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 communities 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. We develop two significant improvements to existing work on anonymous banning systems 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 a variety of settings, and show that they are concretely efficient enough for deployment.**

1. Introduction

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 of anonymity derive from *completely anonymous* systems, in which an anonymous poster could truly be anyone and the audience is everyone. However, anonymity is often wielded in good faith in the context of an established community to great effect: 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 can conduct an extended discussion under pseudonyms [2], [3]. 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, these same tools can be abused.

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 [4], [5] rather than ring signatures [6] (or linkable ring signatures for pseudonymity [7]). But this approach is still one of extremes—jumping from anonymous to non-anonymous—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 [8]. And anonymity may be most 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 [9]), and moderation teams ban these accounts when they transgress community 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. Platforms can attempt to fingerprint users or require difficult-to-change identifiers, e.g., a phone number, on registration, but this undermines the accessible anonymity they aim to provide.

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

*Joint first authors. Author order determined by a coin flip.
†Joint senior authors. Author order determined by a coin flip.

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 a long work of simple anonymous banning systems first introduced by Tsang et al. [10] and extended by many recent works [11]–[13]. Critically, however, users must hold updateable *state*, which was made possible by the recent work of Shih et al. [13].

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 which 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 (or a subset, if appropriate).[1]

We anticipate that cryptographic personas will be most valuable when used *within communities*, 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. As mentioned above, recent work on anonymous banning systems [13] 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 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 [13] 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.

1. To our knowledge, [14] was the first project to propose combining pseudonyms with anonymous credentials. Our work take 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.

1.1. Our Contributions

In this work, we make the following contributions:

- 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 from which they can send messages, but each of these personas is cryptographically connected such that punishments applied to any one of a user's personas will automatically propagate to all of that 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 [13], [13].
- We develop a way to push the expensive proof generation step required during the authentication in Shih et al.'s [13] construction into a pre-processing step without compromising user privacy. Our technique builds on Privacy Pass [15] 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 [16]–[20] into the construction of Shih et al. [13] 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 [13].
- We implement cryptographic personas and integrate our implementation with Signal, facilitating the generation of cryptographic personas within Signal group chats. We provide detailed micro-benchmarks demonstrating that cryptographic personas are concretely efficient to use.

Cryptographic Persona and Identity. While the current version of this work does not explicitly explore this connection, it is clear that cryptographic personas are only valuable

when integrated into a system with some underlying notion of identity. Minimally, we require a mechanism to prevent Sybil accounts, since revocation is worthless if users can trivially make another account. One obvious solution is to compose a cryptographic persona with an anonymous credential derived from legal identity documents or some other Sybil resistance mechanism.

To the best of our knowledge, composing exiting identity documents with anonymous credentials was first proposed in zk-creds [21], which used zero-knowledge proofs over legacy identity documents (e.g., passports[2]) as "zk-supporting-documentation." This proof is then used in issuance for a new full-featured anonymous credential scheme that leverages zkSNARKs for programmable statements, allowing for the composition of identity assertions like age verification with anonymous credential features like pseudonymity and rate-limiting. In contrast, in more recent work Google Wallet [22] deployed a scheme that forgoes this composition. Instead, it uses the zero-knowledge proof over the identity document directly as a limited form of credential, building on a line of work on zero-knowledge proofs for legacy signatures [23], [24] to realize a folklore limited purpose construction. This approach is enabled by the innovative work of Frigo and Shelat [25] that builds a specialized high-performance zero-knowledge proof to parse legacy identity document formats and verify their signatures. Remarkably, this process is fast enough to run even on older mobile devices, allowing the protocol to simply generate a fresh proof for every credential show. However, without the additional composition step, Google's use of this scheme comes at the cost of flexibility, as the credential can only assert claims already present in the original document.

These larger notions of privacy preserving digital identity, broadly construed, promise to be particularly important as the emerging threat of AI actors push more digital services to require demonstrations of real-world identity. Tying cryptographic personas to such these real-world credential would give us a way to have pseudonymous speech on the internet from known to be human parties.

2. Overview and Use Cases

The novel conceptual insight behind cryptographic personas is that the power to post anonymously can be revoked *selectively* for the small subset of users who demonstrate that they cannot be trusted to use the power responsibly. Specifically, all users start with the ability to post anonymously (or pseudonymously), but the community can revoke that ability for users who share content that is not in line with community standards. 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). While this does not prevent abusive content from being shared on a platform, it severely limits the damage individual, abusive users can inflict and ensures that communities are eventually composed of individuals who will use anonymity conscientiously.

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" [26]) and want to add clarifications in response to questions. Similarly, a user in a support community might want to share about their progress over time under a single persona. 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.

2.1. Technical Sketch

Implementing personas as a user-interface feature 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 there is no party that 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 [27] to hide a message sender's identities from the server and has implemented a private group protocol to prevent the server from learning the members of groups. Thus, in order to make cryptographic personas valuable in both these situations, we require that the cryptographic personas provide strong anonymity for each persona—even against a corrupted server.

A natural starting point for constructing cryptographic personas are anonymous signature variants, like ring signatures [6], linkable ring signatures [7], or group signatures [4], [5], as these allow users to validate themselves as a registered member of a group or platform without revealing their identities. These primitives, however, do not naturally provide a means to revoke in the manner described above.[3] Thus, we require a different technical starting point.

Instead, we build cryptographic personas from the powerful zk-promises primitive introduced by Shih et al. [13]. zk-promises can be seen as a *stateful* anonymous credential [29] that allows for callbacks—that is, the server can force the holder of the anonymous credential to update some private

2. Many passports and some national ID cards often contain a signed digital copy of the ID in an RFID chip or smart card.

3. "Revocable" ring signatures in the literature (e.g., [28]) 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.

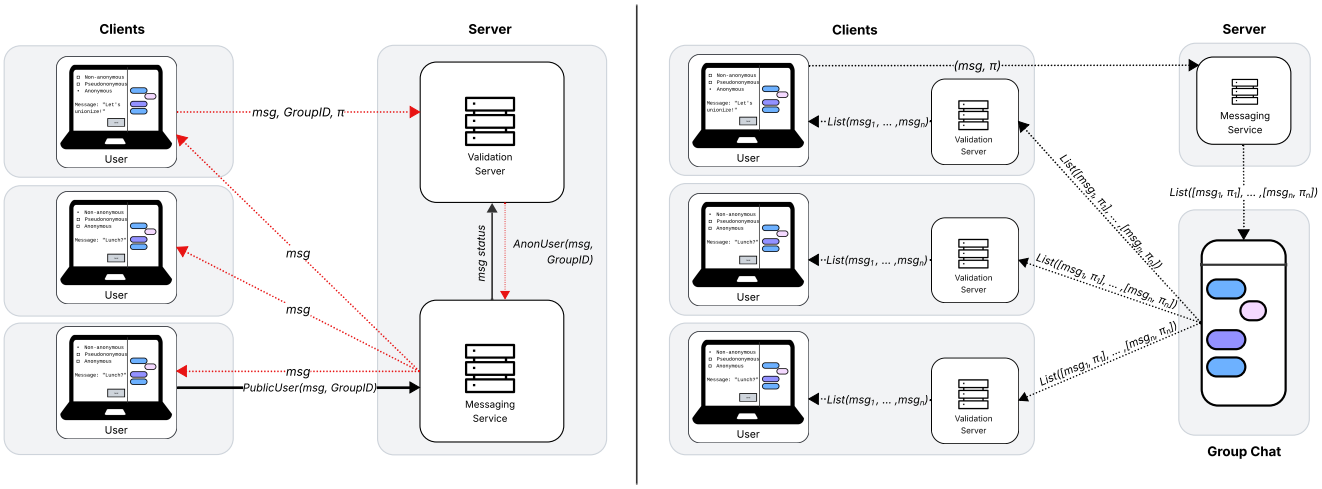


Figure 1: The different system architectures we envision for deploying cryptographic personas, illustrated for a group messaging platform. On the left, we depict the "cryptographic personas as infrastructure" mode. Specifically, users prove that they have access to a pseudonym to the validation server using π, and the server then forwards the message msg onto the main messaging service. This visualization also depicts the "cryptographic personas as a service" mode (as the depicted validation server can be run by a third party). On the right, we illustrate the "serverless cryptographic personas" mode, which lets users run the validation server logic locally. Specifically, the group chat serves as shared state consisting of message-proof tuples $(msg_1, \pi_1), \ldots, (msg_n, \pi_n)$. User devices download this full state and locally check all proofs and display the messages msg_i for which π_i verifies. Solid arrows indicate standard message delivery operations (always present), while dotted arrows represent the additional anonymous/pseudonymous protocol steps introduced by our construction.

state before they authenticate (anonymously) to the server again in the future. Importantly, the user cannot roll their state back, which means that the private state can be seen as oblivious storage for the server, and the user can prove predicates over their private state each time they authenticate to the server. We summarize the zk-promises construction in Section 3.

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. As we discuss below, however, there are non-trivial modifications that we need to develop in order to make this construction practical.

System Architecture. Thus far, we have described cryptographic personas in the context of a client–server model, but in practice, there are three distinct modes in which cryptographic personas could be deployed:

(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 this mode of operation on the left-hand side of Figure 1, where the service provider in question is a message service.

(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 part 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 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. In this model, the group chat itself serves as a shared log of operations, allowing clients to au-

thenticate state locally.[4] This requires modifying clients to generate proofs for messages they send and validate those they receive, but leaves the server unchanged. However, a mechanism for clients to send messages anonymously is still required, either through a shared anonymous account or alterations to the messaging protocol. We illustrate this deployment architecture on the right-hand side of Figure 1.

New constructions for real-time performance. There are two main barriers that must be overcome in order to make cryptographic personas based on zk-promises concretely practical:

(1) We must streamline the authentication interaction between users and the service provider when sending a message. In our measurements described in Section 5.1, 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.
(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 of the 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 measurements described in Section 5.1, we find that this process is prohibitively expensive (both for clients and servers).

We explore the following two protocol improvements to make zk-promises sufficiently performant:

Pushing authentication into pre-processing. We observe that proof generation, which is done when a user wants to issue a request in zk-promises [13], can be pushed to a pre-processing phase using the techniques developed in Privacy Pass [15]. 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 can exchange the zero-knowledge proofs from zk-promises for tickets (e.g., at the beginning of each day) that can be used to send messages or post content later, making sending many messages or making many posts sequentially concretely feasible. Doing this naïvely, however, results in authentication sessions that are linkable. This is because the statement in the proof that must be verified by the verification server contains a callback identifier that must later be presented to the verification server. Instead, during ticket issuance, we have the client prove in zero-knowledge that their local state (the zk-object) is correctly formatted with respect to *a* callback identifier, which will later be revealed during ticket redemption. We note, however, that these tickets would remain valid *even after the ticket holder's ability to use personas has been revoked*. Thus, we think of this optimization as an "optimistic mode," in which the expectation is that revocation rarely happens or has a lag-time before revocations take effect; of course, in an emergency, all outstanding tickets can be revoked in case this optimism was not well-founded.

Folding update proofs. In zk-promises, clients must 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. Note that the circuit for this proof would naturally reveal the number of previous messages sent—a clear privacy violation. Instead, Shih et al. consider two options: (a) padding out the circuit to have some maximum number of allowed callback handles. This is clearly highly inefficient, as this means all users must prove 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, perhaps, working in fixed-size batches). This approach is the one implemented in their work. Concretely, this results in quadratic load on the server (each message requires an interaction *for each previous message*). Clearly, this is prohibitive overhead. We observe that the structural problem here is that the length of the list is unknown to the server and cannot be revealed. As such, we require a proof system that can easily adapt to a variable-length statement without requiring worst-case complexity. We observe that recent proof folding techniques [16]–[20] are the perfect tool for this setting. Specifically, the user can simply download the set of callbacks and then locally iterate through the callback handle list, providing a proof for each element in the list before folding the instances into one another. 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 would verify) at the expense of true succinctness.

In Section 5.1, we show that these optimizations are crucial for getting cryptographic personas to work in practice. Sending many messages or performing frequent moderation checks would add noticeable delays, making the system too sluggish for everyday chat apps like Signal. Our optimizations fix this by pushing proof work off the critical path and folding expensive checks, making cryptographic personas fast enough for real-world group messaging.

2.2. Use Cases

We illustrate the value of cryptographic personas with several representative use cases with differing community sizes and internal power structures.

Departmental communication. Academic departments rely on collaboration and deliberation among faculty members with differing perspectives and priorities in order to make critical decisions on curricula, hiring, promotion, and resource allocation. At the same time, academic departments

4. zk-promises, the starting point for our constructions, supports authenticating state either via server-side signatures or by proving membership in a log.

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, which could be used to quickly gauge faculty opinions. Faculty could be limited to one persona each day or one persona per agenda item during a faculty meeting. At the same time, the accountability features of cryptographic personas—possibly managed by the department chair or democratically—help ensure that anonymous posts stay professional and amicable.

Q&A platforms. As we discuss in Section 6, anonymous posting results in more honest and eager engagement by users on Q&A platforms [30]–[34], but comes with the risk of trolling and harassment [35]–[37]. One example might be a Q&A Signal group containing all of the PhD students within an academic department, in which junior students might want to ask valuable questions but fear appearing foolish. Cryptographic personas offer question-askers and answerers the ability to post anonymously. Both question-askers and answerers can be limited to one persona per question, in order to prevent a single answerer from making it appear as though there is a consensus among answerers. Cryptographic personas could also be extended to achieve "meronymous" communication, as explored by Soliman et al. [34] within the context of discussion platforms, in which posters reveal a subset of the user population of which they are a part (e.g., they are one of a subset of experts on a topic). In order to govern access to cryptographic personas, the platform could use administrators or topic-specific moderators, a successful strategy on many discussion platforms [38]–[41].

Support communities. Support communities are composed of members who share their experiences, struggles, and progress as they move through difficulties. As the topics discussed are often highly sensitive, there can be tremendous value in providing members of these communities with pseudonymity—both from each other and from the service provider itself. Members may want to assume different pseudonyms in different contexts, e.g., when posting an update vs. sharing encouragement. At the same time, it might be helpful for users to establish a reputation for being a supportive member of the community over time before they are allowed to become moderators in the community or share their own stories. Cryptographic personas that allow for building a unified reputation could, therefore, be incredibly valuable as a tool within these contexts.

Activism coordination. Finally, we consider activist coordination, including large, semi-open messaging groups that have been used in recent protest movements [42]. Historically, these groups have been run on platforms that allow for throw-away accounts (e.g., Telegram) in recognition of the anonymity requirements of these users. Another alternative would be to run these groups on encrypted platforms and provide users with pseudonymity using cryptographic personas. These groups often have leaders who are in charge of coordination. Cryptographic personas could allow leaders to create specially annotated personas that reveal only that the underlying user is a member of leadership, but suppresses *which* member of leadership is speaking. Looking ahead, we implement this feature with "badges" that can be attached to user profiles and associated with derived pseudonyms. This minimizes the digital footprint that leaders leave behind in case the group itself is infiltrated.

3. 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 (or, equivalently, be able to reason offline about the contents of the bulletin board based on, e.g., signatures) as $\mathsf{Algo}^{\mathsf{BB}}$.

zk-promises [13]. 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.
- $\textsc{ExecMethodAndCreateCall}(\mathsf{pp}, \mathsf{obj}, \mathsf{pk_{sp}}, \mathsf{meth}, x, w) \to (\mathsf{obj}', \pi, \mathsf{cbData} = \{\mathsf{tik}, \mathsf{obj.sn}\}, \mathsf{aux})$[5] takes in the public parameters pp, the user's current private state obj, the service provider's public key $\mathsf{pk_{sp}}$, 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', a proof of validity π, callback handle $\mathsf{cbData} = \{\mathsf{tik}, \mathsf{obj.sn}\}$ containing at minimum a callback ticket tik and the serial number $\mathsf{obj.sn}$, and auxiliary data aux. After running this function, the client must post cbData onto the bulletin board BB.
- $\textsc{VerifyCreate}^{\mathsf{BB}}(\mathsf{pp}, \mathsf{sk_{sp}}, \mathsf{obj}', \pi, \mathsf{cbData}, \mathsf{aux}) \to \{0, 1\}$ has oracle access to the bulletin board BB, takes in the public parameters pp, the service provider's secret key $\mathsf{sk_{sp}}$, a commitment to the users updated private state obj', a proof of validity π, callback handle cbData, and auxiliary data aux and outputs a binary value.
- $\textsc{Call}(\mathsf{pp}, \mathsf{tik}, \mathsf{args}) \to (\mathsf{tik}, \mathsf{args})$ 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.
- $\textsc{VerifyCall}^{\mathsf{BB}}(\mathsf{pp}, \mathsf{tik}, \mathsf{args}, \mathsf{aux}) \to \{0, 1\}$ 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.

5. Shih et al.'s [13] initial algorithmic interfaces do not expose auxiliary witness data w that is not held in obj. Some features in our construction require the ability to add additional data, and making this change to their construction is trivial.

– $\text{SCANONE}^{\mathsf{BB}}(\mathsf{pp}, \mathsf{obj}, x) \rightarrow (\mathsf{obj}', \pi, \mathsf{cbData} = \{\mathsf{tik}, \mathsf{obj.sn}\})$ 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', update proof π, and callback handle $\mathsf{cbData} = \{\mathsf{tik}, \mathsf{obj.sn}\}$.
– $\text{VERIFYMETHODEXEC}^{\mathsf{BB}}(\mathsf{pp}, \mathsf{obj}', \pi, \mathsf{cbData}) \rightarrow \{0, 1\}$ has oracle access to the bulletin board BB, takes in the public parameters pp, updated user private state obj', update proof π, and callback handle $\mathsf{cbData} = \{\mathsf{tik}, \mathsf{obj.sn}\}$ 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). This process also updates the list of used callback handles within obj to include the one stored in cbData. π can reason over obj and, depending on the statement proved, 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 runs VERIFYCREATE to check if the request is authorized, and processes the request if it is. If the service provider wants to invoke a callback, it uses the CALL interface and sends the arguments to the callback onto 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, the server can batch callback invocations into *epochs* and callbacks eventually expire, in that they can no longer be invoked using CALL. The server gathers callbacks, and then, when they are ready to progress to the next epoch, they make all of these pending callbacks active. Users can demonstrate that they are up to date with respect to all active callbacks, rather than requiring them to stay in one-to-one lockstep.[6] 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. In sum, 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 ideal world-real world paradigm; we refer readers to the original work [13].

Privacy Pass [15]. 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.

6. In principle, this system can be easily adapted to allow users to scan through a list of all posted callbacks (e.g., banned personas) rather than the callback handles associated with the user. This may be more efficient when the expected workload involves each user making many posts but very few invocations of callbacks.

4. Cryptographic Personas

We now give our formal definition and construction of cryptographic personas, additional usability features, and constructions key for real-time systems.

4.1. 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 2.1), 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 in infrastructure or as-a-service mode, this is likely to include the server party.

The ideal functionality exposes five main interfaces in addition to registration: (1) sending a message with a new persona, (2) sending a message with an existing persona, (3) receiving messages, (4) merging personas, and (5) revocation of abusive users.[7] When sending a message using either interface (1) or (2), the calling party provides a message m, a context context, and a set of recipients recipients. Each message results in a pending delivery that is eventually retrieved by the recipients 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 two 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 4.2, we can imagine this function simply returning `true`; we explore another concrete instantiation of this function in Section 4.3

7. We note that we have chosen not to model registration in our system, but rather let the protocol participants be fixed ahead of time. In part, we make this decision because the definitions of the primitives on which we construct cryptographic personas are modeled using static participation. Clearly, dynamic registration can be handled by simply generating new instances of the ideal functionality, or modifying the ideal functionality with a registration interface is trivial.

Algorithm 1 Cryptographic Personas: Client-Server Protocol

Parameters:

H	▷ collision- and second-preimage-resistant hash function
Com	▷ commitment scheme
pp $\coloneqq$ SETUP(Φ)	▷ zk-promises state transitions
BB	▷ authenticated bulletin board (or log)
(pk_{sp}, sk_{sp})	▷ server keys
context	▷ a context where context $\in$ AllowedContexts
AllowedToCreateNewPersona, AllowedToRevoke	▷ policy hooks
obj $= (\mathsf{sk}, \mathsf{revoked}, \mathsf{rep}, \mathsf{badge}_1, \ldots)$	▷ client's private zero-knowledge state

Phase 1 - Registration

Client

1: Initialize new zk-object obj
2: obj.sn $\leftarrow\$ \{0,1\}^\lambda$ ▷ random serial number updated per generated callback
3: obj.sk $\leftarrow\$ \{0,1\}^\lambda$ ▷ consistent secret key
4: obj.revoked $\coloneqq 0$ ▷ pseudonymity initially not revoked
5: obj.rep $\coloneqq 0$ ▷ see Section 4.3 for alternatives
6: $\mathsf{com}_{\mathsf{obj}} \coloneqq \mathsf{Com}(\mathsf{obj}, r)$ for $r \leftarrow\$ \{0,1\}^\lambda$
7: $\mathsf{com}_{\mathsf{obj}} \xrightarrow{\text{send}} \mathsf{BB}$ ▷ zk-promises implicitly commits all zk-objects on BB. $\mathsf{com}_{\mathsf{obj}}$ could be linked to long-term identity to prevent Sybil attacks.

Phase 2 - Send Message

Client

Require: Message m, context context, recipients recipients, zk-object obj

1: nonce $\leftarrow\$ \{0,1\}^\lambda$ ▷ skip this step for sending message with existing persona
2: persona $\coloneqq H(\mathsf{obj.sk} \,\|\, \mathsf{nonce})$ ▷ client-chosen handle
3: $(\mathsf{obj}', \pi, \mathsf{cbData}, \mathsf{aux}) \coloneqq$ ▷ produces updated obj', ZKP π, and the callback handle
EXECMETHODANDCREATECALL(
$\mathsf{pp}, \mathsf{obj}, pk_{sp}, \mathsf{Persona}(\mathsf{obj}, (\mathsf{persona}, \mathsf{nonce})), \perp)$ ▷ alternatively use $\mathsf{PersonaLimited}_k$
4: obj $\coloneqq$ obj'
5: cbData $\xrightarrow{\text{post}}$ BB
6: $(\pi, \mathsf{cbData}, \mathsf{aux}, m, \mathsf{context}, \mathsf{persona}) \xrightarrow{\text{send}}$ Server

Validation Server

7: $b \coloneqq$
VERIFYCREATE$^{\mathsf{BB}}(\mathsf{pp}, sk_{sp}, \mathsf{obj}', \pi, \mathsf{cbData}, \mathsf{aux})$
8: **if** $b = 1$ **then** ▷ In serverless mode, each P $\in$ recipients simulates the validation
9: $(\pi, \mathsf{cbData}, \mathsf{aux}, m, \mathsf{context}, \mathsf{persona}) \xrightarrow{\text{send}} \mathsf{P}$ server locally

Phase 3 - Revocation

Client

Require: Evidence evidence, ticket tik, context context, persona persona

1: $(\mathsf{evidence}, \mathsf{tik}) \xrightarrow{\text{send}}$ Server

Validation Server

2: **if** AllowedToRevoke(P, persona, context, evidence, `time`) $=$ `true` **then**
3: args $\coloneqq$ (persona, evidence, context, ...) ▷ application-specific logic
4: $(\mathsf{tik}, \mathsf{args}) \coloneqq$ CALL(pp, tik, args)
5: $(\mathsf{tik}, \mathsf{args}) \xrightarrow{\text{send}}$ BB ▷ in serverless mode, call is replicated to local views

Client

6: Download new callback records from BB
7: **for all** stored callback handles in obj that are now active **do**
8: $(\mathsf{obj}', \pi, \mathsf{cbData}) \coloneqq$ SCANONE$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{obj}, \mathsf{args})$ ▷ for folding optimization, all callbacks consumed simultaneously
9: obj $\coloneqq$ obj'
10: $(\mathsf{obj}', \pi, \mathsf{cbData}) \xrightarrow{\text{post}}$ BB

Bulletin Board

11: **if** VERIFYMETHODEXEC$^{\mathsf{BB}}(\mathsf{pp}, \mathsf{obj}', \pi, \mathsf{cbData}) = 1$ **then**
12: BB $\leftarrow$ BB $\cup \{(\mathsf{obj}', \mathsf{cbData})\}$

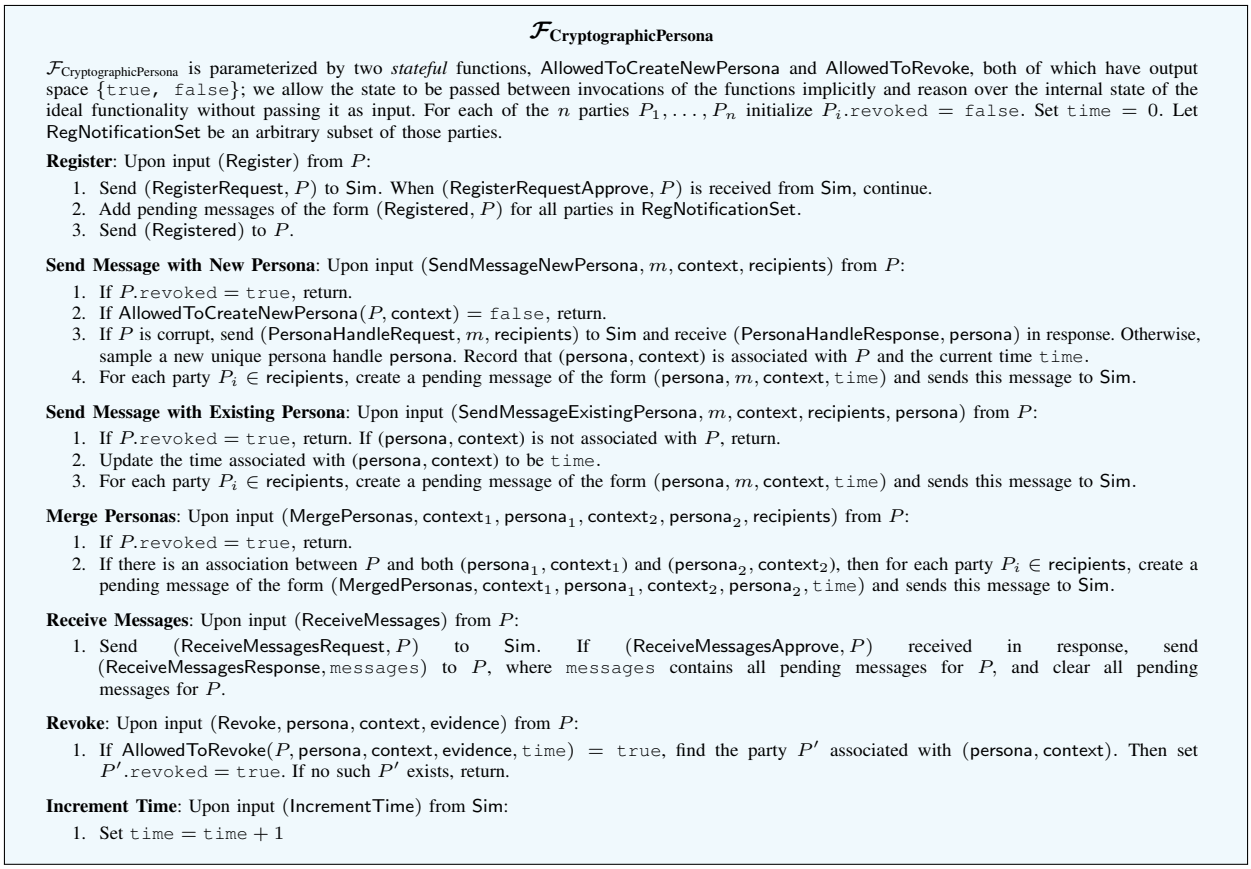


Figure 2: An Ideal Functionality that captures the ideal properties of cryptographic personas.

AllowedToRevoke: As discussed in Section 2, there are numerous ways in which a community might want to manage the power to revoke users' access to cryptographic personas. For example, 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 a thread moderator). Alternatively, the members of the community could decide democratically. AllowedToRevoke takes the identity of the party attempting to call Revoke and some arbitrary evidence and decides if the revocation should happen. For unilateral revocation, this can simply check if P is an authorized party, and for democratic revocation, the evidence string can carry a certified vote. Additionally, AllowedToRevoke allows us to scope which personas are viable for revocation; in practice, our implementation of zk-promises is most efficient when it only needs to consider reasonably recent callbacks. As such, AllowedToRevoke should only allow revocation on personas that have been used recently.

We give a sketch proof showing that the construction outlined below realizes this ideal functionality in Section A.

4.2. Basic Construction

We give an algorithmic description of our basic construction in Algorithm 1.

We begin by specifying the private state that each user holds when running the protocol in Figure 1. It is outlined in Struct 1. Because zk-promises is designed to prove statements in the form of arithmetic constraints, each of these values is encoded as a field element. During initialization, the user samples a secret key and initializes `revoked` to be `false` (i.e., 0 as a field element). 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 set by the group. Alternatively, the group may choose any other initial credibility score. The types and amount of badges are also application-specific. We note that for our basic construction, we only make use of the first two fields.

Struct 1: `MsgUser` zk-object

```
sk          // secret key
revoked     // revoked status
rep         // credibility score
badge1      // variable number of badges
badge2
...
```

Send message with new persona. 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 2. 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 $\mathcal{H}$.

Algorithm 2 Pseudonymous Posting

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

Send message with existing persona. 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.

Revoke. 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 3 below. When appropriate, 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, effectively preventing the user from sending further anonymous or pseudonymous messages, thereby enforcing the collective decision while preserving the anonymity of the voters. 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. If the bulletin board is maintained by consensus among the users, then each client can locally check AllowedToRevoke before updating their local copy of the bulletin board.

Algorithm 3 Revocation Callback

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

4.3. Advanced Messaging Features

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

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 4. This can be done by invoking the EXECMETHODANDCREATECALL interface of zk-promises with meth = PersonaLimited, defined in Algorithm 5. 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 4 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 5, 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 5 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 6. 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 activity performed under different personas without revealing their secret key or any unrelated personas.

Vote anonymously. One particular setting in which rate-limiting responses is 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

Algorithm 6 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
```

in Algorithm 7. Note that by no longer making use of a nonce, there is 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 7 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.

4.4. Adding User Properties

Users may want to hold some information about themselves within their state that can be selectively associated with a persona. We do not model the following features within $\mathcal{F}_{\text{CryptographicPersona}}$ to keep the presentation notationally simple and because adding these extensions is trivial.

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, 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 8. 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 8 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 9, 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.

Algorithm 9 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
```

4.5. Pushing Proof Generation to Preprocessing

In the protocol described above, users must generate a new callback and zero-knowledge proof attesting to the validity of that callback each time they want to post a message. While this ensures both correctness and unlinkability, users' devices must pause while generating this proof. If users' devices are highly performant, this pause might be short enough to evade notice. On many devices—especially legacy devices and light-weight mobile devices—the pause is likely to meaningfully interrupt users' workflows (recall that research shows that users start to notice delays at 100ms and feel disrupted when delays reach 1 second [43]). Concretely, we benchmark proof generation to be

8. As noted in Section 4.2, `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.

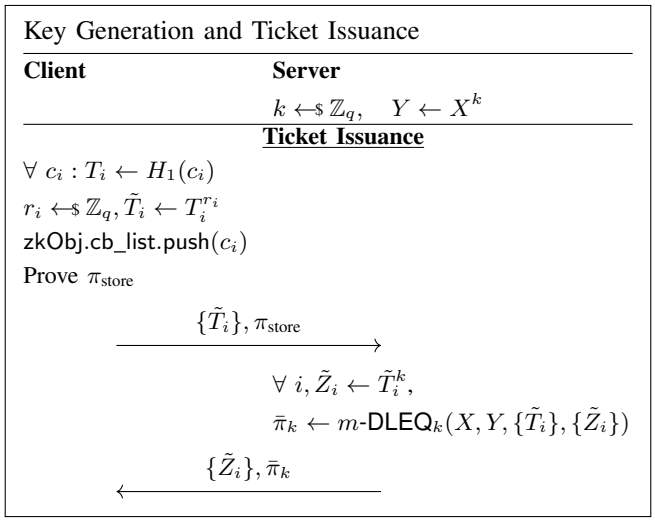


Figure 3: The server's Key Generation phase (run during setup) is shown above the line. Ticket issuance using a VOPRF with batch DLEQ proofs is below the line.

roughly 140ms on a high-end 2023 laptop (Apple M3 Max) and roughly 430ms on a 2018 legacy laptop (Apple Intel Core i5); these benchmarks are documented in Table 2. Thus, even on a slightly older or mid-tier laptop, proof generation introduces noticeable latency, and on mobile or resource-constrained devices, the computational cost is likely prohibitive, potentially interrupting normal user interactions.

In order to move proof generation into a pre-processing phase, one might consider pre-computing a batch of Groth16 proofs locally, e.g., each morning, and submitting them throughout the day as needed. However, this is not possible for two reasons: (1) each successive proof that generates a callback requires an interaction with the server[9] (in which the server signs the previous state), making strictly local computation infeasible, (2) if the client simply interacted with the server to generate a batch of callbacks each morning, this process would reveal a set of callback tickets to the server, making them linkable to one another. For example, if the client prepares a set of new personas that can be used later in the day, the server could determine that these personas are related to the same user. Thus, we require a more intricate approach to pre-processing.

We design our pre-processing technique based on Privacy Pass [15], which makes proof generation and message sending fully unlinked. Our construction enables clients to pre-authorize multiple future interactions via a single batched zero-knowledge proof, while maintaining unlinkability across subsequent message posts. The design 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.

9. This new construction is designed for the deployment configuration of cryptographic personas in which a validation server is deployed as infrastructure. Other approaches would be needed for serverless deployment.

During the ticket issuance (see Figure 3), 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 index i, it computes a blinded input $\tilde{T}_i = H_1(c_i)^{r_i}$, where $H_1 : \{0,1\}^* \to \mathbb{G}^*$ is a hash-to-curve function into a prime-order group $\mathbb{G}$ with generator X. The client constructs a zero-knowledge proof π_{store} that demonstrates knowledge of well-formed pre-images $\{c_i, r_i\}$ with respect to $\tilde{T}_i$ and the callbacks $\{c_i\}$ are stored in the users zk-object. Additionally, the user proves any application-specific predicate (e.g., reputation threshold or non-ban status).[10] 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 [15] for details. After verification, the client unblinds each response to obtain $Z_i = H_1(c_i)^k$, and derives a keyed MAC with key $K_i = H_2(c_i, Z_i)$, where $H_2 : \{0,1\}^* \times \mathbb{G} \to \{0,1\}^\kappa$ binds the token to its callback.

In the ticket redemption phase (see Figure 4), the client selects a callback and its associated PrivacyPass data (c_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 $Z_i' = H_1(c_i)^k$ 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; one-time token usage and state integrity are enforced using the zk-object established during issuance. This design enables asynchronous and stateless redemption across sessions while preserving unlinkability.

4.6. Folding

In zk-promises, callback scans are done one callback handle at a time (or in fixed batches),[11] i.e., a user loops through all previous messages they have sent in the system that *could* have had their callbacks invoked.[12] Users are

10. Note that pseudonyms should not be revealed during pre-processing, but the client can simply pre-process zero-knowledge proofs that associate a given callback with a chosen pseudonym and then submit them during ticket redemption.

11. It is also possible to configure the system such that users are responsible for scanning an explicit list of banned persona as simply proving non-membership, as is done in other anonymous banning systems e.g., [44], [11], [12]. This profile may be better when there are expected to be very few callbacks invoked and users are sending many messages. We describe this work in the more complex model as this makes it easier to maintain complex state (e.g. reputation) efficiently.

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

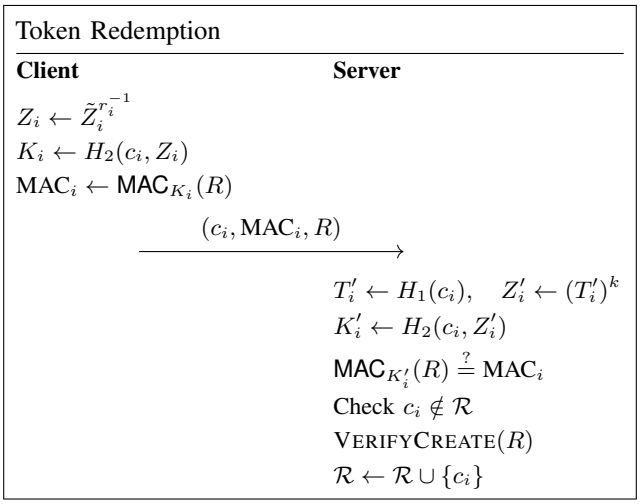


Figure 4: Redemption of a callback ticket for method execution using derived MAC. R is the client's request.

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 [16]–[20] 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 5.1.4, folding also concretely improves the proof generation time on the client by 11% (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 [12]. Specifically, we can commit to the depth, add a commitment to the depth 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.

5. Implementation and Evaluation

To evaluate the practicality of cryptographic personas, we implemented our construction using Groth16 [45] for zero-knowledge proofs, Poseidon [46] for hashing, Nova [16] as our folding scheme as implemented in [47], and Schnorr signatures over BN254 [48] for zk-promises [13].

Across all deployment applications introduced in Section 2, the core functionality is the same: clients generate and scan proofs, servers or peers verify them, and state is updated accordingly. Moreover, the dominant costs, proof generation on the client side and proof verification on the server side, will remain the same. Thus, we pick one application setting with which to integrate in order to get a better understanding of the end-to-end costs of running cryptographic personas in practice, rather than giving micro benchmarks in isolation.

As a proof-of-concept, we realized our prototype within a modern messaging service. Concretely, we integrated our validation logic into Signal via the `signal-cli-client` library [49]. We emphasize that this choice is not intrinsic—our prototype could be instantiated in other platforms such as Slack, WhatsApp, or Discord—but Signal serves as a representative example of a secure, widely-used messaging system. In this prototype, the validation server functions as a "phantom" participant that relays pseudonymous messages on behalf of clients. Each user can send anonymously by routing through this phantom participant, while receivers update state locally.

This design corresponds most closely to a combination of the infrastructure and service deployment models: the validation server is operated as a shared component, but performs a role that could equally be embedded directly into the application provider's infrastructure. Our realization necessarily breaks Signal's native end-to-end encryption, since the phantom account is managed centrally. While this specific trade-off would be unacceptable in a production deployment, it is sufficient to demonstrate feasibility and to quantify efficiency in a realistic system setting. More generally, the integration demonstrates how cryptographic personas can be incorporated into existing messaging platforms without requiring extensive changes to client software or protocol handshakes. If Signal were to integrate cryptographic personas as infrastructure into their live systems, they could make the following change to Signal clients: in each group chat, all participants have a synchronized sender key from which any group participant can send messages, and all group members use this key to send pseudonymous messages.

5.1. Evaluation

5.1.1. Experimental Setup. We report two sets of experimental results to capture performance across both modern and legacy hardware. The first set was obtained on a 2023 Apple MacBook Pro equipped with an Apple M3 Max processor (14-core CPU, 30-core GPU) and 36GB of unified memory, running macOS Sequoia 15.6. The second set was obtained using a 2018 Apple MacBook Pro 13-inch, equipped with an Intel 2.3GHz Quad-Core i5 processor and 8GB of RAM, running macOS Sequoia 15.7. While the 2018 MacBook is older, it is reasonable to assume that many users would want to use cryptographic personas on similar hardware. Moreover, we see this older device as a reasonable proxy for common mobile devices. We denote the two hardware profiles in tables as *Modern* and *Legacy*, respectively. Note that we assume the server will run with modern hardware, so we limit our server-side results to the *Modern* profile.

For both environments, we ran the modified Signal client, the validation server, and a local test deployment of the Signal server on the same machine. This setup allows us to isolate system performance without introducing network variability, while still reflecting realistic end-to-end behavior as experienced by a user running Signal Desktop or a similar integrated messaging client. Each experiment was repeated 100 times and averaged.

5.1.2. Overhead of sending a message. We evaluate the overhead of sending messages with cryptographic personas (over sending a typical message on a messaging service) by measuring the additional load it imposes on the client and server. Specifically, we instrumented our system to record execution times for the following components:

(1) **Micro-benchmark: Proof Generation.** The time required for the client to generate the proof associated with a persona operation (i.e., run EXECMETHODANDCREATECALL for our circuits).
(2) **Micro-benchmark: Verify.** The server-side time spent verifying the received proof. For proofs that result in a state update (i.e., registering a callback), this step also includes any associated signing or cryptographic processing. This is performed upon message reception and directly contributes to total server-side overhead.
(3) **Micro-benchmark: Total Cryptographic Persona Overhead.** The client and server processing time from the moment when a client makes a request to the validation server until verification is completed and the message is prepared for dispatch to the messaging service. This benchmark isolates the cost introduced solely by our cryptographic persona layer—primarily proof verification and state updates—while excluding network latency (recall that both client and server are run locally) associated with the messaging service, since this latency depends on the underlying network infrastructure and is incurred regardless of whether cryptographic personas are deployed. By focusing on this internal contribution, we obtain a consistent, service-agnostic baseline across deployment models (infrastructure-integrated, service-based, or serverless). We denote this measurement as CryptoPersonaOverhead.

We provide measurements for each of these benchmarks, along with R1CS constraint counts, for our five persona-related operations in Table 2.

Predictably, proof generation is the most expensive computational component, given the cost of generating zkSNARKs for even moderately sized circuits. The number of constraints for each operation varies with the complexity of the operation; pseudonym generation incurs the highest constraint count, as it includes logic for generating callbacks and verifying that the pseudonym was correctly derived (along with additional rate- and scope-limiting checks in the case of limited pseudonyms). Because we use Groth16, verification times are constant, and variability in the Verify micro-benchmark accounts for signing operations done by the server.

Similarly, the total server-side cryptographic persona overhead is highest for pseudonym operations, as they require executing a full callback and interaction flow, whereas other operations are simpler and less time-intensive. On modern consumer hardware, these benchmarks suggest that performance is well within the range needed for practical deployment. By contrast, the legacy hardware results show that proof generation is roughly three times slower across all operations, raising client-side latency from ≈ 120–140 ms into the 340–430 ms range. While verification costs remain low, this slowdown makes proof generation the dominant overhead, and for users on older devices, it could noticeably disrupt workflows [43]. The observed proof generation costs on legacy hardware indicate potential gains from performing parts of the computation ahead of time, an approach we evaluate in Section 5.1.4.

5.1.3. Costs associated with callbacks. There are three steps associated with invoking and processing the callbacks used in moderation (i.e., revocation and reputation): (Step 1) the server posts the callback to the bulletin board (i.e., run CALL), (Step 2) the server runs an epoch update, which requires a linear scan over all of the callbacks posted since the last update, and (Step 3) each client updates their state such that they are ready to send a new message when desired. The rate at which these steps are interleaved represents a configurable deployment parameter. If the effects of the callbacks must take place immediately, the server runs Step 1 and Step 2 sequentially, after which all clients must complete Step 3 before continuing interaction with the system. However, this approach quickly becomes impractical in real-world deployments: performing a scan after every message introduces significant overhead on the critical path, increases user-facing latency, and scales poorly with message volume. Moreover, repeated scanning prevents efficient batching or caching of cryptographic proofs. Alternatively, the server can run Step 2 periodically, allowing users to continue interacting without updating their state, meaning revocations and reputation may be slower to propagate. As we discuss below, Step 3 is a slow operation, and allowing users to run it during downtime will improve usability—although the server

TABLE 2: Micro-benchmarks: Cryptographic Personas

Operation Executer	**Constraints**	**ProofGen** Client		**Verify** Server	**CryptoPersonaOverhead** Server
Hardware Unit	#	*Modern* ms	*Legacy* ms	*Modern* ms	*Modern* ms
Lim. Pseudonym	10484	143.67	429.70	5.83	112.72
Unlim. Pseudonym	10206	141.38	428.31	5.32	112.49
Authorship	8805	123.72	370.22	3.28	99.93
Vote	8532	116.96	356.63	3.01	100.42
Badge	8268	113.24	342.46	2.96	99.29

can always update the epoch proactively in an emergency. We defer our benchmarks for Step 3 to Section 5.1.4.

Table 3 reports the average time required for Step 1 and Step 2, both presented on a per-callback basis. The PostCallback column captures the server's time to log and commit a moderation-triggered callback. The UpdateEpoch column measures the time to update the relevant user epoch state, which includes verifying associated commitments and issuing an updated record. To produce these measurements, we simulate 100 total callbacks in batches of 10. After every 10 callbacks, an epoch update is triggered, resulting in 10 epoch updates total. We report the average time per callback across these batched updates. For clarity, these benchmarks exclude the cost of sorting callbacks or user records, focusing solely on core cryptographic operations to isolate their performance impact. The resulting micro-benchmarks capture server-side performance under load and show that, despite being more complex than conventional messaging logic, cryptographic moderation remains efficient enough for real-time deployment.

Following each epoch update, Step 3 involves client-side state reconciliation. In this phase, each user performs a scan over all non-expired callbacks to detect newly invoked entries and to recompute their local state accordingly. Table 4 reports the time required for this linear scan over all callbacks that *may* have been invoked since the previous epoch update. During this process, the client generates a proof attesting that it has correctly scanned the outstanding callbacks and updated its local state accordingly, which is sent to the server. As discussed in Section 4.6, this must be done *sequentially*, either one at a time or in batches, in order to prevent clients from being fingerprinted based on the number of callbacks they must scan. The validation server verifies these proofs and records the updated state, with ServerOverhead capturing the total processing cost from proof receipt to response dispatch. Once the client's state incorporates all active revocations, any subsequent message attempts by revoked users will be rejected (i.e., obj.revoked $=$ false), ensuring that moderation effects are enforced cryptographically and remain consistent across all participants.

5.1.4. New Constructions for Real-Time Performance. We now benchmark the constructions described in Section 4.5 and Section 4.6.

Generating callbacks during pre-processing. The construction described in Section 4.5 shifts the computational burden of sending a message into a pre-processing phase, allowing clients on both modern and legacy devices to efficiently use cryptographic personas.

TABLE 3: Micro-benchmarks: Callback Invocation

Operation Executer *Hardware*	**Post Callback** Server *Modern*	**Update Epoch** Server *Modern*
Moderation Action	3.0 ms	8.91 ms

TABLE 4: Micro-benchmarks: Scanning

Operation Executor *Hardware*	**ProofGen** Client *Modern*	 *Legacy*	**ProofVerify** Server *Modern*	**ServerOverhead** Server *Modern*
Scan	210.48 ms	739.54 ms	2.0 ms	4.0 ms

Table 5 reports the per-ticket cost of issuing and redeeming tickets on modern and legacy hardware, respectively. Client-side performance includes all additional operations relative to the base construction evaluated in Section 5.1.2, including generating the proof π_{store}, creating the blinded tokens $\{\widetilde{T}_i\}$, and unblinding the server's response.

On modern hardware, pre-processing adds only 75.21ms of total computational overhead per ticket, reducing the client's work required when sending a message from $\approx$ 140ms to 1.23ms—over a 99% reduction. Note that this 75.21ms is only to generate the proof π_{store}, and any other proofs that a client might need to generate (e.g., demonstrating that it knows a pre-image associated with a pseudonym) would also need to be done during pre-processing. Legacy hardware exhibits higher pre-processing costs at 327.70ms per ticket, while client-side latency during message sending is reduced from roughly ≈ 430ms to 2.90ms ($> 99\%$), making cryptographic personas practical even on older systems.

Folding for scanning. We instantiate folding for scanned callbacks using Nova [16], [20], implemented via Privacy Scaling Explorations' `sonobe` library [47]. Our construction uses the BN254 and Grumpkin cycle of curves and applies recursive folding to aggregate multiple callback scans into a single, succinct proof.[13]

To evaluate performance, we fix the total number of callbacks to $k = 100$ and vary the folding batch size $n \in$

13. While our folding implementation uses standard Nova and does not hide the step count, we emphasize that step count hiding (as discussed in Section 4.6) can be incorporated with minimal overhead.

TABLE 5: Micro-benchmarks: Ticket Issuance & Ticket Redemption

Executer	**Client**		**Server**
Hardware	*Modern*	*Legacy*	*Modern*
Ticket Issuance	75.21 ms	327.70 ms	1.61 ms
Ticket Redemption	1.23 ms	2.90 ms	1.98 ms

$\{1, 2, 4, 5, 10, 20, 25, 50\}$, where n denotes the number of callbacks folded into each step. Each run performs $t = k/n$ folding steps of size n, followed by a randomized blinding step, thus processing a total of k callbacks. Table 6 presents our measurements.[14] For each n, we measure the client per-callback proof cost, the cost of blinding the proof to ensure zero knowledge (i.e., folding in a random satisfiable instance), server verification time, and the total server overhead. On the client-side, proof costs are reported per callback, while proof blinding is an absolute cost.

Clearly, the performance of folding will be asymptotically better than the baseline iterative scanning approach from zk-promises [13] (see Table 4), as the client's work is asymptotically the same, but the server's work is *constant* when using folding. Using our implementation, we identify the number of callbacks at which folding concretely outperforms the baseline by computing $\mathbf{k}^*$, the number of callbacks at which the costs of the two approaches are the same. Our results are reported in Table 7. For the client, computing $\mathbf{k}^*$ is dependent on the *concrete* costs of each approach, as folding is more concretely efficient than baseline scanning (although not asymptotically). For the server, $\mathbf{k}^*$ is determined by the point at which the asymptotics become dominant. In more detail, we compute $\mathbf{k}^*$ as follows:

– **Client-Side Break-even $\mathbf{k}^*$ (Proof Generation)**: The client-side cost for folded scanning of k proofs is $T_{\text{fold}_{client}}(k) = \mathsf{ProofCostPerCallback} \cdot k + \mathsf{ProofBlind}$. The baseline cost is $T_{\text{base}_{client}}(k) = \mathsf{ProofGen}_{\text{baseline}} \cdot k$. The break-even factor $\mathbf{k}^*$ is found by equating these costs, $T_{\text{fold}_{client}}(k^*) = T_{\text{base}_{client}}(k^*)$, yielding:

$$k^* = \frac{\mathsf{ProofBlind}}{\mathsf{ProofGen}_{\text{baseline}} - \mathsf{ProofCostPerCallback}}$$

– **Server-Side Break-even $\mathbf{k}^*$ (Proof Verification)**: The server's benefit is a result of aggregating k individual proofs into a single verification. $\mathbf{k}^*$ occurs when the cumulative baseline verification time for k proofs, $T_{\text{base}_{server}}(k) = \mathsf{ProofVerify}_{\text{baseline}} \cdot k$, equals the fixed time for the single aggregated verification, $T_{\text{fold}_{server}} = \mathsf{ProofVerify}_{\text{folding}}$. The formal break-even condition is:

$$\mathsf{ProofVerify}_{\text{baseline}} \cdot k^* = \mathsf{ProofVerify}_{\text{folding}}$$

Thus, $\mathbf{k}^*$ is the ratio of verification costs:

$$k^* = \frac{\mathsf{ProofVerify}_{\text{folding}}}{\mathsf{ProofVerify}_{\text{baseline}}}$$

14. We observe that there are engineering optimizations possible that could further improve the server-side performance of our solution. We provide more careful micro-benchmarks in Appendix B.

On the client side, our evaluation encompasses both high-end (*Modern*) and older hardware (*Legacy*) configurations, whereas the server is assumed to be running *Modern* hardware.

It is important to note that the *server*-side analysis strictly isolates the proof verification overhead (and ignores other implementation details like parsing and memory management). This is because our implementation of the server-side code is not optimized, which would result in a severe under-estimate of the saving offered by folding (see Section B for a full discussion).

As shown in Table 7, the client-side folding advantage manifests at comparatively low folding factors. $\mathbf{k}^*$ ranges from approximately 4.3 to 21.5 on modern hardware and a more favorable 3.2 to 14.4 on legacy devices. This disparity confirms that legacy hardware intrinsically benefits more from folding due to the high baseline per-callback latency (739.54 ms) dominating the cost function, allowing the fixed folding overhead to be amortized rapidly.

Given that a single user account with cryptographic personas can be cross-platform compatible, the k values (i.e., number of callbacks) are well within the range of a daily user's messaging habits. For example, a single WhatsApp user sends on average ≈ 17.5 messages per day [50]. Since the highest $\mathbf{k}^*$ on resource-constrained (legacy) clients is 14.4 (for $n = 50$), the requirement for folding to provide a performance gain is met by less than a single day's worth of typical user messaging activity. Thus, folding is well-suited for users, providing a quantifiable mechanism to reduce latency and overhead for mobile and legacy deployments.

The server exhibits remarkably low break-even thresholds, with $\mathbf{k}^*$ spanning a narrow range of 4.7 to 10.0 on modern hardware configurations. This tight range confirms the high efficiency of the aggregated verification step when measured against the baseline proof verification costs. Since the server's primary goal is maximizing throughput, these low $\mathbf{k}^*$ values demonstrate that the performance gain from folding is realized nearly immediately, even at minimal batch sizes. Specifically, folding efficiently amortizes the verification work across the batch, thus reducing the server's effective per-callback computational load and optimizing overall throughput for cryptographic verification cycles. This immediate, low $\mathbf{k}^*$ indicates that folding is well-suited as a server-side throughput optimization, lowering total resource usage for repeated cryptographic verification cycles even across small volumes of callbacks.

6. Related Work

Social benefits and risks of anonymity. Human-computer interaction researchers have demonstrated that anonymity and pseudonymity—a longstanding online tradition [51]—are productive tools for circumventing these risks in a wide variety of digital contexts [52]. For example, anonymous bloggers have been found to be more disclosive and honest [53], [54], anonymity aids idea generation on online platforms [55], and anonymity within comments sections of websites is often considered an essential feature [38], [36],

TABLE 6: Micro-benchmarks: Folded Scanning ($k = 100$) over batch sizes n. ProofCostPerCallback and ProofBlind report client-side witness computation and proof generation times (ms). ProofSize is the serialized proof size (MB). ProofVerify is server-side verification time (ms), and ServerOverhead includes full server processing per batch. As discussed in Section B, a non-significant percentage of the total server load (at least $\approx$ 40-50%) should be possible to reduce with engineering optimizations. Modern hardware results are shown alongside legacy hardware to illustrate performance across device generations.

Operation Executer	**n**	**Constraints**	**ProofSize**	**ProofCostPerCallback** Client		**ProofBlind** Client		**ProofVerify** Server	**ServerOverhead** Server
Hardware Unit	#	#	MB	*Modern* ms	*Legacy* ms	*Modern* ms	*Legacy* ms	*Modern* ms	*Modern* ms
Folded Scan	1	7192400	4.64	173.0	606.1	426.7	1136.3	9.3	64.4
	2	4260800	5.48	98.2	346.0	498.1	1312.1	9.8	72.7
	4	2795000	7.14	61.4	217.8	646.4	1666.3	10.0	82.7
	5	2501840	7.97	54.4	191.8	715.1	1783.2	10.6	90.2
	10	1915520	12.12	41.3	138.5	1083.5	2618.2	12.5	113.0
	20	1622360	20.43	33.6	114.3	1807.0	4471.8	14.4	152.4
	25	1563728	24.59	32.1	105.6	2171.0	5264.2	15.9	173.8
	50	1446464	45.36	28.1	92.7	3929.1	9311.2	20.0	252.7

TABLE 7: Ideal callback factor $\mathbf{k}^*$ (number of callbacks) for client and server folding performance under modern and legacy hardware.

Metric Executer	**n**	$\mathbf{k}^*$ Client		$\mathbf{k}^*$ Server
Hardware Unit	#	*Modern* #	*Legacy* #	*Modern* #
Break-even k^*	1	11.4	8.5	4.7
	2	4.4	3.3	4.9
	4	4.3	3.2	5.0
	5	4.6	3.3	5.3
	10	6.4	4.4	6.2
	20	10.2	7.2	7.2
	25	12.2	8.3	8.0
	50	21.5	14.4	10.0

[56]. On social media sites where anonymity is easy, social anonymity can be an important factor in finding support [57], [58] and building community [59], [60]. Perhaps the digital systems within which anonymity has been studied most closely are question and answer (Q&A) websites [30]–[34]. On these platforms, results show that anonymity reduces the social costs (i.e., risks) of engaging [32], producing more [30] and more sensitive questions [33]. Soliman et al. recently extended these results by designing a Q&A platform for academics that facilitated partial disclosure of identity (which the authors call "meronymous"), in an attempt to side-step barriers in academic discussions [34].

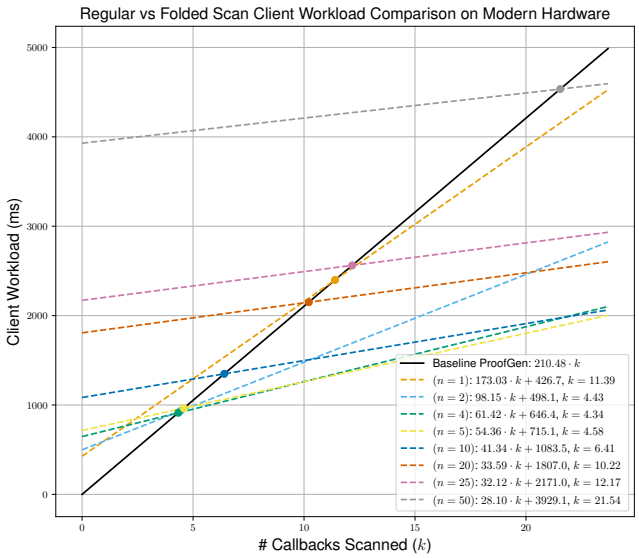
While anonymity and pseudonymity have documented value on online platforms, the disinhibition effect they produce can yield negative effects [61]. Anonymous comments are of lower quality and are more likely to be anti-social [35]–[37]. For example, the Economic Job Market Rumor Form, an anonymous form that is extremely popular among economists, has been widely criticized as abusive and toxic [62], [63]. Indeed, anonymity has been identified as an important factor in facilitating cyberbullying [64], [65]. While there are many types of platforms in which anonymity promotes anti-social behavior, it is important to remember that this is not always the case; Tran et al. showed that Wikipedia editors who used Tor (anonymity-seeking behavior) produced similar quality edits to others [1].

Anonymous credentials and anonymous blocklisting. Our work is a particular instantiation of the anonymous credential [66], [67] paradigm. Within this paradigm, users can prove that they should have access to a particular resource without revealing which authorized user they are. Since their initial proposal, anonymous credentials have grown to be a practical tool primed for deployment, e.g., [68], [14], [69], [70], and can be stateful [29].

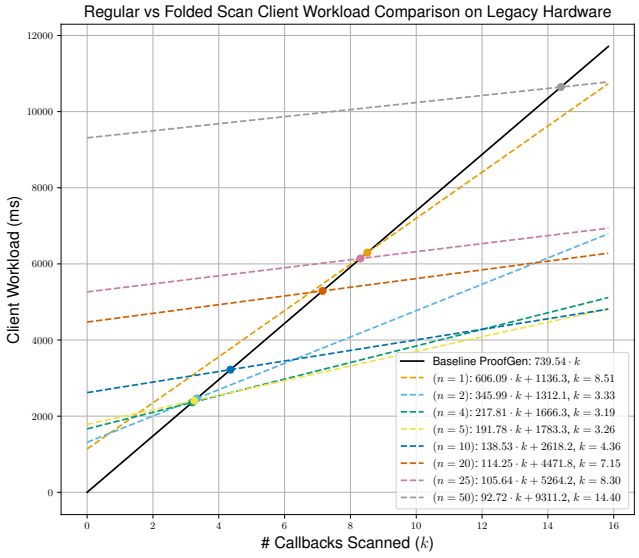
Perhaps foreseeing the risk of full anonymity with unrevocable posting ability, a line of work has considered anonymous blocklisting, for which we refer the reader to the excellent overview of Henry et al. [71]. A particularly relevant line of work, initiated by Tsang et al. in BLAC [10], considers trusted third party (TTP) free blocking schemes. The approach taken by BLAC and subsequent work is simple: users are issued a fixed identity key k and for each action, they generate $tag \coloneqq \mathsf{PRF}(k, nonce)$ and a proof both that tag is correct and that they are not on a blocklist $\mathcal{L}$ by proving $\forall (tag', nonce') \in \mathcal{L}, \mathsf{PRF}(k, nonce') \neq tag'$. Users are blocked by adding their tag to the blocklist $\mathcal{L}$.

Recently, SNARKBlock [11] kicked off a new line of work using modern zk proof techniques for blocklisting. Subsequently, Alpaca [12] used a folding scheme to prove blocklist non-membership, offering both improved performance and allowing new users to avoid iterating over old chunks of the blocklists, resolving an open "cold start" problem identified in [11] that was a major impediment for deployment.

In zk-promises, Shih et al. [13] combine techniques from stateful anonymous credentials [29] with a different blocklisting approach from Tsang et al. [72] that inherently avoids coldstart, building programmable anonymous credentials with anonymous callbacks where function calls and callbacks alter the credential's state while ensuring



(a) Modern Hardware



(b) Legacy Hardware

Figure 5: Comparison of client-side workload for regular scanning (solid black line) and folded scanning (dashed lines) during proof generation for k callbacks on both *Modern* (Figure 5a) and *Legacy* (Figure 5b) hardware. The solid black line (labeled Baseline ProofGen) represents total client-side proof generation time for regular scanning, derived from the original zk-promises implementation [13]. Dashed lines represent folded scanning at different batch sizes n, modeled as $\mathsf{ProofCostPerCallback} \cdot k + \mathsf{ProofBlind}$ (see Table 6). Dots on each intersection indicate the break-even points where folded scanning begins to outperform regular scanning—i.e., the minimum number of callbacks k needed for the corresponding batch size n to provide a client-side performance benefit. This highlights the tradeoff between batch size and callback volume for optimizing client efficiency.

the user remains unlinkable across credential shows and callbacks. By supporting state, this approach allows complex reputation and other logic, in contrast to SNARKblock [11] and Alpaca [12], and far better performance than [72]. We note that the particular approach for checking for revocations in zk-promises and our work here, termed scanning, is best suited to when the number of actions the client must check for is smaller than the ban list. But in certain settings where bans are rare but messages frequent (e.g., a small group chat among friends with anonymity), it may be useful to instantiate the techniques here and indeed in zk-promises with the approach from BLAC [10].

7. Conclusion

Anonymity (or pseudonymity) can empower users to be honest and productive with one another, but some users will abuse the ability to engage with others anonymously (or pseudonymously). In this work, we introduced cryptographic personas as a new primitive to navigate this tension. Our system allows users to communicate pseudonymously, while ensuring that abusive behavior results in the selective revocation of anonymity privileges—but not identity de-anonymization. This balances expressive freedom with communal accountability, a design goal that has proven difficult to achieve in privacy-respecting systems. We build on recent constructions of anonymous blocklists, improving prior work such that cryptographic personas can run on real-time systems like group chats.

A note on deployment: for small groups, Cryptographic Personas may be deployed *without* cryptographic overhead by relying on the chat application itself to enforce the same constraints our proofs do. This limits the security model to UI-bound adversaries using an honest chat application—or one attested by the platform (e.g., via iOS's DeviceCheck framework [73]). For many small groups, this tradeoff seems reasonable; in cases where it proves inadequate, Cryptographic Personas can simply be disabled for that chat. Similarly, on-device trusted hardware [74] can replace zero-knowledge proofs, trading cryptographic integrity for hardware-based guarantees to greatly improve performance while preserving client privacy.

References

[1] Chau Tran, Kaylea Champion, Andrea Forte, Benjamin Mako Hill, and Rachel Greenstadt. Are anonymity-seekers just like everybody else? An analysis of contributions to wikipedia from tor. In *2020 IEEE Symposium on Security and Privacy*, pages 186–202. IEEE Computer Society Press, May 2020. doi:10.1109/SP40000.2020.00053.

[2] Alice E. Marwick and danah boyd. I tweet honestly, i tweet passionately: Twitter users, context collapse, and the imagined audience. *New Media & Society*, 13(1):114–133, 2011. arXiv:https://doi.org/10.1177/1461444810365313, doi:10.1177/1461444810365313.

[3] Alexander Hamilton, James Madison, and John Jay. The federalist papers. 1787.

[4] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Berlin, Heidelberg, April 1991. doi:10.1007/3-540-46416-6_22.

[5] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Berlin, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_38.

[6] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Berlin, Heidelberg, December 2001. doi:10.1007/3-540-45682-1_32.

[7] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 325–335. Springer, Berlin, Heidelberg, July 2004. doi:10.1007/978-3-540-27800-9_28.

[8] Alice 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/, September 2017. Accessed on 4 June, 2025.

[9] 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.

[10] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 72–81. ACM Press, October 2007. doi:10.1145/1315245.1315256.

[11] Michael Rosenberg, Mary Maller, and Ian Miers. SNARKBlock: Federated anonymous blocklisting from hidden common input aggregate proofs. In *2022 IEEE Symposium on Security and Privacy*, pages 948–965. IEEE Computer Society Press, May 2022. doi:10.1109/SP46214.2022.9833656.

[12] Jiwon Kim, Abhiram Kothapalli, Orestis Chardouvelis, Riad S. Wahby, and Paul Grubbs. ALPACA: Anonymous blocklisting with constant-sized updatable proofs. Cryptology ePrint Archive, Paper 2025/767, 2025. URL: https://eprint.iacr.org/2025/767.

[13] Maurice Shih, Michael Rosenberg, Hari Kailad, and Ian Miers. zk-promises: Anonymous Moderation, Reputation, and Blocking from Anonymous Credentials with Callbacks. pages 4995–5014, 2025. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/shih.

[14] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In *2023 IEEE Symposium on Security and Privacy*, pages 790–808. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179430.

[15] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018. doi:10.1515/popets-2018-0026.

[16] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022. doi:10.1007/978-3-031-15985-5_13.

[17] Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. URL: https://eprint.iacr.org/2022/1758.

[18] Abhiram Kothapalli and Srinath Setty. CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Report 2023/1192, 2023. URL: https://eprint.iacr.org/2023/1192.

[19] Liam Eagen and Ariel Gabizon. ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106, 2023. URL: https://eprint.iacr.org/2023/1106.

[20] Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 345–379. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_11.

[21] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 790–808, 2023. doi:10.1109/SP46215.2023.10179430.

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

[23] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. URL: https://eprint.iacr.org/2015/1093.

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

[25] Matteo Frigo and abhi shelat. Anonymous credentials from ECDSA. Cryptology ePrint Archive, Report 2024/2010, 2024. URL: https://eprint.iacr.org/2024/2010.

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

[27] Joshua Lund. Technology Preview: Sealed sender for Signal. https://signal.org/blog/sealed-sender/, October 2018. Accessed on 4 June, 2025.

[28] Dennis YW Liu, Joseph K Liu, Yi Mu, Willy Susilo, and Duncan S Wong. Revocable ring signature. *Journal of Computer Science and Technology*, 22:785–794, 2007.

[29] Scott E. Coull, Matthew Green, and Susan Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 501–520. Springer, Berlin, Heidelberg, March 2009. doi:10.1007/978-3-642-00468-1_28.

[30] Fiona 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*, 72(3):336–363, 1997.

[31] Xiao Ma, Nazanin Andalibi, Louise Barkhuus, and Mor 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*, pages 1781–1793, 2017.

[32] Haiwei Ma, Hao-Fei Cheng, Bowen Yu, and Haiyi Zhu. Effects of anonymity, ephemerality, and system routing on cost in social question asking. *Proceedings of the ACM on human-computer interaction*, 3(GROUP):1–21, 2019.

[33] Cheng Guo and Kelly Caine. Anonymity, user engagement, quality, and trolling on q&a sites. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–27, 2021.

[34] Nouran Soliman, Hyeonsu B. Kang, Matthew Latzke, Jonathan Bragg, Joseph Chee Chang, Amy Xian Zhang, and David R. Karger. Mitigating barriers to public social interaction with meronymous communication. In Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski, editors, *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, pages 151:1–151:26. ACM, 2024. doi:10.1145/3613904.3642241.

[35] Loren Terveen and Will Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1(2001):487–509, 2001.

[36] Nicholas Diakopoulos and Mor Naaman. Towards quality discourse in online news comments. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 133–142, 2011.

[37] Peter G Kilner and Christopher M Hoadley. Anonymity options and professional participation in an online community of practice. In *Computer Supported Collaborative Learning 2005*, pages 272–280. Routledge, 2017.

[38] Cliff Lampe and Paul 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*, CHI '04, page 543–550, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/985692.985761.

[39] Joseph Seering, Robert Kraut, and Laura 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*, CSCW '17, page 111–125, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/2998181.2998277.

[40] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. Steering user behavior with badges. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 95–106, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488388.2488398.

[41] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 2857–2866, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1978942.1979366.

[42] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Collective information security in large-scale urban protests: the case of hong kong. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 3363–3380. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/albrecht.

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

[44] Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. BLAC: Revoking repeatedly misbehaving anonymous users without relying on TTPs. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–33, 2010.

[45] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.

[46] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

[47] Privacy Scaling Explorations. Sonobe: Succinct proof composition and verification framework. https://privacy-scaling-explorations.github.io/sonobe-docs/, 2024. Accessed: 2025-06-04. URL: https://github.com/privacy-scaling-explorations/sonobe.

[48] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.

[49] Sebastian Scheibner. AsamK/signal-cli, May 2025. original-date: 2015-05-11T10:49:42Z. URL: https://github.com/AsamK/signal-cli.

[50] Avi Rosenfeld, Sigal Sina, David Sarne, Or Avidov, and Sarit Kraus. Whatsapp usage patterns and prediction of demographic characteristics without access to message content. *Demographic Research*, 39:647–670, 2018. URL: https://www.jstor.org/stable/26585343.

[51] Judith S Donath. Identity and deception in the virtual community. In *Communities in cyberspace*, pages 37–68. Routledge, 2002.

[52] Ruogu Kang, Stephanie Brown, and Sara 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*, pages 2657–2666, 2013.

[53] Erin E. Hollenbaugh and Marcia K. Everett. The effects of anonymity on self-disclosure in blogs: An application of the online disinhibition effect. *Journal of Computer-Mediated Communication*, 18(3):283–302, 04 2013. arXiv:https://academic.oup.com/jcmc/article-pdf/18/3/283/19492073/jjcmcom0283.pdf, doi:10.1111/jcc4.12008.

[54] Qinying Liao, Yingxin Pan, Michelle X Zhou, and Tingting 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*, pages 315–324, 2012.

[55] Terry Connolly, Leonard M Jessup, and Joseph S Valacich. Effects of anonymity and evaluative tone on idea generation in computer-mediated groups. *Management science*, 36(6):689–703, 1990.

[56] Rolf Fredheim, Alfred Moore, and John Naughton. Anonymity and online commenting: The broken windows effect and the end of drive-by commenting. In *Proceedings of the ACM web science conference*, pages 1–8, 2015.

[57] Munmun De Choudhury and Sushovan De. Mental health discourse on reddit: Self-disclosure, social support, and anonymity. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 71–80, 2014.

[58] Nazanin Andalibi, Oliver L Haimson, Munmun De Choudhury, and Andrea 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*, pages 3906–3918, 2016.

[59] Michael Bernstein, Andrés Monroy-Hernández, Drew Harry, Paul André, Katrina Panovich, and Greg 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*, volume 5, pages 50–57, 2011.

[60] Nicole B Ellison, Lindsay Blackwell, Cliff Lampe, and Penny Trieu. "the question exists, but you don't exist with it": Strategic anonymity in the social lives of adolescents. *Social Media+ Society*, 2(4):2056305116670673, 2016.

[61] John R Suler and Wende L Phillips. The bad boys of cyberspace: Deviant behavior in a multimedia chat community. *CyberPsychology & Behavior*, 1(3):275–294, 1998.

[62] Justin Wolfers. Evidence of a toxic environment for women in economics. *New York Times*, 18, 2017.

[63] Florian Ederer, Paul Goldsmith-Pinkham, and Kyle Jensen. Anonymity and identity online. *arXiv preprint arXiv:2409.15948*, 2024.

[64] Christopher P Barlett, Douglas A Gentile, and Chelsea Chew. Predicting cyberbullying from anonymity. *Psychology of Popular Media Culture*, 5(2):171, 2016.

[65] Rachel Young, Stephanie Miles, and Saleem Alhabash. Attacks by anons: A content analysis of aggressive posts, victim responses, and bystander interventions on a social media site. *Social Media + Society*, 4(1):2056305118762444, 2018. arXiv:https://doi.org/10.1177/2056305118762444, doi:10.1177/2056305118762444.

[66] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982. doi:10.1007/978-1-4757-0602-4_18.

[67] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001. doi:10.1007/3-540-44987-6_7.

[68] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Berlin, Heidelberg, August 2004. doi:10.1007/978-3-540-28628-8_3.

[69] Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 691–721. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_24.

[70] Carsten Baum, Olivier Blazy, Jaap-Henk Hoepman, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, abhi shelat, Daniel Slamanig, and Søren Eller Thomsen. Cryptographers' feedback on the eu digital identity's arf. https://github.com/user-attachments/files/15904122/cryptographers-feedback.pdf, Jun 2024.

[71] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *2011 IEEE Symposium on Security and Privacy*, pages 81–95. IEEE Computer Society Press, May 2011. doi:10.1109/SP.2011.13.

[72] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. PEREA: towards practical TTP-free revocation in anonymous authentication. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 333–344. ACM Press, October 2008. doi:10.1145/1455770.1455813.

[73] Apple Developer Documentation. DeviceCheck. https://developer.apple.com/documentation/devicecheck.

[74] Florian Tramèr, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 19–34, April 2017. URL: https://ieeexplore.ieee.org/document/7961949/authors, doi:10.1109/EuroSP.2017.28.

[75] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017. doi:10.1007/978-3-319-63715-0_20.

[76] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 457–475. Springer, Berlin, Heidelberg, March 2021. doi:10.1007/978-3-662-64322-8_22.

Appendix A. Proof Sketch

The proof of our construction is straightforward, so we sketch it here. Our scheme is secure against an adversary that statically corrupts an arbitrary subset of the parties $P_1, \ldots, P_n$. We assume that all network traffic (e.g., IP addresses) in the real protocol does not reveal the identities of users; Tor could be used in settings where this assumption could not hold.

For the scope of this proof, we can let AllowedToCreateNewPersona simply output `true` or enforce limits on the number of total personas that a user can generate in a particular context (as described in the "rate-limited persona" part of Section 4.3). AllowedToRevoke simply implements the expiry aspects of zk-promises. Namely, only callbacks that have been created recently (where recently is some fixed threshold relative to the current `time` maintained by the ideal functionality) can actually be used for revocation. We define our simulator Sim as follows:

- When a party registers for the system, Sim extracts the secret key from the underlying zk-promises construction and associates it with the identity of the registering party.
- When Sim receives a proof π from the underlying zk-promises construction, along with a persona persona, context context, and message m, it extracts the secret key of the prover from π (which is associated with an identity. If there is no identity associated with the key, then the simulation aborts with an error). Once the identity of the sender is extracted, Sim sends the message on behalf of that corrupted party to the ideal functionality.
- If the extractor fails, the simulator aborts the simulation with an error.
- When prompted by the ideal functionality, Sim supplies the persona chosen by the adversary (note that the persona is in the statement for which π is a proof). If that persona is a collision with any other personas generated by any party, the simulator aborts the simulation with an error.
- When a corrupt party requests their messages using the (ReceiveMessages), Sim simulates the accompanying proof and delivers the messages.

The hybrid argument that follows is simple. Because the underlying zero-knowledge proof system is a proof of knowledge, the probability that the extractor fails is negligible in the security parameter.[15] Next, if any corrupt party manages to select a persona that was used by another party, then they have either broken the pre-image resistance of the hash function (by finding an input that maps to a persona that was uniformly selected at random by the ideal functionality) or broken the second-preimage resistance of the hash function (by allowing two different corrupted parties to hold the same persona).[16] In either case, this happens with negligible probability.

Thus, the simulation is negligibly close to the real world, meaning our protocol securely realizes our ideal functionality.

15. Note that this requires that the zero-knowledge proof system used to instantiate the zk-promise system must be simulation extractable, as we need to extract from proofs in a setting where we produce simulated proofs. Our implementation—as well as Shih et al.'s initial implementation—uses Groth16 [45] as its proof system. Recent work has shown that Groth16 has a weak simulation extractable property, despite being malleable [75], [76].

16. Note that because we allow clients to pick their own secret keys, we require that $\mathcal{H}$ is a hash function, rather than a pseudorandom function. Otherwise, a malicious party could pick their secret key adaptively to find a persona collision.

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 8: 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 6 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 B. Total Server Overhead for Folding

Conceptually, the main thing that the server does in each scanning operation is proof verification. But, as can be observed in Table 6, the ServerOverhead is significantly higher than ProofVerify. To identify the source of this additional load, we further instrumented our implementation to identify the server operations that significantly impacted server performance. We show the results of this instrumentation (where each row represents a single run of the scheme) in Table 8. Unsurprisingly, our instrumentation reveals that there are many server operations whose performance degrades roughly linearly in the batch size n.

Each column in Table 8 corresponds to a distinct server-side processing phase. ExtractBody measures the time to asynchronously receive and buffer the HTTP request body into memory; recall that the proof size grows as n grows, which accounts for this growth. LoadServerParams captures the time spent acquiring a read-lock on shared server state and cloning the public parameters required for verifying folded proofs, including R1CS definitions, folding keys, and hash configurations. Note that in an optimized, deployment-ready implementation, much of this can be removed. DeserializePayload reflects the cost of deserializing the buffered proof data into structured types, including constraint validation. ProofVerify measures the time to verify the randomized IVC proof using the Nova folding scheme, along with additional application-specific checks over public inputs. AppendBulletin includes the time to store the verified result in the server's bulletin board. ServerOverhead captures the overall wall-clock time from request receipt to completion, including overhead from memory management, asynchronous scheduling, and other runtime effects not captured by individual phases.

Our measurements make clear that as n increases, the total server load also increases—but there are many low-level tasks that are not captured by these main tasks, making the total not a direct sum of these measured components. This is due to the innate overhead associated with handling larger proofs, which contributes additional cost not explicitly reflected in the measured components of the system. These costs likely stem from memory allocations, serialization overhead, and other engineering artifacts in our current implementation. While these additional overheads are outside the scope of our core contributions, they highlight important areas for future optimization.

These measurements help clarify the computational costs associated with verifying folded scans and illustrate how server-side performance evolves with increasing batch size. While the cryptographic operations scale predictably, our results indicate that system-level factors—such as memory usage, serialization, and parameter handling—contribute meaningfully to overall runtime. With improved system engineering—such as more efficient memory handling or batched operations—this extraneous overhead can be significantly reduced, improving the practical performance of our system without altering the underlying cryptographic constructions. This breakdown thus serves as a useful reference for identifying areas where future engineering work can improve the efficiency of our cryptographic personas in real-world deployments.