

PEREA: Practical TTP-Free Revocation of Repeatedly Misbehaving Anonymous Users

Man Ho Au

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

Patrick P. Tsang

Department of Computer Science
Dartmouth College, USA

Apu Kapadia

School of Informatics and Computing
Indiana University Bloomington, USA

Indiana University Technical Report TR688

July 2011

(Revises the previous versions dated November 2010 and April 2011)

Abstract

Several anonymous authentication schemes allow servers to revoke a misbehaving user’s future accesses. Traditionally, these schemes have relied on powerful Trusted Third Parties (TTPs) capable of deanonymizing (or linking) users’ connections. Such TTPs are undesirable because users’ anonymity is not guaranteed, and users must trust them to judge ‘misbehavior’ fairly. 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*, which is impractical as the size approaches thousands of entries. We propose PEREA, a new anonymous authentication scheme for which this bottleneck of computation is *independent of the size of the revocation list*. Instead, the time complexity of authentication is linear in the size of a *revocation window* $K \ll L$, the number of subsequent authentications before which a user’s misbehavior must be recognized if the user is to be revoked. We extend PEREA to support more complex revocation policies that take the *severity* of misbehaviors into account. Users can authenticate anonymously if their *naughtiness*, i.e., the sum of the severities of their blacklisted misbehaviors, is below a certain naughtiness threshold. We call our extension PEREA-Naughtiness. We prove the security of our constructions, and validate their efficiency as compared to BLAC both analytically and quantitatively.

1 Introduction

Anonymous access to services can be desirable in many situations such as posting content related to whistle blowing, journalism in oppressive regimes, and activism. Fully anonymous access to services, 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 could not hold users who deface webpages accountable. For this reason, many services (including Wikipedia) have entirely blocked anonymizing networks such as Tor [20].¹ Given the advantages of anonymous access, denying anonymity to all users is a drastic measure. To provide a balance between the two extremes of full anonymity and no anonymity, various anonymous authentication schemes provide some form of *accountable anonymity*. Recognizing the importance and impact of research in the space of accountable anonymity, the Tor Project lists the integration of such systems as an important part of their roadmap,² and in 2009 the PET award committee recognized our recent work for improving such systems.³ Research on accountable anonymity is thus of both practical and theoretical

¹The Abuse FAQ for Tor Server Operators lists examples at <https://www.torproject.org/docs/faq-abuse.html.en>.

²Tor Development Roadmap, 2008–2011. Section 7.3 lists our prior work on Nymble [26, 38] as a priority for further development. <https://www.torproject.org/press/presskit/2008-12-19-roadmap-full.pdf>

³Our conference papers on BLAC [35] and PEREA [36] were recognized as Runners up for the Award for Outstanding Research in Privacy Enhancing Technologies <http://petsymposium.org/award/>. This article extends on our initial work on PEREA.

significance. We now describe the space of existing schemes and provide an overview of our contributions.

Anonymous authentication schemes allow users to authenticate to *service providers (SPs)* as some anonymous member of a group. For accountability several schemes support the revocation of these 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 identities (or pseudonyms⁴) 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, 18, 28] feature an *Open Authority (OA)*, which 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, 29, 32] and hash chains [26, 38] also rely on TTPs. Nymbler [25] and Jack [30] are two recent improvements on the hash chains-based Nymble system [26, 38] that also require TTPs. We omit details on the subtleties of the various schemes, and simply emphasize that all these schemes feature a TTP (or a set of TTPs) that can deanonymize users or link⁵ their accesses.

Having a TTP with such power is undesirable as users are never guaranteed the anonymity of their connections. Users must trust the TTP to judge their ‘misbehavior’ 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 the press.

1.1 Eliminating TTPs, but at a cost

Enhanced Privacy ID (EPID) [9], and our own Blacklistable Anonymous Credentials (BLAC) [35] are two recently proposed schemes that for the first time eliminate the reliance on TTPs for revocation, thus providing *privacy-enhanced revocation* [35].⁶ In BLAC and EPID, SPs can add an entry from an anonymous user’s authentication transcript to a blacklist, following which the user is revoked and cannot authenticate. No TTP is needed to perform these actions, and revoked users remain anonymous. Privacy-enhanced revocation also allows for *subjective judging* [35], 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 [17] and *k*-Times Anonymous Authentication (*k*-TAA) [34] support accountability in only narrowly defined applications where misbehaviors can be mapped to ‘too many authentications’ (such as ‘double spending’ a digital coin, which deanonymizes the offending user).

While BLAC and EPID eliminate the reliance on TTPs, the amount of computation at

⁴Some systems may allow users to establish accounts under a fake name. These fake names are commonly known as *handles* or *pseudonyms*. For accountability, systems may ensure in some way that it is difficult for a single person to establish multiple pseudonyms in the system.

⁵We 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 non-negligibly greater than random guessing.

⁶Brickell and Li [9] refer to this concept as “enhanced revocation” in the context of EPID, and we called this concept “anonymous blacklisting” in the context of BLAC. As will become clear, we now distinguish between the *action* of blacklisting and the *end result* of revocation.

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 the same user) would make the costs of authentication prohibitive and pose a severe bottleneck at the SP. For example, for a blacklist with 10,000 entries, BLAC requires around 68 seconds of computation at the SP for a single authentication on commodity hardware (see analysis in Section 8.2). These numbers would double (to approximately 136 seconds at the SP) for 20,000 entries.⁷ In our paper on BLAC, we acknowledged this limitation and listed “more efficient blacklist checking” as an open problem. As described next, PEREA is designed to address this problem.

1.2 PEREA, an efficient alternative

This article revises and extends the PEREA scheme first published by the ACM Conference on Computer and Communication Security (CCS) [36]. We start by describing this scheme, and in Section 1.3 we describe our new extension *PEREA-Naughtiness* and our additional contributions.

PEREA (Privacy-Enhanced Revocation with Efficient Authentication) is 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. This relaxed semantics of revocation allows for a more efficient solution with the tradeoff that it is possible for a misbehaving user to escape revocation if not caught in time. For example, if $K = 10$, then 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 many cases where users are not expected to authenticate more than a few times a day, $K = 10, T = 2880$ would allow SPs 2 days to catch misbehaviors and allow 5 authentications per day on average. In practice, SPs would want to keep K low to rate limit anonymous authentications. Without such a rate limit, accountable anonymity is not very useful if users can perform, say, 10,000 misbehaviors before being blacklisted. In our paper on BLAC [35], we suggested that the rate of authentication should be limited

⁷As a rough baseline we looked at PhishTank, a community based service that tracks phishing sites’ IP addresses. In April 2011, 15,872 “valid” phishing IP addresses were identified. Thus for comparison we use 10,000–20,000 blacklist entries as a reasonable number to expect for a large service trying to keep anonymous misbehaving users at bay. <http://www.phishtank.com/stats/2011/04/>

for this reason. Thus we argue “at most K authentications in the time it takes to identify misbehaviors” would be a natural requirement in other schemes as well.

In practice we expect K to be in the range of 5–15 (which will be much smaller than L in practice), leading to much better performance than BLAC or EPID in which the computational complexity depends linearly on the number of blacklist entries. For example, L can grow to thousands of entries in an application such as Wikipedia, while K is limited to a constant as small as 5 or 15 (see concrete performance numbers at the end of Section 1.3).

1.3 PEREA-Naughtiness: Supporting flexible revocation policies

Before we describe how our extension PEREA-Naughtiness supports ‘naughtiness policies,’ it is helpful to discuss the d -strikes-out policy first introduced as an extension to BLAC in our recent TISSEC ’10 article [37]. Existing anonymous revocation schemes (including EPID, BLAC, and the original PEREA) implement ‘1-strike-out’ policies—a single blacklisted misbehavior implicates the user for revocation. As a natural extension, the d -strikes-out policy attempts to block a misbehaving user if he/she has misbehaved d or more times. For example, an SP may want to revoke users that have misbehaved 3 or more times and continue to grant access to users who have been blacklisted only once or twice. (Note that unlike other schemes such as k -TAA, the d -strikes-out policy decouples the notion of misbehaviors from authentications. A user may have authenticated a 1,000 times but misbehaved only 3 times. Thus k -TAA schemes cannot implement such policies.)

The drawback of a d -strikes-out policy is it doesn’t differentiate between misbehaviors and treats them all equally. For example, an SP might consider a copyright violation to be more egregious than an inappropriate comment and might want to give more weight to certain misbehaviors. With *PEREA-Naughtiness*, we generalize the d -strikes-out policy to a weighted version called the *naughtiness* policy. Each misbehavior can be scored with a *severity*, and the *naughtiness* of a user is the sum of severities of his/her blacklisted misbehaviors (within the revocation window). If the user’s naughtiness is beyond the specified *naughtiness threshold*, the user will not be able to authenticate. For example, Wikipedia could blacklist inappropriate comments with a severity of 1, and copyright violations with a severity of 3. A threshold naughtiness of 5 would deny authentication to Alice who has made 5 inappropriate comments or Bob who has made one copyright violation and 2 inappropriate comments. Thus with PEREA-Naughtiness, SPs have the option for more flexible blacklisting, in a way that is natural—misbehaviors are scored based on their severity, and users that have been ‘too naughty’ in the recent past (as dictated by the revocation window) are revoked. Providing constructions, models and proofs for naughtiness policies in the context of PEREA-Naughtiness thus provides a significant contribution over the work on d -strikes-out policies as presented in the TISSEC ’10 article on BLAC [37] (we also point out that BLAC uses a completely different construction from PEREA).

We note the d -strikes-out policy is a special case of PEREA-Naughtiness with severity of 1 for all misbehaviors and a naughtiness threshold of d . Since the construction of this special case for PEREA-Naughtiness can be made more efficient, we refer to this efficient construction as PEREA- d -strikes-out. We will use the term *PEREA-based schemes* when we refer to these three constructions collectively. In Section 8.2 we will show the performance for PEREA-based schemes as compared to BLAC. Again, one must keep in mind that PEREA-

based schemes address misbehaviors only within the revocation window, whereas BLAC can address misbehaviors made at any point in the past. Thus the added performance comes with the tradeoff of not being able to penalize users for misbehaviors beyond the revocation window.

Our other significant contribution is that we formally define a model that captures the notion of security (and privacy) for PEREA-Naughtiness and prove the security of our new construction under the proposed model. As opposed to high-level sketches provided in our conference paper [36], this article presents detailed models and proofs. Finally, we complement the complexity analysis of our scheme with empirical measurements that demonstrate their practicality in realistic settings. For a revocation window of size $K = 10$, on commodity hardware our constructions would result in authentication times at the SP of about 4 seconds for PEREA and 14 seconds for PEREA-Naughtiness *regardless of blacklist size* (recall this number was about 136 seconds for BLAC with $L = 20,000$ and would continue to grow linearly with blacklist size). The amount of computation at the user is potentially increased (as compared to BLAC and EPID) because the users’ credentials need to be refreshed when new entries are added to the blacklist. If users periodically (e.g., once a day) refresh their credentials, then the performance is comparable to BLAC during an authentication (see Section 8.2).

Finally, we note authentications in BLAC and PEREA-based schemes would be tied to particular ‘write’ actions such as page edits or comment posting where there is the potential for posting unwanted content, and not the more numerous ‘read’ actions such as webpage downloads. With a single server dedicated to authentication, PEREA-Naughtiness could support 6,000 anonymous actions per day and extra servers could be used to improve this number. For a popular site such as Wikipedia, if up to 5% of the total edits to English webpages were anonymous, a single server’s throughput would be acceptable.⁸ We also note that for authenticated *actions* a user does not have to wait to perform the action, as the authentication can be performed after the action, as part of the process of accepting updates from the SP.

1.4 Summary of 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 present an extension *PEREA-Naughtiness* to support *naughtiness* policies, where individual misbehaviors can be treated with different *severities*, and users with naughtiness levels over a threshold can be revoked. Furthermore, naughtiness policies generalize *d*-strikes-out policies that were introduced as an extension to BLAC (although a completely different construction is needed in the context of PEREA).
- We evaluate the performance of PEREA-based schemes both analytically and quantitatively as compared to BLAC. We demonstrate that PEREA-based schemes outperform BLAC in server computation at the expense of increased computation at the user. We

⁸ Wikipedia Statistics, Edits per day: <http://stats.wikimedia.org/EN/PlotsPngDatabaseEdits.htm>

show that the computation at the user is reasonable, and even *better* than BLAC if one assumes the blacklist is relatively stable (e.g., the blacklist has grown by only 5% since the user’s last authentication).

- We formally define our security model for PEREA-based schemes, and prove the security of our construction under this model without random oracles.

1.5 Paper outline

We present the security goals for PEREA-based schemes and provide an overview of our solution in Section 2. After introducing cryptographic building blocks in Section 3, we present our construction of PEREA in Section 4. In Section 5 we present our extended constructions for PEREA-*d*-strikes-out and PEREA-Naughtiness. Next, we formalize the security definitions in Section 6, and in Section 7 we prove the security of our constructions. We present a detailed analytical and quantitative evaluation in Section 8. When any system is realized as a practical implementation, some issues (such as timing attacks) outside the scope of the security model must be considered. We discuss these issues in Section 9, and conclude in Section 10.

2 Overview of PEREA-based schemes

Before diving into the technical details of our solution in the following sections, we now provide an accessible overview of our constructions. PEREA-based schemes use a cryptographic accumulator (described in Section 2.2) to represent a blacklist, but because PEREA-based schemes have no TTP, the application is not straightforward. We start by describing our security goals, and then describe the limitations of existing accumulator-based schemes, followed by a high-level overview of both PEREA and our extension PEREA-Naughtiness.

2.1 Security Goals for PEREA-based schemes

We now describe the security properties needed by PEREA-based schemes. Here we give informal descriptions of these properties and refer the interested reader to Section 6 for a formal definition of these properties.

PEREA-based schemes must have the basic property of *misauthentication resistance*, i.e., no unregistered user should be able to authenticate. PEREA-based schemes 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 in which a malicious SP recognizes a user as being revoked without the user being aware of his or her reduced privacy.

2.2 Accumulators

A *dynamic accumulator* (going forward, we will call it an *accumulator*), 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 (without the secret key to the accumulator) can prove in zero knowledge that a certain element is ‘in’ the accumulator *if and only if* the element has indeed been accumulated. There seem to be two relatively straightforward ways to apply accumulators to anonymous authentication. In the *whitelisting* approach each user is associated with a unique pseudonym and authenticates himself/herself by proving in zero knowledge that his/her pseudonym is in the accumulator [13]. Offending users could be removed from this whitelist. On the other hand, in the *blacklisting* approach, offending users are added to the accumulator, and the authenticating user proves his/her pseudonym is *not* in the accumulator [29]. In both of 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 the individual actions of) a user, and therefore current accumulator-based solutions do not suffice. We need a solution in which pseudonyms are not added to 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. PEREA-Naughtiness adds the extra functionality of proving that the sum of the severity of those tickets are within a specified threshold. What follows is a high-level description of how PEREA and PEREA-Naughtiness implement this functionality.

2.3 Overview of our construction for PEREA

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 data structures. 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

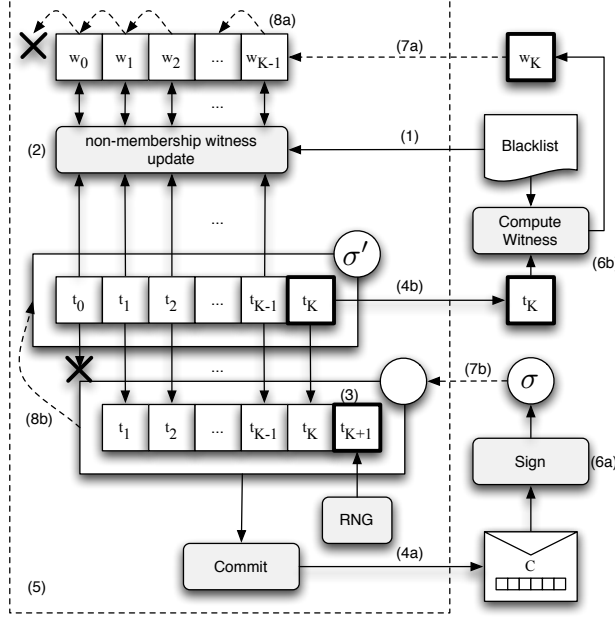


Figure 1: (1) User Alice obtains the SP’s blacklist and (2) updates her list of witnesses (w_0, \dots, w_{K-1}) for the tickets (t_0, \dots, t_{K-1}) in her queue. (3) Alice generates ticket t_{K+1} for use in the next authentication and constructs the new queue (t_1, \dots, t_{K+1}) . (4) Alice generates 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.

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, \dots, 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* \mathbb{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, \dots, t_{K-1} has not been blacklisted *without disclosing those tickets*. This proof is done in zero knowledge, in which the user generates for each ticket 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 ($O(L)$) to generate these witnesses⁹, 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., that 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 SP 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 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.

2.4 Overview of our extension for PEREA-Naughtiness

In PEREA-Naughtiness each misbehavior on the blacklist is associated with a non-negative integer called “severity” indicating the extent or gravity of the misbehavior. A user is authenticated if and only if the sum of severities of his blacklisted misbehaviors, called “naughtiness” is less than a threshold; i.e., naughtiness is tolerated up to a limit. An entry in the blacklist is now of the form $(t, \varsigma, \sigma_{t,\varsigma})$, where t is the ticket associated with the misbehavior, while ς is the corresponding severity, and $\sigma_{t,\varsigma}$ is a signature on (t, ς) . The SP additionally declares a threshold naughtiness value n where authentication succeeds only if the naughtiness of Alice is less than or equal to n .

We extend the original scheme to support naughtiness policies by making use of commitments [33]. Alice prepares commitments $C_0^{(\varsigma_0)}, C_1^{(\varsigma_1)}, \dots, C_{K-1}^{(\varsigma_{K-1})}$ for the severity of all tickets in her queue Q . A ticket has a severity of 0 if it is not put in the blacklist. During the authentication protocol, Alice proves to the SP that these commitments are correctly formed by demonstrating in zero-knowledge that exactly one of the following is true:

- $C_i^{(\varsigma_i)}$ is a commitment of 0, $Q[i]$ is indeed the i -th ticket in her queue and $Q[i]$ has not been accumulated in the accumulator that represents the blacklist;
- $C_i^{(\varsigma_i)}$ is a commitment of severity ς , $Q[i]$ is indeed the i -th ticket in her queue, and the blacklist contains the signature of the tuple $(Q[i], \varsigma)$. Knowledge of a signature $\sigma_{t,\varsigma}$ on the tuple (t, ς) assures the SP that a ticket t of severity ς is indeed on the blacklist and not a severity fabricated by Alice.

Thus Alice cannot lie about the severity associated with each of her tickets. If a ticket is blacklisted, she must commit the actual severity value, and if a ticket is not blacklisted,

⁹As we note in Section 8, blacklists in PEREA are approximately 14% the size of blacklists in BLAC.

she commits a 0. Next, due to the homomorphic property of the commitment scheme, the SP can compute $\prod C_i^{(s)}$, which is a commitment to the naughtiness of Alice. Finally, Alice proves to the SP in zero knowledge that the value committed in $\prod C_i^{(s)}$ is less than the threshold n .

3 Building Blocks

We now outline the various cryptographic primitives that we use to realize PEREA-based schemes. Readers who are not interested in the details of our cryptographic construction and proof of security may choose to skip Sections 3–7.

3.1 Preliminaries

3.1.1 Notation and intractability assumptions

If S is a finite 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 $\{\ell, \ell + 1, \dots, \delta\}$, by Λ_ℓ the set $[0, 2^{\ell+1} - 1]$, i.e., the set of integers of size at most $\ell + 1$ bits, by Π_ℓ the set $\{e \in \Lambda_\ell \mid e \text{ is prime}\}$, 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 integer $N = pq$ is called a *safe-prime product* if p and q are 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 a is the output of an execution of $A(\cdot)$. Finally, $a \doteq b$ defines a to be b .

The security of PEREA-based schemes relies on the *Strong RSA Assumption* [4, 22] and the *Decisional Diffie-Hellman (DDH) Assumption* [5] 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 $v^e = u \pmod N$, 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 $1/2$.

3.1.2 ZKPoK protocols

In a Zero-Knowledge Proof-of-Knowledge (ZKPoK) protocol [24], 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-based schemes make use of ZKPoK protocols.

We follow the notation introduced by Camenisch and Stadler [15]. 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. Symbols appearing on the left of the colon denote values whose

knowledge is being proved while symbols appearing on the right, but not the left, of the colon denote public values.

3.2 Tickets and queues

In PEREA-based schemes, a user picks a *ticket* uniformly at random from the set Π_{ℓ_t} , where ℓ_t is a security parameter. For example, there are at least 2^{224} tickets in this set when $\ell_t = 230$ ¹⁰.

For a reasonably large number, say 2^{40} , of randomly picked tickets from the set of size 2^{224} , the probability that two of them collide is approximately 2^{-112} due to the birthday paradox. We set the ticket domain to be $\mathcal{T} \doteq [-2^{\ell_T} + 1, 2^{\ell_T} - 1]$, where $\ell_T > \ell_t$ is another security parameter.¹¹

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 enqueueing (**Enq**) and dequeueing (**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-based schemes, 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-based schemes make use of *universal dynamic accumulators (UDAs)* recently introduced by Li et al. [29]. 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 such a way that the time for the verifier to check the proof is independent of the number of inputs already accumulated into V .¹² In PEREA-based schemes, 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-based schemes. We call the modified scheme **TicketAcc**.

Key generation On input security parameter $\text{param}_{acc} = \ell_N$, choose an ℓ_N -bit safe-prime product $\mathbf{N} = \mathbf{pq}$ uniformly at random, pick $\mathbf{g} \in_R QR_{\mathbf{N}}$, and output the accumulator private key $\mathbf{sk}_{acc} = \phi(\mathbf{N})$ and public key $\mathbf{pk}_{acc} = (\ell_N, \mathbf{N}, \mathbf{g})$. \mathbf{pk}_{acc} is an implicit input to all the algorithms below.

¹⁰This follows from a result due to Dusart [21]: if $\pi(x)$ is the number of distinct primes less than x , then $\pi(x) > \frac{x}{\ln x} (1 + \frac{0.992}{\ln x})$ for all $x > 598$.

¹¹Note that $\mathcal{T} \supseteq \Pi_{\ell_t}$. This allows a user in PEREA to employ the more efficient range proof, which is not tight, to prove in zero knowledge to the SP that she knows some ticket in \mathcal{T} .

¹²The time required for the prover depends on the number of inputs already accumulated though.

The accumulator allows any input in the domain $\mathcal{X} \doteq \{x \in \mathcal{X}' \mid x \text{ is prime}\}$ to be accumulated, where $\mathcal{X}' = [0, 2^{\ell_x})$ with $\ell_x = \lfloor \ell_{\mathbf{N}}/2 \rfloor - 2$. The choice of $\ell_{\mathbf{N}}, \ell_t$ should ensure $\Pi_{\ell_t} \subset \mathcal{X}$.

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

$$\text{Accumulate}(\mathbf{V}, t) \rightarrow \mathbf{V}' \doteq \mathbf{V}^t \pmod{\mathbf{N}}. \quad (1)$$

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

$$\text{Accumulate}(\mathbf{V}, S_T) \quad (2)$$

to mean the repetitive invocation of `Accumulate` to accumulate tickets t_1, t_2, \dots, t_L , one at a time. An accumulator value is initially \mathbf{g} . The accumulator value \mathbf{V} resulting from accumulating S_T is thus:

$$\text{Accumulate}(\mathbf{g}, S_T) \rightarrow \mathbf{V} \doteq \mathbf{g}^{t_1 t_2 \dots t_L} \pmod{\mathbf{N}}. \quad (3)$$

We abbreviate the above as `Accumulate`(S_T).

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

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

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

We sometimes simply call non-membership witnesses “witnesses”.

Computation of non-membership witnesses If $\mathbf{V} = \text{Accumulate}(S_T)$ for some $S_T \subset \mathcal{T}$, then for any $t \in \mathcal{T} \setminus S_T$, one can compute, using knowledge of \mathbf{sk}_{acc} , a witness $w = (a, \mathbf{d})$ for t with respect to \mathbf{V} :

$$\text{ComputeWitness}(t, \mathbf{V}, \mathbf{sk}_{acc}, S_T) \rightarrow w \quad (5)$$

so that $1 = \text{IsNonMember}(t, \mathbf{V}, w)$. The non-membership witness w can be computed with or without the accumulator private key $\mathbf{sk}_{acc} = \phi(\mathbf{N})$. The former, which is more efficient, is employed by PEREA-based schemes and is reviewed below. On input of a value t , one first checks whether $t \notin S_T$. Let $u = \prod_{t \in S_T} t$ and $u' = u \pmod{\phi(\mathbf{N})}$. If $\gcd(t, u') = 1$, one finds a and b such that $au' + bt = 1$, and sets the non-membership witness w for t as $(a, \mathbf{g}^{-b} \pmod{\mathbf{N}})$. If $\gcd(t, u') \neq 1$, one finds a and b such that $au + bt = 1$, and sets the non-membership witness w for t as $(a, \mathbf{g}^{(-b \pmod{\phi(\mathbf{N})})} \pmod{\mathbf{N}})$. The value u' can be stored to speed up future computation. Note that with the knowledge of \mathbf{sk}_{acc} , it is possible to create a *false* non-membership witness, that is, a non-membership witness for t can be computed even if $t \in S_T$.

Update of non-membership witnesses Given a witness w such that $1 = \text{IsNonMember}(t, \mathbf{V}, w)$, when \mathbf{V} gets updated to \mathbf{V}' via the accumulation of a new ticket $t' \in \mathcal{T} \setminus \{t\}$ into it (i.e., $\mathbf{V}' = \text{Accumulate}(\mathbf{V}, t')$), anyone can compute, *without the knowledge of sk_{acc}* , a witness w' for the same ticket t with respect to the updated accumulated value \mathbf{V}' (i.e., $1 = \text{IsNonMember}(t, \mathbf{V}', w')$) as:

$$\text{UpdateWitness}(w, t, \mathbf{V}, t') \rightarrow w' \quad (6)$$

Let $w = (a, d)$ be the non-membership witness for t with respect to \mathbf{V} . The new non-membership witness $\hat{w} = (\hat{a}, \hat{d})$ for t with respect to \mathbf{V}' can be computed as follows. One finds a_0, r_0 such that $a_0 t' + r_0 t = 1$. This is possible since the tickets are distinct primes. This implies $a_0 a t' + r_0 a t = a$. One then sets $\hat{a} = a_0 a \bmod t$ and finds r such that $\hat{a} t' = a + r t$. Finally, one computes $\hat{d} = d \mathbf{V}^r$ and sets $\hat{w} = (\hat{a}, \hat{d})$. To see \hat{w} is the new non-membership witness, it is true that

$$\mathbf{V}^{\hat{a}} = \mathbf{V}^{\hat{a} t'} = \mathbf{V}^{a + r t} = d^t g \mathbf{V}^{r t} = (d \mathbf{V}^r)^t g = \hat{d}^t g$$

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

$$\text{UpdateWitness}(w, t, \mathbf{V}, S_T) \quad (7)$$

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

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

3.3.2 Proof that a ticket is not accumulated

To prove in zero knowledge that a ticket $Q[i]$ in queue Q is not accumulated in an accumulator value \mathbf{V} , one can conduct

$$PK \{ (Q[i], w) : 1 = \text{IsNonMember}(Q[i], \mathbf{V}, w) \} \quad (8)$$

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. [29, §5]. The construction has a complexity of $O(1)$, i.e., independent of the number of inputs that have been accumulated into \mathbf{V} . Note that the above proof only convinces the verifier that the prover knows a value, denoted as $Q[i]$, that is not accumulated in an accumulator. The proof that $Q[i]$ is the i -th least recently enqueued ticket is addressed in Section 3.4.

3.4 Proving the integrity of the queue

3.4.1 A protocol for queue signing

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

PEREA-based schemes must use a signature scheme in which Alice can request a signature on the queue from the SP 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 of the *signature scheme for blocks of messages* [14, §4] and the *protocol for signing blocks of committed values* [14, §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-based schemes.

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

Key generation On input security parameters $\text{param}_{sig} = (\ell_N, \ell_s, \ell_e, \ell_T, \ell, \delta_r)$, the SP chooses an ℓ_N -bit safe-prime product $N = pq$ uniformly at random, and $b, c, g_0, g_1, \dots, g_K \in_R QR_N$, and then outputs the signature private key $\text{sk}_{sig} = \phi(N)$ and public key $\text{pk}_{sig} = (\text{param}_{sig}, N, b, c, (g_i)_{i=0}^K)$.¹³ The SP keeps sk_{sig} private and publishes pk_{sig} to the public. pk_{sig} is an implicit input to the algorithms 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 :

$$\text{Commit}(Q, r) \rightarrow C \doteq c^r \prod_{i=0}^K g_i^{t_i} \text{ mod } N, \quad (9)$$

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 prove that the commitment was constructed correctly:

$$PK \{(Q, r) : C = \text{Commit}(Q, r) \wedge Q \in \mathcal{Q} \wedge r \in \mathcal{R}\}, \quad (10)$$

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 sk_{sig} :

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

where $r' \in_R \Lambda_{\ell_s}$, $e > 2^{\ell_T+1}$ is a random prime of length ℓ_e and $v = (bc^{r'}C)^{1/e \text{ mod } \phi(N)} \text{ mod } N$.

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

$$\text{Finalize}(\tilde{\sigma}, r) \rightarrow \sigma \doteq (s := r + r', e, v). \quad (12)$$

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

¹³We use N, N to denote the RSA moduli for the accumulator and the signature respectively. They can be the same or different in PEREA-based schemes.

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

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 $\ell_e > \ell_T + 2$, $\ell_s = \ell_N + \ell_T + \ell$ and $\delta_r = \lfloor \frac{\ell_N - 1}{\epsilon} - \ell \rfloor$ for some $1 < \epsilon \in \mathbb{R}$, these properties can be proved to hold for sufficiently large ℓ_N, ℓ under the Strong RSA Assumption in virtually the same way as Camenisch and Lysyanskaya proved theirs [14].

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:

$$PK \{(Q, \sigma) : 1 = \text{Verify}(Q, \sigma) \wedge Q \in \mathcal{Q}\} \quad (14)$$

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* [14, §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 authentication in PEREA-based schemes, Alice updates her current queue from Q' to $Q = Q'.\text{Enq}(t^*).\text{Deq}()$ 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:

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

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 Equation 9 and conducts the following protocol with the SP (ranges omitted):

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

which can in turn be constructed using standard protocols for proving relations among components of a DL 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.

3.5 Range Proof

Two types of range proofs are employed in our constructions. Proofs that a committed value lies in a given integer interval (“interval proof” for short) [14, §5.2], where the prover who knows a number x in an interval \mathcal{I} , convinces a verifier that x lies in a larger interval \mathcal{J} , is employed for a user to prove to an SP that his/her ticket $t \in \Pi_{\ell_t}$ belongs to the larger ticket domain \mathcal{T} . This proof is essential for security reasons since `QueueSig` is unforgeable if t lies within the intended message space \mathcal{T} . On the other hand, an exact range proof is not necessary since the goal is to guarantee the values for the SP to sign are within the designated range. Interval proof is based on the observation that the range of a valid response in the discrete-logarithm-based, zero-knowledge proof is related to the range of the witness given the space of the randomness and the challenge. If the randomness and the challenge space employed in the proof is carefully chosen, then a valid response falls within a designated range if and only if the witness is within a related interval.

Interval proof, while efficient, are insufficient in situations where a proof for an exact range is required. In PEREA-Naughtiness, a user is required to prove to an SP that his/her naughtiness is below a particular threshold. In this case, an exact range proof is necessary. We employ the signature-based range proof in [11] since it is particularly suitable for small intervals. In a nutshell, the verifier provides a set of “digital signatures” on the elements of the required range under a verification key. We consider this set of digital signatures as the public parameter. For the prover to demonstrate that a certain value committed in a commitment is within the range, the prover proves in zero-knowledge that he/she knows a signature under the verification key for the element committed. As the threshold of naughtiness in PEREA-Naughtiness is rather small, this proof, which has a constant complexity, is specifically useful. When instantiated using the signature scheme of [14], the protocol is secure under the Strong RSA Assumption. The factorization-based range proof in [8] can also be used, but is less efficient.

4 Construction of PEREA

We now provide a concise description of our construction of PEREA, making use of the building blocks presented in Section 3. We show how this construction is extended for PEREA- d -strikes-out and PEREA-Naughtiness in Section 5.

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 how many misbehaviors from a particular user an SP is willing to tolerate before the SP can blacklist that user. We expect K to be small, e.g., $K = 5$ or $K = 15$.

On input parameters param_{acc} and param_{sig} as defined in Section 3, the SP then generates a key pair $(\text{sk}_{acc}, \text{pk}_{acc})$ for `TicketAcc` and a key pair $(\text{sk}_{sig}, \text{pk}_{sig})$ 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 $\mathbf{serversk} = (\mathbf{sk}_{sig}, \mathbf{sk}_{acc})$, and then creates and publishes a server public key $\mathbf{serverpk} = (K, \mathbf{pk}_{sig}, \mathbf{pk}_{acc}, \hat{t})$.

Initially the SP's blacklist BL is empty, i.e., $\text{BL} = \emptyset$. The corresponding accumulator value is thus $\mathbf{V} = \mathbf{g}$. 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.¹⁴ The registration protocol goes as follows.

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

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

Next, she picks $r \in_R \Delta(\ell_N, \delta_r)$ and commits Q' as $C = \text{Commit}(Q', r)$. 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).

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

This protocol and hence its construction are similar to the one in Equation 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 creates a signature $\tilde{\sigma}$ on C and computes a non-membership witness \hat{w} for \hat{t} with respect to its current blacklist BL' , i.e., it executes $\tilde{\sigma} \leftarrow \text{Sign}(C, \mathbf{sk}_{sig})$ and $\hat{w} \leftarrow \text{ComputeWitness}(\hat{t}, \mathbf{V}', \mathbf{sk}_{acc}, \text{BL}')$, where $\mathbf{V}' = \text{Accumulate}(\text{BL}')$. The SP returns $(\tilde{\sigma}, \hat{w}, \text{BL}', \mathbf{V}')$ to Alice.
4. (*Credential finalizing.*) Alice computes $\sigma' \leftarrow \text{Finalize}(\tilde{\sigma}, r)$ and proceeds only if $\mathbf{V}' = \text{Accumulate}(\text{BL}')$, $1 = \text{Verify}(Q', \sigma')$ and $1 = \text{IsNonMember}(\hat{t}, \mathbf{V}', \hat{w})$. She stores her credential \mathbf{cred} as:

$$\mathbf{cred} \leftarrow (Q', \sigma', (w'_i)_{i=0}^{K-1}, \text{BL}', \mathbf{V}'),$$

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

¹⁴How 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.

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 certificate. We assume some out-of-band mechanism to obtain the correct public key of the SP. We hope to eliminate the requirement of a server-authenticated channel in the future but one must handle situations such as session hijacking (which could result in framing), relay attacks, and so on. Schemes such as EPID and BLAC also need to assume they can create a server-authenticated channel in their protocols and thus they need some mechanism to obtain the server’s public key as well.

4.3.1 Blacklist examination

Alice first obtains from the SP the current version of its blacklist 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 BL . She proceeds if she is *not* revoked. She drops the connection otherwise.

Denote by $\Delta_{\text{BL}} \doteq \text{BL} \setminus \text{BL}'$, where BL' is the SP’s blacklist she last saw and saved in her credential. Δ_{BL} is thus the set of newly blacklisted tickets.¹⁵

4.3.2 Request for authentication

Now that Alice knows that she has not been revoked and thus `WitnessUpdate` will return the updated witnesses for her queue, she requests to authenticate. Alice picks $t^* \in_R \Pi_{\ell_t}$ and $r \in_R \Delta(\ell_N, \delta_r)$, and computes

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

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

$$w_i \leftarrow \text{WitnessUpdate}(w'_i, Q'[i], V', \Delta_{\text{BL}}). \quad (19)$$

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

4.3.3 Proof of correctness

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

¹⁵We assume for now that SPs only add entries to their blacklists, so that $\text{BL}' \subseteq \text{BL}$ always. We address “unblacklisting” and blacklist manipulation in Section 9.

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

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, (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 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 DL representations of group elements [10].

4.3.4 Credential refreshing: server actions

The SP helps Alice refresh her credential as follows. The SP first creates a signature $\tilde{\sigma}$ on the commitment C , and computes a non-membership witness w_K for t_K with respect to \mathbf{V} , i.e., it executes $\tilde{\sigma} \leftarrow \text{Sign}(Q, \text{sk}_{sig})$ and $w_K \leftarrow \text{ComputeWitness}(t_K, \mathbf{V}, \text{sk}_{acc}, \text{BL})$. The SP then sends $(\tilde{\sigma}, w_K)$ to Alice.

4.3.5 Credential refreshing: user actions

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

$$\begin{array}{l} 1 \stackrel{?}{=} \text{Verify}(Q, \sigma) \\ 1 \stackrel{?}{=} \text{IsNonMember}(t_K, \mathbf{V}, w_K) \end{array}$$

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

$$\begin{array}{ll} (Q', \sigma') & \leftarrow (Q, \sigma) \\ (w'_0, w'_1, \dots, w'_{K-2}) & \leftarrow (w'_1, w'_2, \dots, w'_{K-1}), \\ w'_{K-1} & \leftarrow w_K \\ (\text{BL}', \mathbf{V}') & \leftarrow (\text{BL}, \mathbf{V}) \end{array}$$

4.3.6 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 $\text{BL} \leftarrow \text{BL} \cup \{t_K\}$ and the corresponding accumulated value as $\text{V} \leftarrow \text{AccumulatorAdd}(\text{V}, t_K)$.

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) [34], users remain anonymous and unlinkable (in an identity-escrow-free way) so long as they authenticate within the allowable rate, i.e., at most k times per time period. On the other hand, the SP can recognize and thus refuse connections made by a user who has exceeded that rate limit, because authentication attempts by the same user who has exceeded the rate limit are now linkable by the SP. The main drawback is that a new k -TAA token is required for each time period. Alternatively, one can use Periodic n -Times Anonymous Authentication [12]. The main difference with k -TAA is that now the token can be reused in the next time period anew. Consequently, the user is allowed to authenticate up to n times per time period without having to obtain a new credential in each time period.

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, then 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.

5 Supporting d -strikes-out and Naughtiness Policies

Extending PEREA to support d -strikes-out policies can be easily achieved by the partial proof of knowledge technique due to [19]. Briefly, in PEREA- d -strikes-out, the user computes, additionally, commitments of his/her K tickets, and proves to the SP in zero knowledge that at least d out of K of the commitments opens to a ticket that has not been accumulated. Additional work at the user's side is the construction of K commitments as well as the proof of their correctness, which requires $2K$ multi-based exponentiations (EXPs). The additional work at the SP side is the verification of the correctness of the K commitments, which requires K EXPs.

Extending PEREA- d -strikes-out to support naughtiness policies is non-trivial, and so we focus on the construction of PEREA-Naughtiness in this section. We compare the performance of PEREA- d -strikes-out and PEREA-Naughtiness in Section 8.2. Our extension does not alter the asymptotic time and communication complexities of the authentication protocol.

5.1 Blacklist examination

Alice first obtains from the SP the current version of its blacklist BL and the threshold n . Recall, an entry in the blacklist is now of the form $(t, \varsigma, \sigma_{t, \varsigma})$, where t is the ticket associated with the misbehavior, while ς is the corresponding severity and $\sigma_{t, \varsigma}$ is a signature on (t, ς) .

The SP additionally declares a threshold naughtiness value n where authentication succeeds only if the naughtiness of Alice is less than or equal to n . Alice then checks if she is revoked, and proceeds if she is *not* revoked. She drops the connection otherwise.

5.2 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 $r, r_1, \dots, r_K, \in_R \Delta(\ell_N, \delta_r)$, and then computes

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

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

$$C_i^{(s_i)} = \begin{cases} \text{Commit}(s_i, r_i), & \text{if } (Q'[i], s_i, \cdot) \in \text{BL}, \\ \text{Commit}(0, r_i), & \text{otherwise.} \end{cases} \quad (21)$$

$$w_i \leftarrow \text{UpdateWitness}(w'_i, Q'[i], V', \Delta_{\text{BL}}). \quad (22)$$

The superscript (s_i) in $C_i^{(s_i)}$ is 0 if $(Q'[i], \cdot, \cdot)$ is not in BL.

She sends $(t_K, C, C_0^{(s_0)}, \dots, C_{K-1}^{(s_{K-1})})$ to the SP. The SP proceeds only if t_K is fresh, i.e., $t_K \notin \text{TL}$, and is a prime in Π_{ℓ_t} . The SP then adds t_K to TL.

5.3 Proof of correctness

Alice (as the prover) conducts the following ZKPoK protocol with the SP (as the verifier):¹⁶

$$PK \left\{ \left(\begin{array}{l} Q', \sigma', t^*, Q, r, \\ (w_i, r_i, \sigma_{Q'[i], s_i}, s_i)_{i=0}^{K-1} \end{array} \right) : \left(\begin{array}{l} t_K = Q'[K] \wedge \\ 1 = \text{Verify}(Q', \sigma') \wedge \\ Q = Q'.\text{Enq}(t^*).\text{Deq}() \wedge \\ C = \text{Commit}(Q, r) \wedge \\ Q' \in \mathcal{Q} \wedge \tilde{t} \in \mathcal{T} \wedge r \in \mathcal{R} \wedge \\ \left(\begin{array}{l} 1 = \text{IsNonMember}(Q'[i], V, w_i) \wedge \\ C_i^{(s_i)} = \text{Commit}(0, r_i) \end{array} \right)_{i=0}^{K-1} \vee \\ \left(\begin{array}{l} 1 = \text{Verify}(\sigma_{Q'[i], s_i}, Q'[i], s_i) \wedge \\ C_i^{(s_i)} = \text{Commit}(s_i, r_i) \end{array} \right)_{i=0}^{K-1} \end{array} \right) \wedge \right\} \quad (23)$$

$$PK \left\{ (\xi, r^*) : \prod_{i=0}^{K-1} C_i^{(s_i)} = \text{Commit}(\xi, r^*) \wedge 0 \leq \xi \leq n \right\} \quad (24)$$

¹⁶We abuse the notation and use $\text{Verify}(\cdot, \cdot)$ to denote the verification equation for `QueueSig` as well as the signature used to sign the tickets and their corresponding severities in the blacklist.

The SP proceeds only if the ZKPoK verifies.

The above protocol aims to convince the SP that (1) $C_i^{(s_i)}$ is the severity of misbehavior of the connecting user’s past K connections (a connection that is not blacklisted has a severity of 0), (2) t_K is a well-formed ticket that the SP can later use to blacklist the user, (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, and (4) $\prod_{i=0}^{K-1} C_i^{(s_i)}$ is a commitment to the sum of the severities of the user’s past K connections, which is less than n .

6 Formal Security Definitions

We now give formal definitions of security for PEREA-Naughtiness (recall PEREA and PEREA- d -strikes-out are special cases), and then prove the security of our constructions in Section 7. We use a simulation-based approach to define the security notions.¹⁷ First we summarize the ideas of the model.

In the *real world* there are a number of players who communicate via cryptographic protocols. Then there is an adversary, \mathcal{A} , who controls the dishonest players in the system. We also have an environment, \mathcal{E} , that provides the inputs to the honest players and receives their outputs. \mathcal{E} also interacts freely with the adversary \mathcal{A} .

In the *ideal world*, we have the same players. However, they do not communicate directly. Rather, there exists a trusted party \mathcal{T} who is responsible for handling all inputs and outputs for all players. Specifically, \mathcal{T} computes the outputs of the players from their inputs by applying the functionality the cryptographic protocols are supposed to realize. The environment \mathcal{E} again provides the inputs to, and receives the outputs from, the honest players, and interacts arbitrarily with \mathcal{A} who controls the dishonest players.

Next, we describe the functionalities of PEREA-Naughtiness in the real world as well as the ideal world. We consider a static model in which the number of honest and dishonest users, together with SP, have been fixed earlier. We use \mathcal{U}_{HU} , \mathcal{U}_{AU} to denote the set of honest users and dishonest users respectively. Let $\mathcal{U}_U = \mathcal{U}_{HU} \cup \mathcal{U}_{AU}$ denote the set of all users. We assume all the dishonest parties are under the control of a single PPT algorithm \mathcal{A} . The environment, \mathcal{E} , provides the input to, and receives the outputs from, all the players (users and SP). \mathcal{E} can also interact with \mathcal{A} freely. The following functionalities are supported. We use the word *event* to denote the execution of a functionality. We remark that all communications with \mathcal{T} are *not* anonymous, meaning that \mathcal{T} knows the identity of the communicating party. It is also assumed that communication between honest parties is not observed by the real-world adversary and that when the real-world adversary receives a message, it does not learn the origin of the message. \mathcal{E} can freely schedule any of the events, with the restriction that the first event must be INIT and it can only be run once. The communicating players receive a unique transaction identifier \mathbf{tid} , generated by the environment \mathcal{E} . Players controlled by \mathcal{A} can deviate from the specification of the functionalities freely.

¹⁷Note: The definition we give does not entail all formalities necessary to fit into the universal composability (UC) framework [16]; our goal here is to prove the security of our construction. The UC framework allows proving the security of schemes *that remain secure when composed with other schemes*, which we do not attempt to prove here.

- $\text{INIT}(\text{tid}^{(\text{INIT})}, \mathcal{U}_{HU}, \mathcal{U}_{AU}, \mathbf{b}_{SP})$. The system begins when \mathcal{E} specifies the number of honest and dishonest users in the system. The bit \mathbf{b}_{SP} indicates whether the SP in the system is honest or not.

Real World The SP generates a key pair (spk, ssk) . The public key spk is made available to all players in the system.

Ideal World The trusted party \mathcal{T} initializes a set \mathcal{U} , which is used to record the registration status as well as the past K authentications of users in \mathcal{U}_U .

- $\text{REG}(\text{tid}^{(\text{REG})}, i)$. \mathcal{E} instructs a user $i \in \mathcal{U}_U$ to register with the SP. Note that this procedure is not anonymous in the view of the SP.

Real World User i sends a request for registration $\text{tid}^{(\text{REG})}$ to the SP. The user, as well as the SP, outputs individually the outcome of this transaction $(\text{tid}^{(\text{REG})}, \text{success/failure})$ to \mathcal{E} . If user i has obtained a credential in previous registration event, an honest SP would reject the request. Note that since this procedure is not anonymous in the view of the SP, the SP can identify duplicated requests from the same user. On the other hand, an honest user would discard the second credential it obtains from the SP if it has successfully registered in a previous registration event.

Ideal World User i sends a registration request, $\text{tid}^{(\text{REG})}$, to \mathcal{T} , \mathcal{T} sends $\text{tid}^{(\text{REG})}$ to the SP and informs him/her that user i would like to register for a credential, together with a bit to indicate if user i has successfully obtained a credential before. The SP returns (accept/reject) to \mathcal{T} , who forwards it back to the user. Note that an honest SP will always output reject if \mathcal{T} 's bit indicates user i has obtained a credential before. If the SP returns accept and no tuple (i, \cdot, \cdot) exists in the set \mathcal{U} , \mathcal{T} appends $(i, \text{tid}^{(\text{REG})}, \mathbf{Q}_i)$ to the set \mathcal{U} , where \mathbf{Q}_i is a queue of size K whose elements are all \perp initially. Otherwise, \mathcal{T} does not alter \mathcal{U} . The user, as well as the SP, output individually the outcome of this transaction $(\text{tid}^{(\text{REG})}, \text{success/failure})$ to \mathcal{E} .

- $\text{AUTH}(\text{tid}^{(\text{AUTH})}, i, n)$. \mathcal{E} instructs user $i \in \mathcal{U}_U$ to authenticate to the SP and instructs the SP to use a threshold n . (For PEREA, $n = 1$. For PEREA- d -strikes-out, $n = d$.)

Real World User i sends a request for authentication $\text{tid}^{(\text{AUTH})}$ to the SP. In the first step, the SP sends the current version of the blacklist BL to user i . The user, as well as the SP, output individually the outcome of this transaction $(\text{tid}^{(\text{AUTH})}, \text{success/failure})$ to \mathcal{E} .

Ideal World User i sends an authentication request $\text{tid}^{(\text{AUTH})}$ to \mathcal{T} , \mathcal{T} informs the SP that an anonymous user would like to authenticate, the corresponding transaction identifier being $\text{tid}^{(\text{AUTH})}$. The SP returns $\text{BL} = (\text{tid}_1^{(\text{AUTH})}, s_1), (\text{tid}_2^{(\text{AUTH})}, s_2), \dots$ to \mathcal{T} . \mathcal{T} checks his set \mathcal{U} and locates an entry $(i, \text{tid}^{(\text{REG})}, \mathbf{Q}_i)$. \mathcal{T} computes the naughtiness N of user i , where $N = \sum_{t=\text{tid}_j^{(\text{AUTH})} \wedge t \in \mathbf{Q}_i} s_j$. If no $(i, \text{tid}^{(\text{REG})}, \mathbf{Q}_i)$ exists (user i has not registered) or $N > n$ (user i does not satisfy the authentication policy), \mathcal{T} returns (BL, \perp) to user i and (BL, \top) otherwise. The user replies with a

bit to indicate if he/she chooses to proceed. The SP and user do not output anything if the user chooses not to proceed in this case. If user i chooses to proceed, \mathcal{T} sends a bit to the SP indicating whether the underlying authenticating user satisfies the authentication policy or not. The SP replies with **success/failure** to \mathcal{T} , and \mathcal{T} forwards the result to user i . If the SP replies with **success**, \mathcal{T} invokes $\mathbf{Q}_i.\text{Enq}(\text{tid}^{(\text{AUTH})}).\text{Deq}()$ if an entry $(i, \text{tid}^{\text{REG}}, \mathbf{Q}_i)$ exists in \mathcal{U} and does nothing otherwise. The user, as well as the SP, output individually the outcome of this transaction $(\text{tid}^{(\text{AUTH})}, \text{success/failure})$ to \mathcal{E} .

- $\text{REV}(\text{tid}^{(\text{REV})}, \text{tid}^{(\text{AUTH})}, s)$. \mathcal{E} instructs the SP to add the authentication identified in $\text{tid}^{(\text{AUTH})}$ to its blacklist with severity s . (For PEREA and PEREA- d -strikes-out, $s = 1$.)

Real World If $\text{tid}^{(\text{AUTH})}$ corresponds to an authentication event such that the SP outputs **success**, the SP obtains the corresponding ticket t . If (t, \cdot) does not exist in BL, the SP appends (t, s) to it. Otherwise, the request is ignored. The SP outputs $(\text{tid}^{(\text{REV})}, \text{success/failure})$ to \mathcal{E} upon completion of the algorithm to indicate whether (t, \cdot) is added to BL successfully.

Ideal World \mathcal{T} checks if $\text{tid}^{(\text{AUTH})}$ corresponds to an AUTH event such that the SP outputs **success**. If yes, it checks if there is already an event $\text{REV}(\text{tid}^{(\text{AUTH})}, \cdot)$ where SP returns **accept**. \mathcal{T} informs the SP of the results of the check and forwards $(\text{tid}^{(\text{AUTH})}, s)$ to the SP. The SP returns **accept/reject** to \mathcal{T} to indicate if the request is accepted. The SP also outputs $(\text{tid}^{(\text{REV})}, \text{success/failure})$ to \mathcal{E} , indicating if the ticket is added successfully or not.

Ideal world PEREA-Naughtiness provides all the desired security properties. Firstly, all the transactions, in the view of the SP, are anonymous. \mathcal{T} only informs the SP some anonymous user would like to authenticate and thus anonymity and backward unlinkability is guaranteed. Secondly, \mathcal{T} verifies if the authenticating user has registered successfully and checks if the naughtiness of the user is below the threshold specified by the SP and thus misauthentication resistance and revocability are assured. Finally, \mathcal{T} informs the user if he/she satisfies the authentication policy before proceeding to the authentication and thus revocation auditability is achieved.

We now formally define the security of PEREA-Naughtiness through comparison of the behavior of PEREA-Naughtiness with the ideal world specification. PEREA-Naughtiness is secure if for every real world adversary, \mathcal{A} , and every environment, \mathcal{E} , there exists an ideal world adversary, \mathcal{S} , controlling the same players in the ideal world as \mathcal{A} does in the real world such that, \mathcal{E} cannot tell whether it is running in the real world interacting with \mathcal{A} or it is running in the ideal world interacting with \mathcal{S} , which has black-box access to \mathcal{A} as stated in Definition 1.

Let λ be a security parameter. We say a function $\text{negl}(\lambda)$ is a negligible function in λ , if for all polynomials $f(\lambda)$ and for all sufficiently large λ , $\text{negl}(\lambda) < 1/f(\lambda)$. If A and B are functions of λ , we say $A(\lambda)$ is non-negligibly larger (in λ) than a $B(\lambda)$ if $A - B$ is not negligible as a function of λ .

Definition 1 (Security) Let $\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda)$ (resp. $\mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)$) be the probability that \mathcal{E} outputs 1 when run in the real world (resp. ideal world) with adversary \mathcal{A} (resp. \mathcal{S} having black-box access to \mathcal{A}). PEREA-Naughtiness is secure if for all PPT algorithms \mathcal{E} , \mathcal{A} , the following expression holds:

$$|\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)| = \text{negl}(\lambda)$$

7 Proofs of Security

We analyze the security of our construction for PEREA-Naughtiness (presented in Section 5) since PEREA and PEREA- d -strikes-out are special cases of this construction. PEREA-Naughtiness has misauthentication resistance, revocability, anonymity, backward unlinkability and revocation auditability. We state the following theorem first and then sketch its proof.

Theorem 1 *Our construction for PEREA-Naughtiness is secure under the Strong RSA Assumption and the DDH Assumption over quadratic residues modulo a safe-prime product.*

□

The security of our PEREA-Naughtiness construction is analyzed by proving indistinguishability between an adversary’s actions in the real world and the ideal world. The idea of the proof is that given a real world adversary \mathcal{A} , we show how to construct an ideal world adversary \mathcal{S} such that no PPT environment \mathcal{E} can distinguish whether it is interacting with \mathcal{A} or \mathcal{S} . The proof is divided into two cases according to the subset of players controlled by \mathcal{A} . In the first case, \mathcal{A} controls the SP and a subset of users while in the second case, only a subset of users is dishonest. The first case covers the security requirements identified in Section 2.1 of *anonymity*, *backward unlinkability* and *revocation auditability*, while the second case covers the properties of *misauthentication resistance* and *revocability*. For instance, the properties of *anonymity* and *backward unlinkability* should continue to hold in the presence of a malicious SP.

We prove the theorem by proving the following lemmas handling the relevant combinations of parties controlled by the adversary.

lemma 1 *For all PPT environments \mathcal{E} and all real world adversaries \mathcal{A} controlling a subset of users, there exists an ideal world simulator \mathcal{S} such that*

$$|\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)| = \text{negl}(\lambda).$$

Proof We define a simulator \mathcal{S} which interacts with \mathcal{E} as an adversary, and at the same time having black-box access to a real world adversary \mathcal{A} . \mathcal{S} represents \mathcal{E} , the honest SP and honest users to \mathcal{A} . Note that \mathcal{S} will be maintaining a list of ‘current’ credentials issued to \mathcal{A} during the lifespan of the system. On the other hand, \mathcal{S} acts as an ideal world adversary to the trusted party \mathcal{T} . We restate the goal of \mathcal{S} here: for any PPT \mathcal{A} and \mathcal{E} , \mathcal{S} ’s goal is to provide a view to \mathcal{E} such that \mathcal{E} cannot distinguish whether it is interacting with \mathcal{S} or \mathcal{A} . Firstly, \mathcal{S} simply forwards any messages between \mathcal{E} and \mathcal{A} . Next, we specify how \mathcal{S} responds to each possible event.

- $\text{INIT}(\text{tid}^{(\text{INIT})}, \mathcal{U}_{HU}, \mathcal{U}_{AU}, 0)$. The system begins when \mathcal{E} specifies the number of honest and dishonest users in the system. The bit 0 indicates that the SP in the system is honest.

Representing Honest SP to \mathcal{A} \mathcal{S} generates a key pair (spk, ssk) . The public key spk is given to \mathcal{A} .

Representing Dishonest Users to \mathcal{T} In this event, \mathcal{T} gives no output to \mathcal{S} .

- $\text{REG}(\text{tid}^{(\text{REG})}, i)$. \mathcal{E} instructs a user $i \in \mathcal{U}_U$ to register with the SP.

Representing honest user i to the honest SP If $i \in \mathcal{U}_{HU}$, \mathcal{S} does not need to do anything.

Representing dishonest user i to \mathcal{T} / the honest SP to \mathcal{A} Upon receiving $\text{tid}^{(\text{REG})}$ from \mathcal{A} on behalf of user i such that $i \in \mathcal{U}_{AU}$, \mathcal{S} rejects if user i has obtained a credential from previous transactions. Otherwise, \mathcal{S} extracts from \mathcal{A} the values (Q', r) in Equation 18. If the extraction fails, \mathcal{S} aborts. Otherwise, \mathcal{S} sends $\text{tid}^{(\text{REG})}$ for a registration request to \mathcal{T} on behalf of user i . Upon receiving accept from \mathcal{T} , \mathcal{S} issues a credential cred to \mathcal{A} according to the protocol. Since \mathcal{S} has extracted (Q', r) , it can compute the finalized credential cred as well. \mathcal{S} stores cred_i as the current credential of user i . \mathcal{S} forwards the protocol outcome $(\text{tid}^{(\text{REG})}, \text{success/failure})$ from \mathcal{A} on behalf of user i to \mathcal{E} .

- $\text{AUTH}(\text{tid}^{(\text{AUTH})}, i, n)$. \mathcal{E} instructs user $i \in \mathcal{U}_U$ to authenticate to the SP and instructs the SP to use a threshold n .

Representing honest user i to the honest SP If $i \in \mathcal{U}_{HU}$, \mathcal{S} does not need to do anything.

Representing dishonest user i to \mathcal{T} / the honest SP to \mathcal{A} If $i \in \mathcal{U}_{AU}$, \mathcal{S} does not know which credential \mathcal{A} is using for the authentication. For instance, while \mathcal{E} specifies user i to perform the authentication, it is entirely possible for \mathcal{A} to use the credential from another dishonest user say, \hat{i} , to perform the authentication. Nonetheless, upon receiving $\text{tid}^{(\text{AUTH})}$ from \mathcal{A} , \mathcal{S} submits $\text{tid}^{(\text{AUTH})}$ to \mathcal{T} on behalf of user i . Upon receiving the reply from \mathcal{T} ($\text{BL}, \perp / \top$), \mathcal{S} reconstructs the blacklist based on $\text{BL} = \{(\text{tid}_1^{(\text{AUTH})}, s_1), (\text{tid}_2^{(\text{AUTH})}, s_2), \dots\}$. Specifically, for any $(\text{tid}_i^{(\text{AUTH})}, s_i) \in \text{BL}$ such that $\text{tid}^{(\text{AUTH})}$ corresponds to an authentication event of a dishonest user, \mathcal{S} appends (t_i, s_i) to the re-constructed blacklist BL' , where t_i is the ticket during the authentication identified by $\text{tid}^{(\text{AUTH})}$. For any $\text{tid}_i^{(\text{AUTH})}$ involving an honest user, \mathcal{S} appends (t_i, s_i) to BL' where t_i is a random value generated from the ticket domain. \mathcal{S} sends the reconstructed blacklist BL' as the blacklist to \mathcal{A} . If \mathcal{A} chooses not to proceed, \mathcal{S} replies to \mathcal{T} on behalf of user i that he/she will not proceed. If \mathcal{A} chooses to proceed, then \mathcal{S} has to locate the ‘actual’ user index, \hat{i} , for which \mathcal{A} is using, so as to perform the corresponding action on behalf of user \hat{i} to \mathcal{T} . To achieve this, \mathcal{S} extracts the values $(Q', \sigma', (w_i, r_i, s_i)_{i=0}^{K-1}, t^*, Q, r, \xi, r^*)$ in Equation 23 and 24 from \mathcal{A} and uses them to locate the index \hat{i} of the user. Note that this requires \mathcal{S} to store the credentials

cred of all users and searching for a credential \mathbf{cred}_i that contains σ' . If the extraction fails, or \mathcal{S} cannot locate the index \hat{i} , \mathcal{S} aborts. Otherwise, \mathcal{S} sends $\mathbf{tid}^{(\text{AUTH})}$ to \mathcal{T} for an authentication request on behalf of the dishonest user \hat{i} . \mathcal{S} forwards the protocol outcome $(\mathbf{tid}^{(\text{AUTH})}, \mathbf{success/failure})$ from \mathcal{A} on behalf of user i to \mathcal{E} . Denote by **FAIL** the event that the output of an honest SP interacting with \mathcal{A} is different to the outcome of the honest SP interacting with \mathcal{S} through \mathcal{T} in the ideal world.

- $\text{REV}(\mathbf{tid}^{(\text{REV})}, \mathbf{tid}^{(\text{AUTH})}, s)$. \mathcal{E} instructs the SP to add the authentication identified in $\mathbf{tid}^{(\text{AUTH})}$ to its blacklist with severity s . \mathcal{S} does not need to do anything in this event.

If \mathcal{S} does not abort and the event **FAIL** never occurs, the output of the honest users and the SP, as well as the dishonest users represented by \mathcal{S} in the ideal world, is the same as those output by the honest users and the SP, together with the dishonest users represented by \mathcal{A} in the real world. That is,

$$\mathbf{Real}_{\mathcal{E}, \mathcal{A}}(\lambda) = \mathbf{Ideal}_{\mathcal{E}, \mathcal{S}}(\lambda)$$

It remains to be shown that the probability that \mathcal{S} aborts or the event **FAIL** occurs is negligible. The following are all the possible cases when \mathcal{S} aborts or event **FAIL** occurs. Recall that the security parameter λ consists of the following lengths $(\ell_N, \ell_s, \ell_e, \ell_T, \ell, \delta_r)$. Suppose there are q_R and q_A registration and authentication events. Further suppose the maximum advantage in breaking the soundness of the ZKPoK protocol is $\mathbf{Adv}^{PK}(\lambda)$ for any PPT adversary. In our construction, we take $\{0, 1\}^\ell$ as the challenge space for our ZKPoK protocols and $\mathbf{Adv}^{PK}(\ell) = 2^{-\ell}$.

1. During a **REG** event from \mathcal{A} , \mathcal{S} fails to extract from \mathcal{A} the tuple (Q', r) in the proof of knowledge in Equation 18. This happens with probability $\mathbf{Adv}^{PK}(\ell)$. Probability that none of these extractions fail in all q_R **REG** events is bounded by $q_R \cdot 2^{-\ell}$.
2. During a successful **AUTH** event from \mathcal{A} , \mathcal{S} fails to extract from \mathcal{A} the values $(Q', \sigma', (w_i, r_i, \varsigma_i)_{i=0}^{K-1}, t^*, Q, r, \xi, r^*)$ in the proof of knowledge in Equation 23 and 24. This happens with probability $\mathbf{Adv}^{PK}(\ell)$. Likewise, probability that none of these extractions fail in all q_A **AUTH** events is bounded by $q_A \cdot 2^{-\ell}$.
3. Let **FORGE** denote the event that there exists a successful **AUTH** event from \mathcal{A} such that the extracted values $(Q', \sigma', (w_i, r_i, \varsigma_i)_{i=0}^{K-1}, t^*, Q, r, \xi, r^*)$ involve a queue Q' that has never appeared as an extracted value in a previous **REG** event as (Q', \cdot) or **AUTH** event as $(\cdot, \cdot, (\cdot, \cdot, \cdot)_{i=0}^{K-1}, \cdot, Q', \cdot, \cdot, \cdot)$. This happens the case when \mathcal{A} has forged a signature σ' on the queue Q' (which also includes the case when \mathcal{A} has been able to open the committed value in two different ways). The simulator \mathcal{S} could setup a reduction to break the unforgeability of the CL signature if event **FORGE** happens. Specifically, \mathcal{S} is given the signing oracle of the CL signature, and shall use the oracle to responds to the **REG** and **AUTH** events. If event **FORGE** happens, \mathcal{S} obtains a new signature on message that has not been submitted to the signing oracle. Thus, the probability of success is equivalent to the probability that **FORGE** happens. \mathcal{S} has made at most $q_R + q_A$ queries to the signing oracle and thus event **FORGE** happens

with probability at most $\mathbf{Adv}^{CL}(q_R + q_A, K + 1, \lambda)$, where $\mathbf{Adv}^{CL}(q, K + 1, \lambda)$ denotes the maximum advantage of any PPT adversary in forging a CL signature after making q signature queries on a block of $K + 1$ messages. It has been proven in [31] $\mathbf{Adv}^{CL}(q_R + q_A, K + 1, \lambda) \leq (2(K + 1))(8(q_R + q_A)) \cdot \mathbf{Adv}^{SRSA}(\ell_N)$ where $\mathbf{Adv}^{SRSA}(\ell_N)$ denotes the maximum advantage of any PPT algorithm in solving the strong RSA problem of modulus N of length ℓ_N . Thus, event FORGE happens with probability less than $(16(K + 1)(q_R + q_A)) \cdot \mathbf{Adv}^{SRSA}(\ell_N)$.

4. During a successful AUTH event from \mathcal{A} , the extracted values $(Q', \sigma', (w_i, r_i, \varsigma_i)_{i=0}^{K-1}, t^*, Q, r, \xi, r^*)$ do not satisfy the relationship: $0 \leq \xi \leq n$; or for $i = 0$ to $K - 1$, neither of the following two relationships is satisfied:

$$1 = \text{IsNonMember}(Q'[i], V, w_i)$$

or the relationship:

$$1 = \text{Verify}(\sigma_{t, \varsigma_i}, Q'[i], \varsigma_i).$$

In this case event FAIL occurs. This represents the case when \mathcal{A} has been able to do any of the following: (1) fake the signature-based range proof of ξ described in Section 3.5 in Equation 24, (2) generate a non-membership witness of a member, or (3) existentially forge a signature σ_{t, ς_i} on a tuple $(Q'[i], \varsigma_i)$ chosen by \mathcal{A} . Case (1) happens with probability $\mathbf{Adv}^{CL}(n_{max}, \lambda)$, where $\mathbf{Adv}^{CL}(q, \lambda)$ denotes the maximum advantage of any PPT adversary in forging a CL signature after making q signature queries of messages. It has been proven in [14] that $\mathbf{Adv}^{CL}(q, \lambda) \leq (8q) \cdot \mathbf{Adv}^{SRSA}(\ell_N)$. Thus, Case (1) happens with probability less than $8(n_{max}) \cdot \mathbf{Adv}^{SRSA}(\ell_N)$, where n_{max} is the maximum threshold value for naughtiness. Case (2) happens with probability $\mathbf{Adv}^{Acc}(\ell_N)$, where $\mathbf{Adv}^{Acc}(\ell_N)$ denotes the maximum advantage of any PPT adversary in breaking the security of the universal accumulator (creating a non-membership witness for a member). It has been shown in [29] $\mathbf{Adv}^{Acc}(\ell_N) \leq \mathbf{Adv}^{SRSA}(\ell_N)$. Case (3) happens with probability $\mathbf{Adv}^{CL}(q_A, 2, \lambda) \leq (32q_A) \cdot \mathbf{Adv}^{SRSA}(\ell_N)$. Note that the value q_A appears because it is the upper bound of $|\text{BL}|$. Thus, event FAIL happens with probability less than $[8(n_{max}) + 1 + 32q_A] \cdot \mathbf{Adv}^{SRSA}(\ell_N)$.

5. During an AUTH event, the re-constructed blacklist of entries (t_i, s_i) involving an honest user, t_i appears in some other AUTH event. This represents the case the random value t_i generated by \mathcal{S} collides with some tickets that has been used before. This happens with negligible probability if the ticket domain is large enough and the probability is bounded by $|\text{BL}|2^{-\ell_t}$ where $|\text{BL}|$ is the size of that AUTH event. Since tickets can only be added to the blacklist in PEREA-based schemes and each ticket corresponds to an authentication event, an upper bound for this in all AUTH events is $q_A 2^{-\ell_t}$, assuming \mathcal{S} uses the same random entry for the same tid .

Summing up the probability for the above cases gives us the desired bound:

$$(q_R + q_A) \cdot 2^{-\ell} + q_A \cdot 2^{-\ell_t} + \{16(K + 1)(q_R + q_A) + 8n_{max} + 1 + 32q_A\} \mathbf{Adv}^{SRSA}(\ell_N)$$

which is negligible under the Strong RSA assumption for large enough security parameters under the condition that the naughtiness threshold is polynomial in the security parameter.

Thus, the simulator \mathcal{S} aborts or the event FAIL occurs with negligible probability. This completes our proof of Lemma 1.

□

lemma 2 *For all PPT environments \mathcal{E} and all real world adversaries \mathcal{A} controlling the SP and a subset of users, there exists an ideal world simulator \mathcal{S} such that*

$$|\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)| = \text{negl}(\lambda).$$

Proof We define a simulator \mathcal{S} which interacts with \mathcal{E} as an adversary, and at the same time having black-box access to a real world adversary \mathcal{A} . \mathcal{S} represents \mathcal{E} and the honest users to \mathcal{A} . On the other hand, \mathcal{S} acts as an ideal world adversary to the trusted party \mathcal{T} . We re-state the goal of \mathcal{S} here: for any PPT \mathcal{A} and \mathcal{E} , \mathcal{S} 's goal is to provide the view to \mathcal{E} such that \mathcal{E} cannot distinguish whether it is interacting with \mathcal{S} or \mathcal{A} . Firstly, \mathcal{S} simply forwards any messages between \mathcal{E} and \mathcal{A} . Next, we specify how \mathcal{S} responds to each possible event.

- $\text{INIT}(\text{tid}^{(\text{INIT})}, \mathcal{U}_{HU}, \mathcal{U}_{AU}, 1)$. The system begins when \mathcal{E} specifies the number of honest and dishonest users in the system. The bit 1 indicates that the SP in the system is controlled by \mathcal{A} .

Representing Honest Users to \mathcal{A} \mathcal{S} receives the public key spk from \mathcal{A} .

Representing Dishonest SP to \mathcal{T} In this event, \mathcal{T} gives no output to \mathcal{S} .

- $\text{REG}(\text{tid}^{(\text{REG})}, i)$. \mathcal{E} instructs a user $i \in \mathcal{U}_U$ to register with the SP.

Representing dishonest user i to the dishonest SP If $i \in \mathcal{U}_{AU}$, \mathcal{S} acts on behalf of the dishonest user and the dishonest SP to \mathcal{T} . After that, \mathcal{S} forwards the protocol outcome from \mathcal{A} on behalf of user i as well as SP to \mathcal{E} . Note that the protocol outcome from \mathcal{A} is possibly different from the supposed outcome following the protocol specification. Nonetheless, \mathcal{S} would use \mathcal{A} 's outcome in case there is a difference.

Representing the dishonest SP to \mathcal{T} / the honest user i to \mathcal{A} Upon receiving a registration request $(\text{tid}^{(\text{REG})})$ from \mathcal{T} on behalf of user i such that $i \in \mathcal{U}_{HU}$, \mathcal{S} engages \mathcal{A} in the registration protocol, using the zero-knowledge simulator to simulate the ZKPoK in Equation 18. If \mathcal{S} fails to obtain a valid credential from \mathcal{A} , \mathcal{S} replies `reject` to \mathcal{T} upon receiving $(\text{tid}^{(\text{REG})}, i)$ from \mathcal{T} . \mathcal{S} forwards the protocol outcome $(\text{tid}^{(\text{REG})}, \text{success/failure})$ from \mathcal{A} on behalf of the dishonest SP to \mathcal{E} .

- $\text{AUTH}(\text{tid}^{(\text{AUTH})}, i, n)$. \mathcal{E} instructs user $i \in \mathcal{U}_U$ to authenticate to the SP and instructs the SP to use a threshold n .

Representing dishonest user i to the dishonest SP If $i \in \mathcal{U}_{AU}$, \mathcal{S} acts on behalf of the dishonest user and the dishonest SP to \mathcal{T} . After that, \mathcal{S} forwards the protocol outcome from \mathcal{A} on behalf of user i as well as the SP to \mathcal{E} . Note that the protocol outcome from \mathcal{A} is possibly different to the supposed outcome following the protocol specification. Nonetheless, \mathcal{S} would use \mathcal{A} 's outcome in case there is a difference.

Representing the dishonest SP to \mathcal{T} / the honest user to \mathcal{A} Upon receiving an authentication request $(\mathbf{tid}^{(AUTH)})$ from \mathcal{T} on behalf of an anonymous user such that $i \in \mathcal{U}_{HU}$, \mathcal{S} engages \mathcal{A} in the authentication protocol and receives from \mathcal{A} a blacklist $\mathbf{BL} = \{(t_1, s_1), (t_2, s_2), \dots\}$. For all $(t, \cdot) \in \mathbf{BL}$, \mathcal{S} appends $(\mathbf{tid}^{(AUTH)}, s)$ to a re-constructed blacklist \mathbf{BL}' , where t is the ticket used in the authentication identified by $\mathbf{tid}^{(AUTH)}$. Upon receiving the authentication request on behalf of an honest user from \mathcal{T} , \mathcal{S} replies to \mathcal{T} with the blacklist \mathbf{BL}' . Note that \mathbf{BL}' is different from \mathbf{BL} but it would not affect the authentication result for the anonymous user i because \mathbf{BL}' contains all entries from honest users' past authentications. If \mathcal{T} replies with a bit indicating that the underlying user would proceed and satisfies the authentication policy, \mathcal{S} uses the zero-knowledge simulator to simulate the ZKPoK in Equations 23 and 24 with a random ticket t . If \mathcal{A} rejects the authentication, \mathcal{S} replies `reject` to \mathcal{T} . \mathcal{S} forwards the protocol outcome $(\mathbf{tid}^{(AUTH)}, \text{success/failure})$ from \mathcal{A} on behalf of the dishonest SP to \mathcal{E} .

- $\text{REV}(\mathbf{tid}^{(REV)}, \mathbf{tid}^{(AUTH)}, s)$. \mathcal{E} instructs the SP to add the authentication identified in $\mathbf{tid}^{(AUTH)}$ to its blacklist with severity s . \mathcal{S} simply forwards this to \mathcal{A} and forwards the protocol outcomes from \mathcal{A} back to \mathcal{E} on behalf of the dishonest SP.

The simulation to \mathcal{A} is perfect due to the statistical zero-knowledgeness of the ZKPoK protocols. Specifically, given the current choice of security parameter λ such that $\delta_r = \lfloor \frac{\ell_N - 1}{\epsilon} - \ell \rfloor$ for some $\epsilon > 1$, the statistical zero-knowledgeness of the ZKPoK protocol we employed is bounded by $\frac{2}{2^{(\ell_N + \ell)(\epsilon - 1)}}$ for each response in the proof, which is bounded by $q_A \cdot 27K \cdot \frac{1}{2^{(\ell_N + \ell)(\epsilon - 1) - 1}}$ and is negligible for sufficiently large security parameter ℓ_N, ℓ and suitably chosen ϵ .

Thus, the output of the honest users, as well as the dishonest SP and users represented by \mathcal{S} in the ideal world is the same as those output by the honest users, together with the dishonest SP and users represented by \mathcal{A} in the real world. That is,

$$\mathbf{Real}_{\mathcal{E}, \mathcal{A}}(\lambda) = \mathbf{Ideal}_{\mathcal{E}, \mathcal{S}}(\lambda).$$

This completes the proof of Lemma 2. □

7.1 Concrete Choice of Parameter

Given the reduction, we could now suggest concrete choice of parameters.

Our goal is to provide a guarantee of security equivalent to 80-bit, 112-bit and 128-bit respectively. We made the following assumptions.

- We assume $\mathbf{Adv}^{SRSA}(\ell_N) = 2^{-118}$, 2^{-150} and 2^{-166} when $\ell_N = 2734$, 4592 and 5749 respectively.
- We assume $q_R = 2^{20}$, $q_A = 2^{30}$, $|\mathbf{BL}| < q_A$, $K = 10$, $n_{max} = 2^{12}$.
- Details of the tightness of reduction in BLAC is not given. Nonetheless, the key building block, BBS+ signature, is secure if the q -SDH assumption holds where q is the number of signatures obtained by the adversary. A closer look in the security proof of BBS+ signature reveals that it suffers from a security loss of $1/q$. Namely, the maximum advantage in forging a BBS+ signature, after making q queries, is bounded by $O(q) \cdot Adv^{qSDH}(\ell_p)$, where $\mathbf{Adv}^{qSDH}(\ell_p)$ denotes the maximum advantage of any PPT algorithm that solves the q -SDH assumption in group of order p of length ℓ_p . In terms of BLAC, q represents the number of successful registration queries. Thus, a rough estimate states that BLAC is secure if $q_R \cdot \mathbf{Adv}^{qSDH}(\ell_p)$ is negligible.
- We assume $\mathbf{Adv}^{qSDH}(\ell_p) = 2^{-100}$, 2^{-132} , and 2^{-148} when $\ell_p = 186$, 249 , and 281 respectively.

Table 1 summarizes the choice of parameters for different levels of security.

Security Level	Schemes	
	PEREA-based schemes $(\ell_N, \ell_s, \ell_e, \ell_T, \ell_t, \ell, \delta_r)$	BLAC (ℓ_p)
80-bit	$(2734, 3224, 333, 330, 166, 160, 2324)$	(186)
112-bit	$(4592, 5274, 461, 458, 230, 224, 3949)$	(249)
128-bit	$(5749, 6527, 525, 522, 262, 256, 4969)$	(281)

Table 1: Concrete Security Parameter and their Security Levels taking security loss into account

8 Performance Evaluation

We now evaluate the performance of PEREA-based schemes in comparison with BLAC both analytically and quantitatively.

8.1 Complexity analysis

Table 2 summarizes the asymptotic complexities of PEREA-based schemes and BLAC/EPID. The asymptotic complexity is unchanged for all PEREA-based schemes. In both PEREA-based schemes 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 PEREA-based schemes users can compute witnesses efficiently in $O(\Delta_L)$, where Δ_L is the size of $\Delta_{\mathbf{BL}} = \mathbf{BL} \setminus \mathbf{BL}'$, i.e., the difference between the current blacklist and the previously observed blacklist (we discuss timing attacks in Section 9).

During an authentication, PEREA-based schemes require only $O(K)$ computation at the server, as compared to $O(L)$ in BLAC/EPID. This computation at the SP is the main

Schemes	Authentication Efficiency					
	Communication		Computation			
	Downlink	Uplink	User (Check+Prove)	Server		
Accumulator-based*	$O(L)$	$O(1)$	$O(L)$	+	$O(\Delta_L)$	$O(1)$
BLAC/EPID	$O(L)$	$O(L)$	$O(L)$	+	$O(L)$	$O(L)$
PEREA-based schemes (this paper)	$O(L)$	$O(K)$	$O(L)$	+	$O(K\Delta_L)$	$O(K)$

Table 2: Comparison of Communication and Computation Complexities. PEREA-based schemes provide enhanced privacy like BLAC and EPID do, but the server’s computation does not grow with the blacklist size; *We use the term accumulator-based schemes to denote schemes in which unrevoked users’ pseudonyms are placed in an dynamic accumulator. We remark that PEREA-based schemes are less efficient than accumulator-based schemes, but they have better privacy, since accumulators do not support privacy-enhanced revocation like the other schemes listed here.

Schemes	Computation	
	User (Check+Prove)	Server
BLAC	$7E1 + 2ET + (2L + 1)EG + 1P$	$4E1 + 2ET + (L + 1)EG + 2P$
PEREA	$[(K + 1)\Delta_L]EN1 + [5K + 2\lceil \frac{K+1}{3} \rceil + \lceil \frac{K-1}{3} \rceil + 3]EN2$	$[4K + 2\lceil \frac{K+1}{3} \rceil + 3]EN2$
PEREA- d -strikes-out	$[(K + 1)\Delta_L]EN1 + [7K + 2\lceil \frac{K+1}{3} \rceil + \lceil \frac{K-1}{3} \rceil + 3]EN2$	$[5K + 2\lceil \frac{K+1}{3} \rceil + 3]EN2$
PEREA-Naughtiness	$[(A + 1)\Delta_L]EN1 + [16K + \lceil \frac{K-1}{3} \rceil + 12]EN2$	$[15K + \lceil \frac{K}{3} \rceil + 8]EN2$

Table 3: Quantitative Comparison of Computational Complexities of BLAC, PEREA, PEREA- d -strikes-out, and PEREA-Naughtiness

bottleneck in the system, and is therefore the most relevant metric for comparing the schemes. In all schemes, the computational complexity at the user to check (via simple bit-string comparison) if the user has been blacklisted is the same — $O(L)$ time. Generating the proofs takes $O(L)$ time in BLAC/EPID, and $O(K\Delta_L)$ in PEREA as each of the K witnesses must be updated Δ_L times.

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

Downlink communication is linear in the size of the blacklist in all schemes, $O(L)$. The uplink communication complexities are the same as the computational complexities at the server: $O(K)$ for PEREA-based schemes, $O(L)$ for BLAC/EPID.

8.2 Quantitative analysis

Table 3 outlines the number of multi-based exponentiations (EXPs) in PEREA-based schemes, where we assume each EXP multiplies up to three bases. Thus, computing $\prod_{i=1}^K \mathbf{g}_i^{e_i}$ requires $\lceil K/3 \rceil$ EXPs. At the user’s side, an authentication requires $(K + 1)\Delta_L + 5K + 2 \cdot \lceil \frac{K+1}{3} \rceil + \lceil \frac{K-1}{3} \rceil + 3$ and $(A + 1)\Delta_L + 16K + \lceil \frac{K-1}{3} \rceil + 12$ EXPs respectively for PEREA and PEREA-Naughtiness, where A is the number of tickets that do not appear in the blacklist (for our analysis we assume the worst case, i.e., $A = K$). At the SP’s side, the figures are $4K + 2 \cdot \lceil \frac{K+1}{3} \rceil + 3$ and $15K + \lceil \frac{K}{3} \rceil + 8$ EXPs, respectively.

Benchmark of the various operations given the choice of security parameters are given

below.

Security Level	EN1	EN2	E1	ET	P	EG
80-bit ($\ell_N = 2736, \ell_p = 186$)	11.54	20.58	3.04	6.76	22	same as E1
112-bit ($\ell_N = 4593, \ell_p = 249$)	40.64	89.63	6.77	13.24	42.61	same as E1
128-bit ($\ell_N = 5749, \ell_p = 281$)	69.96	169.97	6.99	15.12	50.46	same as E1

Table 4: Benchmark of Operations Given the Required Security levels (in ms). Due to the nature of curve generation, one cannot specify the group order. Rather, a random curve is generated and the number of points (group order) is computed afterwards. Thus, in the above, $\ell_p = 185, 247$ and 289 is used for 80-bit, 112-bit and 128-bit security respectively. An element in the group can be represented by $\ell_p + 1$ -bit at the optimal case when the group order and the base field of the curve is of the same size. As for ℓ_N , after generation of two $\ell_N/2$ -bit primes, their product is not always of ℓ_N bits and thus in our experiment $\ell_N = 2376, 4593$ and 5749 respectively.

8.2.1 Benchmarking various operations

The constants E1, ET, and EG are the times of multi-exponentiations in $\mathbb{G}_1, \mathbb{G}_T$, and \mathbb{G} , respectively, and P is that of pairing operations, assuming a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. \mathbb{G} is an arbitrary group such that the DDH Problem is hard in \mathbb{G} . The benchmarks shown in Table 4 are obtained on a Lenovo X200s with an Intel Core 2 Duo CPU L9400 and 4GB RAM running Windows Vista as the host. We used Oracle VirtualBox 4.0.4 to emulate a guest machine of 512MB RAM running Ubuntu 10.10. Timings of E1, ET, EG, and P are obtained using test code based on the Pairing-Based Cryptography (PBC) library¹⁸ (version 0.5.11) based on type D pairing. \mathbb{G} could be an arbitrary group. For simplicity, we assume $\mathbb{G} = \mathbb{G}_1$, so that $EG = E1$. This implicitly assumes the DDH problem is hard in group \mathbb{G}_1 and is known as the XDH assumption. EN1 is the exponentiation of a random base with relatively small exponent modulo N and EN2 is the multi-exponentiation of fixed bases with full range exponent modulo N . EN1 and EN2 are the major operations in PEREA-based schemes in which EN1 is mainly used in the update of non-membership witnesses while EN2 is used in the normal operations for proof generation and verification. The test code for EN1 and EN2 is written based on the MIRACL library¹⁹ (version 5.4.2) and runs directly on the Windows platform. The bases in EN2 are assumed to be known in advance and pre-processing based on the known bases has been done, using a storage of 35.875 MB.

8.2.2 Analysis of computation

For our quantitative analysis we use a security level of 112 bits for a fair comparison between BLAC and PEREA-based schemes. Figure 2(a) shows the estimated authentication time at the SP for BLAC and PEREA-based schemes for various blacklist sizes L . The time required for authentication increases linearly for BLAC and remains constant for PEREA-based schemes. For PEREA, when $K = 5$, the SP takes 2.4 sec per authentication on

¹⁸<http://crypto.stanford.edu/pbc/>

¹⁹<http://www.shamus.ie/>

average; when $K = 10$, it takes 4.4 sec. For PEREA-Naughtiness, the SP takes 7.5 sec per authentication when $K = 5$ and 14.4 sec when $K = 10$. Contrast these numbers with 67.8 sec for BLAC when $L = 10,000$ and grows to 135.5 sec when $L = 20,000$. We note that if the simpler d -strikes-out policy is implemented instead of the naughtiness policy, the computational cost will be reduced as no zero-knowledge proof-of-knowledge of the SP’s signature on each severity is required. For only d -strikes-out policies, therefore, the SP takes 3.4 sec per authentication when $K = 5$ and 6.5 sec when $K = 10$. Figure 2(b) illustrates the authentication times at the SP for PEREA-based schemes as K increases.

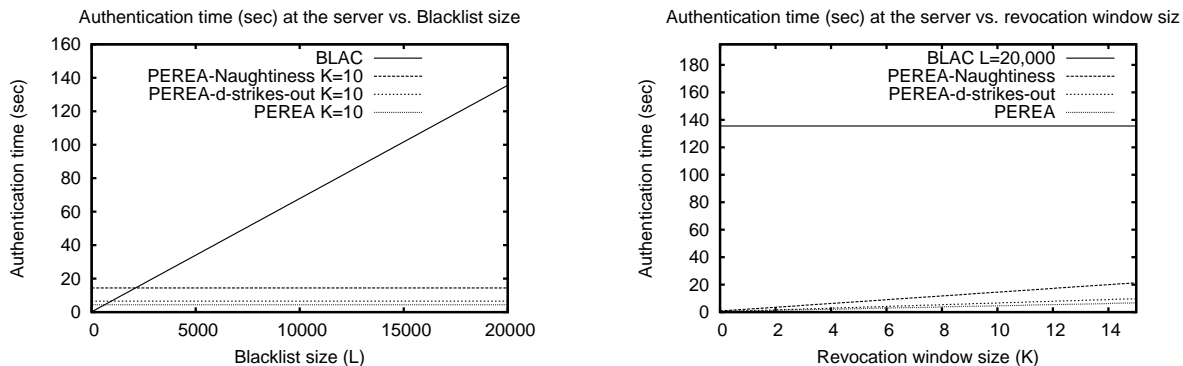
Figure 2(c) shows the authentication time at the user. In PEREA-based schemes, when a new entry is added to the blacklist, users must refresh their credential (`WitnessUpdate`) to take that accumulator update into account. If users wait to refresh their credential until their next authentication, a large amount of time will be spent processing all the updates. The ‘(No Updates)’ curve corresponds to this situation. In practice we expect a mode of operation where users refresh their credentials once a day or once a week for example (note, these updates can be downloaded anonymously so as not to reveal when the user may have authenticated last). Thus during an actual authentication we expect very few witness updates will be needed to accommodate blacklist updates since the last refreshing. We show results for “(2% New)”, which corresponds to the situation where only 2% of the entries on the blacklist are new since the last refreshing. BLAC is more efficient where no updates are performed, but PEREA-Naughtiness is slightly better when only 2% of the entries have changed.

For BLAC, authentication times at the user for $L = 10,000$ and $L = 20,000$ are 135.5 sec and 270.9 sec respectively. The values for all PEREA-based schemes are similar so we present the values for PEREA-Naughtiness at the user. For $L = 10,000$, with no prior witness updates, authentication at the user takes 2446.7 sec for $K = 5$ and 4486.1 sec for $K = 10$. For $L = 20,000$, authentication at the user takes 4885.1 sec for $K = 5$ and 8956.5 sec for $K = 10$. Now, if only 400 users have been blacklisted since the user last authenticated, then for $K = 5$, authentication at the user takes only 105.9 sec for $K = 5$ and 194.5 sec for $K = 10$, which are both better than BLAC. BLAC/EPID and PEREA-based schemes in general therefore require the user to wait for ‘a couple of minutes’ before an authenticated action takes effect at the SP. The user can be offline during the period when the SP completes the authentication. Note that if entries are removed from the blacklist (SPs might forgive certain misbehaviors) periodically, users must compute their witnesses for the entire blacklist getting worst case performance at the next credential refresh after a forgiveness cycle.

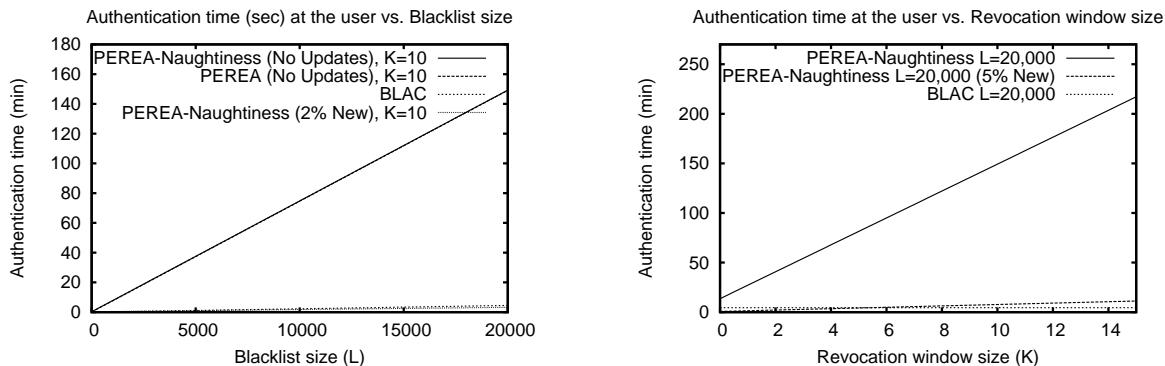
Figure 2(d) illustrates the authentication times at the SP for PEREA-based schemes as K increases and shows the performance is the same for all PEREA-based schemes.

8.2.3 Analysis of communication

For downlink communication, in PEREA and PEREA- d -strikes-out each entry is 230 bits, as compared to 472 bits (taking $\mathbb{G} = \mathbb{G}_1$) in BLAC and 10569 bits for PEREA-Naughtiness. For example, for a blacklist of size 10,000, users would need to download 280.76 KB in PEREA, 576.17 KB in BLAC, and 12.6 MB in PEREA-Naughtiness. For a blacklist of size 20,000, users would need to download 561.52 KB in PEREA, 1.125 MB in BLAC, and 25.2 MB



(a) Authentication time at the SP vs. Blacklist size. (b) Authentication time at the SP vs. Revocation window size.



(c) Authentication time at the user vs. Blacklist size. Note the curves for PEREA-Naughtiness and PEREA are nearly identical. Since PEREA-*d*-strikes-out has performance “in between” the two, we omit that curve for clarity. (d) Authentication time at the user vs. Revocation window size.

Figure 2: PEREA-based schemes are more efficient with authentication at the SP, and outperform BLAC. Authentication at the user is slower than BLAC, but comparable to BLAC when users periodically refresh their credentials, and e.g., only 5% of the blacklist entries have changed since the last refreshing.

in PEREA-Naughtiness. As mentioned earlier, the time taken for all the witness updates can be amortized, and we expect a client side process that updates blacklists periodically, downloading only the new portion of the blacklist since the daily update. For example, if only 400 new entries have been added to the blacklist since the last update, the size of download for PEREA-Naughtiness is only 504 KB.

One reason for the larger download in PEREA-Naughtiness is that the extended black-

list now contains the ticket of the misbehavior, its severity, and the SP’s signature on each severity. Thus the functionality of *naughtiness* policies comes with the overhead of larger blacklists. In today’s web, we feel that downloading a blacklist of a few MB for an authentication (such as anonymous edits to Wikipedia pages) is within reason, although we hope to reduce such overheads in future work. We would like to remark that, if a simple d -strikes-out policy is to be implemented, the communication overhead of the extension is roughly the same as that of PEREA, as each misbehavior is now of the same severity and it is not necessary to include the SP’s signature to bind the ticket of misbehavior and its severity. Commitments of the severities of each ticket in the user’s queue is also unnecessary, since we would simply construct a threshold proof, using the technique in [19], to demonstrate fewer than d tickets in the user’s queue are on the server’s blacklist.

9 Discussion

Timing attacks. Our protocol includes an optimization in which users need to perform computation only for the new entries in the blacklist (See Equation 19). An SP therefore could link users’ connections if they were 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 . Based on our performance analysis, we believe setting K between 5–15 represents a reasonable tradeoff between authentication latency for the user, and the size of the revocation window. For $K = 10$, if it takes a site like Wikipedia an hour ($T = 60$) 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. Whatever the time window for recognizing misbehaviors, we argue websites would not want more than 10 or so misbehaviors before a user is caught anyway. If SPs want higher authentication rates ($\frac{K}{T}$), it is more prudent to shorten the time T it takes to recognize misbehaviors rather than increase K , which allows more misbehaviors before revocation and also comes with decreased performance.

Concurrency. The instantiation of ZKPoK protocol in Equation 20 is *not* secure against concurrent attacks [23] and must be executed sequentially. Our presented constructions therefore contain 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. The time spent in this critical section is dominated by network latency (the computation at the user in the critical section is estimated at less than 1 ms based on our experimentation). The SP can include a timeout for unresponsive clients to maintain a high authentication throughput. While it is possible to utilize a generic UC-secure ZKPoK to eliminate this constraint, the construction would be impractical. A possible denial of service attack would be for a user or a set of users to keep trying to authenticate and reduce authentication throughput because of this critical section. We point out that users must first authenticate via a rate-limiting scheme first, and so individual users

will be limited in this attack. Furthermore, this requires users to be legitimate, enrolled users in the system. Nevertheless, a group of malicious enrolled users could slow down the authentication rate. In cases where the SP realizes it is under DoS attack, it can issue client puzzles of increasing complexity to users before entering the critical section to mitigate such attacks. Nevertheless, we hope to eliminate this critical section in the future.

Gaming the blacklist. Since SPs can construct arbitrary blacklists at any time (by adding and removing entries), an SP can game the system by presenting crafted versions of the blacklist to authenticating users. For example, an SP may try to link two transactions by presenting different versions of the blacklist with and without these entries and compare authentication rates. Since the blacklist is publicly available, we assume such behavior will be noticed by users. In particular, users can refuse to authenticate if they notice a particular entry disappearing and then reappearing. This scenario would signal a gaming attack. An honest SP is expected to add entries to a blacklist, possibly remove entries, and never add removed entries again. In any case, these attacks are hard to perform on PEREA-based schemes (as compared to BLAC/EPID) because with revocation windows of $K = 5$ or $K = 15$, SPs will not be able collect enough statistical data about two previous authentications after the user has authenticated that many times.

Forgiving misbehaviors. To forgive the blacklisted user who provided ticket t_K , the SP removes t_K from BL and updates the current accumulator value from \mathbf{V} to $\mathbf{V}^{1/t_K \bmod \phi(\mathbf{N})} \bmod \mathbf{N}$. If such “unblacklisting” is allowed in PEREA-based schemes, however, users must update their witnesses in $O(L)$ time (rather than the optimized $O(\Delta_L)$ time) during an authentication (Equation 19), even if only one ticket has been removed since their last authentication.

10 Summary

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*. We extend PEREA to support more complex revocation policies such as the *d-strikes-out* policy (PEREA-*d-strikes-out*) and a weighted version called the *naughtiness* policy (PEREA-Naughtiness). Through analytical and experimental validation, we show that for realistic parameters, PEREA-based schemes provide more efficient authentication at the server than existing schemes.

11 Acknowledgments

Patrick P. Tsang passed away on October 27, 2009 as a victim to cancer. We dedicate this paper to his memory. We thank Sean W. Smith for his helpful comments and earlier support for this project. This work was supported in part by the Institute for Security Technology Studies (BJA 2005-DD-BX-1091) and NSF (CNS-0524695). The views and conclusions are our own.

We thank our anonymous reviewers for their extensive and helpful feedback.

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 *Lecture Notes in Computer Science*, 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 *Lecture Notes in Computer Science*, pages 183–197. Springer, 2002.
- [3] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In M. K. Franklin, editor, *Financial Cryptography*, volume 1648 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1999.
- [4] N. Bari and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
- [5] D. Boneh. The decision diffie-hellman problem. In J. Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 1998.
- [6] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, 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 P. Ning and T. Yu, editors, *WPES*, pages 21–30. ACM, 2007.
- [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, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.
- [12] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *ACM Conference on Computer and Communications Security*, pages 201–210. ACM, 2006.
- [13] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2002.

- [14] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.
- [15] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In Kaliski, Burton S. Jr [27], pages 410–424.
- [16] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
- [17] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
- [18] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
- [19] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
- [20] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation Onion router. In *Usenix Security Symposium*, pages 303–320, Aug. 2004.
- [21] 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.
- [22] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Kaliski, Burton S. Jr [27], pages 16–30.
- [23] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.
- [24] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
- [25] R. Henry, K. Henry, and I. Goldberg. Making a Nymble Nymble using VERBS. In M. J. Atallah and N. J. Hopper, editors, *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2010.
- [26] P. C. Johnson, A. Kapadia, P. P. Tsang, and S. W. Smith. Nymble: Anonymous IP-address blocking. In *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 113–133. Springer, 2007.
- [27] Kaliski, Burton S. Jr, editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997.
- [28] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589. Springer, 2004.

- [29] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2007.
- [30] Z. Lin and N. Hopper. Jack: Scalable accumulator-based Nymble system. In *WPES '10: Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 53–62, New York, NY, USA, 2010. ACM.
- [31] A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Massachusetts Institute of Technology, 2002.
- [32] L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.
- [33] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.
- [34] I. Teranishi, J. Furukawa, and K. Sako. k -times anonymous authentication (extended abstract). In *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2004.
- [35] 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.
- [36] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. PEREA: Towards practical TTP-free revocation in anonymous authentication. In *Proceedings of The 15th ACM Conference on Computer and Communications Security (CCS)*, pages 333–344. ACM, Oct. 2008.
- [37] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. BLAC: Revoking repeatedly misbehaving anonymous users without relying on TTPs. *ACM Transactions on Information and System Security*, 13:39:1–39:33, December 2010.
- [38] P. P. Tsang, A. Kapadia, C. Cornelius, and S. W. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Transactions on Dependable and Secure Computing*, 8(2):256–269, March–April 2011.