PEREA: Towards Practical TTP-Free Revocation in Anonymous Authentication

Patrick P. Tsang[†], Man Ho Au[‡], Apu Kapadia[†], Sean W. Smith[†]

[†]Department of Computer Science
Dartmouth College
USA

[‡]Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Australia

{patrick, akapadia, sws}@cs.dartmouth.edu, mhaa456@uow.edu.au

ABSTRACT

Several anonymous authentication schemes allow servers to revoke a misbehaving user's ability to make future accesses. Traditionally, these schemes have relied on powerful TTPs capable of deanonymizing (or linking) users' connections. Recent schemes such as *Blacklistable Anonymous Credentials (BLAC)* and *Enhanced Privacy ID (EPID)* support "privacy-enhanced revocation" — servers can revoke misbehaving users without a TTP's involvement, and without learning the revoked users' identities.

In BLAC and EPID, however, the computation required for authentication at the server is *linear in the size (L) of the revocation list*. We propose PEREA, a new anonymous authentication scheme for which this bottleneck computation is *independent of the size of the revocation list*. Instead, the time complexity of authentication is linear in the size ($K \ll L$) of a *revocation window*, the number of subsequent authentications before which a user's misbehavior must be recognized if the user is to be revoked. We prove the security of our construction, and have developed a prototype implementation of PEREA to validate its efficiency experimentally.

Categories and Subject Descriptors

K.6.5 [**Operating Systems**]: Security and Protection—*Authentication*; E.3 [**Data Encryption**]: Public key cryptosystems

General Terms

Algorithms, Security

Keywords

anonymous authentication, privacy-enhanced revocation, subjective blacklisting, non-membership proofs

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

1. INTRODUCTION

Anonymous authentication schemes allow users to authenticate to *service providers (SPs)* as some anonymous member of a group. Fully-anonymous authentication, however, can give users the license to misbehave since they cannot be held culpable for their actions. For example, a website such as Wikipedia may allow anonymous postings, but then cannot hold users who deface webpages accountable. To mitigate this problem, several schemes support revocation of anonymous users, where a *trusted third party (TTP)* can take action against misbehaving users. At a high level, authentication in these schemes requires users to send SPs their pseudonyms encrypted with the TTP's key; SPs can present a misbehaving user's escrowed identity to the TTP as part of a complaint procedure. For example, schemes based on group signatures [1, 6, 15, 22] feature an *Open Authority (OA)*, who uses privileged information in combination with the offending user's authentication transcript to revoke users. Optionally, the OA can provide the SP with a *linking token* to recognize the offending user's connections. Other classes of schemes based on dynamic accumulators [2, 7, 13, 23, 24] and hash chains [21] also rely on TTPs. We omit details on the subtleties of the various schemes, and simply emphasize that all these schemes feature a TTP that can deanonymize users or link[1] their accesses.

Having a TTP with such power, however, is undesirable—users are never guaranteed the anonymity of their connections. Users must trust the TTP to judge their "misbehaviors" fairly and not be susceptible to bribery or coercion by powerful adversaries. This potential for reduced privacy may be unacceptable for users such as whistleblowers, activists, and journalists in countries with restricted freedom of press.

Eliminating TTPs, but at a cost.

Enhanced Privacy ID (EPID) [9], and our own Blacklistable Anonymous Credentials (BLAC) [26] are two recently proposed schemes that for the first time eliminate the reliance on TTPs for revocation, thus providing *privacy-enhanced revocation* [26].[2] SPs can add an entry from an

[1]We say that an entity X can *link* a user's connections if X can infer that the connections belong to a single user with probability better than random guessing.

[2]Brickell and Li refer to this concept as "enhanced revocation" in the context of EPID [9], and we called this concept "anonymous blacklisting " in the context of BLAC. As will become clear, we now distinguish between the *action* of

anonymous user's authentication transcript to a blacklist (or revocation list), following which the revoked user cannot authenticate. No TTP is needed to perform these actions, and revoked users remain anonymous. Privacy-enhanced revocation also allows for *subjective judging* [26], where SPs can revoke users at their discretion since the privacy of users is not at risk. In contrast, TTP-free schemes such as e-cash [4] and k-Times Anonymous Authentication (k-TAA) [25] support accountability in only narrowly defined applications where misbehaviors can be mapped to too many authentications (such as "double spending" a digital coin).

While BLAC and EPID eliminate the reliance on TTPs, the amount of computation at the SP required for authentication is *linear in the size of the blacklist*, i.e., $O(L)$ where L is the number of entries in the blacklist. At a high level, the client has to prove in zero knowledge that each entry in the blacklist was not produced by him/her, resulting in L such proofs. A blacklist with thousands of entries (several entries may correspond to misbehaviors by the same user) would make the costs of authentication prohibitive and pose a severe bottleneck at the SP. For example, for a blacklist with 1,600 entries, BLAC requires 1.68s of computation at the SP and 431.8 KB of communication for a single authentication [26]. These numbers would approximately double for 3,200 entries. In our paper on BLAC, we acknowledged this limitation and listed "more efficient blacklist checking" as an open problem.

PEREA, an efficient alternative.

We propose *PEREA (Privacy-Enhanced Revocation with Efficient Authentication)*, an anonymous authentication scheme without TTPs in which the time complexity of authentication at the SP (the bottleneck operation) is *independent of the size of the blacklist*. Instead, the amount of computation is linear in the size K of the *revocation window*, the number of authentications before which a misbehavior must be recognized and blacklisted for a user to be revoked. For example, if $K = 10$, the SP must blacklist a user's misbehavior before that user has made 10 subsequent authentications. Note that a blacklisted user is not revoked if he or she has already made K subsequent authentications, and therefore we differentiate between the action of blacklisting and the end result of whether the user is actually revoked. Since the SP may take some time to recognize misbehaviors (e.g., malicious edits on Wikipedia may not be detected immediately), these K authentications can be rate limited to K authentications every T minutes. Combined with rate limiting, SPs have enough time (T) to recognize misbehaviors, and honest users can authenticate anonymously at an acceptable rate (one authentication every $\frac{T}{K}$ minutes on average). For example, for $K = 10, T = 60$, SPs must judge misbehaviors within 60 minutes, and users can authenticate once every 6 minutes on average.

In practice we expect $K \ll L$, leading to much better performance that is independent of the number of blacklist entries. For example, L can grow to thousands of entries in an application such as Wikipedia, while K can be limited to a constant as small as 10 or 20 in the context of PEREA. We note that the amount of computation at the user is increased (as compared to BLAC and EPID), but we show that this trade-off is well worth the benefit (Section 5).

blacklisting and the *end result* of revocation.

On rate limiting.

We argue that *rate limiting is an implicit scalability requirement in both BLAC and EPID.* If a user performs 10,000 misbehaviors before being detected in BLAC or EPID, the number of entries L in the blacklist grows by 10,000 for that single misbehaving user, resulting in unacceptable overhead ($O(L)$). Furthermore, blacklisting is less effective at deterring misbehavior if users can do all the desired damage before being detected and blacklisted. In our paper on BLAC [26], we suggested that the rate of authentication should be limited for this reason.

Our contributions.

- We present PEREA, an anonymous authentication scheme without TTPs that supports privacy-enhanced revocation. PEREA is the first such scheme with computation at the service provider *independent of the size of the blacklist.*
- We introduce the concept of a *revocation window*, the number of authentications within which a misbehaving user must be blacklisted if he or she is to be revoked. These semantics allow for a more efficient solution than with existing approaches.
- We evaluate the performance of PEREA both analytically and experimentally, and prove the security of our construction without random oracles. We have built a prototype implementation, and compare its performance with BLAC. We demonstrate that PEREA outperforms BLAC in server computation.[3]

Paper outline.

We present an overview of our solution and its security requirements in Section 2. After presenting the individual cryptographic building blocks in Section 3, we present our construction in Section 4. We present a detailed evaluation, both analytically and experimentally, in Section 5. We present a discussion of several issues related to PEREA in Section 6, and conclude in Section 7.

2. OVERVIEW

Since our solution is based on accumulators, we start by describing the limitations of existing accumulator-based schemes, followed by a high-level overview of our scheme. We then describe the security goals of our solution.

2.1 Accumulators

A *dynamic accumulator*, or simply an *accumulator* as we call it in this paper, is a constant-size cryptographic construct that represents set membership. Elements may be added to (i.e., "accumulated"), or removed from, the accumulator. Furthermore, anyone can prove in zero knowledge that certain element is "in" the accumulator *if and only if* the element has indeed been accumulated. In the context of anonymous authentication, users can authenticate by proving in zero knowledge that their pseudonym is in the accumulator, where the accumulator represents a "whitelist" of valid pseudonyms [13]. Using a recent scheme [23], users could instead prove their non-membership in an accumulator representing a "blacklist" of revoked pseudonyms. In both

[3]We did not compare PEREA experimentally with EPID, which has the same asymptotic performance as BLAC.

these approaches, authentication is performed in constant time at the server.

Why current approaches fail. The limitation of current accumulator-based approaches is that *someone* must add or remove pseudonyms from the accumulator. In existing schemes, the user must provide the SP with his/her pseudonym encrypted with a TTP's key. The SP can revoke that user's access by providing the user's encrypted pseudonym to the TTP, which then decrypts and adds or removes (depending on whether the accumulator is a blacklist or whitelist) the pseudonym from the accumulator. Optionally, the TTP can announce the pseudonym to the SP, who then updates the accumulator itself. As mentioned earlier, we seek to eliminate TTPs that can identify (or link) users, and therefore current accumulator-based solutions do not suffice. We need a solution in which pseudonyms are not added or removed from the accumulators.

Our use of accumulators. PEREA uses accumulators as a blacklist in a novel way. Instead of presenting an encrypted pseudonym during authentication, the user presents a *ticket*, which is an unlinkable "one-show" token generated by the user. These tickets give users the desired unlinkability across authentications. Simply putting a ticket into the accumulator, however, does not revoke users because users can produce any number of new tickets (unlike systems in which users have unique pseudonyms). PEREA solves this problem by making the user also prove in zero knowledge that the last K tickets he/she presented are not in the accumulator. What follows is a high-level description of how PEREA implements this functionality.

2.2 Overview of our construction

In PEREA, *users* register with a *service provider (SP)* and obtain a *credential* for anonymous authentication at that SP. The credential consists of various initialized elements and datastructures. Figure 1 is a pictorial representation of how these datastructures are used in authentication as described in this section. All elements within the dashed box belong to the user and are proved in zero knowledge to the SP. Arrows represent the flow of information during the current authentication, and dashed arrows represent the flow of information in preparation for the next authentication.

Authentication. As part of the authentication process, users provide the SP with a *ticket* t_K. Users maintain a *queue* of the past K authentication *tickets* $t_0, \ldots, t_{K-1}$ in addition to the current ticket t_K, and the SP maintains a blacklist of tickets belonging to misbehaving users. The user proves to the SP that the current ticket t_K is valid, and that none of the previous tickets in the queue are on the SP's blacklist. If authentication succeeds, the user generates a new blind ticket t_{K+1} for the next authentication, and constructs a new queue by dropping t_0 and appending t_{K+1}. Once a user's ticket has been added to the blacklist within the revocation window, the user can no longer authenticate and acquire new tickets for subsequent authentications.

Proving that the user is not revoked. The SP stores its blacklist in the form of an *accumulator* V (as will be explained later, the blacklist is also stored and communicated as a list). Since the SP should not be able to link any of the user's previous transactions, the user must prove that his/her queue of previous tickets $t_0, \ldots, t_{K-1}$ has not been revoked *without disclosing those tickets.* This proof is done in zero knowledge, in which the user generates for each ticket

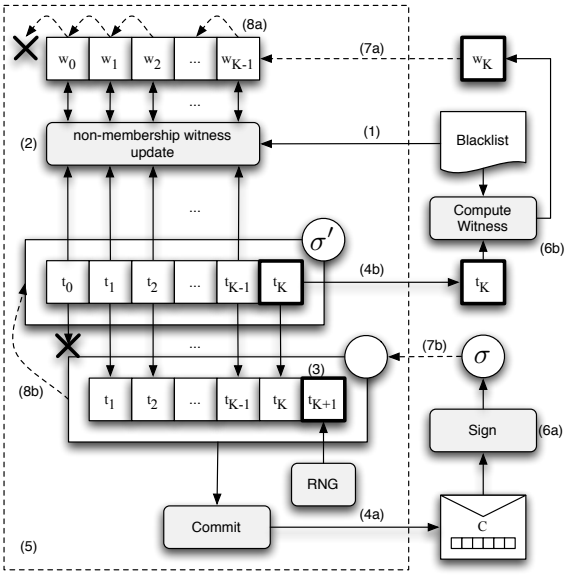


Figure 1: (1) User Alice obtains the SP's Blacklist and (2) updates her list of witnesses $(w_0, \ldots, w_{K-1})$ for the tickets $(t_0, \ldots, t_{K-1})$ in her queue. (3) Alice generates ticket t_{K+1} for use in the next authentication and constructs the new queue $(t_1, \ldots, t_{K+1})$. (4) Alice generates a commitment C of the new queue, and sends C and t_K to the SP. (5) Alice proves the integrity of her datastructures in zero knowledge, and that C and t_K are well formed. If the proof succeeds, (6) the SP generates a new signature σ for C, and computes a witness w_K for ticket t_K, and (7) sends these items to the user, who (8) refreshes her list of witnesses and stores the new queue for subsequent authentication.

t_i a *witness* w_i, which attests that ticket t_i has not been "accumulated" in the accumulator (representing the blacklist). While the user must download the blacklist $(O(L))$[4] to generate these witnesses, the accumulator-based proof sent to the SP is of size $O(K)$ and verification takes time $O(K)$.

Proving the integrity of the queue. The SP needs the user to prove that none of the user's *most recent* K tickets has been revoked. A user could circumvent revocation by fabricating a queue with an incorrect set of K tickets. The SP therefore needs to verify the integrity of the queue, i.e., the queue contains the correct sequence of the most recently used K tickets. Again, this proof must be performed in zero knowledge because disclosing a queue (of tickets) to the SP immediately links the user's previous actions. To prove the integrity of the queue, the user makes use of a signature obtained from the SP in the previous authentication. Likewise, the user must now obtain a new signature for use in the next authentication. The user provides the SP with a *commitment* (a blinded queue) of the new queue, along with the current authentication ticket t_K. If authentication succeeds, the server generates a signature of this commitment and sends it back to the user. During the user's subsequent authentication, the user can prove the integrity of the new

[4]As we note in Section 5, blacklists in PEREA are approximately 14% the size of blacklists in BLAC.

queue in zero knowledge by using the new signature as part of the zero knowledge protocol.

Now that we have described the various aspects of our construction, we refer the reader to Figure 1 for the actual sequence of actions during authentication.

2.3 Security Goals

We now describe the security properties of PEREA. Here we give informal descriptions and refer the interested reader to Appendix A for a formal definition of these properties.

PEREA must have the basic property of *misauthentication resistance*, i.e., no unregistered user should be able to authenticate. PEREA must also support *revocability*, i.e., users blacklisted within the revocation window should not be able to authenticate successfully. Furthermore, any coalition of revoked and/or unregistered users should not be able to authenticate successfully.

The *anonymity* property requires that SPs should not be able to identify authenticating users within the *anonymity set* of registered users and their authenticated connections should be *unlinkable*. The SP should be able to infer only whether the authenticating user is revoked or not. We also require *identity-escrow freeness*, i.e., there should exist no TTP that can infer the identity or pseudonym of a user behind an authentication.

Backward unlinkability [3] requires that upon revocation, all the user's past authentications should remain anonymous and unlinkable. *Revocation auditability* requires that users should be able to check their revocation status before performing any actions at the SP. This property avoids the situation when a malicious SP recognizes a user as being revoked without the user being aware of his or her reduced privacy.

3. BUILDING BLOCKS

We now outline the various cryptographic primitives that we use to realize PEREA.

3.1 Preliminaries

Notation and intractability assumptions.

If S is a set, then $|S|$ denotes its cardinality and $a \in_R S$ means that a is an element picked from S uniformly at random. If ℓ, δ are integers, we denote by $[\ell, \delta]$ the set $\{\ell, \ell+1, \ldots, \delta\}$, by Λ_ℓ the set $[0, 2^{\ell+1}-1]$, i.e., the set of integers of size at most ℓ bits, by Π_ℓ the set $\{e \in \Lambda_\ell | e \text{ is prime}\}$, and by $\Delta(\ell, \delta)$ the set $[2^{\ell-1}, 2^{\ell-1}+2^\delta-1]$. A *safe prime* is a prime p such that $\frac{p-1}{2}$ is also prime. An ℓ-bit *safe-prime product* is a product of two $\lfloor \frac{\ell}{2} \rfloor$-bit safe primes. If N is an integer, then QR_N is the set of quadratic residues modulo N and $\phi(N)$ is the Euler's totient of N. If $A(\cdot)$ is a (possibly probabilistic) algorithm, then we write $a \leftarrow A(\cdot)$ or $A(\cdot) \rightarrow a$ to mean that a is the output of an execution of $A(\cdot)$. Finally, $a \doteq b$ defines a to be b.

The security of PEREA relies on the *Strong RSA Assumption* [5, 18] and the *Decisional Diffie-Hellman (DDH) Assumption* over the quadratic residues modulo a safe-prime product. Let N be a random λ-bit safe prime product. The Strong RSA Assumption says that there exists no PPT algorithm which, on input N and $u \in \mathbb{Z}_N^*$, returns $e > 1$ and v such that $v^e = u \bmod N$, with non-negligible probability (in λ). The DDH Assumption over QR_N says that there exists no PPT algorithm which, on input of a quadruple $(g, g^a, g^b, g^c) \in QR_N^4$, where $a, b \in_R \mathbb{Z}_{|QR_N|}$, and $c \in_R \mathbb{Z}_{|QR_N|}$ or $c = ab$ with equal probability, correctly distinguishes which is the case with probability non-negligibly (in λ) greater than $1/2$.

ZKPoK protocols.

In a Zero-Knowledge Proof-of-Knowledge (ZKPoK) protocol [20], a *prover* convinces a *verifier* that some statement is true without the verifier learning anything except the truth of the statement. In many existing anonymous credential systems, a client uses some variants of ZKPoK protocols to prove to a server her possession of a credential during an authentication without revealing the credential. PEREA makes use of ZKPoK protocols.

We follow the notation introduced by Camenisch and Stadler [14]. For example, $PK\{(x) : y = g^x\}$ denotes a ZKPoK protocol that proves the knowledge of an integer x such that $y = g^x$ holds, where g generates a group in which discrete logarithms are hard to compute.

3.2 Tickets and queues

In PEREA, a user picks a *ticket* uniformly at random from the set Π_{ℓ_t}, where $\ell_t = 166$. As there are at least 2^{160} tickets in this set,[5] two tickets picked uniformly at random collide with probability at most 2^{-80}, because of the birthday paradox. We set the ticket domain to be $\mathcal{T} \doteq [-2^{\ell_T}+1, 2^{\ell_T}-1]$, where $\ell_T = 330$.[6]

A *queue* of size k is a sequence of k tickets. The domain of all k-sized queues is thus $\mathcal{Q}_k \doteq \mathcal{T}^k$. A queue supports the enqueuing (`Enq`) and dequeuing (`Deq`) operations in the usual sense. We denote by $Q[i]$ the i-th least recently enqueued ticket in Q, with $i = 0$ being the least recent (oldest). In PEREA, all queues are of size $(K+1)$, where K is the revocation window as explained before. We therefore sometimes abbreviate their domain $\mathcal{Q}_{K+1}$ as simply $\mathcal{Q}$.

3.3 Proving that a user is not revoked

3.3.1 An accumulator scheme for tickets

PEREA makes use of *universal dynamic accumulators (UDAs)* recently introduced by Li et al. [23]. Compared to conventional *dynamic accumulators (DAs)*, UDAs additionally allow for an efficient zero-knowledge proof of *non-membership*. Specifically, for any input x (within some domain) that has not been accumulated into an accumulator value V, anyone can prove to anyone else that this is indeed the case, without revealing x, in time independent of the number of inputs already accumulated into V. In PEREA, the SPs blacklist users by accumulating their tickets into UDAs.

The following describes a construction of UDAs we adapted from the one due to Li et al. The differences are mostly at the presentation level; we make notational changes and retain only the functionality needed by PEREA. We call the modified scheme `TicketAcc`.

Key generation On input security parameter $\mathtt{param}_{acc} = \ell_\mathsf{N}$, choose an ℓ_N-bit safe-prime product $\mathsf{N} = \mathsf{pq}$ uni-

[5] This follows from a result due to Dusart [17]: if $\pi(x)$ is the number of distinct primes less than x, then $\pi(x) > \frac{x}{\ln x}\left(1 + \frac{0.992}{\ln x}\right)$ for all $x > 598$.

[6] Note that $\mathcal{T} \supsetneq \Pi_{\ell_t}$. This allows a user in PEREA who knows a ticket in Π_{ℓ_t} to efficiently prove in zero knowledge to the SP that she knows some ticket in $\mathcal{T}$.

formly at random, pick $\mathsf{g} \in_R QR_{\mathsf{N}}$, and output the accumulator private key $\mathtt{sk}_{acc} = \phi(\mathsf{N})$ and public key $\mathtt{pk}_{acc} = (\ell_{\mathsf{N}}, \mathsf{N}, \mathsf{g})$. $\mathtt{pk}_{acc}$ is an implicit input to all the algorithms below.

We have chosen $\ell_{\mathsf{N}} = 1024$ as required by the security of the scheme and so that tickets can be accumulated.[7]

Accumulating tickets Accumulating ticket $t \in \mathcal{T}$ to an accumulator value V can be computed as:

$$\mathtt{Accumulate}(\mathsf{V}, t) \to \mathsf{V}' \doteq \mathsf{V}^t \bmod \mathsf{N}. \tag{1}$$

Let $S_T = \{t_1, t_2, \ldots, t_L\} \subset \mathcal{T}$. We overload

$$\mathtt{Accumulate}(\mathsf{V}, \mathcal{S}_T) \tag{2}$$

to mean the repetitive invocation of `Accumulate` to accumulate tickets $t_1, t_2, \ldots, t_L$, one at a time. (The order does not matter, due to quasi-commutativity [13].) An accumulator value is initially g. The accumulator value V resulted from accumulating S_T is thus:

$$\mathtt{Accumulate}(\mathsf{g}, S_T) \to \mathsf{V} \doteq \mathsf{g}^{t_1 t_2 \cdots t_L} \bmod \mathsf{N}. \tag{3}$$

We abbreviate the above as $\mathtt{Accumulate}(S_T)$.

Non-membership witnesses If $\mathsf{V} = \mathtt{Accumulate}(S_T)$ for some $S_T \subset \mathcal{T}$ and $t \in \mathcal{T} \backslash S_T$, then there exists a non-membership witness w in the form $(a, \mathsf{d}) \in \mathbb{Z}_{\lfloor \frac{\mathsf{N}}{4} \rfloor} \times QR_{\mathsf{N}}$ for t w.r.t. V such that $1 = \mathtt{IsNonMember}(t, \mathsf{V}, w)$, where

$$\mathtt{IsNonMember}(t, \mathsf{V}, (a, \mathsf{d})) \doteq \begin{cases} 1, & \text{if } \mathsf{V}^a \equiv \mathsf{d}^t \mathsf{g}, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

As we will see soon (in Section 3.3.2), the witness w for a ticket t w.r.t. an accumulated value V allows a prover to convince a verifier that t was not accumulated in V, without revealing t or w.

We sometimes simply call non-membership witnesses "witnesses."

Computation of non-membership witnesses If $\mathsf{V} = \mathtt{Accumulate}(S_T)$ for some $S_T \subset \mathcal{T}$, then for any $t \in \mathcal{T} \backslash S_T$, one can compute, using knowledge of $\mathtt{sk}_{acc}$, a witness $w = (a, \mathsf{d})$ for t w.r.t. V:

$$\mathtt{ComputeWitness}(t, \mathsf{V}, \mathtt{sk}_{acc}) \to w \tag{5}$$

so that $1 = \mathtt{IsNonMember}(t, \mathsf{V}, w)$. Please refer to [23, §3.2] for how to find such a w.

Update of non-membership Witnesses Given witness w such that $1 = \mathtt{IsNonMember}(t, \mathsf{V}, w)$, when V gets updated to V' via the accumulation of a new ticket $t' \in \mathcal{T} \backslash \{t\}$ into it (i.e., $\mathsf{V}' = \mathtt{Accumulate}(\mathsf{V}, t')$), anyone can compute, *without the knowledge of* $\mathtt{sk}_{acc}$, a witness w' for the same ticket t w.r.t. the updated accumulated value V' (i.e., $1 = \mathtt{IsNonMember}(t, \mathsf{V}', w')$) as:

$$\mathtt{UpdateWitness}(w, t, \mathsf{V}, t') \to w' \tag{6}$$

Again, please refer to [23, §4.2] for details.

If $t \in \mathcal{T} \backslash S_T$ for some $S_T \subset \mathcal{T}$, we overload

$$\mathtt{UpdateWitness}(w, t, \mathsf{V}, S_T) \tag{7}$$

to denote the repetitive invocation of `UpdateWitness` to update w for t when tickets in S_T are accumulated into V, one at a time, in any order.

The complexity of the operations in Equations 1, 4, 5 and 6 is $O(1)$, i.e., independent of the number of tickets that have been accumulated into V. The complexity of those in Equations 2, 3 and 7 is thus $O(|S_T|)$.

3.3.2 Proof that a ticket is not accumulated

To prove in zero knowledge that the i-th least recently enqueued ticket $Q[i]$ in queue Q is not accumulated in an accumulator value V, one can conduct

$$PK\ \{(Q[i], w) : 1 = \mathtt{IsNonMember}(Q[i], \mathsf{V}, w)\} \tag{8}$$

using the knowledge of the corresponding witness w. The construction of the above protocol and its security proof have been given by Li et al. [23, §5]. The construction has a complexity of $O(1)$, i.e., independent of the number of inputs that have been accumulated into V.

3.4 Proving the integrity of the queue

3.4.1 A protocol for queue signing

In PEREA, the SP signs user Alice's queue during an authentication so that next time when Alice tries to authenticate, the server can be convinced of the queue's integrity.

PEREA must use a signature scheme in which Alice can request the SP for a signature on the queue and also later prove to the SP her possession of a valid signature on a queue, without revealing the queue and the signature. Hence, a conventional digital-signature scheme would not work.

For this purpose, we construct `QueueSig`, which is an adaptation from the *signature scheme for blocks of messages* [12, §4] and the *protocol for signing blocks of committed values* [12, §6.3], both due to Camenisch and Lysyanskaya. Our adaptation is again at the presentation level: we think of blocks of messages as queues of tickets, and present, with notational changes, only those parts that are relevant to PEREA.

As will become clear, `QueueSig` provides the skeleton for both the *Registration* protocol and the *Authentication* protocol in PEREA. We now describe `QueueSig`.

Key generation On input security parameters $\mathtt{param}_{sig} = (\ell_N, \ell_s, \ell_e, \ell_T, l, \delta_r)$, the SP chooses an ℓ_N-bit safe-prime product $N = pq$ uniformly at random, and $b, c, g_0, g_1, \ldots, g_K \in_R QR_N$, and then outputs the signature private key $\mathtt{sk}_{sig} = \phi(N)$ and public key $\mathtt{pk}_{sig} = (\mathtt{param}_{sig}, N, b, c, (g_i)_{i=0}^K)$. The SP keeps $\mathtt{sk}_{sig}$ private and publishes $\mathtt{pk}_{sig}$ to the public. $\mathtt{pk}_{sig}$ is an implicit input to the algorithms defined below.

Request for signature To request a signature on a committed queue $Q = (t_i)_{i=0}^K \in \mathcal{Q}$, Alice picks $r \in_R \Delta(\ell_N, \delta_r)$, commits Q:

$$\mathtt{Commit}(Q, r) \to C \doteq c^r \prod_{i=0}^{K} g_i^{t_i} \bmod N, \tag{9}$$

and then sends the commitment C to the SP.

Proof of correctness Alice (as the prover) then conducts the following protocol with the SP (as the verifier) to

[7] The accumulator allows any input in the domain $\mathcal{X} \doteq \{x \in \mathcal{X}' | x \text{ is prime}\}$ to be accumulated, where $\mathcal{X}' = [0, 2^{\ell_x})$ with $\ell_x = \lfloor \ell_{\mathsf{N}}/2 \rfloor - 2$. Since $\ell_t = 166$, the choice of $\ell_{\mathsf{N}} = 1024$ gives $\Pi_{\ell_t} \subset \mathcal{X}$.

prove that the commitment was constructed correctly:

$$PK\left\{(Q,r) : C = \texttt{Commit}(Q,r) \ \wedge \ Q \in \mathcal{Q} \ \wedge \ r \in \mathcal{R}\right\}, \tag{10}$$

where $\mathcal{R} \doteq [0, 2^{\ell_N})$. The SP proceeds only if the protocol succeeds.

Signing The SP signs and returns to Alice a signature $\tilde{\sigma}$ on C using its private key $\texttt{sk}_{sig}$:

$$\texttt{Sign}(C, \texttt{sk}_{sig}) \rightarrow \tilde{\sigma} \doteq (r', e, v), \tag{11}$$

where $r' \in_R \Lambda_{\ell_s}$, $e \in_R \Pi_{\ell_e}$ and $v = \left(bc^{r'}C\right)^{1/e \bmod \phi(N)} \bmod N$.[8]

Finalizing Alice finalizes the signature $\tilde{\sigma} = (r', e, v)$ on the commitment C into a signature σ on her queue Q:

$$\texttt{Finalize}(\tilde{\sigma}, r) \rightarrow \sigma \doteq (r + r', e, v). \tag{12}$$

She proceeds only if the signature verifies, i.e., $\texttt{Verify}(Q, \sigma) = 1$, where

$$\texttt{Verify}(Q,\sigma) = \begin{cases} 1, & \text{if } v^e \equiv bc^s \prod_{i=0}^{K} g_i^{t_i} \ \wedge \ e > 2^{\ell_e - 1}, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

This construction of the protocol has an $O(K)$ computational complexity at — and an $O(K)$ communication complexity between — Alice and the SP. The security of the protocol requires that (1) the signatures are unforgeable and (2) the SP learns nothing (e.g., its content, and who owns it) about the queue that it is signing. When $\ell_N \geq 1024$, $l \geq 160$, $\ell_e > \ell_T + 2$, $\ell_s = \ell_N + \ell_T + l$ and $\delta_r = \lfloor \frac{\ell_N - 1}{\epsilon} - l \rfloor$ for some $1 < \epsilon \in \mathbb{R}$, these properties can be proved to hold under the Strong RSA assumption in virtually the same way as Camenisch and Lysyanskaya proved theirs [12]. We have chosen $(\ell_N, \ell_s, \ell_e, \ell_T, l, \delta_r) \doteq (1024, 1514, 333, 330, 160, 862)$.

3.4.2 Proof of knowledge of a signed queue

As alluded to earlier, Alice must prove to the SP the possession of a valid signature issued by the SP for her queue, without revealing the queue and the signature themselves. The following protocol does exactly that:

$$PK\left\{(Q,\sigma) : 1 = \texttt{Verify}(Q,\sigma) \ \wedge \ Q \in \mathcal{Q}\right\} \tag{14}$$

A construction for the above protocol and its security proof closely follow the one for the *ZKPoK protocol for proving the knowledge of a signature on blocks of committed values* [12, §6.3] and its security proof, respectively. We thus omit the details. The construction has an $O(K)$ computational complexity at — and an $O(K)$ communication complexity between — Alice and the SP.

3.4.3 Proof of relation between two queues

During a PEREA authentication, Alice updates her current queue from Q' to $Q = Q'.\texttt{Enq}(t^*).\texttt{Deq}()$ for her use during the next authentication, where t^* is a new random ticket. Alice must obtain the SP's signature on this new queue to convince the SP of its integrity during her next authentication. On the other hand, the SP should only sign Q if it is indeed correctly updated from Q'. The following protocol allows Alice to convince the SP that this is indeed the case, without revealing the contents of either queue:

$$PK\left\{(Q', Q, t) : \begin{array}{c} Q = Q'.\texttt{Enq}(t^*).\texttt{Deq}() \ \wedge \\ Q' \in \mathcal{Q} \ \wedge \ t^* \in \mathcal{T} \end{array}\right\} \tag{15}$$

This protocol can be constructed as follows. Alice first picks $r_0, r_1 \in_R \Delta(\ell_N, \delta_r)$ and commits both Q' and Q according to Eq. 9 and conducts the following protocol with the SP (ranges omitted):

$$PK\left\{(r_0, r_1, (t_i)_{i=0}^{K+1}) : \bigwedge_{b=0,1} C_b \equiv c^{r_b} \prod_{i=0}^{K} g_i^{t_{i+b}}\right\}, \tag{16}$$

which can in turn be constructed using standard protocols for proving relations among components of a discrete-logarithm representation of a group of elements [10]. The construction has an $O(K)$ computational complexity at — and an $O(K)$ communication complexity between — Alice and the SP.

4. CONSTRUCTION

We now provide a concise description of our construction, making use of the building blocks presented in Section 3.

4.1 Server setup

The SP first decides on the size of the revocation window K based on system requirements. As discussed in Section 1, K will depend on the time it takes to identify misbehaviors, and the expected rate of authentication among users. We expect K to be small, e.g., $K = 10$.

On input parameters $\texttt{param}_{acc}$ and $\texttt{param}_{sig}$ as defined in Section 3, the SP then generates a key pair $(\texttt{sk}_{acc}, \texttt{pk}_{acc})$ for `TicketAcc` and a key pair $(\texttt{sk}_{sig}, \texttt{pk}_{sig})$ for `QueueSig` according to Section 3.3.1 and 3.4.1, respectively. The SP also picks a prime $\hat{t} \in \Pi_{\ell_t}$, which is used to fill a user's queue as the default value during registration. The SP creates a server private key $\texttt{serversk} = (\texttt{sk}_{sig}, \texttt{sk}_{acc})$, and then creates and publishes a server public key $\texttt{serverpk} = (K, \texttt{pk}_{sig}, \texttt{pk}_{acc}, \hat{t})$.

Initially the SP's blacklist `BL` is empty, i.e., $\texttt{BL} = \emptyset$. The corresponding accumulator value is thus $\mathsf{V} = \mathsf{g}$. Additionally, the SP maintains a ticket-list `TL` to record tickets that it has seen for checking the freshness of tickets.

4.2 Registration

User Alice registers with the SP to obtain a credential for PEREA authentication. Alice must be authenticated by the SP to register.[9] The registration protocol goes as follows.

1. *(Request for credential.)* Alice picks $t^* \in_R \Pi_{\ell_t}$ and initializes her queue Q' as

$$Q' = (\hat{t}, \hat{t}, \ldots, \hat{t}, \hat{t}, t^*) \in \mathcal{Q}. \tag{17}$$

   Next, she picks $r \in_R \Delta(\ell_N, \delta_r)$ and commits Q' as $C = \texttt{Commit}(Q', r)$. She then sends commitment C to the SP.

2. *(Proof of correctness.)* Alice (as the prover) conducts the following protocol with the SP (as the verifier).

$$PK\left\{\begin{array}{lcll} (Q', r) : & & & \\ \bigwedge_{i=0}^{K-1} \hat{t} & = & Q'[i] & \wedge \\ C & = & \texttt{Commit}(Q', r) & \wedge \\ Q' \in \mathcal{Q} & \wedge & r \in \mathcal{R} & \end{array}\right\} \tag{18}$$

[8] To allow for more efficient ZKPoK of a signature, the signer should pick e from a slightly smaller range instead. We refer the reader to [8, 12] for the details.

[9] How this authentication happens is application-dependent. The SP may ask Alice to, e.g., present her driver's license in person, or register via a client-authenticated TLS session.

This protocol and hence its construction are similar to the one in Eq. 10, except that Alice has to prove additionally that the K least recent tickets in the queue correspond to the default ticket $\hat{t}$. The SP proceeds only if this protocol terminates successfully.

3. *(Credential issuing.)* The SP signs a signature $\tilde{\sigma}$ on C and computes a non-membership witness $\hat{w}$ for $\hat{t}$ w.r.t. its current blacklist $\mathtt{BL}'$, i.e., it executes $\tilde{\sigma} \leftarrow \mathtt{Sign}(C, \mathtt{sk}_{sig})$ and $\hat{w} \leftarrow \mathtt{ComputeWitness}(\hat{t}, \mathsf{V}', \mathtt{sk}_{acc})$, where $\mathsf{V}' = \mathtt{Accumulate}(\mathtt{BL}')$. The SP returns $(\sigma', \hat{w}, \mathtt{BL}', \mathsf{V}')$ to Alice.
4. *(Credential finalizing.)* Alice computes $\sigma' \leftarrow \mathtt{Finalize}(\sigma', r)$ and proceeds only if $\mathsf{V}' = \mathtt{Accumulate}(\mathtt{BL}')$, $1 = \mathtt{Verify}(Q', \sigma')$ and $1 = \mathtt{IsNonMember}(\hat{t}, \mathsf{V}', \hat{w})$. She stores her credential $\mathtt{cred}$ as:

$$\mathtt{cred} \leftarrow (Q', \sigma', (w_i')_{i=0}^{K-1}, \mathtt{BL}', \mathsf{V}'),$$

where $w_i = \hat{w}$ for all $i = 0$ to $K-1$.

4.3 Authentication

We now describe the authentication protocol executed between user Alice and the SP. Alice has previously registered with the SP and has hence obtained a credential, although she may or may not have PEREA-authenticated to the SP before. The protocol is executed over an SP-authenticated channel, which can be established using, e.g., SSL/TLS based on the SP's X.509 certificate.

Blacklist examination.

Alice first obtains from the SP the current version of its blacklist $\mathtt{BL}$. This is the version of the SP's blacklist from which the SP wants to be convinced that the connecting user is absent, despite the fact that the blacklist might get updated in the course of authentication. Alice then checks if she is revoked, i.e., if one or more tickets in her ticket-queue appear in $\mathtt{BL}$. She proceeds if she is *not* revoked. She drops the connection otherwise.

Denote by $\Delta_{\mathtt{BL}} \doteq \mathtt{BL} \backslash \mathtt{BL}'$, where $\mathtt{BL}'$ is the SP's blacklist she last saw and saved in her credential. $\Delta_{\mathtt{BL}}$ is thus the set of newly blacklisted tickets.[10]

Request for authentication.

Now that Alice knows that she has not been revoked, she requests to authenticate. Alice picks $t^* \in_R \Pi_{\ell_t}$ and $r \in_R \Delta(\ell_N, \delta_r)$, and then computes

$$\begin{array}{rcl} t_K & \leftarrow & Q'[K] \\ Q & \leftarrow & Q'.\mathtt{Enq}(t^*).\mathtt{Deq}() \\ C & \leftarrow & \mathtt{Commit}(Q, r) \\ \mathsf{V} & \leftarrow & \mathtt{Accumulate}(\mathsf{V}', \Delta_{\mathtt{BL}}) \end{array}$$

and, for $i \in [0, K)$,

$$w_i \leftarrow \mathtt{WitnessUpdate}(w_i', Q'[i], \mathsf{V}', \Delta_{\mathtt{BL}}). \tag{19}$$

She sends (t_K, C) to the SP. The SP proceeds only if t_K is fresh, i.e., $t_K \notin \mathtt{TL}$, and is a prime in Π_{ℓ_t}. The SP then adds t_k to $\mathtt{TL}$.

[10]We assume for now that SPs only add entries to their blacklists, so that $\mathtt{BL}' \subseteq \mathtt{BL}$ always. We address "unblacklisting" and blacklist manipulation in Section 6.

Proof of correctness.

Alice (as the prover) conducts the following ZKPoK protocol with the SP (as the verifier):

$$PK\Big\{ \begin{array}{l} (Q', \sigma', (w_i)_{i=0}^{K-1}, t^*, Q, r) : \\ \quad \begin{array}{rcll} t_K & = & Q'[K] & \wedge \\ 1 & = & \mathtt{Verify}(Q', \sigma') & \wedge \\ \bigwedge_{i=0}^{K-1} \; 1 & = & \mathtt{IsNonMember}(Q'[i], \mathsf{V}, w_i) & \wedge \\ Q & = & Q'.\mathtt{Enq}(t^*).\mathtt{Deq}() & \wedge \\ C & = & \mathtt{Commit}(Q', r) & \wedge \end{array} \\ \quad Q' \in \mathcal{Q} \;\wedge\; \tilde{t} \in \mathcal{T} \;\wedge r \in \mathcal{R} \end{array} \Big\} \tag{20}$$

The SP proceeds only if the ZKPoK verifies.

As explained earlier, the above protocol aims to convince the SP that (1) the connecting user's past K connections have not been blacklisted, and (2) t_K is a well-formed ticket that the SP can later use to blacklist the user, and (3) C is a well-formed commitment of the user's next queue, a signature on which allows the user to authenticate in her next connection.

We have described in Section 3 how to construct protocols for proving individual statements that appear in the above protocol. Constructing the above protocol is thus fairly straightforward: we put together all the individual proofs, and make sure that the common secrets in them are indeed the same, by using a suitable commitment scheme such as the one we used to commit a queue, and standard techniques for proving relations among components of discrete-logarithm representations of group elements [10].

Refreshment issuing.

The SP helps Alice refresh her credential as follows. The SP first signs a signature $\tilde{\sigma}$ on the commitment C, and computes a non-membership witness w^* for t^* w.r.t. V, i.e., it executes $\tilde{\sigma} \leftarrow \mathtt{Sign}(Q, \mathtt{sk}_{sig})$ and $w_K \leftarrow \mathtt{ComputeWitness}(\mathsf{V}, t_K, \mathtt{sk}_{acc})$. The SP then sends (σ, w^*) to Alice.

Credential refreshment.

Alice finalizes the signature σ, i.e., $\sigma = \mathtt{Finalize}(\tilde{\sigma}, Q, r)$, and checks for correctness:

$$\begin{array}{rcl} 1 & \stackrel{?}{=} & \mathtt{Verify}(Q, \sigma) \\ 1 & \stackrel{?}{=} & \mathtt{IsNonMember}(t_K, \mathsf{V}, w_K) \end{array}$$

She updates $\mathtt{cred} = \langle (Q', \sigma'), (w_i')_{i=0}^{K-1}, (\mathtt{BL}', \mathsf{V}') \rangle$ for her next authentication as follows.

$$\begin{array}{lcl} (Q', \sigma') & \leftarrow & (Q, \sigma) \\ (w_0', w_1', \ldots, w_{K-2}') & \leftarrow & (w_1', w_2', \ldots, w_{K-1}'), \\ w_{K-1}' & \leftarrow & w_K \\ (\mathtt{BL}', \mathsf{V}') & \leftarrow & (\mathtt{BL}, \mathsf{V}) \end{array}$$

Service Provision.

Following a successful authentication, the SP serves the user and audits the user's behavior. The SP stores t_K along with the auditing information for potential blacklisting in the future.

4.4 Revocation

To attempt to revoke the user who provided t_K, the SP updates its blacklist as $\mathtt{BL} \leftarrow \mathtt{BL} \cup \{t_K\}$ and the Corresponding accumulated value as $\mathsf{V} \leftarrow \mathtt{AccumulatorAdd}(\mathsf{V}, t_K)$.

Schemes	Privacy-Enhanced Revocation?	Authentication Efficiency			
		Communication		Computation	
		Downlink	Uplink	User (Check+Prove)	Server
Accumulator-based [13, 24]	No	$O(L)$	$O(1)$	$O(L) + O(\Delta_L)$	$O(1)$
BLAC [26]/EPID [9]	Yes	$O(L)$	$O(L)$	$O(L) + O(L)$	$O(L)$
PEREA (this paper)	Yes	$O(L)$	$O(K)$	$O(L) + O(K\Delta_L)$	$O(K)$

Table 1: PEREA provides enhanced privacy like BLAC and EPID do, but the server's computation does not grow with the blacklist size; PEREA is less efficient than Accumulators, but it has better privacy.

4.5 Rate limiting

Standard techniques exist to enforce rate limiting in PEREA without eroding its guarantee on user privacy. For instance, in k-times anonymous authentication (k-TAA) [25], users remain anonymous and unlinkable (in an identity-escrow-free way) so long as they authenticate within the allowable rate, i.e., at most once per time period. On the other hand, the SP can recognize and thus refuse connections made by a user who has exceeded that rate, as authentication attempts by the same user within a single time period are linkable by the SP. Alternatively, one can use *periodic n-times periodic anonymous authentication* [11].

With rate limiting enforced, user Alice first authenticates to the SP using one of the schemes suggested above, over an SP-authenticated channel. If the authentication succeeds, Alice then carries out a PEREA authentication over the same channel. If this authentication also succeeds, the SP serves Alice over the same channel. Alice should never try to connect if she has reached the allowable authentication rate.

4.6 Security analysis

Our PEREA construction has accountability and user privacy. We state the following theorem and sketch its proof in Appendix B.)

THEOREM 1. *Our construction is secure under the Strong RSA assumption and the DDH assumption over quadratic residues modulo a safe-prime product.* □

5. PERFORMANCE EVALUATION

We now evaluate the performance of PEREA both analytically and empirically.

5.1 Complexity analysis

Table 1 summarizes the performance of PEREA in comparison with existing schemes. In both PEREA and BLAC/EPID, the number of entries in the blacklist grows with the number of misbehaviors L, which can be much larger than the number of registered users. In schemes based on dynamic accumulators, L is bounded from above by the number of registered users. In PEREA users can compute witnesses efficiently in $O(\Delta_L)$, where Δ_L is the size of $\Delta_{\mathtt{BL}} = \mathtt{BL}\backslash\mathtt{BL}'$, i.e., the difference between the current blacklist and the previously observed blacklist. [11]

During an authentication, PEREA requires only $O(K)$ computation at the server, as compared to $O(L)$ in BLAC/EPID. This computation is the main bottleneck in the system, and is therefore the most relevant metric for comparing the schemes. Accumulator-based approaches are much faster ($O(1)$), but do not provide privacy-enhanced revocation. In all schemes, the computational complexity at the user is the same — $O(L)$ time to check (via simple bit-strings comparison) if the user has been blacklisted. Generating the proofs takes $O(L)$ time in BLAC/EPID, and $O(K\Delta_L)$ in PEREA as each witness must be updated Δ_L times.

Downlink communication complexity is linear in the size of the blacklist in all schemes. In PEREA, however, each entry is 166 bits, as compared to 1164 bits in BLAC. The traffic from the SP to the user in PEREA is therefore only about 14% of that in BLAC. For example, for a blacklist of size 800, users would need to download only 16.24 KB in PEREA, but 113.67 KB in BLAC. The uplink communication complexities are the same as the computational complexities at the server: $O(K)$ for PEREA, $O(L)$ for BLAC/EPID, and $O(1)$ for dynamic accumulators.

Lastly, setup at the SP and the registration between the SP and a user grow from $O(1)$ in BLAC/EPID to $O(K)$ in PEREA, but these computations are infrequent. Revocation for all schemes requires $O(1)$ computation at the server.

5.2 Empirical evaluation

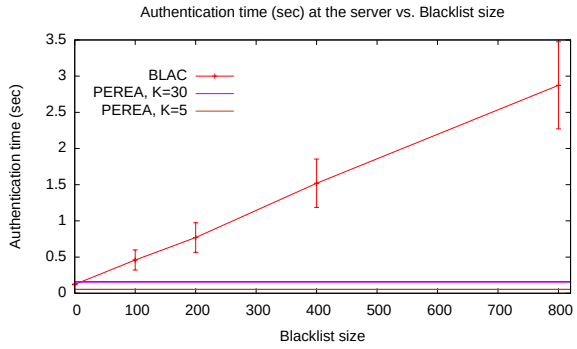
We have prototyped PEREA in C, making use of the multi-precision integer arithmetics package in OpenSSL (version 0.9.8g). Our prototype consists of several pieces: an implementation of Li et al.'s Universal Accumulator scheme (with necessary modifications), Camenisch and Lysyanskaya's signature scheme and their protocol for proving a signature on blocks of committed messages, as well as an implementation of the ZKPoK protocol in Eq. 20.

Using our source code for BLAC, we compared its efficiency with PEREA. Since our PEREA prototype is not multi-threaded, we disabled multi-threading in BLAC's code for a fair comparison. We expect the efficiency of our prototype to scale well with multi-threading, because most operations that grow with K can be parallelized.

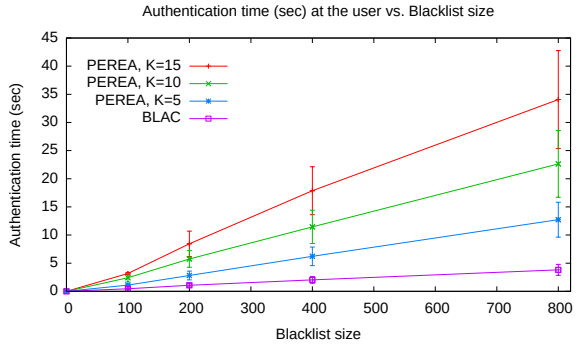
The test machine was a Lenovo T60 laptop with a 2.0 GHz Intel Core2 T7200 CPU and 1.5 GB of RAM running Ubuntu 7.10 with GNU/Linux kernel 2.6.22. In our measurements, we did not include the time taken to generate random numbers because randomness generation introduces too much noise in the timing measurements. We point out that PEREA (and BLAC) need randomness mostly for the user to compute various commitments, most of which can be precomputed before the actual authentication.

Figure 2(a) shows the authentication time at the SP for BLAC and PEREA for various blacklist sizes L. As expected, the time required for authentication increases linearly for BLAC and remains constant for PEREA. For PEREA, when $K = 5$, the SP takes 0.05s per authentication on average; when $K = 30$, it takes 0.16s. Contrast these numbers with 2.9s for BLAC when $L = 800$.

[11] We discuss timing attacks in Section 6.



(a) The authentication time at the SP grows linearly with blacklist size for BLAC. The error bars represent one standard deviation. The authentication time for PEREA is constant; the two horizontal bands represent one standard around the means.



(b) The authentication time at the user grows linearly for BLAC. K is a multiplicative factor for PEREA, and authentication at the user is slower than BLAC.

Figure 2: We see that PEREA is efficient with authentication at the SP, and outperforms BLAC. Authentication at the user is slower than BLAC, but acceptable for reasonable values of K.

Figure 2(b) shows the authentication time at the user. BLAC is more efficient, but we point out that in PEREA the computation at the user can be much less than the worst case if Δ_L is small. For example, if only 100 users have been blacklisted since the user last authenticated, then for $K = 10$, the authentication time for the user is 0.7s, compared to the worst case of 5.9s for a blacklist of 800 entries. Furthermore, we believe that users can tolerate several seconds of authentication, and even for worst-case performance, $K = 10$ represents a reasonable trade off.

6. DISCUSSION

Timing attacks. Our protocol includes an optimization in which users need to perform computation only for the new entries in the blacklist (See Eq. 19). An SP therefore could link users' connections if they are recent enough by observing low latency during authentication. To counter this attack, we require that users add a delay to make up for the difference. Our protocol therefore does not improve the delay perceived at the user, but spares users from performing unnecessary computation. We also note that this delay does not affect authentication throughput at the SP.

Choice of K. As we argued earlier, we believe that $K = 10$ represents a reasonable tradeoff between authentication latency for the user, and the size of the revocation window. If it takes a site like Wikipedia an hour to identify misbehaviors, users would be able to make an anonymous connection once every six minutes, a reasonable amount of time to accommodate small edits. If, however, it takes a site like Wikipedia a day to identify misbehaviors, $K = 10$ would limit users to only 10 anonymous connections per day. Thus, PEREA may not support frequent edits for longer periods of detection, while other schemes such as BLAC and EPID would support frequent, small edits within a day.

Concurrency. The ZKPoK protocol in Eq. 20 is *not* secure against concurrent attacks [19] and must be executed sequentially. PEREA's authentication protocol hence contains a critical section, in which the SP must wait until it has received an authenticating user's *response* before it *challenges* another authenticating user, potentially limiting the SP's authentication throughput. As indicated by our experiments, the time spent in this critical section is dominated by network latency (the computation at the user in the critical section is less than 1 ms in all experiments). The SP can include a timeout for unresponsive client to maintain a high authentication throughput. We hope to eliminate this critical section in future work.

Gaming the blacklist. Since SPs can construct arbitrary blacklists at any time (by adding and removing entries), one may wonder if an SP can game the system by presenting crafted versions of the blacklist to authenticating users. Since authentications are unlinkable in PEREA, and the SP learns only whether the user is revoked, the modified blacklists across authentications do not leak information. If an SP, however, can link multiple authentications to a single user by using external knowledge, the SP might be able to narrow down which entry in the blacklist corresponds to that user. We note that BLAC and EPID face the same issues.

Forgiving misbehaviors. We now describe briefly how PEREA supports "unblacklisting." To forgive the blacklisted user who provided ticket t_K, the SP removes t_K from $\mathtt{BL}$ and updates the current accumulator value from V to $\mathsf{V}^{1/x_K \bmod \phi(\mathsf{N})} \bmod \mathsf{N}$. If unblacklisting is allowed in PEREA, however, users must update their witnesses in $O(L)$ time (rather than the optimized $O(\Delta_L)$ time) during an authentication (Eq. 19), even if only one ticket has been removed since their last authentication.

7. CONCLUSION

We present PEREA, an anonymous authentication scheme that supports *privacy-enhanced revocation*, where anonymous users can be revoked without relying on trusted third parties. Previous schemes supporting privacy-enhanced revocation have required computation at the server that is linear in the size of the blacklist. We introduce the concept of a *revocation window* and show how the server computation is reduced to be linear in the size of the revocation window, and more importantly *independent of the size of the blacklist.* Through analytical and experimental validation, we show that for realistic parameters, PEREA provides more efficient authentication at the server than existing schemes.

8. ACKNOWLEDGMENTS

This work was supported in part by the Institute for Security Technology Studies, under Grant number 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance, and the National Science Foundation, under grant CNS-0524695. The views and conclusions do not necessarily represent those of the sponsors.

9. REFERENCES

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, volume 1880 of *LNCS*, pages 255–270. Springer, 2000.

[2] G. Ateniese, D. X. Song, and G. Tsudik. Quasi-efficient revocation in group signatures. In *Financial Cryptography*, volume 2357 of *LNCS*, pages 183–197. Springer, 2002.

[3] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *FC '99: Proceedings of the Third International Conference on Financial Cryptography*, pages 196–211, London, UK, 1999. Springer-Verlag.

[4] M. H. Au, S. S. M. Chow, and W. Susilo. Short e-cash. In *INDOCRYPT*, volume 3797 of *LNCS*, pages 332–346. Springer, 2005.

[5] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.

[6] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.

[7] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In *ACM CCS*, pages 168–177. ACM, 2004.

[8] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[9] E. Brickell and J. Li. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. In *WPES '07: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, pages 21–30, New York, NY, USA, 2007. ACM.

[10] J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, 1998. Reprint as vol. 2 of *ETH Series in Information Security and Cryptography*, ISBN 3-89649-286-1, Hartung-Gorre Verlag, Konstanz, 1998.

[11] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *ACM CCS*, pages 201–210. ACM, 2006.

[12] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[13] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[14] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[15] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

[16] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.

[17] P. Dusart. The k^{th} prime is greater than $k(\ln k + \ln \ln k - 1)$ for $k \geq 2$. *Mathematics of Computation*, 68:411–415, 1999.

[18] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[19] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[20] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[21] P. C. Johnson, A. Kapadia, P. P. Tsang, and S. W. Smith. Nymble: Anonymous ip-address blocking. In *Privacy Enhancing Technologies*, volume 4776 of *LNCS*, pages 113–133. Springer, 2007.

[22] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT*, volume 3027 of *LNCS*, pages 571–589. Springer, 2004.

[23] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, volume 4521 of *LNCS*, pages 253–269. Springer, 2007.

[24] L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *LNCS*, pages 275–292. Springer, 2005.

[25] I. Teranishi, J. Furukawa, and K. Sako. *k*-times anonymous authentication (extended abstract). In *ASIACRYPT*, volume 3329 of *LNCS*, pages 308–322. Springer, 2004.

[26] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without TTPs. In *ACM CCS*, pages 72–81. ACM, 2007.

APPENDIX

A. FORMAL SECURITY DEFINITIONS

We formally define the security notions as games played between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. $\mathcal{A}$'s capabilities are modeled as arbitrary and adaptive, but non-interleaving accesses to various oracles, which together share a private state state that contains counters m, n and a, and sets $\mathcal{U}_P, \mathcal{U}_A, \mathcal{U}_B, \mathcal{A}_A$, which are initialized to 0 and $\emptyset$, respectively. Here we define the oracles.

- P-REG allows $\mathcal{A}$ to register an honest user with the honest SP. Upon invocation, the oracle increments n by 1, simulates the registration protocol between an honest user and the honest SP, appends $\langle n, trans_n, cred_n \rangle$ to state, where $trans_n$ is the resulting protocol transcript and $cred_n$ is the resulting user credential, adds n to $\mathcal{U}_P$ and returns $(trans_n, n)$ to $\mathcal{A}$. The user is indexed by n.

- A-REG (resp. B-REG) allows $\mathcal{A}$ to register a corrupt (resp. honest) user with the honest (resp. corrupt) SP. Upon invocation, the oracle increments n by 1, plays the role of the SP (resp. a user) and interacts with $\mathcal{A}$ in the registration protocol, appends $\langle n, trans_n, \perp\rangle$ (resp. $\langle n, \perp, cred_n\rangle$) to state, where $trans_n$ is the protocol transcript (resp. $cred_n$ is the credential issued to the user by $\mathcal{A}$), adds n to $\mathcal{U}_A$ (resp. $\mathcal{U}_B$) and returns n to $\mathcal{A}$. The user is indexed by n.
- CORRUPT-U(i) allows $\mathcal{A}$ to corrupt an honest user. On input i, the oracle removes i from $\mathcal{U}_B$ or $\mathcal{U}_P$, adds i to $\mathcal{U}_A$, and returns $cred_i$ to $\mathcal{A}$.
- P-AUTH(i) allows $\mathcal{A}$ to eavesdrop an authentication run between an honest user and an honest SP. On input i) such that $i \in \mathcal{U}_P \cup \mathcal{U}_B$, the oracle increments a by 1, simulates (using $cred_i$) the authentication protocol between honest user i and honest SP, appends $\langle \pi_a, a\rangle$ to state, where π_a is the resulting protocol transcript, and returns (π_a, a) to $\mathcal{A}$.
- A-AUTH (resp. B-AUTH(i)) allows a corrupt user (resp. SP) to be authenticated by an honest SP (resp. user). The oracle increments a by 1, plays the role of the SP (resp. user i to be authenticated by the SP, for input $i \in \mathcal{U}_B \cup \mathcal{U}_P$) and interacts with $\mathcal{A}$ in the authentication protocol, adds a to $\mathcal{A}_A$ in case of A-AUTH, appends $\langle \pi_a, a\rangle$ to state, where π_a is the resulting protocol transcript, and returns a to $\mathcal{A}$.
- ADD-TO-BL(k) allows $\mathcal{A}$ to influence an honest SP to think that an authenticated session involves a misbehavior. On input $k \leq a$, the oracle adds the ticket $\tau_k = \text{Extract}(\pi_k)$ to the SP's blacklist.
- REMOVE-FROM-BL(τ) allows $\mathcal{A}$ to influence an honest SP to think that an authenticated session does not involve a misbehavior. On input τ such that τ is in the SP's blacklist, the oracle removes τ from that blacklist.

A.1 Accountability

Setup Phase. $\mathcal{C}$ takes a sufficiently large security parameter and generates spk and ssk. spk is given to $\mathcal{A}$.

Probing Phase. $\mathcal{A}$ is allowed to issue queries to all the oracles except B-REG.

End Game Phase. $\mathcal{A}$ outputs $j \in \mathcal{S}_P$. $\mathcal{A}$ wins the game if both of the following are true: (1) There exists a sub-sequence S of the sequence of all oracle queries in the same order as they were made, where $S = \langle a_1 :=$ A-AUTH, ADD-TO-BL(a_1), $a_2 :=$ A-AUTH, ADD-TO-BL(a_2), ..., $a_k :=$ A-AUTH, ADD-TO-BL$(a_k), a_{k+1} :=$ A-AUTH$(j)\rangle$, such that $a_x \in \mathcal{A}_A$ for all $x = 1$ to k, and in between $a_i :=$ A-AUTH and ADD-TO-BL(a_i), there are less than K A-AUTHqueries. (2) $k \geq |\mathcal{U}_A| + Q_R$, where Q_R is the number of REMOVE-FROM-BL(j, Extract(π_{a_i})) queries such that $i \in \{1, \ldots, k\}$.

A.2 User privacy

Setup Phase. $\mathcal{C}$ takes a sufficiently large security parameter and generates spk and ssk, which are given to $\mathcal{A}$.

Probing Phase. $\mathcal{A}$ is allowed to issue queries to all the oracles except P-REG and A-REG. Oracle queries can be interleaved and/or span the Challenge Phase and Probing Phase 2.

Challenge Phase. $\mathcal{A}$ outputs $i_0, i_1 \in \mathcal{U}_B$. $\mathcal{C}$ then flips a fair coin $b \in \{0, 1\}$. $\mathcal{A}$ queries P-AUTH($\perp$) if the SP is honest and B-AUTH($\perp$) otherwise, without specifying i. $\mathcal{C}$ answers the query assuming i_b.

Probing Phase 2. $\mathcal{A}$ is allowed to issue queries as in the Probing Phase, except that queries to CORRUPT-U(i_0) or CORRUPT-U(i_1) are not allowed.

End Game Phase. $\mathcal{A}$ outputs a guess bit b'. $\mathcal{A}$ wins if $b = b'$ and has never invoked ADD-TO-BL(k) such that π_k is an authentication transcript from user i_0 or i_1.

B. PROOF OF THEOREM 1 (SKETCH)

Accountability.

Suppose there exists a PPT adversary $\mathcal{A}$ which can win in game Accountability with non-negligible probability, we show how to construct a PPT simulator $\mathcal{C}$ that can forge a CL signature [12]. $\mathcal{C}$ is given access to a signing oracle O^{CL}_{Sign} and its goal is to output a new message-signature pair of the CL signature. $\mathcal{C}$ uses the public key of the CL signature to set up the parameter of PEREA. For each registration request (indexed by request i) due to a malicious user, $\mathcal{C}$ invokes O^{CL}_{Sign} to obtain a CL signature σ^{CL}_i. With σ^{CL}_i, $\mathcal{C}$ is able to provide the correct response. For each authentication request (indexed by request j) due to a malicious user, $\mathcal{C}$ conducts a rewind simulation to extract the underlying CL signature $\hat{\sigma}^{CL}_j$. Due to the soundness property of the protocol, $\hat{\sigma}^{CL}_j$ must be a valid CL signature. Other oracle queries related to honest users can be handled in a straightforward fashion, as $\mathcal{C}$ is in possession of the user secret. Due to the zero-knowledge property of the PoK protocols, $\mathcal{C}$ is able to simulate the related transaction from honest users. In the end-game phase, $\mathcal{A}$ has issued $k + 1$ A-Auth Oracle query and Add-To-BL oracle query. Due to the setting of the game (and the soundness of the proof of knowledge protocol in authentication request), at least one of the signatures $\hat{\sigma}^{CL}_j$ is different from all σ^{CL}_i. $\mathcal{C}$ wins the game because it has obtained a new CL signature that is not the output of O^{CL}_{Sign}. Since CL signature is unforgeable under the Strong RSA assumption, our system possesses Accountability under the same assumption.

User privacy.

The PoK protocols in user registration and authentication can be constructed to be general zero-knowledge proof-of-knowledge protocols with zero-knowledgeness and soundness under the Strong RSA and the DDH assumptions using standard technique [16]. It is obvious that all components presented to the server but ticket t^* do not reveal any information about the underlying user. For instance, the commitment scheme is statistically hiding and reveal nothing about the content committed. It remains to argue that t^* presented during the authentication protocol does not reveal any information for the server to link different transactions. Observe that each t^* appears in plain once only and it only appears in a commitment for other transactions. Due to the hiding property of the commitment scheme, for any t^* there exists an opening that matches any commitments in previous transactions. Hence, PEREA possesses user privacy, as implied by the zero-knowledge property of the instantiation of the protocols and the hiding property of the commitment, which are in turn implied by the Strong RSA assumption and the DDH assumption.