S. Sina, D. Sarne, O. Avidov, and S. Kraus, "Whatsapp usage patterns and prediction of demographic characteristics without access to message content," *Demographic Research*, vol. 39, pp. 647–670, 2018. [Online]. Available: https://www.jstor.org/stable/26585343

APPENDIX A
PROOF

Before proceeding to our main proof statement, we require that the zkSNARK used in the underlying zk-promises instance is *simulation extractable*, as the simulator will be simulating proofs for honest parties while extracting witnesses from the corrupt clients. In particular, we rely on the fact that Groth16 provides a form of *weak* simulation extractability [42], [70], [71], which guarantees that for any PPT adversary interacting with a simulator that produces simulated proofs, there exists an extractor Ext such that for any accepting proof π output by the adversary, $\mathrm{Ext}(\pi)$ outputs a valid witness for the associated statement, except with negligible probability in the security parameter. By "weak" simulation extractability, we require that the proofs from which we extract need only produce a valid witness if the simulator has not produced a proof for the same statement.

To prove Theorem 1, we define a series of hybrids H_i.

Hybrid H_0 (Real Execution). This hybrid corresponds to a real execution of the protocol in Figure 3 with adversary $\mathcal{A}$ interacting with honest parties.

Hybrid H_1 (Simulated Proofs). Modify H_0 by replacing all honestly generated proofs with simulated proofs produced by the zero-knowledge simulator.

Lemma 2. $\mathsf{H}_0 \approx_c \mathsf{H}_1$.

Proof. Follows directly from the zero-knowledge property of the underlying zkSNARK. □

Hybrid H_2 (Extraction for registration). Modify H_1 so that whenever a message $\mathsf{pk}, \pi_{\mathsf{init}}, \mathsf{com}_{\mathsf{obj}}, \mathsf{ID}_{\mathsf{client}}$, such that pk is unique and $\mathsf{ID}_{\mathsf{client}}$ is a valid identity of a corrupt client, is received from an adversarial client, we run the extractor to recover the secret key in obj. If the extractor fails, the experiment aborts.

Lemma 3. $\mathsf{H}_1 \approx_c \mathsf{H}_2$.

Proof. By weak simulation extractability, the extractor only fails with non-negligible probability if the statement is a statement for which simulations were provided. However, by construction, we only consider unique values of pk, making uniqueness trivial to satisfy. □

Hybrid H_3 (Extraction for sending). Modify H_2 so that whenever a proof π is accepted, sending a new message that was not previously queried to the simulation oracle, we run the extractor Ext to obtain a witness w for the corresponding statement. If extraction fails or produces an invalid witness, the experiment aborts.

Lemma 4. $\mathsf{H}_2 \approx_c \mathsf{H}_3$.

Proof. By weak simulation extractability, any accepting proof for a statement not previously queried to the simulator yields a valid witness except with negligible probability. Note that in sending flows, all zero-knowledge proofs that will result in a message being sent have unique statements except with negligible probability, as the statement reasons over commitments and the uniformly sampled serial numbers for each object. An adversarial client can sample their serial numbers to be identical to a preexisting, announced serial number, but this would only result in any honest validation server rejecting the message. As such, the unique statement is trivially satisfied. Thus, extraction fails with only negligible probability, and the probability that the experiment aborts differs negligibly between H_2 and H_3. □

Hybrid H_4 (Aborting on secret key extraction). Modify H_2 so that if at any point the extractor returns the secret key corresponding to the public key posted for a simulated honest client during registration, the experiment aborts.

Lemma 5. $\mathsf{H}_3 \approx_c \mathsf{H}_4$.

Proof. If this abort happens with non-negligible probability, then we can build a new adversary that breaks the pre-image resistance of the hash function with high probability. □

Hybrid H_5 (Persona Reuse). Modify H_4 so that the experiment aborts if the adversary outputs a valid message involving a persona that was previously assigned to an honest party.

Lemma 6. $\mathsf{H}_4 \approx_c \mathsf{H}_5$.

Proof. An abort of this kind in H_5 implies that the adversary outputs a valid proof π corresponding to a persona previously assigned to an honest party. Since the associated statement was not previously simulated, extractability yields a witness $(\mathsf{obj.sk}, \mathsf{nonce})$ such that

$$\mathsf{persona} = H(\mathsf{obj.sk} \,\|\, \mathsf{nonce}).$$

This computes a preimage of an output of the hash function on a high-entropy input, violating the preimage resistance of H. Therefore, the probability that H_5 aborts in this way is negligible. □

Hybrid H_6 (Randomized Personas). Modify H_6 so that personas corresponding to honest parties are sampled from the output distribution of H when evaluated on uniform inputs, rather than as

$$\mathsf{persona} = H(\mathsf{obj.sk} \,\|\, \mathsf{nonce}),$$

while maintaining consistency across the execution.

Lemma 7. $\mathsf{H}_5 \approx_c \mathsf{H}_6$.

Proof. For honest parties running the protocol, take uniform, high-entropy inputs and feed them to the hash function H. Thus, we can simply sample these from the output space uniformly. If an adversary can distinguish between these distributions, then it implies that they can break the preimage resistance of the hash function. □

Hybrid H_7 (Ideal World). Replace the real protocol execution with an interaction with the ideal functionality $\mathcal{F}_{\text{CryptographicPersona}}$ and the simulator $\mathcal{S}_{\text{CryptographicPersona}}$.

$\mathcal{S}_{\text{CryptographicPersona}}$

$\mathcal{S}_{\text{CryptographicPersona}}$ maintains a table KeyTable mapping secret keys sk to party identities.

Register: Upon input $(\mathsf{RegisterRequest}, P)$ from $\mathcal{F}_{\text{CryptographicPersona}}$:

1. Generate a random sk for P.
2. Record (sk, P) in KeyTable.
3. Honestly generate pk and π_{init} according to the protocol and simulate posting the registration record to the bulletin board.
4. Send $(\mathsf{RegisterRequestApprove}, P)$ to $\mathcal{F}_{\text{CryptographicPersona}}$.

Adversarial Register: Upon input $(\mathsf{pk}, \pi_{\text{init}}, \mathsf{com}_{\mathsf{obj}}, P)$ from $\mathcal{A}$:

1. Run the extractor on π_{init} to obtain the sk associated with P's registration in the underlying zk-promises execution. Abort if the extractor fails.
2. Look up $P' \leftarrow \mathsf{KeyTable}[\mathsf{sk}]$. If P' already exists and P' is honest, abort the simulation with an error.
3. If P is not already registered and pk is unique, then record (sk, P) in KeyTable and send $(\mathsf{RegisterRequestApprove}, P)$ to $\mathcal{F}_{\text{CryptographicPersona}}$.

Send Message with New Persona: Upon receiving $(\mathsf{com}_{\mathsf{obj}'}, \pi, \mathsf{cbData}, \mathsf{aux}, m, \mathsf{context}, \mathsf{persona})$ from $\mathcal{A}$:

1. Run the extractor on π to obtain the sk and the nonce used to randomly sample a persona.
2. Look up $P \leftarrow \mathsf{KeyTable}[\mathsf{sk}]$. If it does not exist, abort.
3. Send $(\mathsf{SendMessageNewPersona}, m, \mathsf{context}, \mathsf{recipients})$ to $\mathcal{F}_{\text{CryptographicPersona}}$.
4. Upon receiving $(\mathsf{PersonaHandleRequest}, m, \mathsf{recipients})$ from $\mathcal{F}_{\text{CryptographicPersona}}$, send $(\mathsf{PersonaHandleResponse}, \mathsf{persona})$ to $\mathcal{F}_{\text{CryptographicPersona}}$.

Send Message with Existing Persona: Upon receiving $(\pi, m, \mathsf{context}, \mathsf{recipients}, \mathsf{persona})$ from $\mathcal{A}$:

1. Run the extractor on π to obtain the sk and the nonce used to randomly sample a persona. If extraction fails, abort.
2. Look up $P \leftarrow \mathsf{KeyTable}[\mathsf{sk}]$. If it does not exist, abort.
3. Check if persona was correctly generated using the sk and the nonce. If not, abort.
4. Otherwise, send $(\mathsf{SendMessageExistingPersona}, m, \mathsf{context}, \mathsf{recipients}, \mathsf{persona})$ to $\mathcal{F}_{\text{CryptographicPersona}}$.

Pending Message: Upon receiving $(\mathsf{persona}, m, \mathsf{context}, \mathtt{time})$ from $\mathcal{F}_{\text{CryptographicPersona}}$:

1. Simulate a proof π that shows the message m was sent by a user with persona and simulate the appropriate state update on the bulletin board.

Revoke: Upon input $(\mathsf{Revoked}, \mathsf{persona}, \mathsf{context}, \mathsf{evidence}, \mathtt{time})$ from $\mathcal{F}_{\text{CryptographicPersona}}$:

1. This message will only be received if the party being revoked is not corrupt. In this case, Sim looks up the tik it generated when simulating the message associated with context, evidence (associated with the underlying zk-object protocol) and simulates posting this to the bulletin board.

Adversarial Revoke: Upon input $(\mathsf{Revoked}, P', \mathsf{persona}, \mathsf{context}, \mathsf{evidence}, \mathtt{time})$ from $\mathcal{F}_{\text{CryptographicPersona}}$:

1. This message will only be received if P' is corrupt. Sim looks up the tik received from the underlying zk-object protocol associated with the adversarially generated message context, evidence and simulates posting this to the bulletin board.

Adversarial Scanning: Upon input $\mathsf{com}_{\mathsf{obj}}, \pi, \mathsf{cbData}$ from $\mathcal{A}$:

1. Simulate the bulletin board honestly.

Receive Messages: Upon receiving $(\mathsf{ReceiveMessagesRequest}, P)$ from $\mathcal{F}_{\text{CryptographicPersona}}$:

1. Ask $\mathcal{A}$ if they want to allow the messages to be delivered.
2. If yes, return $(\mathsf{ReceiveMessagesApprove}, P)$. If P was corrupt, then also receive $(\mathsf{ReceiveMessagesResponse}, \mathtt{messages})$ from $\mathcal{F}_{\text{CryptographicPersona}}$ and forward them to $\mathcal{A}$.

Increment Time: Upon input $(\mathsf{IncrementTime})$ from $\mathcal{A}$:

1. Send $(\mathsf{IncrementTime})$ to $\mathcal{F}_{\text{CryptographicPersona}}$.

Fig. 7: Simulator for Cryptographic Personas.

Lemma 8. $\mathsf{H}_6 \approx_c \mathsf{H}_7$.

Proof. In H_3, all values provided to the adversary are simulated, and all actions of corrupted parties are mediated via extraction and consistency checks. This is identical to the behavior of the simulator interacting with the ideal functionality. Thus, the two hybrids are perfectly indistinguishable. □

With all these hybrids in place, we are now ready to prove our main theorem:

Theorem 1. *Assuming the existence of a weakly simulation-extractable zkSNARK (e.g., Groth16) and a collision-resistant and pre-image resistant hash function H, and a hiding and binding commitment scheme* Com*, the protocol in Figure 3 securely realizes the functionality $\mathcal{F}_{\text{CryptographicPersona}}$ in the stand-alone simulation model against adversaries that statically corrupt any number of parties.*

Proof. The proof follows from the hybrid argument:

$$\mathsf{H}_0 \approx_c \mathsf{H}_1 \approx_c \mathsf{H}_2 \approx_c \mathsf{H}_3 \approx_c \mathsf{H}_4 \approx_c \mathsf{H}_5 \approx_c \mathsf{H}_6 \approx_c \mathsf{H}_7 \approx_c \mathsf{H}_8.$$

Thus, by the Hybrid Lemma, the real-world execution is computationally indistinguishable from the ideal-world execution. □

APPENDIX B
ADVANCED MESSAGING FEATURES

We now continue to describe the ways to build a richer set of features on top of our main construction.

A. Definitions

Create rate-limited persona. In some cases, it may be valuable to limit the number of personas that a user can create within a given context (e.g., a day or a topic of discussion). Specifically, it may be helpful to set AllowedToCreateNewPersona that parameterizes $\mathcal{F}_{\text{CryptographicPersona}}$ to be the one shown in Algorithm 3. This can be done by invoking the EXECMETHODANDCREATECALL interface of zk-promises with meth = PersonaLimited, defined in Algorithm 4. In this function, the caller supplies the number i of personas that they have created for a given context so far as a secret witness to the proof. i can be any number from 0 to k-1, which limits the number of *unique* personas that can be generated to k.

Algorithm 3 Limited Pseudonym Creation Predicate

```
1: procedure AllowedToCreateNewPersona_k(P, context)
2:     if P has previously invoked this function at least k
       times for context context: then
3:         return false
4:     else if context ∉ AllowedContexts then
5:         return false
6:     else
7:         return true
```

Create scope-limited persona. In addition to rate-limiting the number of personas a user may create, it may also be useful to restrict *which* contexts a given persona is authorized to post in. For example, a user might only be permitted to post anonymously in a specific thread, channel, or group. This can be enforced by encoding the thread or group identifier as part of the hash pre-image and verifying that the persona being used was derived for the intended context. Like rate-limiting, this can also be done by invoking the EXECMETHODANDCREATECALL interface of zk-promises with meth = PersonaLimited in Algorithm 4, by specifying AllowedContexts. In practice, this ensures that a persona created for one topic or group cannot be reused in another, enabling scope control over anonymous identities.

Algorithm 4 Rate and Scope Limiting

```
1: procedure PersonaLimited_k(
       obj, x = (persona, context), w = i)
2:     assert(obj.revoked = false)
3:     assert(i < k)
4:     assert(context ∈ AllowedContexts)
5:     assert(persona = H(obj.sk||i||context))
6:     return
```

Merge personas. In some applications, it may be useful for a user to retroactively link two previously unrelated pseudonyms. To enable this, our system provides a mechanism for persona merging via the MergePersonas method, as defined in Algorithm 5. This method takes as input two personas, $\texttt{persona}_1$ and $\texttt{persona}_2$, and their corresponding nonces, $\texttt{nonce}_1$ and $\texttt{nonce}_2$, and proves that both were derived from the same secret key. The same functionality can be applied to limited pseudonyms. This allows the user to link activities performed under different personas without revealing their secret key or any unrelated personas.

Algorithm 5 Retroactively Merging Personas

```
1: procedure MergePersonas(
       obj, x = (persona_1, nonce_1, persona_2, nonce_2)
2:     assert(obj.revoked = false)
3:     assert(persona_1 = H(obj.sk||nonce_1))
4:     assert(persona_2 = H(obj.sk||nonce_2))
5:     return
```

Vote anonymously. One particular setting in which rate-limiting responses are particularly valuable is the creation of anonymous polls. We implement anonymous polls by encoding the poll identifier into the context and setting the maximum number of personas that can be created for polls to be 1. Specifically, the user can generate a *unique* persona for that poll by calling EXECMETHODANDCREATECALL interface of zk-promises with meth = AnonVote, as defined in Algorithm 6. Note that by no longer making use of a nonce, there's no way for a user to generate two personas for a specific poll.

We note that there is no corresponding interface in the ideal functionality for anonymous polls. This is because it is implicitly captured as long as the AllowedToCreateNewPersona function limits each party to a single persona for each context that is associated with a poll. In this case, parties can vote using the SendMessageNewPersona interface, setting context = $poll_id$ and $m \in \{0, 1\}$ to be their vote.

Algorithm 6 Anonymous Voting

```
1: procedure AnonVote(obj, x = (persona))
2:     assert(obj.revoked = false)
3:     assert(persona = H(obj.sk||poll_id))
4:     return
```

Revoke as result of voting anonymously. Groups may also wish to collectively enforce revocations. They can do so by participating in an anonymous vote. If the outcome of this anonymous vote meets a predefined threshold, this collective decision triggers the revocation process. Specifically, the result of the vote's execution implicitly invokes the CALL interface associated with the revocation callback for the targeted user. In this case, the AllowedToRevoke parameterizing $\mathcal{F}_{\text{CryptographicPersona}}$ should only return true when the supplied evidence is of a poll.

Change reputation. In some settings, it might be valuable for a user to maintain a reputation score, which they can later associate with the persona they create. For example,

TABLE II: Personas operation benchmarks (Without optimizations from Section IV-D and Section IV-E)

Operation Executer	**R1CS**	**Proof Generation** Client		**Proof Verification** Server	**Proof Verification & Message Handling** Server
Hardware Unit	# 	*Modern* ms	*Legacy* ms	*Modern* ms	*Modern* ms
Lim. Pseudonym	10484	143.7	429.7	5.8	112.7
Unlim. Pseudonym	10206	141.4	428.3	5.3	112.5
Authorship	8805	123.7	370.2	3.3	99.9
Vote	8532	117.0	356.6	3.0	100.4
Badge	8268	113.2	342.5	3.0	99.3
Scan (baseline)	21614	210.5	739.5	2.0	4.0

anyone could up-vote or down-vote a message shared by a user (under any of their personas), and then when a user creates a new persona, they can annotate that persona with "high reputation," making them more trustworthy. We implement this using the reputation callback described in Algorithm 7. Creating a new persona can be modified in a direct way by adding an additional **assert** statement. The input value n could be negative or positive. The updated reputation, however, can only be positive.[8] The group can customize the details of this credibility score to fit their specific needs. For example, the score could start at a minimum (like 0) and increase as users gain positive feedback. Alternatively, it could begin at a maximum (like 100) and decrease with negative feedback. Or, it could start at a neutral mid-point (like 50), allowing for both positive and negative movement. This enables a flexible system for rewarding positive contributions and reflecting negative feedback anonymously.

Algorithm 7 Reputation Callback

```
1: procedure Reputation(obj, x = (n))
2:     obj.rep ← min(max(obj.rep+n, 0), |𝔽|)
3:     return obj
```

Prove badge ownership. Users may wish to selectively prove possession of specific credentials or roles. For example, a user might be one of a set of administrators for a group or one of a set of users with known expertise. To support this, our system allows a user to prove ownership of a particular badge without revealing any unrelated attributes. As defined in Algorithm 8, the Badge method verifies that the user is not revoked and that the claimed badge matches one of the explicitly stored badge slots within the zk-object. The proof confirms that for a given index `i`, the badge at position `i` equals the claimed badge, enabling disclosure of application-specific privileges while preserving the privacy of other badges or metadata.

B. Evaluation

As shown in Table Table II, the evaluated features primarily increase the complexity of the underlying zero-knowledge statement rather than introducing additional protocol rounds or server-side computation. Consequently, the dominant overhead remains proof generation on the client.

[8]As noted in Section IV-C, `reputation` is stored as a field element, and thus cannot decrement below 0. As such, we encode the baseline to be half the maximum field value set by a group, allowing the value to be adjusted in either direction.

Algorithm 8 Badge Ownership

```
1: procedure Badge(obj, x = (badge, i))
2:     assert(obj.revoked = false)
3:     match i:
4:         case 1: assert(obj.badge1 = badge)
5:         case 2: assert(obj.badge2 = badge)
6:         ...
7:     return
```

APPENDIX C
CALLBACK INVOCATION

Table III reports the computational costs associated with callback invocation and epoch updates. When a moderation action is issued, the server posts the corresponding callback to the bulletin board and commits to the moderation-invoked state, incurring the per-callback cost reported as Post Callback. Upon the lapse of the current epoch, the server performs an epoch update that linearly scans callbacks issued since the previous epoch to update per-user epoch state, with the resulting amortized per-callback cost reported as Update Epoch. As shown in the table, both operations incur low server-side overhead and are amortized across many moderation actions, and therefore do not lie on the critical path of message delivery.

TABLE III: Callback Invocation

Operation	**Post Callback**	**Update Epoch**
Executor	Server	Server
Unit	ms	ms
Moderation Action	3	8.91

APPENDIX D
BREAK-EVEN ANALYSIS

We define the *break-even callback factor* as the minimum number of callbacks x at which folded scanning achieves the same cost as baseline iterative scanning. This factor is computed separately for the client (proof generation) and the server (proof verification) using the empirical cost components reported in Table V.

Client-side break-even. For folded scanning, the client-side cost of processing x callbacks is modeled as

$$T_{\text{fold}_c}(x) = \mathsf{ProofCostPerCallback} \cdot x + \mathsf{ProofBlind}$$

where ProofCostPerCallback is the per-callback folding cost and ProofBlind is the one-time cost of blinding the aggregated proof. Baseline iterative scanning incurs linear cost

$$T_{\text{base}_c}(x) = \mathsf{ProofGen}_{\text{baseline}} \cdot x$$

where $\mathsf{ProofGen}_{\text{baseline}}$ is the per-callback proof generation time in the original zk-promises protocol [4]. The client-side break-even factor is obtained by equating these expressions, $T_{\text{fold}_c}(x) = T_{\text{base}_c}(x)$, which yields

$$x = \frac{\mathsf{ProofBlind}}{\mathsf{ProofGen}_{\text{baseline}} - \mathsf{ProofCostPerCallback}}$$

Server-side break-even. On the server, folded scanning aggregates all callbacks into a single proof verified once, incurring constant cost

$$T_{\text{fold}_s} = \mathsf{ProofVerify}_{\text{folding}}$$

In contrast, baseline verification scales linearly with the number of callbacks,

$$T_{\text{base}_s}(x) = \mathsf{ProofVerify}_{\text{baseline}} \cdot x$$

The server-side break-even factor is therefore defined by

$$\mathsf{ProofVerify}_{\text{baseline}} \cdot x = \mathsf{ProofVerify}_{\text{folding}}$$

which gives

$$x = \frac{\mathsf{ProofVerify}_{\text{folding}}}{\mathsf{ProofVerify}_{\text{baseline}}}$$

All reported break-even values in Table V are computed directly from these expressions using measured client- and server-side costs.

Empirical evaluation. We present the amortized, per-callback proof generation times (**Amortized Cost\Callback**) and proof sizes (**Proof Size**) in Table V. As the batch size increases, the per-callback performance of the client improves, as there are fewer times the client needs to run the folding procedure; these savings outpace the increased cost of the final blinding step, which is proportional to n. The concrete size of the final proofs, however, also increases with n. As such, our solution is configurable and permits different tradeoffs between proof size and amortized proving cost depending on anticipated usage patterns.

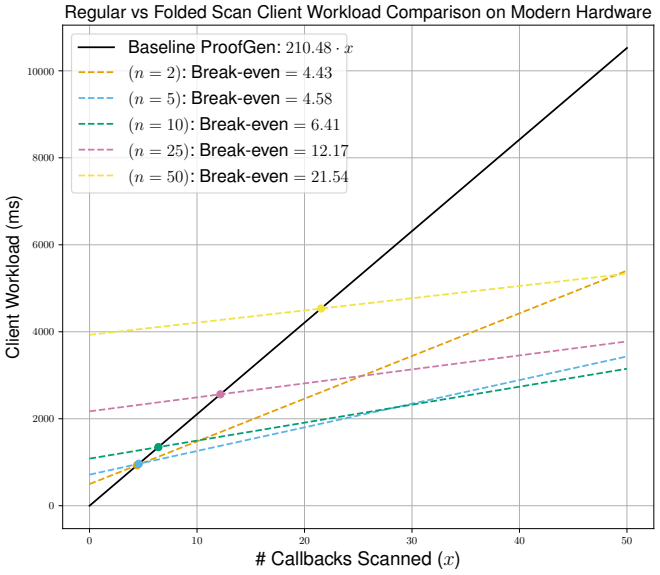


Fig. 8: Client-side workload on *modern* hardware, comparing regular scanning (solid) and folded scanning (dashed) for batch sizes $n \in \{2, 5, 10, 25, 50\}$. Dots indicate break-even points where folded scanning becomes more efficient.

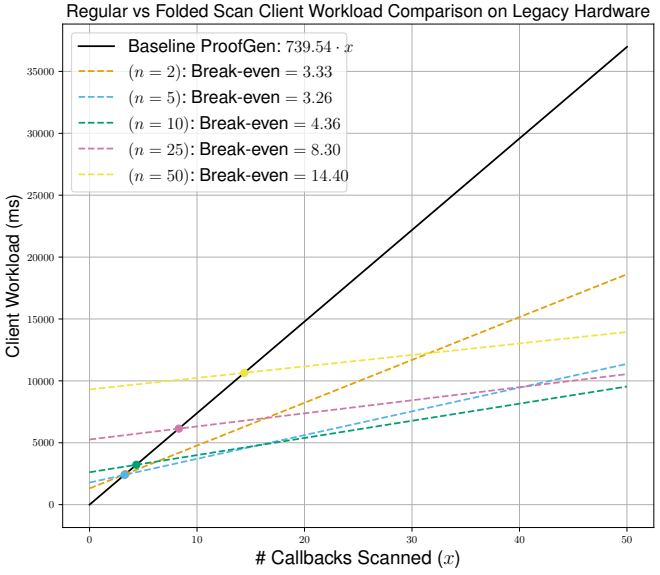


Fig. 9: Client-side workload on *legacy* hardware, comparing regular scanning (solid) and folded scanning (dashed) for batch sizes $n \in \{2, 5, 10, 25, 50\}$. Dots indicate break-even points where folded scanning becomes more efficient.

We further evaluate the concrete behavior of folded scanning in Figures 8 and 9, which plot the client-side proof generation cost as a function of the number of callbacks x under both baseline iterative scanning and folded scanning for batch sizes $n \in \{2, 5, 10, 25, 50\}$. The solid black line represents the baseline cost $\mathsf{ProofGen}_{\text{baseline}} \cdot x$, while dashed curves correspond to folded scanning at different batch sizes n, modeled

TABLE IV: Micro-benchmarks: Folded Scanning & Break-even Callback Factor, with $x = 100$ total callbacks across folding batch sizes n. Results are shown for modern and legacy hardware.

Operation Executer	**n**	**ProofCostPerCallback** Client		**ProofBlind** Client		**ProofVerify** Server	**ServerOverhead** Server	**Break-even** Client		**Break-even** Server
Hardware Unit	#	*Modern* ms	*Legacy* ms	*Modern* ms	*Legacy* ms	*Modern* ms	*Modern* ms	*Modern* #	*Legacy* #	*Modern* #
Optimized: Folded Scan for Revocation	1	173.0	606.1	426.7	1136.3	9.3	64.4	11.4	8.5	4.7
	2	98.2	346.0	498.1	1312.1	9.8	72.7	4.4	3.3	4.9
	4	61.4	217.8	646.4	1666.3	10.0	82.7	4.3	3.2	5.0
	5	54.4	191.8	715.1	1783.2	10.6	90.2	4.6	3.3	5.3
	10	41.3	138.5	1083.5	2618.2	12.5	113.0	6.4	4.4	6.2
	20	33.6	114.3	1807.0	4471.8	14.4	152.4	10.2	7.2	7.2
	25	32.1	105.6	2171.0	5264.2	15.9	173.8	12.2	8.3	8.0
	50	28.1	92.7	3929.1	9311.2	20.0	252.7	21.5	14.4	10.0

TABLE V: Folded Scanning & Break-even Callback Factor, with $x = 100$ total callbacks across folding batch sizes n. Results are shown for modern and legacy hardware.

Operation Executer	**n**	**Proof Size**	**Amortized Cost/Callback** Client	
Hardware Unit	#	MB	*Modern* ms	*Legacy* ms
Baseline: Scan for Revocation	1	2.56×10^{-4}	210.5	739.5
Optimized: Folded Scan for Revocation	1	4.64	177.3	617.5
	2	5.48	103.2	359.1
	4	7.14	67.9	234.5
	5	7.97	61.6	209.6
	10	12.12	52.1	164.6
	20	20.43	51.67	159.0
	25	24.59	53.81	158.2
	50	45.36	67.4	185.8

as $\mathsf{ProofCostPerCallback} \cdot x + \mathsf{ProofBlind}$. Intersection points (marked by dots) indicate the client-side *break-even* callback factor, i.e., the smallest number of callbacks for which folding becomes more efficient than the baseline construction [4]. As predicted by the analytical cost model, increasing the batch size n decreases the marginal per-callback folding cost while increasing the fixed proof blinding overhead, inducing a tradeoff that shifts the break-even point across hardware profiles.

On legacy hardware, where baseline proof generation is comparatively expensive, folding becomes beneficial at small callback volumes (e.g., $x \geq 4$ for $n = 4$). On modern hardware, the break-even point occurs at slightly larger but similar volumes (e.g., $x \geq 5$ for $n = 5$), reflecting lower baseline costs but the same asymptotic behavior. More broadly, folding becomes beneficial at relatively modest usage levels across all batch sizes considered. Even the largest configuration we evaluate ($n = 50$) reaches its break-even point after only 14.4 callbacks, which remains below typical daily messaging activity (e.g., ≈ 17.5 messages/day [72]). These results validate the analytical model and suggest that users would realize the concrete performance benefits of folding during ordinary day-to-day use.

Operation	**n**	**ExtractBody**	**LoadServerParams**	**DeserializePayload**	**ProofVerify**	**AppendBulletin**	**ServerOverhead**
Executor		Server	Server	Server	Server	Server	Server
Unit	#	ms	ms	ms	ms	ms	ms
Folded Scan	1	8	40	5	8	< 1	64.4
	2	9	45	9	8	< 1	72.7
	4	12	46	10	9	< 1	82.7
	5	13	48	11	9	< 1	90.2
	10	17	63	14	11	< 1	113.0
	20	25	87	20	14	< 1	152.4
	25	26	90	23	15	< 1	173.8
	50	41	151	39	20	< 1	252.7

TABLE VI: Server-side processing time (ms) for folded scan operations with varying batch sizes (n). These numbers were collected from a single run, and thus are not averaged across runs.—and differ very slightly from the results in Table V due to natural variation. While core operations scale moderately with n, the total server load reflects additional overhead from processing larger proofs. These implementation-level costs, such as memory and I/O overhead, are orthogonal to our core contributions and can be optimized in future engineering work.

APPENDIX E
TOTAL SERVER OVERHEAD FOR FOLDING

Although folded scanning requires only a single proof verification per batch, the measured ServerOverhead in Table V exceeds ProofVerify. To isolate the source of this gap, we instrument the validation service and report a fine-grained breakdown of server-side execution in Table VI.

Each column corresponds to a distinct processing phase. ExtractBody captures request ingestion and buffering, which grows with proof size. LoadServerParams measures acquisition and cloning of shared verification parameters (e.g., R1CS, folding keys, hash configuration), which dominate in our unoptimized prototype. DeserializePayload reflects decoding and structural validation of the folded proof. ProofVerify measures Nova verification plus application-level checks, while AppendBulletin records persistence of verified results.

As batch size n increases, total server cost rises primarily due to proof-size–dependent system overheads (memory allocation, serialization, and parameter handling), rather than cryptographic verification itself. These effects explain why ServerOverhead is not the sum of individual components and why verification remains a minor fraction of end-to-end runtime. Importantly, these costs are orthogonal to the folding construction: verification remains constant in k, and the additional overhead reflects engineering artifacts of the current implementation. Standard optimizations—such as parameter caching, zero-copy deserialization, and batched memory management—can substantially reduce this overhead without modifying the underlying cryptographic protocol.