

zk-BAN: An efficient anonymous blocklisting system with signature-based revocation

Kosei Akama ^{*}
NICT/Keio University

Yoshimichi Nakatsuka [†]
NICT

Koichi Moriyama [‡]
Keio University

Keisuke Uehara [§]
Keio University

Abstract

Anonymous blocklisting enables services to revoke malicious users while preserving anonymity for honest users. Signature-based revocation supports revocation from past authentications, but existing systems incur substantial overhead at large scale and for users returning after long offline periods. Window-based alternatives reduce this overhead but impose restrictive revocation-timing policies. This work presents zk-BAN, a signature-based anonymous blocklisting system that reduces proof-generation and verification costs through two design improvements: constructing a carefully filtered revocation list and partially unifying PRF tags to reduce repeated zero-knowledge computations. We formalize zk-BAN and prove that it satisfies blocklistability and unlinkability. Under modeled large-scale real-world workloads, zk-BAN generates proofs in approximately 600 ms and verifies them in approximately 2 ms. We show zk-BAN displays 54.4 times faster proving than our naive baseline and 136-fold faster proving and 32-fold faster verification than prior work.

1 Introduction

Problem statement. Online services increasingly contend with malicious user behavior, such as defamatory posts on social media and cheating in online games. A common mitigation is account suspension (i.e., *revocation*), which prevents offenders from continuing to abuse the service.

To make revocation effective, services often bind accounts to real-world identities, e.g., through SMS verification, to prevent revoked users from returning under new accounts. However, such identity binding creates privacy risks: the service provider could misuse identifying information to track users across sessions, e.g., by linking a user’s public-facing and private accounts. These concerns may also provoke backlash from privacy-conscious users [22, 24].

Prior work and Motivation. These tensions have motivated *anonymous blocklisting* techniques, which seek to provide both (i) the ability to revoke misbehaving users and (ii) anonymity for honest users. These techniques typically use zero-knowledge proofs to demonstrate that a user has not been revoked; we refer to such proofs as *non-revocation proofs*.

A representative approach is *signature-based revocation*, in which a user can be revoked using only a past signature [11, 39]. In such systems, signatures contain PRF/Hash values derived from a user’s secret key and a tag which is often chosen freshly for each authentication. To revoke a user, the service adds the corresponding PRF/Hash value from the offending signature to a revocation list. To prove non-revocation, a user proves in zero knowledge that the value derived from their secret key and the relevant tag does not match any value in the revocation list. However, because proof-generation and verification costs grow with the number of revoked users, classical signature-based revocation systems face substantial scalability challenges. This limitation is particularly problematic for modern services that suspend accounts at large scale, ranging from several million to tens of millions of accounts per year [5, 42].

One way to reduce this overhead is by restricting the proof target to the difference since the user’s last credential update, i.e., the revocation entries added during that interval, which we call *differential revocation-proof method*, introduced by [7]. However, large-scale services can still accumulate many revocations between updates, leaving proof generation and verification costly. Moreover, users who remain offline for extended periods must process an increasingly large accumulated difference, making credential update impractical.

A different family of approaches, which we call *window-based*, addresses scalability by bounding authentication overhead according to the number of user-service sessions within a time window, rather than the total size of the revocation list [6, 8, 14, 26, 44, 45]. However, these schemes either require services to make revocation decisions under tight deadlines [8] or nudge services toward conservative, resource-intensive policies, e.g., deferring final decisions to avoid

^{*}Email: akama@nict.go.jp

[†]Email: yoshimichi.nakatsuka@nict.go.jp

[‡]Email: moriyama@sfc.keio.ac.jp

[§]Email: kei@sfc.keio.ac.jp

missed revocations [6, 14, 26, 44, 45].

More recently, zk-SNARK-based systems have revisited signature-based revocation to improve scalability via efficient, lightweight non-revocation proofs [23, 30]. Nevertheless, proof generation can remain expensive at large scale. For example, for a service such as X (formerly Twitter) which revokes 9.49 million accounts per year [42, 43], a user would need to perform approximately 6 minutes of computation, including precomputation, per day. This overhead compounds when users interact with multiple services, each maintaining an independent blocklist. In addition, users returning after an offline period must process all revocations accumulated during their absence, which can be prohibitively expensive.

Our work. This work constructs zk-BAN, a novel anonymous blocklisting system designed to overcome the limitations of prior work from the latency and scalability point of view. This is achieved through two key design improvements.

As discussed above, proof-generation costs in zk-SNARK-based signature-based systems are still expensive under large-scale settings and particularly increase when users remain offline for extended periods. One way to address this issue is to use the differential revocation-proof method, although this still incurs a high overhead. Our first design improvement addresses this limitation. Specifically, we found that we can reduce the size of the revocation list considered during proof generation, hereafter called the *sub-revocation list*, by carefully selecting which newly revoked entries must be processed since the user’s last update. zk-BAN exploits the fact that, once revoked, a user can no longer obtain the credentials required to produce future valid authentications. This optimization allowed us to reduce the revocation list size from every revoked entry since the last update to roughly about one period-worth of entries in the ideal scenario.

While the above improvement list lowers proof-generation and verification costs, the resulting latency remained high. The dominant source of this overhead was repeated PRF-based zero-knowledge computations during proof generation. Our second design improvement addresses this bottleneck by partially unifying the PRF tags used during authentication. Rather than choosing an independent random tag for each authentication, zk-BAN derives tags from contextual information, such as the current time period and a bounded session counter within that period. Here, a session denotes a sequence of service interactions; in account-based services, such sessions are typically associated with a user account. This optimization reduces the number of PRF computations from δ_L , the number of revoked signatures in the sub-revocation list, to $T \times K$, where T is the number of periods during which those signatures were generated and K is the maximum number of authentications performed by each user per period. Achieving this reduction is challenging; it requires careful co-design of the proofs and protocols to avoid introducing privacy leakage or additional performance overhead.

In the zk-BAN design, the time required to generate a non-

revocation proof grows linearly with δ_L , T , and K . Although δ_L can be large in modern systems, we show that zk-BAN is more efficient than prior signature-based revocation systems under realistic conditions derived from real-world use cases, which comprise $\delta_L = 9.49$ million revocations per year, a $T = 30$ -day revocation-decision grace period, and $K = 5$ sessions per user per day. Compared with our naive baseline implementation, zk-BAN achieves 54.4 times faster proving. Compared to related work, it achieves a 136-fold and 10800-fold faster proving compared to [30] and [23] respectively.

Limitations. zk-BAN has two protocol-level limitations: (1) during authentication, a user’s anonymity set is limited to users who are online within that period, rather than all users; (2) zk-BAN is most applicable to services in which each user creates relatively few sessions per period and most abusive session can be revoked within a short delay. Additionally, zk-BAN has one implementation-level limitation; it requires users to store a large zk-SNARK proving key.

Contributions. Our key contributions are as follows:

- We design zk-BAN, a novel anonymous blocklisting system that supports scalable signature-based revocation, relaxed revocation deadlines, and efficient authentication for users who remain offline for extended periods.
- We introduce two novel techniques to reduce proof-generation/-verification costs; sub-revocation-list construction and partial PRF tag unification.
- We formalize and prove that zk-BAN satisfies blocklistability and unlinkability.
- We evaluate zk-BAN under modeled large-scale real-world workloads and show that it achieves substantially lower non-revocation proof and verification latency than prior signature-based revocation systems.

2 Background

zk-SNARKs. zk-SNARKs allow one to prove properties of hidden information without revealing the information itself. In this setting, the relation between the hidden and public information, typically represented as a *circuit*, is evaluated.

zk-SNARKs typically satisfy: (1) *Zero-Knowledge*: No information about the hidden data is leaked; (2) *Completeness*: A correct statement always yields a valid proof; and (3) *Soundness*: A proof for a false statement does not pass verification. Additionally, zk-SNARKs generally involve the following four procedures: (i) **Compilation**: Compile the relation between hidden and public information and generate a constraint system cs ; (ii) **Setup**: Using cs , generate a proving key pk and a verification key vk ; (iii) **Proving**: Using cs and pk , generate a proof π showing that the hidden and public inputs satisfy the relation defined by the circuit; and (iv) **Verification**: Using vk and the public input, verify π .

In general, the verification time increases with the number of public inputs. To reduce this latency, one can precompute part of the verification procedure based on a subset of the

public input. As a result, the online verification time depends only on the public inputs not provided during precomputation, thereby reducing latency.

This work uses the same notation as SNARKBlock [30] to represent statements proved by zk-SNARKs. Specifically, a zero-knowledge argument for public information $w_1, \dots, w_W$ and secret information $s_1, \dots, s_S$ satisfying the relation $R(s_1, \dots, s_S, w_1, \dots, w_W)$ is written as follows: $PK\{(w_1, \dots, w_W; s_1 \dots s_S) : R(s_1, \dots, s_S, w_1, \dots, w_W)\}$.

Signature-Based Revocation. Signature-based revocation is a zero-knowledge-proof-based anonymous blocklisting system, which appears in many previous works, including BLAC [39], SNARKBlock [30] and ALPACA [23]. In signature-based revocation, signers prove in zero knowledge that the secret key (usk) used for signing does not match any key corresponding to an entry in the revocation list.

The signature includes a random tag and a pseudo-random function value $h(= \text{PRF}(\text{tag}, \text{usk}))$. To claim non-revocation, the signer proves that h differs from all PRF values in the revocation list. Suppose that the revocation list is given by $RL = \{(h_1, \text{tag}_1), \dots, (h_L, \text{tag}_L)\}$. Then the zero-knowledge proof can be written as follows:

$$PK\{(\text{tag}, h, h_1, \dots, h_L; \text{usk}) : \\ h = \text{PRF}(\text{tag}, \text{usk}) \bigwedge_{l=1}^L h_l \neq \text{PRF}(\text{tag}_l, \text{usk})\}$$

3 Related Work

Classical Signature-Based Revocation. Representative examples of signature-based anonymous blocklisting techniques include BLAC [39] and EPID [11]. However, these approaches are known to have scalability limitations: proofs are generated with respect to the entire revocation list, so signing/verification latency and signature size grow with the revocation-list size. For example, with a 50,000-entry revocation list, it takes 150 sec and 75 sec for signing and verification, respectively, in EPID [41]. Given that large-scale services can perform around 10 million suspensions per year [42], applying such techniques at scale is impractical.

BLACR [7] addresses BLAC’s scalability issues by requiring zero-knowledge proofs only for newly added revocation entries since a designated checkpoint. However, BLACR’s authentication cost remains high even for moderately sized daily updates. For example, if 20,000 entries are added to the revocation list in a single day (hypothetically assuming a total of 1 million entries and a 2% daily addition rate), a user needs about 88 seconds of computation and the service side about 30 seconds [6]. This update volume is still several times smaller than real-world systems such as X, where around 30,000 suspensions occur daily on average ($\sim 5,296,870 + 4,191,171 \times 2M \div 365$) [42, 43].

Window-Based Blocklisting. To address the scalability challenges of signature-based revocation, researchers have proposed a more scalable family of systems often referred to as *window-based blocklisting*, first introduced by PEREA [8]. In this approach, verification time does not depend on the revocation-list size L , but instead on the number of user-service sessions K within a fixed time window. Since K is typically bounded per user and is much smaller than L , this approach is considered scalable. A key drawback, however, is that the revocation window must be fixed in advance, and a service cannot revoke a malicious user after that window has elapsed. Consequently, if a service fails to detect abuse within the prescribed period, it loses the ability to revoke the offender later, effectively forcing providers to make revocation decisions under a tight deadline (two days in PEREA). While many instances of abuse are detected quickly, some are only discovered after substantial investigation or after observing a larger volume of illicit activity, making such rigid deadlines unsuitable in practice.

Motivated by these limitations, FARB [44], PERM [6], and PE(AR)² [45] propose *deferred revocation* mechanisms. These systems follow a “Block First, Forgive Later” strategy [14], where services initially block suspicious users and later reinstate them if warranted. However, this approach risks temporarily blocking benign (but suspicious) users. Moreover, once a user’s block is lifted, the systems do not support revoking that user later, even if subsequent evidence reveals malicious behavior. As a result, services must be conservative when both revoking and reinstating users, which further increases operational pressure.

SAC [14] and SMART [26] aim to mitigate these constraints by allowing users to continue accessing the service while the provider decides whether to lift the block. Nevertheless, in SAC, users’ anonymity sets shrink depending on the number of deferred revocations which is controlled by the server. In SMART, the number of revocations that can be deferred is limited. Moreover, in both approaches, the service cannot revoke a malicious user if their activity is discovered after the decision.

zk-Promises [33], and its derivative Cryptographic Personas [38] use zk-SNARK-based “callbacks” and, while not strictly window-based anonymous blocklisting systems, still require the service to decide whether to revoke a user by a fixed deadline, much like PEREA.

Scalable Signature-Based Revocation. To avoid missed revocations and the operational time pressure inherent in window-based systems, recent work has revisited signature-based revocation. In particular, zk-SNARKs have renewed interest in this approach by improving scalability.

SNARKBlock [30] uses zk-SNARKs to realize anonymous blocklisting that scales with the number of revocations. However, SNARKBlock requires substantial batch computation for users to synchronize blocklist additions and removals, which

can significantly degrade client performance. For example, for every 8,192 entries added to the revocation list, users incur roughly 100 seconds of processing (see Figure 5 in [30]). At 30,000 new entries per day, this amounts to about 6 minutes of daily computation. Additionally, this work incurs extremely high overhead when users are offline for a while, as they must perform the costly batch computation for a large number of revoked signatures. Even if users are online everyday, since they may interact with multiple services that maintain independent blocklists, this synchronization overhead raises serious deployability concerns.

ShadowBlock [15] and ALPACA [23] are concurrent works aimed at achieving constant-time proof generation and verification by leveraging IVCs or RSA accumulators. However, these schemes still require user-side batching, which is heavier than that of SNARKBlock. For example, while ShadowBlock achieves constant-time verification using an RSA accumulator, its user-side batching cost is $O(L \times \delta_L)$ ¹, which is not scalable. Similarly, ALPACA also requires user-side batching, which incurs a linear computational cost and is more expensive than that of SNARKBlock.

FAB [35] is another concurrent work, which proposes a fast anonymous blocklisting scheme for multi-service settings. However, it assumes that each user has only a single pseudonym for each service, which is fundamentally different from our motivation.

4 System & Threat model

We consider the following ecosystem in zk-BAN, which comprises three types of principals: (1) signers, (2) verifiers, and (3) group managers (GMs). For convenience, we describe these principals as distinct roles; however, the verifier and GM may be operated by the same entity and are only logically separated.

Signer: Performs signing using a secret key and credential; typically an end-user.

Verifier: Verifies signatures; typically a service provider.

GM: Issues credentials to signers.

zk-BAN considers all three principals to be potentially malicious. A malicious signer aims to bypass an imposed service ban, including attempting to generate signatures that remain valid even after the signer has been blocklisted. Malicious verifiers and GMs may collude with each other; their goal is to track signers based on their signatures.

To prevent these threats, the proposed system must satisfy the following security requirements (detailed in Appendix B):

Blocklistability: Verifiers can revoke signers by adding them to a blocklist with the help of GMs. Revoked users are rejected by services in the next period or later.

¹Although not explicitly stated in the paper, this can be inferred from the fact that generating a non-membership witness with an RSA accumulator incurs a comp. cost linear in L for each new entry added to the RL (δ_L).

Unlinkability: Even under collusion, verifiers and GMs cannot track signers via signatures across “sessions”.

The term *session* here refers to a single unlinkability domain defined over a specified period of time. Actions within the same session may be linked to one another through temporary pseudonyms, accounts, cookies, or similar mechanisms. By contrast, actions belonging to different sessions must not be linkable. For example, a session in a social media service would be an account the user uses to submit posts; posts from the same account is expected to be linked but posts from separate accounts held by the same user must not be linked.

As with previous work, we make several assumptions. First, we assume that the GM and the verifier do not undermine zk-BAN’s blocklistability (e.g., by issuing credentials to malicious signers), as they have no incentive to do so. Second, GMs are assumed to not conduct attacks that are publicly detectable since they risk losing their reputation, public image, and user base. For example, we assume that the GM does not engage in large-scale rejection attacks to track individual signers. Additionally, global parameters (i.e., the GM’s public key gpk and the revocation list RL) are assumed to be published transparently, and we do not consider the possibility that the GM tracks signers by distributing different global parameters. Moreover, our system does not consider violations during the zk-SNARKs trusted setup, as such issues can be detected using a subversion-resistant zk-SNARKs [19] (see Section 10.1 for a related discussion). Third, side-channel attacks (e.g., timing correlation attack) are considered out of scope in our work. Our work also does not address threats arising from malware, the leakage of credentials and secret keys, or Denial-of-Service (DoS). Finally, we assume that the signers do not exceed their authentication limit.

5 Design

5.1 Conceptual Design

In this section, we present the basic design of zk-BAN. Following recent trends, we base the design on a signature-based revocation method that uses zk-SNARKs. However, as discussed previously, in signature-based revocation, the cost of proof generation and verification increases with the size of the revocation list. Therefore, zk-BAN reduces the proof-generation overhead by considering only the entries that were generated since the signer’s previous credential update. We refer to this subset as the *sub-revocation list* Δ_{RL} . During each period the signer proves its non-revocation status to the GM through a non-revocation proof over Δ_{RL} . After verifying the proof, the GM issues a credential that is valid for that period. Using the updated credential, the signer can authenticate to the service until the next period. In this paper, we refer to this approach as the *differential revocation-proof method*.

However, this method has a slight limitation; it reveals the period $period'$, the point at which a signer last updated their

credential. A signature containing period' therefore indicates that its signer has remained offline since that period. This reduces the size of the anonymity set from N signers to those who have not updated since period' . Since most signers in large-scale services are online throughout the day (whether actively participating, passively viewing content, or connected through background app communication), this reduction can be especially substantial for signers who have remained offline for long periods.

Our solution is to decouple the update and authentication protocols, which are typically unified in anonymous blocklisting systems, and remove any linkable information between them. Consequently, even if the adversary learns that a signer updated their credential in period' , it cannot cryptographically link the update request to the signer's authentication request. Thus, each authentication request can be attributed only to one of the N_{period} signers who updated to period , which roughly corresponds to the number of online signers. Since most signers in large-scale services are online throughout the day, this yields a sufficiently large anonymity set.²

In summary, zk-BAN consists of the following procedures:

1. **Registration:** After the signer proves its identity to the GM, the GM issues an initial credential to the signer.
2. **Authentication:** The signer signs service-specific information, and the verifier verifies the signature.
3. **Update:** The signer proves its non-revocation status, and the GM verifies the proof. If the proof is valid, the signer receives a refreshed credential from the GM.
4. **Revocation:** The verifier sends the signature to the GM, and the GM adds that signature to the revocation list.

5.2 Design Improvements

When realizing the above conceptual design, we introduced the following two key design improvements.

5.2.1 Improvement 1: Filtering the Sub-Revocation List

As described above, zk-BAN reduces the overhead of non-revocation proofs by restricting the proof target to the sub-revocation list Δ_{RL} . However, if a signer does not generate such a proof for an extended period, for example when the signer has been offline for a long time, both the GM and the signer may incur substantial computational costs. In that case, the number of revocations accumulated since the previous update may become very large. For instance, based on X's report that it revoked 9.5 million ad-related accounts per year [42], a user generating a non-revocation proof for the first time in a year would need to account for a sub-revocation list of that size. Using SNARKBlock, generating such a proof would take approximately 1.3 days, including batch processing.

²This guarantee concerns cryptographic unlinkability and does not account for correlations derived from update and authentication request timing.

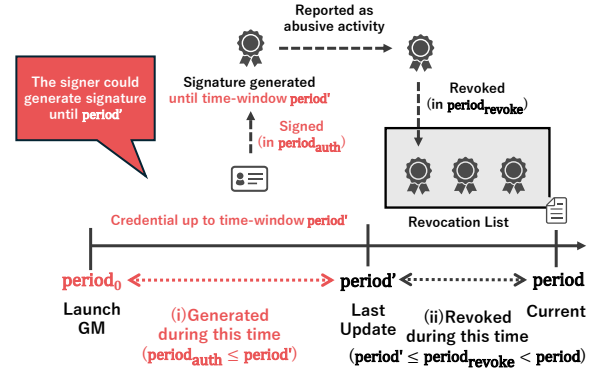


Figure 1: Conditions under which the malicious user's signatures appear in the sub-revocation list (Improvement 1)

Therefore, in this study, we propose a novel method to further reduce the size of the sub-revocation list. Figure 1 illustrates the idea. The key observation is that signatures generated after the signer's last update can be excluded from the sub-revocation list. This is because, once the signer has been revoked during period' , the GM no longer issues the valid credential required to generate signatures in later periods. Since the signer can obtain such a credential only through the update process, any valid signature generated after period' cannot belong to that signer if the signer did not complete the update. Based on this observation, we limit the signatures considered in the non-revocation proof to those satisfying both of the following conditions: (i) signatures revoked during the interval from the previous period period' to the current period period , and (ii) signatures generated at or before period' . Let $\text{period}_{\text{Auth}}$ denote the time at which a signature was generated and $\text{period}_{\text{Revoke}}$ denote the time at which it was revoked. Then, the above restriction is expressed as follows:

$$(\text{period}' \leq \text{period}_{\text{Revoke}} < \text{period}) \wedge (\text{period}_{\text{Auth}} \leq \text{period}')$$

Note that, even after narrowing the target in this manner, the sub-revocation list may still be large. In particular, if a large amount of abuse that occurred before period' is discovered only later, the size of the sub-revocation list may still increase. However, we believe such cases are rare in practice. This is because, in many services, the vast majority of abuse is detected and revoked shortly after it occurs rather than much later. For example, Wikipedia reverts 90% of problematic edits within five hours [2]. Additionally, Call of Duty aims to detect and remove misconduct within one hour [29]. Nevertheless, because not all abuse can be detected immediately, it remains important to preserve the ability to revoke misconduct discovered at a later point.

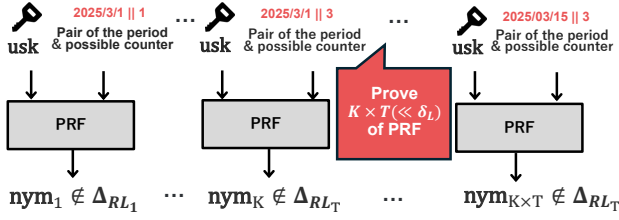


Figure 2: Method to reduce PRF computations (Improvement 2)

5.2.2 Improvement 2: Reducing PRF Computations

In signature-based revocation, the primary signer-side bottleneck is the repeated PRF-based zero-knowledge-argument computation, which requires computing the PRF for each tag in the sub-revocation list Δ_{RL} .

To address this issue, we propose a novel method that reduces the number of PRF proofs by defining the PRF tag tag using the current period period. We also include nym alongside the signature as part of the authentication artifact. In this case, it is sufficient to prove that nym is not contained in the sub-revocation list $\Delta_{RL} (= \{nym'_1, \dots, nym'_{\delta_L}\})$:

$$PK\{(\Delta_{RL}; usk, nym) : nym = PRF(\text{period}, usk) \wedge nym \notin \Delta_{RL}\}$$

When the signer proves non-revocation in a batch over multiple past periods, $\Delta_{PL} = \{\text{period}_1, \dots, \text{period}_T\}$, with a sub-revocation list $\Delta_{RL} = (\Delta_{PL}, \Delta_{RL_1}, \dots, \Delta_{RL_T})$, where t denotes the index of period period_t in Δ_{PL} , and Δ_{RL_t} is the subset of the sub-revocation list containing the signatures corresponding to period t , i.e., $\Delta_{RL_t} = (nym'_{t,1}, \dots, nym'_{t,\delta_{L_t}})$, this becomes:

$$PK\{(\Delta_{RL}; usk, nym_1, \dots, nym_T) : \bigwedge_{t=1}^T (nym_t = PRF(\text{period}_t, usk) \wedge nym_t \notin \Delta_{RL_t})\}$$

However, in this form, each signer can generate only one nym, and thus only one session, per period. We address this issue by appending a counter cnt to the tag, i.e., tag = (period, cnt). This increases the number of available tags and, in turn, the number of sessions, similar to the construction taken in anonymous rate-limiter [36]. Then, the zero-knowledge argument becomes as follows (see Figure 2):

$$PK\{(\Delta_{RL}; usk, nym_{1,1}, \dots, nym_{T,K}) : \bigwedge_{t=1}^T \bigwedge_{cnt=1}^K (nym_{t,cnt} = \text{Hash}(\text{period}_t, cnt, usk) \wedge nym_{t,cnt} \notin \Delta_{RL_t})\}$$

Including cnt in tags creates a problem: proving $0 < cnt \leq K$ during verification would require revealing cnt, allowing the service or other entities to link pseudonyms that reuse

cnt and violating unlinkability. zk-BAN avoids this by hiding cnt: it compares $nym_{t,cnt}$ with every $nym'_{t,cnt} \in \Delta_{RL_t}$, even if generated with a different counter, and proves $0 < cnt \leq K$ in zero knowledge at each authentication.

However, this introduces another issue: it increases non-membership proof costs. This is because the signer must prove $nym_{t,cnt} \notin \Delta_{RL_t}$ for every $nym' \in \Delta_{RL}$, even when nym' was not generated with cnt. Even still, our analysis shows that zk-BAN remains performant under realistic revocation volumes (Sections 7 and 8), since the reduction in hash-proof costs outweighs the increase in non-membership-proof costs in zk-SNARKs.

6 Protocol

In this section, we present the details of the zk-BAN protocol. To describe the protocol steps clearly, we use the notation summarized in Table 1 in Appendix A. Throughout this section, Sign and Verify denote the signing and verification functions, respectively. We omit the details of the zk-SNARK trusted setup, the constraint systems $CS_{Reg}, CS_{Auth}, CS_{Upd}$, the proving keys $PK = \{pk_{Reg}, pk_{Auth}, pk_{Upd}\}$, and the verification keys $VK = \{vk_{Reg}, vk_{Auth}, vk_{Upd}\}$. We refer the reader to Section 7.2 for a more detailed explanation. We assume that the GM has already generated its signing key pair (gsk, gpk) and published gpk.

6.1 Registration

A signer registers with the GM through the following steps:

1. Signer generates a registration request as follows:
 - (a) Signer randomly chooses a secret key usk.
 - (b) Signer computes its public key $upk_{\text{period}} \leftarrow \text{Hash}(\text{period}, \text{flag}_{upk}, usk)$ from the secret key usk, the current period period, and the flag flag_{upk} .
 - (c) Using the proving key pk_{Reg} , the signer generates the following proof π_{Reg} :

$$PK\{(upk_{\text{period}}, \text{period}; usk) : upk_{\text{period}} = \text{Hash}(\text{period}, \text{flag}_{upk}, usk)\}$$

2. Signer proves its identity to the GM and then sends upk_{period} and π_{Reg} .
3. GM verifies the registration request and issues a credential $cred_{\text{period}}$ as follows:
 - (a) GM uses the verification key vk_{Reg} to verify π_{Reg} .
 - (b) GM computes $cred_{\text{period}} \leftarrow \text{Sign}(upk_{\text{period}}, gsk)$.
4. GM sends $cred_{\text{period}}$ to the signer.
5. Signer stores usk, upk_{period} , $cred_{\text{period}}$, and cnt (= 1).

6.2 Authentication

Verifier authenticates the signer through the following steps:

1. Signer generates a signature over the message msg by:
 - (a) Signer chooses a random value r .
 - (b) Signer checks that $\text{cnt} \leq K$.
 - (c) Signer computes the message-derived value $\sigma \leftarrow \text{Hash}(\text{msg}, r, \text{usk})$ and the pseudonym $\text{nym} \leftarrow \text{Hash}(\text{period}, \text{cnt}, \text{usk})$.
 - (d) Signer generates the following proof π_{Auth} :

$$\begin{aligned}
 PK\{ & (\sigma, \text{nym}, \text{msg}, \text{gpk}, \text{period}; \\
 & \text{cnt}, \text{usk}, \text{upk}_{\text{period}}, \text{cred}_{\text{period}}, r) : \\
 & 0 < \text{cnt} \leq K \wedge \sigma = \text{Hash}(\text{msg}, r, \text{usk}) \\
 & \wedge \text{nym} = \text{Hash}(\text{period}, \text{cnt}, \text{usk}) \\
 & \wedge \text{upk}_{\text{period}} = \text{Hash}(\text{period}, \text{flag}_{\text{upk}}, \text{usk}) \\
 & \wedge 1 = \text{Verify}(\text{cred}_{\text{period}}, \text{upk}_{\text{period}}, \text{gpk}) \}
 \end{aligned}$$

- (e) Signer increments $\text{cnt} \leftarrow \text{cnt} + 1$.
2. Signer sends $(\sigma, \text{nym}, \pi_{\text{Auth}})$ and msg to the verifier.
3. Verifier uses vk_{Auth} to verify π_{Auth} .

6.3 Update

The signer updates its credential as follows. Suppose that the system has advanced from the previous period period' to the current period period , and that a signer holding a credential $\text{cred}_{\text{period}'}$ for period period' wishes to obtain a new credential $\text{cred}_{\text{period}}$ for period period . Note that the update phase is treated as a batch process and, therefore, does not need to be completed immediately in real time. For more information on the specific timing of updates, see Section 10.1.

1. Signer obtains a sub-revocation list Δ_{RL} as follows:
 - (a) Signer sends period' to GM.
 - (b) From the revocation list RL , GM extracts a sub-revocation list Δ_{RL} (i.e., a subset of RL) such that $(\text{period}' \leq \text{period}_{\text{Revoke}} < \text{period}) \wedge (\text{period}_{\text{Auth}} \leq \text{period}')$, where $\text{period}_{\text{Revoke}}$ is the period in which the revocation occurred and $\text{period}_{\text{Auth}}$ is the period in which the signature was created. Let the set of periods in Δ_{RL} be $\Delta_{PL} = \{\text{period}_1, \dots, \text{period}_T\}$, and for each period period let the associated pseudonyms be $\{\text{nym}'_{1,1}, \dots, \text{nym}'_{T, \delta_{LT}}\}$, where δ_{LT} denotes the number of signatures generated from period_t .
 - (c) GM sends Δ_{RL} to signer.
2. Signer generates an update request as follows:
 - (a) Signer computes the public keys $\text{upk}_{\text{period}} = \text{Hash}(\text{period}, \text{flag}_{\text{upk}}, \text{usk})$ and $\text{upk}_{\text{period}'} = \text{Hash}(\text{period}', \text{flag}_{\text{upk}}, \text{usk})$, as well as the one-time update ticket $\text{ticket}_{\text{period}'} = \text{Hash}(\text{period}', \text{flag}_{\text{ticket}}, \text{usk})$.

- (b) Signer generates the following proof π_{Upd} :

$$\begin{aligned}
 PK\{ & (\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \\
 & \text{period}, \text{period}', \text{gpk}, \Delta_{RL}, \text{usk}, \\
 & \text{upk}_{\text{period}'}, \text{cred}_{\text{period}'}, \text{nym}_{1,1}, \dots, \text{nym}_{T, K}) : \\
 & \text{upk}_{\text{period}} = \text{Hash}(\text{period}, \text{flag}_{\text{upk}}, \text{usk}) \\
 & \wedge \text{upk}_{\text{period}'} = \text{Hash}(\text{period}', \text{flag}_{\text{upk}}, \text{usk}) \\
 & \wedge \text{ticket}_{\text{period}'} = \text{Hash}(\text{period}', \text{flag}_{\text{ticket}}, \text{usk}) \\
 & \wedge 1 = \text{Verify}(\text{cred}_{\text{period}'}, \text{upk}_{\text{period}'}, \text{gpk}) \\
 & \bigwedge_{t=1}^T \bigwedge_{\text{cnt}=1}^K (\text{nym}_{t, \text{cnt}} = \text{Hash}(\text{period}_t, \text{cnt}, \text{usk}) \\
 & \wedge \text{nym}_{t, \text{cnt}} \notin \Delta_{RLt}) \}
 \end{aligned}$$

3. Signer sends the update request $(\text{period}', \text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}})$ to GM.
4. GM issues the credential $\text{cred}_{\text{period}}$ as follows:
 - (a) GM checks that $\text{ticket}_{\text{period}'}$ is not in signer's ticket list TL .
 - (b) As in step 1b, GM filters Δ_{RL} from RL .
 - (c) GM uses vk_{Upd} to verify the proof π_{Upd} (this can be precomputed in advance).
 - (d) GM appends $\text{ticket}_{\text{period}'}$ to TL .
 - (e) GM computes $\text{cred}_{\text{period}} \leftarrow \text{Sign}(\text{upk}_{\text{period}}, \text{gsk})$.
5. GM sends $\text{cred}_{\text{period}}$ to Signer.
6. Signer stores $\text{cred}_{\text{period}}$ and resets $\text{cnt} \leftarrow 1$.

6.4 Revocation

The GM and the verifier revoke a signer as follows. Let $\text{period}_{\text{Revoke}}$ be the revocation period and $\text{period}_{\text{Auth}}$ the period in which the signature was generated.

1. Verifier sends $\sigma, \text{msg}, \text{period}_{\text{Auth}}, \text{nym}, \pi_{\text{Auth}}$ to GM.
2. GM verifies the zero-knowledge argument π_{Auth} and confirms that msg is deemed abusive.
3. GM appends $(\text{nym}, \text{period}_{\text{Revoke}}, \text{period}_{\text{Auth}})$ to RL .

7 Implementation

The implementation of zk-BAN consists of four components³: **Core Library**, **System Library**, **Signer App**, and **Server**.

7.1 Core Library

This library is responsible for cryptographic operations and is implemented in Go for security and performance reasons. We used the Gnark zk-SNARK library [10], as it is highly performant compared to other libraries [17]. We also implemented a verification-preprocessing library for Gnark, since it does not natively support this functionality required by our

³The source code of zk-BAN is available at <https://github.com/nict-sflab/zk-ban-readme>.

update protocol. We used Groth16 for the zk-SNARK proofs, BLS12-381 as the elliptic curve, the SNARK-friendly Poseidon2 hash [21], and EdDSA as our signature scheme.

Challenges & Solutions

We encountered the following challenges during implementation of the core library.

Challenge & Solution 1. Although the method described in Section 5.2.2 reduces the number of PRF proofs from δ_L to $T \times K$, efficiently proving $\text{nym}_{t,\text{cnt}} \notin \Delta_{RL_t}$ remains challenging. One possible solution is to use a cryptographic accumulator, such as an Indexed Merkle Tree [40] or an RSA accumulator [25]. Although these constructions offer favorable asymptotic scalability, in our setting, each non-membership proof is evaluated over a relatively small list, as detailed later. At this scale, we do not expect accumulator-based approaches to provide meaningful performance improvements.⁴

Therefore, in this work, we choose to use many inequality proofs. This choice is based on our observation that constructing an accumulator for each Δ_{RL_t} would be more costly than proving multiple inequalities in our expected use-cases, especially because we have reduced the δ_{L_t} size. For instance, assuming X (Twitter) revokes 30,000 accounts per day over 30 days, the average revocation list size per period would be only 1,000 ($= 30,000/30 = \delta_L/T$). At this scale, a Merkle tree-based accumulator requires 10 ($\sim \log_2(1,000)$) hash proofs plus range proof for each $\text{nym}_{t,\text{cnt}}$, and since a single hash proof is hundreds of times more costly than an inequality proof [20], computing 1,000 inequality proofs is more efficient as inequality proofs only requires single constraints.

Challenge & Solution 2. If one implements the non-revocation proof naively, a different circuit must be prepared for each possible size of the sub-revocation list. This is because zk-SNARK circuits and their proving keys depend on the input size, i.e., the size of the sub-revocation list. However, this approach makes proving-key generation challenging. For example, although the GM could dynamically compile circuits for each non-revocation proof, circuit compilation is computationally expensive and therefore impractical.

To address this issue, SNARKBlock splits large revocation lists into smaller segments and uses a single circuit to prove non-revocation for each segment. This is because generating proofs for the entire list at once would produce massive proving and verification keys, increasing storage / download costs and memory requirements. Although reusing a single circuit reduces this overhead, multiple proofs must still be generated. SNARKBlock therefore aggregates them to reduce the proof size and verification time. However, aggregation causes both

the proof size and the verification latency to grow logarithmically with the list size, and the additional verification steps cannot be precomputed, which further increases latency as the revocation list grows.

In zk-BAN, we address this issue by having the GM pre-compile circuits for several possible sub-revocation list sizes and provide them to the signer in advance. During an update, a “suitable circuit” is selected, allowing the system to maintain a smaller proof size and lower verification latency. However, the signer and the GM may not always have a circuit of the appropriate size. We discuss such cases in Section 10.2.

At first glance, because zk-BAN does not split the revocation list into smaller chunks, the prover’s computational cost might appear high. In practice, however, zk-BAN adopts the differential revocation-proof method (see Section 5.1), which keeps the amount of input data small and therefore makes the cost of computing and storing the keys practical.

Challenge & Solution 3. Defining what qualifies as a “suitable circuit” is itself a challenge. In this work, we define such a circuit as follows: (a) A circuit in which the number of periods T and the number of revocations per period do not exceed the circuit’s τ and δ_{λ_t} , respectively; and (b) Among the circuits that satisfy condition (a), the one with the lowest proof-generation cost.

For condition (a), suppose that the signer’s sub-revocation list has T periods and that each period t has δ_{L_t} revocations. Then, the circuit must satisfy: $(T \leq \tau) \wedge (\delta_{L_t} \leq \delta_{\lambda_t}) (\forall t \in \{1, \dots, T\})$. Note that it is sufficient for τ and each δ_{λ_t} to be greater than or equal to T and δ_{L_t} , respectively. Even if some periods or pseudonyms in the sub-revocation list are absent from the circuit, all inputs must still be filled, using 0 in place of missing periods or pseudonyms. This works because the non-revocation condition $\text{Hash}(\text{period}, \text{cnt}, \text{usk}) \neq \text{nym}$ is satisfied if period or nym is 0. For condition (b), let the latency-increment factors for increasing the period count and the revocation count be c_τ and c_λ , respectively, and let τ and δ_λ denote the circuit’s capacities for periods and revocations. Then, the circuit minimizing proof-generation time is the one for which $\delta_\lambda \cdot c_\lambda + \tau \cdot c_\tau$ is smallest.

7.2 System Library

This library is responsible for invoking the core library, querying the database, and managing communications. We implemented the system library also in Go for the same security and performance reasons. All databases use SQLite and are managed through the ent ORM library. The server API was built using the echo framework. The library consists of signer, verifier, and GM sub-libraries. The signer library supports registration, signing, and updates while interfacing with the core library, GM, and verifier. The verifier library exposes signature-verification endpoints and outputs submitted signatures for revocation. The GM library provides endpoints for retrieving sub-revocation lists, registering signers and issuing

⁴Indexed tree accumulators are inefficient: for $\delta_{L_t} \approx 1,000$, they require about $200 \log_2(1,000) \approx 2,000$ constraints, roughly twice the cost of a simple non-equality proof. RSA accumulators are similarly inefficient, as each $\text{nym}_{t,\text{cnt}}$ requires $\delta_{L_t} \approx 1,000$ RSA-group operations [12], with no witness reuse across independent period-specific sub-revocation lists.

credentials, and processing updates according to Section 6.3 using stored state such as ticket , $\text{cred}_{\text{period}}$, and $\text{upk}_{\text{period}}$.

For simplicity and performance reasons, the precomputation result is kept in memory rather than in the database. The verification key is also loaded from disk into memory at startup. The precomputation result and verification keys are stored in a hashmap keyed by the pair of periods (period , period').

7.3 Signer App

The signer app is a client application that performs registration, authentication, and updates. We provide two versions of this app: a smartphone app and a desktop app. Appendix E shows the detail of the signer app specification.

7.4 Servers

We provide server implementations for both the GM and the verifier. The GM server extends the GM system library with Firebase-based SMS identity verification. The verifier server is built on the verifier server library and, in addition to the standard endpoints, hosts a website containing a custom URL that allows the signer to launch the signer app.

8 Performance Evaluation

This section reports various performance metrics of zk-BAN which are evaluated using parameters derived from real-world settings. For our experiment, we used a PC ($32 \times 2.0\text{GHz}$ (default) / 5.5GHz (turbo mode) CPU, 64GB RAM).

Baseline environment parameters. We first estimated the size of the sub-revocation list required for current large-scale systems. The numbers stated here are referred to as *baseline setting* in this section. See our scalability evaluation (Section 8.3) for an evaluation of zk-BAN that exceeds these baseline settings. Note that, in all the following experiments, we used $\tau = T$ and $\delta_{\lambda_i} = \delta_L$ for the number of circuit inputs.

First, we set the number of revoked signatures in the sub-revocation list to $\delta_L = 30,000$. This value was calculated from the yearly number of account revocations in X (see Section 3), assuming that updates are performed daily.⁵

Additionally, we set the maximum number of sessions to $K = 5$, derived from the maximum number of accounts that can be made with a single device in X [1, 3, 13]. This value should be sufficient for most users, as it exceeds the average number of X accounts (1.13 active accounts / user [34]). The actual number of sessions may be smaller than that of the accounts, as some users may allow linkage between some of their accounts. In addition to the case of X, this value of K is sufficiently large even compared to Google Chrome’s

⁵Strictly speaking, X reported that it suspended 9.49 million accounts ($= 4.19 + 5.30$ million) [42, 43], implying that δ_L should be $26000 = 9.49\text{M}/365$. Nevertheless, we use $30,000 = 30 \times 1000$, representing an upper bound.

Private State Tokens feature, which allows only two sessions per website within a 48-hour period [31].

Next, we set the number of time windows contained in the sub-revocation list to $T = 30$ based on the fact that the typical log retention period is 30 days [9, 27, 32], and that many services detect malicious activities within 30 days. For example, a previous study reported that Twitter (now X) suspended 92% of spam accounts within 3 days of their first post [37]. A similar trend is likely to persist in X. Moreover, most illicit activities are detected within *a few hours* in other services, e.g., Wikipedia [2], CoD [29]. Therefore, $T = 30$ can be considered a sufficiently conservative value. The periods in sub-revocation lists do not need to be contiguous in zk-BAN, e.g., if you have sub-revocation lists for only yesterday and 30 days ago, zk-BAN treats this as $T = 2$.

8.1 Sub-revocation list Filtering Evaluation (Improvement 1)

This section evaluates the effectiveness of our first design improvement (sub-revocation list reduction) using the Twitch moderation data from the EU DSA Transparency Database [18]. We use this dataset because it provides the timestamps of violation occurrence and moderation enforcement necessary to accurately calculate δ_L and T . Note that the parameters in the Twitch dataset do not match the target deployment setting for Improvement 2; the overall performance of zk-BAN in the Twitch case, including Improvement 2, is reported in Appendix G.

Setup. We fix period to July 31, 2026, and vary period' to evaluate the extent to which Improvement 1 reduces δ_L and T . To model the worst-case scenario, we assume that a revocation that is added to the list is never removed.

Result. Figure 3 shows T and δ_L with and without filtering. Under the classical signature-based revocation scheme, both values increase as period' ages and reaches $\delta_L = 544763$, $T = 375$. In contrast, with zk-BAN, δ_L remains uniform approximately 3726 and T below approximately 18, largely independent of period' . When period' is one year old, zk-BAN reduces δ_L to approximately $1/146$ and T to approximately $1/20.8$ of the conventional scheme. Thus, the benefit of Improvement 1 grows as signers stay offline for longer periods.

The above results provide sufficient evidence of the effectiveness of Improvement 1. Hence, the remaining evaluations focus only on showing the effectiveness of Improvement 2.

8.2 Baseline Measurement (Improvement 2)

Setup. We evaluated the performance of authentication and update latencies in the baseline environment.



Figure 3: Comparison of δ_L and T between classical signature based revocation and zk-BAN

The update latency may vary depending on the distribution of per-session revocations in the sub-revocation list. Therefore, in these experiments, we conducted three configurations of sub-revocation lists: (i) with constant revocation counts per period number, (ii) with linearly increasing revocation counts per period number, and (iii) with revocation counts following a (truncated) Gaussian distribution per period number, and observed the differences in latency.

Results. The numbers for the authentication process are low (53.1 ms for proof generation and 2.46 ms for verification), showing that this process is efficient. The update process requires a closer look. The update verification latency is sufficiently lightweight for precomputation and extremely efficient for the main computation. Also, we found that in the update process, the configurations (i)–(iii) do not affect the latency, and on the prover side, it takes about 600 ms ((i) 600 ms / (ii) 599 ms / (iii) 611 ms). These numbers may seem large at first glance, but considering that immediate responsiveness is not required for the update process, this latency is sufficiently acceptable. Similarly, for the GM side, the configurations do not significantly increase latency, since it only took about 2 ms ((i) 1.64 ms / (ii) 2.65 ms / (iii) 2.70 ms) for verification. Thus, we claim that zk-BAN’s verification is fast.

8.3 Scalability Evaluation (Improvement 2)

Setup. We evaluated zk-BAN’s scalability capabilities in environments with parameters larger than the baseline setting by measuring: (1) update latency while varying δ_L in the sub-revocation list, with K fixed at 5 and T fixed at 30; (2) update latency while varying T , with δ_L fixed at 30,000 and K fixed at 5; and (3) update latency while varying K , with δ_L fixed at 30,000 and T fixed at 30.

As with the previous section, the latency may vary due to how the revocations in the sub-revocation list are distributed

among sessions. Hence, as before, we used the configurations (i)–(iii) in our experiments.

Additionally, this experiment was performed only on the PC. This is because the purpose of this experiment is not to measure the latency values themselves but to clarify the trend by which the latency increases.

Results. Figure 4 shows the results of the scalability evaluation. As shown in the figure, signer-side latency increases linearly with δ_L (the number of revocations in the sub-revocation list). The results show that, even when the number of revocations in the sub-revocation list reaches 300,000 (i.e., about 110 million revocations per year), the signer-side latency is only about 3.47 s. We can observe that the GM’s verification latency hardly changes with respect to δ_L , taking only about 1 ms even as δ_L varies. This is because the process dependent on δ_L was moved to the precomputation verification phase (see Section 8.6 for more details) in Groth16, allowing the main verification phase to remain constant.

In addition, we can see that the signer-side latency increases linearly with T (the number of periods in the sub-revocation list), while the GM-side latency did not change significantly. For $T = 720$, which represents two-years-worth of revocations ($\sim 365 \times 2$), the signer-side latency was around 4.05 s. This shows that zk-BAN is practical even in large-scale environments, given that updates are handled as a batch process.

Furthermore, the figure shows that the signer-side latency increases linearly with K (the maximum number of sessions within the sub-revocation list), while the GM-side latency again does not change significantly, similar to what we observed when varying δ_L . In fact, for $K = 100$ (i.e., users can create 100 unlinkable sessions per day), the signer-side latency was around 10.2 s. Although this may seem large, this is still smaller than that of our comparison implementation and SNARKBlock in the ablation study (see Section 8.4 and Section 8.5). From these results, we further conclude that zk-BAN is scalable under modeled real-world conditions.

8.4 Ablation Evaluation (Improvement 2)

Setup. To further evaluate the efficacy of our hash reduction strategy, we performed an ablation study where we compare the baseline-condition non-revocation proof generation latency of zk-BAN (cf Section 5.2.2) and a classical signature-based revocation system (cf Section 2, proof of $h = \text{PRF}(\text{tag}, \text{usk})$ is removed). Note that, in this experiment, we used the uniform-distribution variant (i.e., configuration (i)) for the sub-revocation list, since the previous experiment showed that the distribution does not affect latency.

Result. Our results show that zk-BAN takes 647 ms, whereas the classical signature-based revocation approach takes 35.2 s, showing that our reduction strategy incurs an approximately 54.4-fold performance improvement.

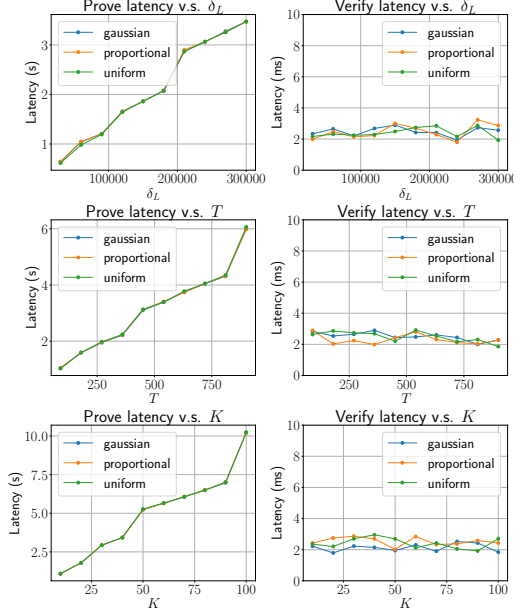


Figure 4: Results of the Scalability Experiment

8.5 Comparison Experiment with Prior Work (Improvement 2)

Setup. We measured and compared the latency of generating non-revocation proofs, specifically the prover latency of the update process in zk-BAN (uniform Δ_{RL} setting) and the sync processes in two prior works, SNARKBlock [30] and ALPACA [23]. We selected these two works because they are among the most performant modern signature-based revocation schemes and their source code is publicly available. The update process in zk-BAN is essentially identical to the sync processes in SNARKBlock and ALPACA, except that zk-BAN additionally requires online communication with the GM. This communication is introduced solely to realize Improvement 1 and is independent of Improvement 2. Because the PRF/Hash computation targeted by Improvement 2 occurs only during the zk-BAN’s update process, comparing its update latency with the sync latency of SNARKBlock and ALPACA provides a suitable evaluation of Improvement 2. For this evaluation, we set $\delta_L = 32768$ and tuned the parameters to maximize the performance of the two prior works while keeping their configs close to X’s use case, as detailed in Appendix H.

Results. Sync latencies for SNARKBlock and ALPACA were 88.4 s and 1.94 h, respectively. In contrast, zk-BAN’s update process took 648 ms, achieving 136 times lower latency than SNARKBlock and 10800 times lower latency than ALPACA.

8.6 Additional Measurements

Comparison of Other Latencies with Prior Work. We compare the proof-generation and verification latencies of zk-BAN (uniform Δ_{RL}) during authentication, as well as its update-verification latency, with those of SNARKBlock and ALPACA. For a fair comparison, we report the sum of zk-BAN’s authentication- and update-verification latencies, since zk-BAN splits non-revocation proof verification across these two procedures.

Our results show that zk-BAN generates authentication proof more than $15.1\times$ and $26.4\times$ faster than SNARKBlock and ALPACA, respectively. Specifically, while SNARKBlock and ALPACA require 805 ms and 1.35 s, respectively, zk-BAN requires only 51.2 ms. This lower online cost stems from zk-BAN requiring signers to prove only possession of a valid credential during the authentication, whereas related work additionally requires proving possession of a valid non-revocation proof generated during the sync phase. Consequently, zk-BAN does not natively support immediate revocation as authentication does not check the signer’s revocation status, exposing an explicit trade-off between authentication performance and support for immediate revocation.

For verification, zk-BAN is $32\times$ more efficient than SNARKBlock and ALPACA. Specifically, verification in zk-BAN takes only 4.25 ms ($= 2.41 \text{ ms} + 1.83 \text{ ms}$) ms, compared with 136 ms for SNARKBlock and 168 ms for ALPACA. Thus, zk-BAN achieves an order-of-magnitude lower verification latency than both systems, primarily due to avoiding aggregation-proof verification.

Proving Latency on a Smartphone. We also evaluate the baseline performance and scalability of zk-BAN on a smartphone ($1 \times 3.1\text{GHz} + 3 \times 2.5\text{GHz} + 4 \times 1.95\text{GHz}$ CPU, 16GB RAM). We conduct the measurements necessary to characterize performance on mobile devices. Accordingly, we use only the uniform revocation trend, as latency independent of the revocation trend. For the scalability evaluation, we measure three endpoint configs, varying one parameter at a time: $\delta_L = 300,000$, $T = 720$, or $K = 100$. This is sufficient because latency is known to increase linearly with each parameter.

Under the baseline setting, zk-BAN requires only 184 ms for signing and 4.92 s for updates, which shows its feasibility given that the update process is batched (despite of being 10 times larger than on a PC). Although the zk-BAN’s latency on a smartphone is approx. 10 times higher than on a PC, it remains within a tolerable range: 39.8 s for $\delta_L = 300,000$ and 29.1 s for $T = 720$ and 103 s for $K = 100$. Note that this does not imply that zk-BAN is less efficient than related work; rather, results suggest that the latency of related work would still be higher than our baseline smartphone latency.

Verification Pre-computation Latency. At period boundaries, the GM performs precomputation for verification as batch processes. To quantify this overhead, we conduct both a baseline evaluation and a scalability evaluation, following

the above methodology.

In the baseline setting, the GM’s pre-verification overhead is moderate. Pre-verification takes about 60 ms (i.e., (i) 63.7 ms / (ii) 61.6 ms / (iii) 60.0 ms) per period. Thus, even assuming that the GM precomputes two-years worth of verification data, the computation time is only about 44 s ($= 60 \text{ ms} \times 365 \times 2$). This result suggests that the overhead is practical for real-world deployment. As the revocation list size and the number of periods increase, the GM’s pre-verification cost increases. Nevertheless, zk-BAN remains scalable. Figure 5 in Appendix D.1 shows the results. The pre-verification latency grows linearly with both the revocation list size and the number of periods. Even with 300,000 revoked users, the latency remains as low as 4.87 min ($= 400 \text{ ms} \times 365 \times 2$) for two years, indicating that zk-BAN scales.

Bandwidth Consumption (Detailed in Appendix D.2). During authentication, the signer sends only the signature elements: nym , σ (each 32 B), and π_{Auth} (244 B), totaling to 308 B. In the update phase, the signer sends the 316 B update request, which consists of 8 B period' , 32 B $\text{upk}_{\text{period}}$, ticket, and a proof 244 B. The total request size 624 B is 41x smaller than SNARKBlock (75.87 kB with L in Section 8.5) and ALPACA (25.6 KB). In addition, the signer downloads a modestly-sized 64 B credential and a 961 kB sub-revocation list ($961 \text{ kB} = (T + \delta_L) \times 32 \text{ B}$). This is smaller compared with SNARKBlock and ALPACA sub-revocation lists; their overhead can reach $1.92 \text{ MB} = (2 \times \delta_L \times 32 \text{ B})$ per day.

Storage Consumption (Detailed in Appendix D.3). The signer stores small keys and credentials (e.g., 32-byte keys usk and gpk , and a 64-byte cred_i), together with circuit-related data, such as constraint systems and proving keys, for a total storage cost of 1.15 GB. Although this cost may appear large, it accounts for at most approximately 0.5% of the storage capacity of modern devices; desktop systems commonly provide terabyte-scale storage, while smartphones now commonly offer 256 GB or more. Most importantly, unlike related work, the proving keys in zk-BAN can be reused across different services and GM, allowing their storage cost to be amortized across deployments. For example, when shared across five services, the effective per-service storage cost is only 228 MB.

Importantly, this proving-key storage overhead is an artifact of our current implementation rather than an inherent requirement of the zk-BAN protocol. Our implementation uses a total of nine proving keys and corresponding constraints for the update circuits (Appendix D.3). By adopting SNARKBlock’s aggregation technique, zk-BAN could instead use one proving key across all circuits, at the cost of increased latency. Under this design, the single proving key (and constraint system, if required) for the update would be smaller than those of the alternatives: 102 MB in the setting of Section 8.4 and 108 MB in the setting of Section 8.5, compared with 7.42 GB for the signature-based revocation implementation used in our ablation study, 153 MB for SNARKBlock with chunk size 1024, 4.73 GB with chunk size 32768, and 479 MB for ALPACA.

However, as previously noted, storage cost is a direct trade-off with GM update verification performance; in this work, we chose to sacrifice the former for the latter.

Verifier-side storage is minimal, accounting for gpk and verifying keys (32 B plus KB-level or smaller artifacts). The GM is required to store gsk , gpk , RL , TL , and any user public-key/credential pairs needed for multi-device sharing. If we assume the baseline scenario for all X accounts (i.e., 557 million users) each updating 365 times, tickets alone can require terabytes (e.g., 6.50 TB) of storage, although this is normal for any large-scale service. For instance, services of a similar scale consume petabytes for profile images, which is significantly larger compared to what the GM must store.

9 Security Analysis

We show that zk-BAN satisfies the blocklistability and unlinkability requirements. Due to page constraints, a more detailed analysis is given in Appendix C, which presents game-based definitions of these requirements, together with the corresponding theorems and proofs.

To establish blocklistability, it suffices to show that authentication by a revoked user cannot succeed after the revocation period begins. To this end, we show that it is infeasible to generate authentication requests or update requests that pass verification in a post-revocation period. This property follows from the fact that no user can forge a credential without gsk , together with the requirement that each request satisfy the properties enforced by the zero-knowledge proof. In addition, as shown in Section 5, a user is revoked when its signature is included in the sub-revocation list. Moreover, in the update phase, replay attacks such as reusing an old credential can be detected by the GM through duplicate tickets. Therefore, we claim that zk-BAN satisfies blocklistability.

Two signatures in zk-BAN are unlinkable if it is infeasible to determine whether they were produced by the same signer. To establish this property, it suffices to show that two authentication requests in zk-BAN remain indistinguishable even when both are observed, and that other requests or responses do not reveal information that helps distinguish them. Each authentication request in zk-BAN contains independent hash values computed from the secret key usk , the public parameter period , and fresh values cnt and r . These values reveal no information for linking two authentication requests. Similarly, an update request contains a one-time ticket and independent hash values derived from usk , period , and period' , together with period' . Since neither authentication nor update requests contain information that links them, update requests cannot help link authentication requests. Therefore, zk-BAN satisfies unlinkability.

Anonymity set size analysis. Although signers that have not updated in the current period cannot authenticate and are excluded from the anonymity set, the anonymity set in

each period comprises N_{period} signers that updated during that period, and is sufficiently large for large-scale services.

10 Discussion

10.1 Additional Security Improvements

Single Member Groups In any anonymous blocklisting protocol, including zk-BAN, the GM can easily identify users by placing each user in a separate group. However, this requires the GM to reject many registration or update requests, allowing the public to easily detect such an attack [4].

Transparently Managing Global Parameters zk-BAN requires several parameters, such as gpk and RL , to be globally consistent; otherwise, a malicious GM could abuse these parameters to track or de-anonymize a target signer, e.g., by issuing a unique gpk to that signer or revoking all other signers. This threat can be mitigated by publishing the relevant parameters and their updates on a transparent bulletin board (e.g., a Certificate Transparency log or a blockchain).

Compromised Trusted Setup zk-SNARKs require a trusted setup to generate pk, vk . Without this setup, a malicious GM could subvert these keys to extract usk from proofs or signatures. Using a subversion-resistant zk-SNARK [19] could mitigate this threat. Although verifying the setup output may become expensive, this is a one-time cost per circuit. Signers can also share verification results and publish pk, vk on a transparent bulletin board to ensure consistent parameters.

Framing Attacks and Countermeasures If the trusted setup is compromised, the GM can also forge signatures and use them to falsely revoke legitimate signers. Subversion-resistant zk-SNARK alone cannot fully prevent this attack, as it guarantees either completeness or soundness of the zero knowledge proofs, but not both simultaneously. If a signer detects such an attack, they can challenge the GM with the assistance of another GM by running a fresh trusted setup and showing that the secret key in the forged signature differs from their own.

Defenses against Timing Correlation Attacks If a user registers or updates and signs soon afterward, the verifier or GM may infer that these actions were performed by the same user. Although this kind of timing correlation attack is out-of-scope in zk-BAN, we can consider several mitigations: 1) hide timing differences via simultaneous updates; 2) randomizing update times; and 3) pre-distributing credentials for the next period.

Adding Immediate Revocation Although we do not support immediate revocation for simplicity’s sake, zk-BAN can be extended to support this by adding a proof of non-revocation during authentication, with the added cost of increased proof-generation latency reported in 8.4.

Limiting Per-signer Revocation Several signature-based anonymous blocklisting protocols are vulnerable to DoS attacks in which an adversary generates excessive signatures over a short period [7, 11, 23, 30, 39], which increases the size of the revocation list and thus the prover’s latency [41]. zk-BAN does not have this issue, because it limits the number of sessions per signer; GM performs only K revocations per user per period by checking the nym.

10.2 Additional Discussions on Efficiency

Cases where no appropriate proving/verification key is found If δ_L or T exceeds the range supported by the keys generated during initial setup, no suitable proving/verification key may be available. We expect such cases to be rare as the system is provisioned with keys covering a sufficiently broad parameter range. Nevertheless, we consider two remedies: 1) generating new keys on demand by having the GM run setup for the required parameter range; or 2) splitting the proof across multiple smaller circuits. For the latter, our measurements (Sections 8.5 and 8.6) show that, under the baseline setting, zk-BAN remains faster than the online phase of SNARKBlock/ALPACA even with up to 30 partitions and without proof aggregation.

11 Limitations

Despite its advantages, zk-BAN has several limitations:

Protocol: First, the anonymity set in zk-BAN is limited to users who were online during the relevant period, rather than the entire user population. This limitation arises from the differential revocation scheme, which reveals period' , the signer’s last credential-update period. To mitigate this problem, zk-BAN’s design requires signers to contact the GM periodically for updates, although this increases the GM’s request volume. In addition, zk-BAN requires a sufficiently large amount of users to be online each day through various means including background communication, which we assume is the case for large-scale services targeted by zk-BAN.

Second, zk-BAN provides substantial performance improvements primarily for services in which users initiate relatively few sessions per period and most abusive behavior is detected and revoked within a short delay. Although some delayed detections are tolerable, zk-BAN’s performance depends jointly on the per-user per-period session count and the delay between the onset of abuse and its detection.

Implementation: Our current zk-BAN implementation, though not the protocol itself, imposes a non-trivial burden on users as it requires them to store large zk-SNARK proving keys. Although the key can be reused across different services and GMs, reducing proving-key size, for example through more succinct proof systems, tailored circuit designs, or cryptographic mechanisms such as proof aggregation and IVC remains important future work.

12 Conclusion & Future Work

We proposed zk-BAN, a novel signature-based anonymous blocklisting protocol that overcomes the performance limitations of prior work by introducing two overlooked design improvements. We show that our proof-of-concept implementation achieves feasible performance under real-world large-scale constraints and outperforms the state-of-the-art signature-based system. Future work includes implementing the additional features presented in the discussion section.

References

- [1] How many X (Twitter) accounts from one device and from one IP? <https://www.blackhatworld.com/security/how-many-x-twitter-accounts-from-one-device-and-from-one-ip.1680295/>. [Online; accessed 2026-04-04].
- [2] Baseline: Median Time to Revert (Probable Vandalism). https://nbviewer.org/github/wikimedia-research/automoderator-measurement/blob/main/baselines/T348860_median_time_to_revert.ipynb#Time-to-Revert-Percentiles, 2 2024. [Online; accessed 2025-08-26].
- [3] Motoki Abe. How to Add an Account to X/Twitter and Switch Between Accounts (Translated from Japanese). <https://news.mynavi.jp/article/20231130-2830116/>. [Online; accessed 2026-04-27].
- [4] Kosei Akama, Yoshimichi Nakatsuka, Masaaki Sato, and Keisuke Uehara. Scrappy: SeCure Rate Assuring Protocol with PrivacY. In *Proceedings 2024 Network and Distributed System Security Symposium*. Internet Society. doi:10.14722/ndss.2024.24445.
- [5] Electronic Arts. Apex Legends™: Anti-Cheat. "https://www.ea.com/games/apex-legends/news/anti-cheat-update-082124?setLocale=en-us", 8 2024. [Online; accessed 2025-01-11].
- [6] Man Ho Allen Au and Apu Kapadia. PERM: Practical reputation-based blacklisting without TTPs. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 929–940, 2012. doi:10.1145/2382196.2382294.
- [7] Man Ho Allen Au, Apu Kapadia, and Willy Susilo. BLACR: TTP-free blacklistable anonymous credentials with reputation. In *Proceedings 2012 Network and Distributed System Security Symposium*. Internet Society, 2012.
- [8] Man Ho Allen Au, Patrick P. Tsang, and Apu Kapadia. PEREA: Practical TTP-free revocation of repeatedly misbehaving anonymous users. *ACM Transactions on Information and System Security (TISSEC)*, 14(4):1–34, 2011. doi:10.1145/2043628.2043630.
- [9] Auth0. Log Data Retention. <https://auth0.com/docs/deploy-monitor/logs/log-data-retention>. [Online; accessed 2025-08-23].
- [10] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.14.0, June 2025. doi:10.5281/zenodo.5819104.
- [11] Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing. 2009.
- [12] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–469, 2022.
- [13] João Carrasqueira. Twitter details recent improvements to its PWA, including support for multiple accounts. <https://www.neowin.net/news/twitter-details-recent-improvements-to-its-pwa-including-support-for-multiple-accounts/>. [Online; accessed 2026-04-27].
- [14] Sherman S. M. Chow, Jack P. K. Ma, and Tsz Hon Yuen. Scored anonymous credentials. In *International Conference on Applied Cryptography and Network Security*, pages 484–515. Springer, 2023. doi:10.1007/978-3-031-33491-7_18.
- [15] Haotian Deng, Mengxuan Liu, Chuan Zhang, Wei Huang, Licheng Wang, and Liehuang Zhu. ShadowBlock: Efficient Dynamic Anonymous Blocklisting and Its Cross-chain Application. *arXiv preprint arXiv:2512.19124*, 2025.
- [16] Android Developers. Integrate Credential Manager with your credential provider solution. <https://developer.android.com/identity/sign-in/credential-provider>. [Online; accessed 2025-08-26].
- [17] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur

- Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. In *International Conference on Security and Cryptography for Networks*, pages 46–72. Springer, 2024. doi:10.1007/978-3-031-71070-4_3.
- [18] European Commission, DG CONNECT. Digital Services Act Transparency Database, 2023. doi:10.2906/134353607485211.
- [19] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In *Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25–29, 2018, Proceedings, Part I 21*, pages 315–347. Springer, 2018. doi:10.1007/978-3-319-76578-5_11.
- [20] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schafnegg, and Roman Walch. Reinforced concrete: A fast hash function for verifiable computation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1323–1335, 2022.
- [21] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schafnegg. Poseidon2: A faster version of the poseidon hash function. In *International Conference on Cryptology in Africa*, pages 177–203. Springer, 2023.
- [22] Joel Hruska. Riot Will Use Windows 11’s TPM 2.0 Requirement to Ban Cheaters From Valorant. <https://www.extremetech.com/gaming/326740-riot-will-use-windows-11s-tpm-2-0-requirement-to-ban-cheaters-from-valorant>, 9 2021. [Online; accessed 2026-03-31].
- [23] Jiwon Kim, Abhiram Kothapalli, Orestis Chardouvelis, Riad S. Wahby, and Paul Grubbs. ALPACA: Anonymous Blocklisting with Constant-Sized Updatable Proofs. *IEEE Symposium on Security and Privacy*.
- [24] Robert Lemos. Intel chip ID has ‘no rules, no privacy’. <https://www.zdnet.com/article/intel-chip-id-has-no-rules-no-privacy/>, 2 1999. [Online; accessed 2025-08-23].
- [25] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *International Conference on Applied Cryptography and Network Security*, pages 253–269. Springer, 2007.
- [26] Jack P. K. Ma and Sherman S. M. Chow. SMART credentials in the multi-queue of slackness (or Secure management of anonymous reputation traits without global halting). In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 896–912. IEEE, 2023.
- [27] OpenAI. Data controls in the OpenAI platform. <https://platform.openai.com/docs/guides/your-data#data-retention-controls-for-abuse-monitoring>. [Online; accessed 2025-08-23].
- [28] Stack Overflow. How to open a web page in Google chrome incognito from android app. <https://stackoverflow.com/questions/37696033/how-to-open-a-web-page-in-google-chrome-incognito-from-android-app>. [Online; accessed 2025-08-26].
- [29] Team RICOCHET. RICOCHET Anti-Cheat Progress Report Black Ops 6 Launch. <https://www.callofduty.com/blog/2024/10/call-of-duty-ricochet-anti-cheat-progress-report-black-ops-6>, 10 2024. [Online; accessed 2025-08-26].
- [30] Michael Rosenberg, Mary Maller, and Ian Miers. Snark-block: Federated anonymous blocklisting from hidden common input aggregate proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 948–965. IEEE, 2022.
- [31] Google Privacy Sandbox. Private State Tokens developer guide. <https://privacysandbox.google.com/protections/private-state-tokens/developer-guide>, 3 2025. [Online; accessed 2025-08-26].
- [32] Jen Sells. Cloudflare enables native monitoring and forensics with Log Explorer and custom dashboards. <https://blog.cloudflare.com/monitoring-and-forensics>, 3 2025. [Online; accessed 2025-08-26].
- [33] Maurice Shih, Michael Rosenberg, Harikesh Kailad, and Ian Miers. zk-Promises: Making Zero-Knowledge Objects Accept the Call for Banning and Reputation. 2024.
- [34] Similarweb. Estimating Twitter’s Bot-Free Monetizable Daily Active Users (mDAU). <https://dfdn.ews.com/2022/09/15/estimating-twitthers-bot-free-monetizable-daily-active-users-mdau/>. [Online; accessed 2025-08-23].
- [35] David Soler, Carlos Dafonte, Manuel Fernández-Veiga, Ana Fernández Vilas, and Francisco J Nóvoa. Federated Anonymous Blocklisting across Service Providers and its Application to Group Messaging. *arXiv preprint arXiv:2511.03486*, 2025.
- [36] Isamu Teranishi, Jun Furukawa, and Kazue Sako. K-times anonymous authentication. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 308–322. Springer, 2004.

- [37] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258, 2011.
- [38] Rachel Thomas, Oliwia Kempinski, Hari Kailad, Emma Margaret Shroyer, Ian Miers, and Gabriel Kaptchuk. Cryptographic Personas: Responsible Pseudonyms Without De-Anonymization. *Cryptology ePrint Archive*, 2025.
- [39] Patrick P. Tsang, Man Ho Allen 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. doi:10.1145/1880022.1880033.
- [40] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. 2022.
- [41] Stephan van Schaik, Alexander Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. Sok: Sgx. fail: How stuff gets exposed. 2024. URL: <https://sgx.fail/files/sgx.fail.pdf>, doi:10.1109/SP54263.2024.00260.
- [42] X. GLOBAL TRANSPARENCY REPORT H1 2024. <https://transparency.x.com/content/dam/transparency-twitter/2024/x-global-transparency-report-h1.pdf>. [Online; accessed 2025-01-16].
- [43] X. GLOBAL TRANSPARENCY REPORT H2 2024. <https://transparency.x.com/en/reports/global-reports/2025-transparency-report>. [Online; accessed 2025-04-04].
- [44] Li Xi and Dengguo Feng. FARB: fast anonymous reputation-based blacklisting without TTPs. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 139–148, 2014. doi:10.1145/2665943.2665947.
- [45] Kin-Ying Yu, Tsz Hon Yuen, Sherman S. M. Chow, Siu-Ming Yiu, and Lucas Chi Kwong Hui. PE (AR) 2: Privacy-enhanced anonymous authentication with reputation and revocation. In *Computer Security—ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings 17*, pages 679–696. Springer, 2012. doi:10.1007/978-3-642-33167-1_39.

A Notation

We use standard notations in mathematical formulas and elsewhere in the paper, which are summarized in Table 1.

B List of zk-BAN functionalities

This section provides a list of zk-BAN functionalities. All queries return $\perp$ if it fails.

$\text{Setup}() \rightarrow (\text{gsk}, \text{gpk}, \mathbf{PK}, \mathbf{VK}, \mathbf{RL}(= \emptyset), \mathbf{TL}(= \emptyset))$: GM generates group public and group secret keys (§ 6).

$\text{Reg}_{\text{Signer}}(\text{pk}_{\text{Reg}}, \text{period}) \rightarrow ((\pi_{\text{Reg}}, \text{upk}_{\text{period}}, \text{id}), \text{usk})$: Signer generates key pair and registration proof (§6.1, (1)).

$\text{Reg}_{\text{GM}}(\text{id}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{vk}_{\text{Reg}}, \text{gsk}, \text{period}, \mathbf{IL}) \rightarrow (\text{cred}_{\text{period}}, \mathbf{IL})$: GM verifies signer's registration proof and issues a credential (§6.1, (3)).

$\text{Auth}_{\text{Signer}}(\text{usk}, \text{cred}_{\text{period}}, \text{msg}, \text{pk}_{\text{Auth}}, \text{gpk}, \text{period}, \text{cnt}) \rightarrow ((\sigma, \text{nym}, \pi_{\text{Auth}}), \text{cnt})$: Signer generates a signature (§6.2, (1)).

$\text{Auth}_{\text{Verifier}}(\text{msg}, \sigma, \text{nym}, \pi_{\text{Auth}}, \text{vk}_{\text{Auth}}, \text{gpk}, \text{period}) \rightarrow 1$: Verifier verifies the signature (§6.2, (3)).

$\text{Fetch}(\text{period}, \text{period}', \mathbf{RL}) \rightarrow \Delta_{\mathbf{RL}}$: GM retrieves revocation-list difference $\Delta_{\mathbf{RL}}$ for the period from period' to period (§6.3, (1)).

$\text{Upd}_{\text{Signer}}(\text{usk}, \text{cred}_{\text{period}'}, \text{pk}_{\text{Upd}}, \text{gpk}, \text{period}, \text{period}', \Delta_{\mathbf{RL}}) \rightarrow (\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}})$: Signer generates an update request (§6.3, (2)).

$\text{Upd}_{\text{GM}}(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{gsk}, \text{gpk}, \text{period}, \text{period}', \Delta_{\mathbf{RL}}, \mathbf{TL}) \rightarrow (\text{cred}_{\text{period}}, \mathbf{TL})$: GM verifies the update request and issues a new credential for period period and updated $\mathbf{TL}$ (§6.3, (4)).

$\text{Revoke}(\text{nym}, \text{period}_{\text{Auth}}, \text{period}_{\text{Revoke}}, \mathbf{RL}) \rightarrow \mathbf{RL}$: GM updates revocation list $\mathbf{RL}$ with tuple $(\text{nym}, \text{period}_{\text{Auth}}, \text{period}_{\text{Revoke}})$ (§6.4).

C Formal Security Analysis

In this section, we provide a proofs showing that zk-BAN's protocol satisfies *blocklistability* and *unlinkability*, defined in Section 4. We assume that the underlying zk-SNARK satisfies soundness and zero-knowledge and that the signature scheme is existentially unforgeable, as well as random oracle models, under assumptions shown in Section 4.

Oracle Settings

For the security analysis, we introduce the following oracles:

$\mathcal{O}_{\text{block}}^{\text{Reg}}() \rightarrow (\text{cred}_{\text{period}}, \text{usk})$: Call $\text{Reg}_{\text{Signer}}(\text{pk}_{\text{Reg}}, \text{period})$ to obtain $((\pi_{\text{Reg}}, \text{upk}_{\text{period}}, \text{id}), \text{usk})$, and then call $\text{Reg}_{\text{GM}}(\text{id}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{vk}_{\text{Reg}}, \text{gsk}, \text{period}, \mathbf{IL})$ to obtain $(\text{cred}_{\text{period}}, \mathbf{IL})$. Then insert $(\text{usk}, \text{cred}_{\text{period}}, \text{period})$ into $\mathbf{UL}_{\text{block}}$ and return $(\text{cred}_{\text{period}}, \text{usk})$.

$\mathcal{O}_{\text{unlink}}^{\text{Reg}}() \rightarrow (\text{cred}_{\text{period}}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{id})$: Call $\text{Reg}_{\text{Signer}}(\text{pk}_{\text{Reg}}, \text{period})$ to obtain $((\pi_{\text{Reg}}, \text{upk}_{\text{period}}, \text{id}), \text{usk})$, and then call $\text{Reg}_{\text{GM}}(\text{id}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{vk}_{\text{Reg}}, \text{gsk}, \text{period}, \mathbf{IL})$ to obtain $(\text{cred}_{\text{period}}, \mathbf{IL})$. Then insert $(\text{id}, (\text{usk}, \text{cred}_{\text{period}}, \text{cnt}=1, \text{period}))$ into $\mathbf{UL}_{\text{unlink}}$ and return $(\text{cred}_{\text{period}}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{id})$.

$\mathcal{O}_{\text{unlink}}^{\text{Auth}}(\text{msg}, \text{id}) \rightarrow (\sigma, \text{nym}, \pi_{\text{Auth}})$: Pick up $(\text{id}, (\text{usk}, \text{period}', \text{cnt}', \text{period}')) \in \mathbf{UL}_{\text{unlink}}$, and call $\text{Auth}_{\text{Signer}}(\text{usk}, \text{cred}_{\text{period}}, \text{msg}, \text{pk}_{\text{Auth}}, \text{gpk}, \text{period}, \text{cnt}')$ to obtain $((\sigma, \text{nym}, \pi_{\text{Auth}}), \text{cnt})$. Note that period is the current time, not period' in $\mathbf{UL}_{\text{unlink}}$. Therefore, return $\perp$ for this proof failure if $\text{period} \neq \text{period}'$. Update the entry for id in $\mathbf{UL}_{\text{unlink}}$ from $(\text{id}, (\text{usk}, \text{cred}_{\text{period}}, \text{cnt}', \text{period}'))$ to $(\text{id}, (\text{usk}, \text{cred}_{\text{period}}, \text{cnt}, \text{period}'))$, and return $(\sigma, \text{nym}, \pi_{\text{Auth}})$.

$\mathcal{O}_{\text{block}}^{\text{Upd}}(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{period}, \text{period}') \rightarrow \text{cred}_{\text{period}}$: Retrieve the difference in the revocation-list by $\Delta_{\mathbf{RL}} \leftarrow \text{Fetch}(\text{period}, \text{period}', \mathbf{RL})$. Then call $\text{Upd}_{\text{GM}}(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{gsk}, \text{gpk}, \text{period}, \text{period}', \Delta_{\mathbf{RL}}, \mathbf{TL})$ to obtain $\text{cred}_{\text{period}}$. Update the entries for usk corresponding to $\text{upk}_{\text{period}}$ and $\text{ticket}_{\text{period}'}$ (i.e., $\text{upk}_{\text{period}} = \text{Hash}(\text{period}, \text{flag}_{\text{upk}}, \text{usk}) \wedge \text{ticket}_{\text{period}'} = \text{Hash}(\text{period}', \text{flag}_{\text{ticket}}, \text{usk})$) in $\mathbf{UL}_{\text{block}}$ from $(\text{usk}, \text{cred}_{\text{period}'})$ to $(\text{usk}, \text{cred}_{\text{period}}, \text{period})$, and return $\text{cred}_{\text{period}}$.

$\mathcal{O}_{\text{unlink}}^{\text{Upd}}(\text{id}) \rightarrow (\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{period}')$: Check $\text{period}' < \text{period}$ and pick up $(\text{id}, (\text{usk}, \text{cred}_{\text{period}'}, \text{cnt}, \text{period}')) \in \mathbf{UL}_{\text{unlink}}$. Recall the difference in the revocation-list by $\Delta_{\mathbf{RL}} \leftarrow \text{Fetch}(\text{period}, \text{period}', \mathbf{RL})$, and call $\text{Upd}_{\text{Signer}}(\text{usk}, \text{cred}_{\text{period}'}, \text{pk}_{\text{Upd}}, \text{gpk}, \text{period}, \text{period}', \Delta_{\mathbf{RL}})$ to obtain $(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}})$. Then call $\text{Upd}_{\text{GM}}(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{gsk}, \text{gpk}, \text{period}, \text{period}', \Delta_{\mathbf{RL}}, \mathbf{TL})$ to obtain $(\text{cred}_{\text{period}}, \mathbf{TL})$. Update the entry for id in $\mathbf{UL}_{\text{unlink}}$ from $(\text{id}, (\text{usk}, \text{cred}_{\text{period}'}, \text{cnt}, \text{period}'))$ to $(\text{id}, (\text{usk}, \text{cred}_{\text{period}}, \text{cnt}(=1), \text{period}))$, and return $(\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{period}')$.

$\mathcal{O}_{\star}^{\text{Fetch}}(\text{period}, \text{period}') \rightarrow \Delta_{\mathbf{RL}}$: Retrieve the difference in the revocation-list using $\Delta_{\mathbf{RL}} \leftarrow \text{Fetch}(\text{period}, \text{period}', \mathbf{RL})$ and return it.

$\mathcal{O}_{\star}^{\text{Revoke}}(\text{nym}, \text{period}_{\text{Auth}}) \rightarrow 1$: Update the revocation list using $\mathbf{RL} \leftarrow \text{Revoke}(\text{nym}, \text{period}_{\text{Auth}}, \text{period}, \mathbf{RL})$, and return 1.

$\mathcal{O}_{\star}^{\text{Next}}() \rightarrow 1$: Set $\text{period}' \leftarrow \text{period}$ and increment $\text{period} \leftarrow \text{period} + 1$. Return 1.

C.1 Blocklistability

Blocklistability in zk-BAN means that, once a signer has been revoked by the GM, they must not be able to generate any new valid authentication requests after the current time-window. More concretely, after being revoked in period $\text{period}'_{\text{Revoke}}$, the signer should not be able to produce a valid signature for any period $\text{period}_{\text{Test}} (> \text{period}'_{\text{Revoke}})$. We took inspiration from [23] for the definition of blocklistability; an adversary cannot generate a valid signature once all signers have been revoked based on their previously generated signatures. In this section, we give a game-based definition of blocklistability and show that zk-BAN meets this requirement.

Blocklistability Game. Let λ be the security parameter. Consider the following game between a challenger C and an adversary $\mathcal{A}$.

1. C initializes the system with the global parameters $(\text{gsk}, \text{gpk}, PK, VK, RL, TL \leftarrow \text{Setup}())$ and $UL_{\text{block}} (= \emptyset)$, then gives all of them except gsk to $\mathcal{A}$.
2. $\mathcal{A}$ may issue any number of queries $O_{\text{block}}^{\text{Reg}}, O_{\text{block}}^{\text{Upd}}, O_{\star}^{\text{Fetch}}, O_{\star}^{\text{Revoke}}, O_{\star}^{\text{Next}}$. For each query, C honestly executes the corresponding protocol and returns the resulting transcript

$$\begin{aligned} T_{\text{Reg}} &= (\text{cred}_{\text{period}}, \text{usk}) \leftarrow O_{\text{block}}^{\text{Reg}}() \\ T_{\text{Upd}} &= \text{cred}_{\text{period}'} \\ &\leftarrow O_{\text{block}}^{\text{Upd}}(\text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{period}, \text{period}') \\ T_{\text{Revoke}} &= 1 \leftarrow O_{\star}^{\text{Revoke}}(\text{nym}, \text{period}) \\ T_{\text{Fetch}} &= \Delta_{RL} \leftarrow O_{\star}^{\text{Fetch}}(\text{period}, \text{period}') \end{aligned}$$

(or $\perp$ when it fails) to $\mathcal{A}$. $\mathcal{A}$ is allowed to observe transcripts for multiple signers. Note that $\mathcal{A}$ can directly compute the following values, since it has usk and $\text{cred}_{\text{period}'}$:

$$\begin{aligned} ((\sigma, \text{nym}, \pi_{\text{Auth}}), \text{cnt}) &= R_{\text{Auth}} \leftarrow \text{Auth}_{\text{Signer}}(\dots) \\ (\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}) &= R_{\text{Upd}} \leftarrow \text{Upd}_{\text{Signer}}(\dots) \end{aligned}$$

3. $\mathcal{A}$ chooses and sends a pair of (msg, cnt) (where $0 < \text{cnt} \leq K$) to C . C then revokes all signers by calling $((\sigma, \text{nym}, \pi_{\text{Auth}}), \text{cnt}) \leftarrow \text{Auth}_{\text{Signer}}(\text{usk}, \text{cred}_{\text{period}}, \text{msg}, \text{pk}_{\text{Auth}}, \text{gpk}, \text{period}, \text{cnt})$ with received msg and $O_{\star}^{\text{Revoke}}(\text{nym}, \text{period})$ for each $(\text{usk}, \text{cred}_{\text{period}}, \text{period}) \in UL_{\text{block}}$. Subsequently, C records the current period as $\text{period}'_{\text{Revoke}} \leftarrow \text{period}$.
4. $\mathcal{A}$ may continue queries to $O_{\text{block}}^{\text{Upd}}, O_{\star}^{\text{Fetch}}, O_{\star}^{\text{Revoke}}, O_{\star}^{\text{Next}}$ and obtain $T_{\text{Upd}}, T_{\text{Revoke}}, T_{\text{Fetch}}$.
5. Finally, $\mathcal{A}$ outputs an authentication request

$$R^* = (\text{msg}^*, \text{nym}^*, \sigma^*, \pi_{\text{Auth}}^*)$$

on any arbitrary message msg^* .

We say that $\mathcal{A}$ wins the game if R^* is a valid authentication request at arbitrary period $\text{period}_{\text{Test}}$ as follows: $\exists \text{period}_{\text{Test}} > \text{period}'_{\text{Revoke}}$ such that

$$1 = \text{Auth}_{\text{Verifier}}(\text{msg}^*, \sigma^*, \text{nym}^*, \pi_{\text{Auth}}^*, \text{vk}_{\text{Auth}}, \text{gpk}, \text{period}_{\text{Test}})$$

Theorem 1 (Blocklistability of zk-BAN.). *In the above blocklistability game, the success probability of any polynomial-time adversary $\mathcal{A}$ is negligible:*

$$\text{Adv}_{\text{zkban}}^{\text{block}}(\mathcal{A}) \leq \text{negl}(\lambda).$$

Proof. Assume, for the sake of contradiction, that there exists an adversary $\mathcal{A}$ that wins the blocklistability game. Then, $\mathcal{A}$ outputs a fresh valid authentication request, R^* , for the period after revocation, during which all keys have been revoked. Thus, it suffices to show that the probability of successfully forging each type of request is negligible.

Since $\mathcal{A}$ must forge some parameters in order to win the blocklistability game, we can prove the blocklistability of zk-BAN by establishing the unforgeability of zk-BAN. More specifically, to obtain such a valid R^* , $\mathcal{A}$ must effectively produce either the authentication request R^* itself or an update request R_{Upd} corresponding to $\text{period}_{\text{Test}}$. This requires $\mathcal{A}$ to forge the relevant parameters: an obviously invalid R^* would fail $\text{Auth}_{\text{Verifier}}(\dots)$, while a valid R_{Upd} requires that $\forall \text{nym}_t \notin \Delta_{RL_t}$ for all t . Note that, in the update phase for $\text{period}_{\text{Test}}$ from $\text{period}'_{\text{Test}}$, Δ_{RL} surely contains nym values for all signers as shown in step 3) in the blocklistability game. This is because, in this game, the last period $\text{period}'_{\text{Test}}$ is no later than the revocation period $\text{period}_{\text{Revoke}}$, while the current period $\text{period}_{\text{Test}}$ is later than $\text{period}_{\text{Auth}}$ and $\text{period}'_{\text{Test}}$ is no earlier than $\text{period}_{\text{Auth}}$. Hence, $\mathcal{A}$'s signatures are included in Δ_{RL} for the query:

$$\begin{aligned} &(\text{period}'_{\text{Test}} \leq \text{period}_{\text{Revoke}} < \text{period}_{\text{Test}}) \\ &\wedge (\text{period}_{\text{Auth}} \leq \text{period}'_{\text{Test}}) \end{aligned}$$

$\mathcal{A}$ may also attempt a replay attack to during the update phase by reusing an old credential; however, this attack is prevented because the GM would detect duplicate $\text{ticket}_{\text{period}'}$ values. Hence, $\mathcal{A}$ must also forge $\text{ticket}_{\text{period}'}$.

Therefore, we show the unforgeability of relevant parameters (i.e., $\pi_{\text{Reg}}, \pi_{\text{Auth}}, \pi_{\text{Upd}}, \text{upk}_{\text{period}}, \text{cred}_{\text{period}}, \text{period}', \text{period}$) in zk-BAN to prove blocklistability by reducing it to the unforgeability of zero-knowledge proofs and digital signatures (i.e., EUF-CMA).

1. **Hyb1. Forging the Zero-Knowledge Proof.** Suppose $\mathcal{A}$ succeeds by forging the relations among the internal values contained in the transcript. That is, $\mathcal{A}$ forges $\text{period}, \text{period}', \text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \text{cred}_{\text{period}}$ and Δ_{RL} and thereby passes verification. The relations between these parameters are enforced by the zk-SNARK

proofs π_{Auth} and π_{Upd} attached to the authentication or update phase. Hence, if $\mathcal{A}$ manages to make a transcript that does not satisfy these relations pass verification, this contradicts the soundness of the underlying zk-SNARK.

2. **Hyb2. Forging the Digital Signature.** $\mathcal{A}$ might try to generate an update request by forging the credential $\text{cred} = \text{Sign}(\text{upk}_{\text{period}'}, \text{gsk})$ generated under the issuer's secret key gsk and period period' . In other words, $\mathcal{A}$ may attempt to generate a valid credential cred on a public key $\text{upk}_{\text{period}'}$ corresponding to a forged secret key gsk . However, this can contradict the unforgeability of the signature scheme. $\square$

C.2 Unlinkability

Another security requirement of zk-BAN is unlinkability. That is, zk-BAN must guarantee that signers cannot be traced from their signatures, i.e., given two signatures, no polynomial-time adversary should be able to determine, with non-negligible advantage, whether they were produced by the same signer. Note that we do not have to consider the timing correlation attack, forgery of global parameters, and linking with single-member groups, as discussed in Section 4.

In this section, we give a game-based definition and show that zk-BAN satisfies the unlinkability requirement.

Unlinkability Game. Let λ be the security parameter. Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. $\mathcal{C}$ initializes the system with the global parameters $(\text{gsk}, \text{gpk}, \text{PK}, \text{VK}, \text{RL}, \text{TL} \leftarrow \text{Setup}())$ and $UL_{\text{unlink}} (= \emptyset)$, then gives all of them except UL_{unlink} to $\mathcal{A}$.
2. $\mathcal{A}$ may call polynomial many queries $O_{\text{unlink}}^{\text{Reg}}, O_{\text{unlink}}^{\text{Auth}}, O_{\text{unlink}}^{\text{Upd}}, O_{\text{unlink}}^{\text{Fetch}}, O_{\text{unlink}}^{\text{Revoke}}, O_{\text{unlink}}^{\text{Next}}$ and observe the following transcripts:

$$\begin{aligned} T_{\text{Reg}} &= (\text{cred}_{\text{period}}, \text{upk}_{\text{period}}, \pi_{\text{Reg}}, \text{id}) \leftarrow O_{\text{unlink}}^{\text{Reg}}() \\ T_{\text{Auth}} &= (\text{nym}, \sigma, \pi_{\text{Auth}}) \leftarrow O_{\text{unlink}}^{\text{Auth}}(\text{msg}, \text{id}) \\ T_{\text{Upd}} &= (\text{upk}_{\text{period}}, \text{ticket}_{\text{period}'}, \pi_{\text{Upd}}, \text{period}') \\ &\quad \leftarrow O_{\text{unlink}}^{\text{Upd}}(\text{id}) \\ T_{\text{Revoke}} &= 1 \leftarrow O_{\text{unlink}}^{\text{Revoke}}(\text{nym}, \text{period}) \\ T_{\text{Fetch}} &= \Delta_{\text{RL}} \leftarrow O_{\text{unlink}}^{\text{Fetch}}(\text{period}, \text{period}') \end{aligned}$$

Note that $\mathcal{A}$ can directly verify authentication requests similar to the blocklistability game:

$$1 = R_{\text{AuthVerifier}} \leftarrow \text{Auth}_{\text{Verifier}}(\dots)$$

3. $\mathcal{A}$ chooses two distinct user IDs id_0, id_1 ($\text{id}_0 \neq \text{id}_1$) which are not revoked by the signer, and a message msg_{Test} .

4. $\mathcal{C}$ generates authentication request $R_{\text{Test}} (\neq \perp)$ from id_0 as follows

$$R_{\text{Test}} \leftarrow O_{\text{unlink}}^{\text{Auth}}(\text{msg}_{\text{Test}}, \text{id}_0)$$

and send R_{Test} to $\mathcal{A}$.

5. $\mathcal{A}$ may continue querying the $O_{\text{unlink}}^{\text{Reg}}, O_{\text{unlink}}^{\text{Auth}}, O_{\text{unlink}}^{\text{Upd}}, O_{\text{unlink}}^{\text{Fetch}}, O_{\text{unlink}}^{\text{Revoke}}, O_{\text{unlink}}^{\text{Next}}$ oracles, and obtain $T_{\text{Reg}}, T_{\text{Auth}}, T_{\text{Upd}}, T_{\text{Revoke}}, T_{\text{Fetch}}$.
6. $\mathcal{A}$ chooses a message m'_{Test} and sends it to $\mathcal{C}$.
7. $\mathcal{C}$ samples a random bit $b \in \{0, 1\}$ and generates authentication requests $R'_{\text{Test}} (\neq \perp)$ from id_0 and $R''_{\text{Test}} (\neq \perp)$ from id_1 as follows:

$$R'_{\text{Test}} \leftarrow O_{\text{unlink}}^{\text{Auth}}(\text{msg}'_{\text{Test}}, \text{id}_0)$$

$$R''_{\text{Test}} \leftarrow O_{\text{unlink}}^{\text{Auth}}(\text{msg}'_{\text{Test}}, \text{id}_1)$$

Then, $\mathcal{C}$ chooses R'_{Test} if $b = 0$ (**0. same signer**) R''_{Test} else if $b = 1$ (**1. different signer**) as R'_{Test} , and sends R'_{Test} to $\mathcal{A}$. In addition, $\mathcal{C}$ defines this period as $\text{period}'_{\text{Test}}$.

8. $\mathcal{A}$ may additionally continue to query $O_{\text{unlink}}^{\text{Reg}}, O_{\text{unlink}}^{\text{Auth}}, O_{\text{unlink}}^{\text{Upd}}, O_{\text{unlink}}^{\text{Fetch}}, O_{\text{unlink}}^{\text{Revoke}}, O_{\text{unlink}}^{\text{Next}}$ oracles, and obtain $T_{\text{Reg}}, T_{\text{Auth}}, T_{\text{Upd}}, T_{\text{Fetch}}$. In this time, $O_{\text{unlink}}^{\text{Revoke}}()$ with R'_{Test} is prohibited to prevent a trivial link with failed update request after revocation, as well as BLAC [39].
9. Based on the two authentication requests and all information obtained so far, $\mathcal{A}$ outputs a bit b' : $b' = 0$ if it judges that the two requests come from the same signer and 1 otherwise. $\mathcal{A}$ wins if $b = b'$.

In the above game, we avoid trivial linking of signers that falls under the cases we mention in our limitation and security assumption sections, i.e., signers that have not updated until the latest period and signers who have exceeded their access limit K . We therefore describe these cases in the above game by excluding cases in which signature generation fails.

Theorem 2 (Unlinkability of zk-BAN). *For any polynomial-time adversary $\mathcal{A}$ in the above unlinkability game, its advantage is negligible:*

$$\text{Adv}_{\text{zkban}}^{\text{unlink}}(\mathcal{A}) = |\Pr[b' = b] - \frac{1}{2}| \leq \text{negl}(\lambda).$$

Proof. Suppose that there exists an adversary $\mathcal{A}$ with non-negligible advantage in the above game. Then $\mathcal{A}$ can distinguish, with non-negligible advantage, whether two authentication requests R_{Test} and R'_{Test} came from the same signer. This includes not only the case where it directly distinguishes R_{Test} and R'_{Test} , but also the case where the transcripts $T_{\text{Reg}}, T_{\text{Auth}}, T_{\text{Upd}}$ obtained from its various queries allow R_{Test} and R'_{Test} to be linked. In this setting, we prove unlinkability by showing that authentication requests are unlinkable (case A) and by non-extractability of additional knowledge from all transcripts (case B):

Case A (Unlinkability of authentication requests in zk-BAN). In zk-BAN, under the assumption of the random oracle model and the zero-knowledge property of zk-SNARKs, $\mathcal{A}$ gains no advantage in determining b' from the parameters R_{Test} and R'_{Test} :

1. **Hyb1. Linking signatures from Hashed Values.** In zk-BAN, the values $\sigma_{\text{Test}}, \text{nym}_{\text{Test}}$ (and $\sigma'_{\text{Test}}, \text{nym}'_{\text{Test}}$) are all outputs of a hashed value from the secret key usk and one-time fresh value (msg, r) or $(\text{period}, \text{cnt})$. Therefore, if $\mathcal{A}$ can gain more knowledge from σ, nym , then we could construct $f(\sigma, \text{nym})$ that distinguishes the hash output from a truly random sequence, contradicting the random oracle model. The tuples σ, nym can thus be treated, for each usk, as independently generated random strings under the random oracle model.
2. **Hyb2. Linking signatures from Zero-Knowledge Proofs.** The zero-knowledge property of the zk-SNARK proofs π_{AuthTest} and π_{AuthTest}' prevents the adversary $\mathcal{A}$ from knowing whether the signers of the π_{AuthTest} and π_{AuthTest}' are identical. This is because if one could distinguish the signer from π_{AuthTest} and π_{AuthTest}' , then we could construct an adversary that distinguishes real proofs from simulator outputs, contradicting zero-knowledge.

The remaining data, i.e., msg , is specified by $\mathcal{A}$. Therefore, $\mathcal{A}$ can respond only to randomly chosen authentication requests with $b' \xleftarrow{\$} \{0, 1\}$, under the zero-knowledge property of zk-SNARKs and in the random oracle model.

Case B (Non-Extractability of additional knowledge in zk-BAN). $\mathcal{A}$ cannot extract additional information from the transcripts $T_{\text{Reg}}, T_{\text{Auth}}, T_{\text{Upd}}$ under the assumptions of zero-knowledge in zk-SNARKs and random oracle models:

1. **Hyb1. Extract Advantage Knowledge from Hashed Values.** In zk-BAN, the values $\text{ticket}_{\text{period}'}, \text{upk}_{\text{period}}, \text{nym}, \sigma$ are all outputs of a Hash value from the secret key usk and a one-time fresh value $(\text{period}', \text{flag}_{\text{ticket}}), (\text{period}, \text{flag}_{\text{upk}}), (\text{msg}, r), (\text{period}, \text{cnt})$. Therefore, if $\mathcal{A}$ could compute some function value $f(\text{ticket}_{\text{period}'}, \text{upk}_{\text{period}}, \text{nym}, \sigma)$ that provides $\mathcal{A}$ an advantage, then we could construct an adversary that distinguishes the Hash output from a truly random sequence, contradicting the random oracle model. Therefore, under random oracle models, the tuples $\text{ticket}_{\text{period}'}, \text{upk}_{\text{period}}, \text{nym}, \sigma$ can be treated, for each usk, as independently generated random strings (subject only to consistency for the same signer and same input).
2. **Hyb2. Extract Advantage Knowledge from Zero-Knowledge Proofs.** By the zero-knowledge property of the zk-SNARK proofs $\pi_{\text{Reg}}, \pi_{\text{Auth}}, \pi_{\text{Upd}}$, the adversary $\mathcal{A}$ obtains no additional information about any secrets such as usk, cred from these proofs. This is because if one could compute from the proofs some new function value that makes advantage to win the game, then we could

construct an adversary that distinguishes real proofs from simulator outputs, contradicting zero-knowledge.

The remaining data, i.e., id, msg , and period' , are not helpful in determining b' . Indeed, msg is chosen by $\mathcal{A}$, id is merely a user identifier and is not related to the request data, and period' is a public time period, which is also not related to requests. Hence, none of these values gives $\mathcal{A}$ any useful advantage. Therefore, under the zero-knowledge property of zk-SNARKs and in the random oracle model, $\mathcal{A}$ cannot do better than to generate a random guess $b' \xleftarrow{\$} \{0, 1\}$. $\square$

Additionally, we show the anonymity set of zk-BAN as follows:

Lemma 3 (Anonymity set of zk-BAN). *In zk-BAN, let U_{period} be the set of users that update to period period , where $|U_{\text{period}}| = N_{\text{period}}$. Given a signature generated by a uniformly chosen user $R^* \xleftarrow{\$} U_{\text{period}}$, any PPT adversary $\mathcal{A}$ can identify R^* with probability at most*

$$\frac{1}{N_{\text{period}}} + \text{negl}(\lambda).$$

Proof. Let $R^* \xleftarrow{\$} U_{\text{period}}$ be the signer of the challenge signature σ^* , and let $\hat{R}$ denote the output of $\mathcal{A}$.

Define the adversary's anonymity advantage as

$$\text{Adv}_{\text{zkban}}^{\text{Anon}}(\mathcal{A}) := \Pr[\hat{R} = R^*] - \frac{1}{N_{\text{period}}}.$$

By Theorem 2, for any two users $R_i, R_j \in U_{\text{period}}$, signatures generated by R_i and R_j in the same period period are computationally indistinguishable. Hence, for every PPT adversary, its advantage in distinguishing whether a given signature was generated by R_i or R_j is negligible.

Assume, for contradiction, that there exists a PPT adversary $\mathcal{A}$ such that

$$\text{Adv}_{\text{zkban}}^{\text{Anon}}(\mathcal{A})$$

is non-negligible. Then $\mathcal{A}$ identifies the signer with a non-negligible advantage over uniform guessing. In particular, this implies that $\mathcal{A}$ can distinguish the signature distributions of at least two users in U_{period} with non-negligible advantage, contradicting Theorem 2.

Therefore,

$$\text{Adv}_{\text{zkban}}^{\text{Anon}}(\mathcal{A}) \leq \text{negl}(\lambda),$$

and consequently,

$$\Pr[\hat{R} = R^*] \leq \frac{1}{N_{\text{period}}} + \text{negl}(\lambda).$$

$\square$

D Additional Performance Measurements

D.1 Verification Precomputation

In this section, we show how the precomputation latency scales with δ_L , T , and K in Figure 5. As shown in the figure, the latency increases with δ_L , whereas no clear trend is observed for T or K . Although the precomputation latency is expected to increase slightly with T , because larger verification parameters affect the verification latency in Groth16, we could not observe such a trend in our experiments. This is likely because the maximum value of T was too small for the effect to be noticeable. We also confirm that K does not noticeably affect the latency in our measurements.

D.2 Bandwidth Consumption

In this section, we analyze the bandwidth consumption in the authentication and update phases.

Authentication During authentication, the signer sends only the signature. The signature consists of nym and σ , each of size 32 B, and π_{Auth} of size 244 B, for a total of 308 B, which is extremely small.

Update During an update, the signer sends the previous period period' to the GM, receives the proving keys and the sub-revocation list from the GM, sends the update request, and finally receives a new credential. The total communication amounts to 961 kB, which is modest. Below we provide a breakdown.

- **Revocation List Request:** This request contains only period' , which is 8 B.
- **Return of the Sub-Revocation List:** The size of the sub-revocation list that the GM sends to the signer is proportional to the total number of period T and the total number of revoked subscriptions δ_L^6 . Each element is 32 B, so the total size is $(T + \delta_L) \times 32$ B. In the baseline case, the size of the sub-revocation list is 961 kB $= (30 + 30,000) \times 32$ B.
- **Update Request:** The update request consists of period' (8 B), $\text{upk}_{\text{period}}$ and $\text{ticket}_{\text{period}'}$ (each 32 B), and π_{Upd} (244 B).
- **Credential Return:** The credential $\text{cred}_{\text{period}}$ returned by the GM has size 64 B.

D.3 Storage Consumption

In this section, we measure the storage required by each entity. The signer stores usk , upk , $\text{cred}_{\text{period}}$, gpk , the constraint systems (CSs) cs_{Reg} , cs_{Auth} , cs_{Upd} , and the proving keys

pk_{Reg} , pk_{Auth} , pk_{Upd} . The GM stores gpk , gsk , TL , the list of user ticket-public key pairs, the CSs cs_{Reg} , cs_{Upd} , and the verifying keys vk_{Reg} , vk_{Upd} . The verifier stores gpk , cs_{Auth} , and vk_{Auth} .

Keys and Credentials The signer stores usk , upk , gpk , and $\text{cred}_{\text{period}}$, the verifier stores gpk , and the GM stores gsk and gpk . usk , upk , and gpk are each 32 B, gsk is 96 B, and $\text{cred}_{\text{period}}$ is 64 B. Thus, the total storage required for these items is 128 B for the signer, 32 B for the verifier, and 128 B for the GM. These sizes are negligible in practice.

Revocation List (RL) RL stores, for each revoked signature, the tuple $(\text{nym}, \text{period}_{\text{Auth}}, \text{period}_{\text{Revoke}})$. Assuming the baseline requirements, we compute the GM's annual storage consumption. Since nym are each 32 B, and $\text{period}_{\text{Auth}}$, $\text{period}_{\text{Revoke}}$ are 8 B, a single entry is 48 B $(= 32 \text{ B} + 8 \text{ B} \times 2)$. If there are 9.49 million such entries, the total size is 455 MB $(= 48 \text{ B} \times 9.49 \times 10^6)$.

Ticket List (TL) TL stores the signer's ticket history $\text{ticket}_{\text{period}'}$, with a new entry added whenever the signer updates their keys. To mimic current large-scale services, we set the total number of users to 557 million (cf. the current number of all accounts in X). If we assume 557 million signers each updating their keys every day, then the total number of entries in TL becomes $203 \times 10^9 (= 557 \times 10^6 \times 365)$. Since each $\text{ticket}_{\text{period}'}$ is 32 B, the total storage consumption is 6.50 TB $= 203 \times 10^9 \times 32 \text{ B}$.

List of Credentials and Public Keys In our implementation of zk-BAN, the GM stores pairs of credentials and public keys to share credentials across the signer's devices. Here, we calculate the associated storage consumption. The GM needs to retain only one period's worth of pairs; credentials and public keys from previous periods can be discarded when a new period begins. Each entry consists of a 32 B upk_i and a 64 B cred_i , and we assume 557 million users as above. Therefore, the total size is 53.5 GB $(= (32 \text{ B} + 64 \text{ B}) \times 557 \times 10^6)$. This storage overhead arises from our implementation for cross-device credential sharing and is not inherent to the zk-BAN protocol itself.

Storage Related to the Circuit Itself In zk-SNARKs, key pairs and CSs consume storage on the GM and the signer. We evaluate this overhead below.

- **Registration (cs_{Reg} , pk_{Reg} , vk_{Reg}):** We generate the CS, the proving key, and the verification key required for registration and measure their sizes: 42.1 kB for the CS, 97.6 kB for the proving key, and 588 B for the verification key. Each is sufficiently small for practical use.

⁶In our implementation, to simplify deserialization, the GM provides a sub-revocation list zero-padded to match the input size of the circuit. As a result, the transmitted revocation list size is proportional to τ and δ_L .

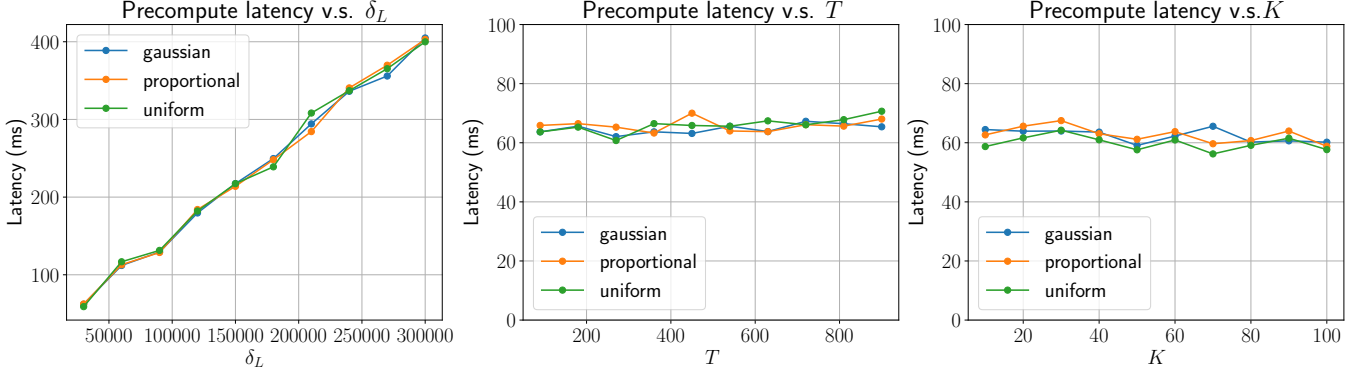


Figure 5: Results of the scalability experiment for verification precomputation

- **Authentication** (cs_{Auth} , pk_{Auth} , vk_{Auth}): We also generate the CS, the proving key, and the verifying key required for authentication. Their sizes are 371 kB for the CS, 2.22 MB for the proving key, and 780 B for the verifying key. Again, all are sufficiently small.
- **Update in the baseline environment** (cs_{Upd} , pk_{Upd} , vk_{Upd}): Next, we measured the sizes of the verification and verification keys required for updates. Because these sizes depend on the size of the sub-revocation list, we estimate the required size under the baseline parameters and then measure it. Specifically, we tested $\tau \in \{15, 30, 60\}$ and $\delta_\lambda \in \{15,000, 30,000, 60,000\}$, and used a revocation list distribution that was fitted to the real service, yielding a total of $9 = 3 \times 3$ sets of keys (proving/verifying).

Summing all proving keys yielded 1.01 GB, confirming that the storage consumption under the above setting is within a practical range. Although this may appear large at first glance, it is not unusual for modern mobile apps to require this amount of storage. The total size of the CSs was 133 MB, which is sufficiently small, and the total size of all verifying keys was 30.3 MB, which is also small.

Update with Massive Parameters (cs_{Upd} , pk_{Upd} , vk_{Upd})

We next examine how the sizes of the CS, proving key, and verifying key for the update process vary. Because these sizes depend on the size of the sub-revocation list, we measured how they grow under different conditions.

Figure 6 shows the variations in the CS, proving key, and verifying key. The proving and verifying key sizes increase linearly as δ_λ and κ grow. This trend arises because the proving key scales with the number of constraints, while the verifying key size grows with the number of public inputs, primarily those for the sub-revocation list. The CS size follows the same overall trend.

E Detailed Implementation: Signer App

As mentioned above, we provide two versions of this application: a smartphone app and a desktop app. Both versions implement authentication and update functions. The update function uses randomized batch timing to prevent timing correlation attacks, as described in Section 10.

Common Challenge & Solution: To prevent verifiers from linking multiple authentication requests made by the same signer, zk-BAN requires signers not to reuse a counter value cnt. This restriction limits each signer to at most K sessions.

For this problem, the implementation of zk-BAN provides an extension that allows signers to authenticate more than K by intentionally reusing a counter value. This extension can be used, for example, when a signer logs in again to an account that they have previously accessed or authenticates to multiple accounts for which linkability is acceptable.

Reusing a counter value, however, makes the corresponding authentication requests linkable as originating from the same signer, thereby reducing privacy. Moreover, because signers select counter values themselves, they may unintentionally reuse a value. This risk can be mitigated by showing the signer which counter values have already been used for the account in question.

E.1 Signer Mobile App

The signer mobile app is an Android app that is responsible for registration, updates, and signing through the Android system library. We used Kotlin for app development and the Gomobile tool to invoke Go-based signing operations from the app. Additionally, for identity verification during the registration phase, we used Firebase’s SMS authentication service and its accompanying library.

Challenge & Solution. A key challenge was determining how the signer mobile app should interact with websites. We considered using a browser extension, but they are not supported in most mobile browsers. We also considered Passkey,

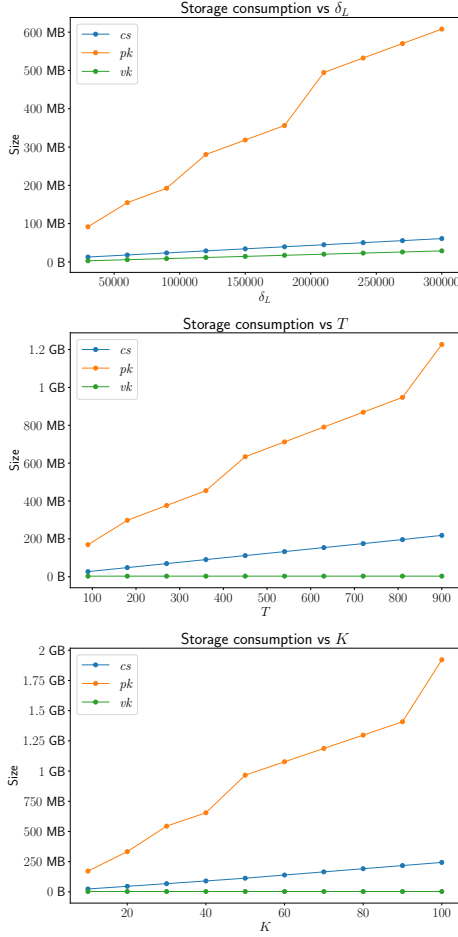


Figure 6: Measurement results for storage sizes

which allows an authenticator (e.g., an Android phone) to sign web requests; however, this approach requires users to select zk-BAN from a list of OS authenticators, which increases user friction as it is complex [16]. We also explored Custom URLs, in which the verifier’s site invokes the signer app to attach and return a signature, similar to prior anonymous credential systems [4]. However, many mobile browsers do not allow external applications to open URLs in incognito mode [28]. Ultimately, we adopted Custom URLs to invoke the signer app from the browser and then let the signer app communicate directly with the verifier. This approach allows the signer app to submit signatures to the verifier even when the user is browsing in private mode.

E.2 Signer Desktop App

The signer desktop app is an application that enables signing on a desktop PC. It is implemented in TypeScript and Electron, with React used for the front end.

This desktop app supports the main user-side processes,

including signing and updates. Its signing process is similar to that of the signer mobile app: it is invoked by the browser through a custom URL. After the user selects a counter, the signature is generated and sent to the verifier. However, the registration and update processes require additional design considerations.

Challenges & Solutions. The first challenge we encountered was how to handle registration in the desktop app, since the GM allows only one registration per user. If the user first registers on the smartphone, registration from the desktop app would be rejected as a duplicate. Allowing two separate credentials for the same user would lead to inconsistent revocation handling.

To address this challenge, zk-BAN performs registration only once on the user’s side using the smartphone. After obtaining the credential and key, the smartphone shares them with the desktop via text copy or a QR code, serving as a single point of revocation across both devices if malicious behavior occurs, thereby maintaining consistency.

Another challenge was deciding on which device to perform updates. Smartphones tend to remain online longer, making them better suited for batch tasks, but they are generally less performant. PCs, by contrast, are typically more performant but may often be offline. In our design, both devices can run the update process, and the device that first obtains the updated credential distributes it to the others.

However, this introduces another challenge: how can the device that receives the updated credential share it securely with the other devices? Text or QR-code transfer is feasible, but it is cumbersome if repeated after every update. A secure channel such as E2EE is possible, but implementing it would add substantial complexity. We therefore adopted a simpler approach in which the GM provides an API through which a device sends its public key upk_i and receives $cred_i$. The key observation is that revealing the credential alone is harmless: valid signatures still require the secret key usk_i , and the credential does not reveal the signer’s identity in zk-BAN.

F Distribution of Δ_{RL} in Latency Measurement

In Section 8, we prepare 3 distributions of Δ_{RL} , such as: (i) constant, (ii) linearly increasing, and (iii) truncated Gaussian. Figure 7 shows the details of the number of nym per t . In this figure, the horizontal axis is the period index t , and the vertical axis shows the number of nym corresponding to that period index t . Importantly, the area of the colored region represents the sub-revocation list size, which is 30,000 for all three cases. Since some periods could end up with zero pseudonyms for some configurations, these periods were set to 1, and the total period count was adjusted accordingly. These changes had a negligible impact on the latency.

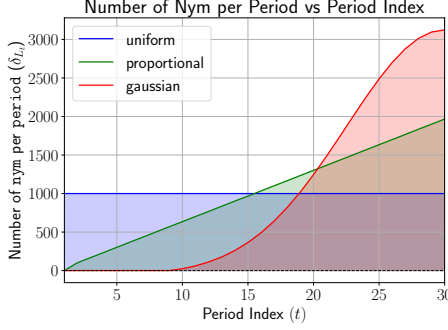


Figure 7: Distribution of Δ_{RL}

G Latency Improvement in Twitch for EU citizens (Improvement 2)

In this section, we show that zk-BAN still achieves a clear reduction in latency even using the Twitch data set, although zk-BAN has a disadvantage in this setting because the number of revocations is small.

For the experiment, we ran zk-BAN with a single circuit. This was possible by setting its per-period capacity to $[3697, 113, 99, 68, 48, 6, 4, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ which accommodates all period’ values in the dataset. For the ablation configuration, we used 2569 and 796 as Δ_L after filtering by Improvement 1 for the one-day and one-year update settings, respectively. For the one-day update, zk-BAN took 229ms, whereas the ablation configuration required 2.99s seconds, yielding a $13.1 \times$ speedup. Although the improvement is relatively modest in this setting because the workload itself is small, this result shows that zk-BAN can still clearly reduce latency on real-world data. For the one-year update, the ablation configuration required 1.15s, whereas zk-BAN achieved an approximately $5.02 \times$ speedup.

H Detailed Performance Evaluation Settings

This section details the settings used for the comparison evaluation in Section 8.5 and Section 8.6.

When measuring authentication in SNARKBlock and ALPACA, we first made sure that synchronization has completed and excluded the small synchronization step included in the default authentication procedure. This separation aligns the measurement with zk-BAN. We also tune the parameters of both prior works to maximize their performance while keeping the experimental setup close to X’s use case.

SNARKBlock additionally requires configuring both the chunk size (i.e., the number of revocation entries per chunk) and the total number of chunks. For the authentication benchmark, we set the chunk size to $32768 = 2^{15}$, matching the update size δ_L in our target use case of X. For the synchronization benchmark, we instead use a chunk size of $1024 = 2^{10}$,

which yields better synchronization performance for SNARKBlock. Although SNARKBlock normally uses a fixed chunk size for authentication and synchronization, we allow different chunk sizes across the two benchmarks to evaluate each phase under its best-performing configuration.

SNARKBlock also requires the number of proofs to be of the form $2^k - 2$ for some $k > 4$; otherwise, it returns the following error: “*HiCIAP must have #proofs = $2^k - 2$ for some $k > 4$* ”. To approximate our baseline as closely as possible under this constraint, we use $254 = 2^8 - 2$ chunks. Accordingly, we set the total number of revocations to $L = 8,323,072 = 32768 \times (256 - 2)$, which is comparable to one year of revocations in X. To align the workloads, zk-BAN uses the same $\delta_L = 32768$, $T = 30$, and $K = 5$, following our baseline setting.

Table 1: Notation & Definitions

Notation	Definition
gsk	GM private key
gpk	GM public key
usk	Signer private key
$upk, upk_{period}, upk'_{period'}$	Signer public key
$ticket, ticket_{period'}$	Signer's ticket used for updates
$cred, cred_{period}, cred'_{period'}$	Signer's credential
r	Random value
msg, msg'	Message / message with random value
σ	Signature
nym, nym'	Pseudonym / revoked pseudonym
$nym'_{t,l}$	The l -th revoked pseudonym in the list for the t -th period
h	PRF value
$period, period, period'$	Period / current period / the last period in which update was run
$period_{Auth}, period_{Revoke}$	Authenticated period / revoked period
cnt	Counter
tag	Tag
$flag, flag_{upk}, flag_{ticket}$	Flag / flag indicating upk / flag indicating ticket
id, id_0, id_1	Identifier / challenge identifiers
$handle$	User handle
$Test$	Test value / test instance
$\pi_{Reg}, \pi_{Auth}, \pi_{Upd}$	Zero-knowledge proofs (corresponding to $\pi_{Reg}, \pi_{Auth}, \pi_{Upd}$ in the appendix)
L	Total number of revocations
δ_L	Sub-revocation list size
δ_{L_t}	Number of revoked signatures in the t -th period of the sub-revocation list
K	Number of sessions
T	Number of time-windows / number of periods in the sub-revocation list
τ	Circuit capacity for the number of periods
δ_λ	Number of signatures that the circuit receives as input
δ_{λ_t}	Number of signatures bound with the t -indexed period that the circuit receives as input
$\Delta_{PL} = \{period_1, \dots, period_T\}$	Set of periods contained in the sub-revocation list
$\Delta_{RL}, \Delta_{RL_t}$	Sub-revocation list / t -th period component of the sub-revocation list
$period_t$	The t -th period in Δ_{PL}
c_τ, c_λ	Latency increment factors used to choose a suitable circuit
TL	List of tickets
RL	Revocation list
$CS = \{cs_{Reg}, cs_{Auth}, cs_{Upd}\}$	zk-SNARK constraint systems (appendix notation: $\{cs_{Reg}, cs_{Auth}, cs_{Upd}\}$)
$PK = \{pk_{Reg}, pk_{Auth}, pk_{Upd}\}$	zk-SNARK proving keys (appendix notation: $\{pk_{Reg}, pk_{Auth}, pk_{Upd}\}$)
$VK = \{vk_{Reg}, vk_{Auth}, vk_{Upd}\}$	zk-SNARK verification keys (appendix notation: $\{vk_{Reg}, vk_{Auth}, vk_{Upd}\}$)
PRF	Pseudorandom function
$Hash$	Hash function
$Sign$	Digital signature generation
$Verify$	Digital signature verification
$Signer, Verifier, GM$	Signer / verifier / group manager
$\perp$	Failure symbol returned when verification fails
IL	Identity list used in the formal definitions
PL	Set/list of periods maintained in the security game
UL_{block}, UL_{unlink}	User lists maintained in the blocklistability / unlinkability games
Δ_{UL}	Subset of users returned by the trace oracle
$period_{Test}, period'_{Test}$	Challenge periods in the security games
usk_{Test}	Challenge signer secret key in the blocklistability game
msg_{Test}, msg'_{Test}	Challenge messages in the security games
$\sigma_{Test}, nym_{Test}, \pi_{AuthTest}$	Challenge signature components in the blocklistability game
$T_{Reg}, T_{Auth}, T_{Upd}, T_{AddRL}, T_{FetchRL}$	Transcript variables in the formal definitions / security games
$R_{Auth}, R_{Upd}, R_{AuthVerifier}$	Request / response variables derived during the games
R_{Test}, R'_{Test}, R^*	Challenge authentication requests in the security games
b, b'	Challenge bit / adversary's guess bit
$\mathcal{A}, C\langle$	Adversary / challenger
λ	Security parameter
$negl$	Negligible function
$unlink, block$	Unlinkability / blocklistability notion