Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs (Extended Version)*†

Patrick P. Tsang‡, Man Ho Au§, Apu Kapadia‡¶, and Sean W. Smith‡

Dartmouth Computer Science
Technical Report TR2007-601

September 30th 2007

Abstract

Several credential systems have been proposed in which users can authenticate to services anonymously. Since anonymity can give users the license to misbehave, some variants allow the selective deanonymization (or linking) of misbehaving users upon a complaint to a trusted third party (TTP). The ability of the TTP to revoke a user's privacy at any time, however, is too strong a punishment for misbehavior. To limit the scope of deanonymization, systems such as "e-cash" have been proposed in which users are deanonymized under only certain types of well-defined misbehavior such as "double spending." While useful in some applications, it is not possible to generalize such techniques to more subjective definitions of misbehavior.

We present the first anonymous credential system in which services can "blacklist" misbehaving users without contacting a TTP. Since blacklisted users remain anonymous, misbehaviors can be judged subjectively without users fearing arbitrary deanonymization by a TTP.

---

*This paper is the extended version of the paper to appear in CCS '07 under the same title [TAKS07].

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

‡Department of Computer Science, Dartmouth College, Hanover, NH, USA

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

¶Institute for Security Technology Studies, Dartmouth College, Hanover, NH, USA

Contents

1 Introduction

Several cryptographic schemes allow users to authenticate to *service providers (SPs)* anonymously. While anonymous authentication offers users a high degree of privacy, it can give users the license to misbehave without the fear of punishment. For example, Wikipedia[1] has allowed editors to modify content anonymously, and as a result several users have misbehaved by posting inappropriate content. SPs, therefore, desire some level of accountability against misbehaving users. Several anonymous credential systems have been proposed in which users can be selectively deanonymized or have their accesses linked (pseudonymized) under special circumstances. As we will discuss, for certain applications the existing schemes are either too punitive—deanonymization (or linking) is unreasonably harsh, and often relies on *trusted third parties (TTPs)* capable of revoking a user's privacy at any time—or too restrictive—allowing deanonymization under only certain narrowly defined types of misbehavior.

Deanonymizing a user is not always necessary to discourage misbehavior; in some cases it is sufficient to simply block misbehaving users from making future accesses, while maintaining their anonymity. We call this property *anonymous blacklisting*. For example, anonymous access at SPs such as Wikipedia and YouTube[2] empowers users to disseminate content without the fear of persecution—a user may add political content on Wikipedia that is forbidden by his or her government, or post a video of police brutality to YouTube. In such cases, while Wikipedia and YouTube may want to penalize users who deface webpages or post copyrighted material, it is of paramount importance for SPs to preserve the anonymity of their well-behaving users. By guaranteeing anonymity to *all* users, anonymous blacklisting allows SPs to penalize misbehavior without the risk of exposing legitimate users such as political dissenters. We now discuss why existing solutions are not desirable for such applications.

Anonymous credential systems that support accountability (such as Camenisch and Lysyanskaya's [CL01, CL04] and schemes based on group signatures [CvH91, ACJT00, BBS04, KY05]) feature a TTP called the *Open Authority (OA)*. The OA is capable of identifying (or linking) the user behind any anonymous authentication. Anonymous credential systems with dynamic membership revocation [CL02a, AST02, BS04, Ngu05], mostly constructed from dynamic accumulators [CL02a], also feature a TTP that is capable of deanonymizing (or linking) users. The existence of such a TTP, however, is undesirable—users can never be assured that their privacy will be maintained by the TTP. Defining the circumstances under which a TTP can expose a user, and ensuring its trustworthiness to judge fairly, is an undue burden on SPs. For such applications, therefore, *a system without TTPs is desirable*.

To eliminate the reliance on TTPs, certain "threshold-based" approaches such as e-cash [ACS05, CHL05, CHL06] and *k-Times Anonymous Authentication (k-TAA)* [TFS04, NSN05, TS06, ASM06] have been proposed. In these schemes, users are guaranteed anonymity unless they authenticate more than a certain number of threshold times. For example, spending an e-coin twice (an undesirable action) or authenticating $k+1$ times in a k-TAA scheme, provides the SP with enough information to compute the user's identity. Linkable ring signatures [LWW04, TW05, TWC+04] and periodic n-times anonymous authentication [CHK+06] also fall into this category. Unfortunately, misbehavior cannot always be defined in terms of threshold values. For example, "inappropriate" edits to a Wikipedia page, or "offensive" video uploads to YouTube are usually identified based on human subjectivity. For such applications, therefore, *subjective judging is desirable*.

[1] `http://www.wikipedia.org`
[2] `http://www.youtube.com`

1.1 Related Solutions

To reiterate, it is important to have an anonymous credential system in which users can be blacklisted in a way that (1) preserves their anonymity, (2) is based on subjective definitions of misbehavior, and (3) does not rely on a TTP. Though not intended for anonymous blacklisting, Syverson et al. present a scheme [SSG97] that ensures that users can perform anonymous and serial transactions at an SP. The SP issues blind tokens to users, which are renewed at the end of a user's transaction. The SP can block future connections from a user by simply not issuing a new token at the end of a transaction (e.g., if the user fails to pay for continued service). The major drawback to this approach is that misbehavior must be judged while the user is online. Indeed, their scheme was not designed for blacklisting users since misbehavior is usually identified long after a user has disconnected. Recently, some of the authors of this paper proposed the Nymble system [JKTS07] to allow SPs to block misbehaving users hiding behind an anonymizing network such as Tor [DMS04]. Nymble makes several practical considerations for anonymous IP-address blocking based on subjective judging, but it does rely on multiple entities that can collude to deanonymize (or link) a misbehaving user.

Even though it may seem that the ability to block future accesses from subjectively-judged misbehaving users inherently requires a TTP capable of deanonymizing (or linking) users, we show that this is not the case.

1.2 Our Contributions

We propose the *BLacklistable Anonymous Credential* (BLAC) system, the first cryptographic construction of an anonymous credential system that supports anonymous blacklisting and subjective judging without relying on TTPs that are capable of revoking the privacy of users at will. We formalize the security model for such a system and prove that our construction is secure under this model. Furthermore, we provide an implementation of our BLAC system and evaluate its performance both analytically and experimentally.

Paper Outline We provide a high-level overview of our BLAC system in Section 2 and formalize the model and security properties in Section 3. In Section 4 we present preliminary information on the various cryptographic tools and assumptions used in our construction, which we present in Section 5. We present an experimental evaluation of our construction in Section 6, a discussion of several issues in Section 7, and finally conclude in Section 8.

2 Our Approach

We provide a high-level overview of our *BLacklistable Anonymous Credential (*BLAC*)* system in this section, and defer cryptographic details to the subsequent sections.

In our system, *users* authenticate to *Service Providers (SPs)* anonymously using *credentials* issued by a *Group Manager (GM)*. The GM is responsible for *enrolling* legitimate users into the system by issuing credentials to them.[3] These credentials are private to the user, and not known by the GM. We emphasize that the GM is *not* a TTP that can compromise the privacy of users, and is trusted only to enroll legitimate users into the system, and issue at most one credential per user. SPs are willing to serve anonymous users as long as they are legitimate users in the system (by enrolling themselves with the GM), and have never misbehaved thus far, where misbehavior

[3] Who is a legitimate user and how to verify such legitimacy are application-dependent.

may be arbitrarily defined and subjectively judged by each individual SP. We describe this process next.

The novelty of our approach is that SPs maintain their own *blacklists* of misbehaving users without knowing the identity of the misbehaving users. Users anonymously authenticating to the SP must first prove that they are *not* on the SP's blacklist (otherwise authentication will fail). Following a user's authentication, SPs store a *ticket* extracted from the *protocol transcript* of the authentication and if the user is later deemed to have misbehaved during the *authenticated session*, possibly long after the user has disconnected, the SP can add the ticket as an entry in its blacklist.[4] If a user Alice detects that she is on the blacklist, she terminates the authentication and disconnects immediately. The SP, therefore, learns only that some anonymous blacklisted user was refused a connection, i.e., the SP does not learn the identity of the blacklisted user, and the user is anonymous within the set of blacklisted users. Users not on the blacklist will be able to authenticate successfully, and the SPs learn only that the user is not on the blacklist. Furthermore, our system allows SPs to *remove* entries from the blacklist, thereby forgiving past misbehaviors.[5] Depending on the severity of misbehavior, a user may be blacklisted for varying periods of time—using inappropriate language could correspond to being blacklisted for one week, whereas posting copyrighted material could correspond to blacklisting for one month. Users are always assured that if they successfully authenticate to an SP their access will always remain anonymous—all that an SP can do is block future accesses by a misbehaving user.

A Glimpse into Tickets Tickets are a vital object in our BLAC system. A ticket is the only piece in the authentication protocol transcript that contains information about the identity of the authenticating user. Here we describe some of the properties these tickets must possess for the system to be secure. First, tickets have to be the output of some non-invertible mapping of the user's credential. If this were not the case, the system would have no anonymity. Also, tickets from the same user should be unlinkable, for otherwise SPs would be able to tell if two authentications are from the same user. This property implies that the mapping just mentioned must also take as input some randomness, as a deterministic mapping implies linkability. Furthermore, tickets must be such that it is possible to prove and verify that a ticket is correctly formed, and that a ticket does not belong to a given user (if it is indeed the case). Without such a property, a user blacklisted by an SP would still be able to authenticate to the SP. As we will see, this property is also a necessary condition to prevent misbehaving users from being "framed" (by other users for example).

Remark. Our BLAC system may be configured to allow or disallow the sharing of blacklist entries (tickets) between SPs. Sharing a blacklist entry would allow multiple SPs to block a user who misbehaved at one of the SPs. We will first present the system where such sharing is *disallowed* and then point out how to allow sharing in Section 7.

3 Model

We present the syntax of the Blacklistable Anonymous Credential (BLAC) system, followed by security properties that any construction of the BLAC system must satisfy.

[4] In practice, the SP may privately log arbitrary information about an authenticated session that is necessary for it to judge at a later time whether the anonymous user misbehaved during that session.

[5] Adding and removing blacklist entries are atomic actions as will be discussed in Section 7.

3.1 Syntax

The entities in the BLAC system are the Group Manager (GM), a set of Service Providers (SPs) and a set of users. The BLAC system consists of the following protocols:

3.1.1 Setup

This algorithm is executed by the GM to set up the system. On input of one or more security parameters, the algorithm outputs a pair consisting of a group public key *gpk* and a group private key *gsk*. The GM publishes *gpk* and keeps *gsk* private.

3.1.2 Registration

This protocol is executed between the GM and a legitimate user to register the user into the system. Upon successful completion of the protocol, the user obtains a credential *cred*, which she keeps private to herself, and is thereby enrolled as a member in the group of registered users.

3.1.3 Authentication

This protocol is executed between a user with credential *cred* and an SP. When an execution of the protocol terminates, the SP outputs a binary value of `success` or `failure`. If the SP outputs `success` in an execution of the protocol, we call the execution a successful authentication and say that the authenticating user has succeeded in authenticating herself; otherwise the authentication is unsuccessful and the user has failed. Only upon a successful authentication does the SP establish an authenticated session with the authenticating user during which the user can access the service provided by the SP. Note that the *protocol transcript* of a successful authentication as seen by the SP is useful for the SP to blacklist the authenticating user, as described next.

3.1.4 Blacklist Management

This is a suite of three algorithms: Extract, Add and Remove, which are executed by SPs for managing their blacklists. On input of an authentication protocol transcript, Extract extracts and returns a *ticket* from the transcript. A *blacklist* is a collection of tickets. On input of a blacklist and a ticket, Add returns a new blacklist that contains all the tickets in the input blacklist as well as the input ticket. On the other hand, on input of a blacklist and a ticket, Remove returns a new blacklist that contains all the tickets in the input blacklist, except the one(s) equivalent to the input ticket.[6]

When we say that a user Alice is *blacklisted* by an SP Bob, we mean that there exists an authentication between Alice and Bob such that Bob has added the ticket extracted from the authentication transcript to his blacklist and has not removed it (yet). Otherwise Alice is *not blacklisted* by Bob. Also, we say that Alice is *misbehaving* with respect to Bob if she is blacklisted by Bob. Otherwise, she is *well-behaving*.

Correctness Any construction of the BLAC system must be correct:

Definition 1 (Correctness) A construction of the BLAC system is correct if all entities in the system are honest (i.e., they follow the system's specification) implies that for any registered legitimate user Alice and for any SP Bob, Alice is able to successfully authenticate herself to Bob with overwhelming probability if Alice is not blacklisted by Bob during the authentication.

[6] We don't define the equivalence of tickets here because it is construction-dependent.

3.2 Security Notions

We now give informal definitions of the various security properties that a construction of the BLAC system must possess. Their formal definition will be given in the next subsection.

3.2.1 Mis-authentication Resistance

Mis-authentication occurs when an unregistered user successfully authenticates herself to an SP. In a BLAC system with *mis-authentication resistance*, SPs are assured to accept authentication only from registered users.

3.2.2 Blacklistability

Any SP Bob may blacklist a user, who has authenticated successfully, at any later time. As a consequence, the blacklisted user will no longer be able to successfully authenticate herself to Bob until the user is unblacklisted by Bob. In a BLAC system with *blacklistability*, SPs are assured to accept authentication only from well-behaving users, i.e., users who are not blacklisted.

3.2.3 Anonymity

In a system with *anonymity*, all that SPs can infer about the identity of an authenticating user is whether the user is or was blacklisted at the time of protocol execution, regardless of whatever the SPs do afterwards, such as arbitrarily manipulating their blacklists.

3.2.4 Non-frameability

A user Alice is framed if she is not currently blacklisted by an honest SP Bob, but is unable to successfully authenticate herself to Bob. In a BLAC system with *non-frameability*, well-behaving users can always successfully authenticate themselves to honest SPs.

Security Any construction of the BLAC system must be secure:

Definition 2 (Security) A construction of the BLAC system is secure if it has mis-authentication resistance, blacklistability, anonymity and non-frameability.

3.3 Formal Definitions

We use a game-based approach to define the security formally. The adversary's capabilities are modeled by arbitrary and adaptive queries to oracles, which are stateful and together share a private state denoted by state. state contains three counters m, n and a, which are initialized to 0, and six sets $\mathcal{U}_P, \mathcal{U}_A, \mathcal{U}_B, \mathcal{S}_P, \mathcal{S}_A, \mathcal{A}_A$, which are initialized to $\emptyset$. The oracles are described as follows.

- P-Reg. This oracle allows the adversary to register an honest user with the honest GM. Upon invocation, the oracle increments n by 1, simulates the registration protocol between an honest user and the honest GM, sets $\mathsf{state} := \mathsf{state}||\langle n, trans_n, cred_n\rangle$, where $trans_n$ is the resulting protocol transcript and $cred_n$ is the resulting user credential, adds n to $\mathcal{U}_P$ and returns $(trans_n, n)$ to the adversary. The user is indexed by n.

- A-Reg. This oracle allows the adversary to register a corrupt user with the honest GM. Upon invocation, the oracle increments n by 1, plays the role of the GM and interacts with

the adversary in the registration protocol, sets $\mathsf{state} := \mathsf{state}||\langle n, trans_n, \perp\rangle$, where $trans_n$ is the protocol transcript, adds n to $\mathcal{U}_A$ and returns n to the adversary. The user is indexed by n.

- B-Reg. This oracle allows the adversary to register an honest user with the corrupt GM. Upon invocation, the oracle increments n by 1, plays the role of a user and interacts with the adversary in the registration protocol, sets $\mathsf{state} := \mathsf{state}||\langle n, \perp, cred_n\rangle$, where $cred_n$ is the credential issued to the user by the adversary, adds n to $\mathcal{U}_B$ and returns n to the adversary. The user is indexed by n.

- Corrupt-U(i). This oracle allows the adversary to corrupt an honest user. On input i, the oracle removes i from $\mathcal{U}_B$ or $\mathcal{U}_P$, adds i to $\mathcal{U}_A$, and returns $cred_i$ to the adversary.

- Add-SP(ID). This oracle allows the adversary to introduce an SP with identity $ID \in \{0,1\}^*$ into the system. Upon invocation, the oracle increments m by 1, adds it to $\mathcal{S}_P$, and returns it to the adversary. The SP is indexed by m.

- Corrupt-SP(j). This oracle allows the adversary to corrupt an honest SP. On input j, the oracle removes j from $\mathcal{S}_P$ and adds it to $\mathcal{S}_A$.

- P-Auth(i, j). This oracle allows the adversary to eavesdrop an authentication run between an honest user and an honest SP. On input (i, j) such that $i \in \mathcal{U}_P \cup \mathcal{U}_B$ and $j \in \mathcal{S}_P$, the oracle increments a by 1, simulates (using $cred_i$) the authentication protocol between honest user i and honest SP j, sets $\mathsf{state} := \mathsf{state}||\langle \pi_a, a\rangle$, where π_a is the resulting protocol transcript, and returns (π_a, a) to the adversary.

- A-Auth(j). This oracle allows a corrupt user to be authenticated by an honest SP. On input $j \in \mathcal{S}_P$, the oracle increments a by 1, plays the role of SP j and interacts with the adversary in the authentication protocol, adds a to $\mathcal{A}_A$, sets $\mathsf{state} := \mathsf{state}||\langle \pi_a, a\rangle$, where π_a is the resulting protocol transcript, and returns a to the adversary.

- B-Auth(i, j). This oracle allows a corrupt SP to authenticate an honest user. On input $i \in \mathcal{U}_B \cup \mathcal{U}_P$ and $j \in \mathcal{S}_A$, the oracle increments a by 1, plays the role of user i to be authenticated by SP j and interacts with the advesary in the authentication protocol, sets $\mathsf{state} := \mathsf{state}||\langle \pi_a, a\rangle$, where π_a is the resulting protocol transcript, and returns a to the adversary.

- Add-To-BL(j, k). This oracle allows the adversary to influence an honest SP to think that an authenticated session involves a misbehavior. On input $j \in \mathcal{S}_P$ and $k \leq a$, the oracle adds the ticket $\tau_k = \text{Extract}(\pi_k)$ to SP j's blacklist.

- Remove-From-BL(j, τ). This oracle allows the adversary to influence an honest SP to think that an authenticated session does not involve a misbehavior. On input $j \in \mathcal{S}_p$ and τ such that τ is in SP j's blacklist, the oracle removes τ from that blacklist.

We remark that queries to P-Reg and A-Reg do not interleave because the honest GM registers user one at a time; queries to Add-To-BL($j, \cdot$) and Remove-From-BL($j, \cdot$) do not interleave with one another, or with queries to P-Auth or A-Auth because honest SPs update their blacklists only when no authentication is in progress. Queries to P-Auth is atomic, but we allow interleaving among queries to P-Auth, A-Auth and B-Auth.

3.3.1 Mis-authentication Resistance and Blacklistability

Mis-authentication resistance is in fact implied by Blacklistability: if someone can authenticate to an SP without having registered, she can authenticate after being blacklisted by mounting an attack against mis-authentication resistance. The following game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ formally defines Blacklistability.

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

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

End Game Phase. $\mathcal{A}$ outputs $j \in \mathcal{S}_P$. $\mathcal{A}$ wins the game if both of the following are true:

- There exists a sub-sequence S of the sequence of all oracle queries in the same order as they were made, where $S =$

$$\begin{aligned} \langle a_1 &:= \text{A-Auth}(j), \quad \text{Add-To-BL}(j, a_1), \\ a_2 &:= \text{A-Auth}(j), \quad \text{Add-To-BL}(j, a_2), \\ &\vdots \\ a_k &:= \text{A-Auth}(j), \quad \text{Add-To-BL}(j, a_k), \\ a_{k+1} &:= \text{A-Auth}(j)\rangle \end{aligned}$$

  such that $a_x \in \mathcal{A}_A$ for all $x = 1$ to k, and all these $k+1$ A-Auth queries are successful.
- $k \geq |\mathcal{U}_A| + Q_R$, where Q_R is the number of Remove-From-BL(j, Extract(π_{a_i})) queries such that $i \in \{1, \ldots, k\}$.

3.3.2 Anonymity

The following game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ formally defines anonymity.

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

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

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

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

End Game Phase. $\mathcal{A}$ outputs a guess bit b'. $\mathcal{A}$ wins the game if $b = b'$ and at least one of the following is true:

- *(Case I.)* For any Add-To-BL(j, k) queried by $\mathcal{A}$ during the Probing Phase such that π_k is an authentication transcript from user i_0 or i_1, $\mathcal{A}$ later queried, during the same phase, Remove-From-BL(j, τ) where $\tau =$ Extract(π_k). In Probing Phase 2, $\mathcal{A}$ never queried Add-To-BL(j, k) such that π_k is an authentication transcript from user i_0 or i_1.

- *(Case II.)* There exist two queries ADD-TO-BL(j, k_0) and ADD-TO-BL(j, k_1), in Probing Phase, such that π_{k_0} and π_{k_1} are authentication transcripts from user i_0 and i_1 respectively and $\mathcal{A}$ did not later make queries to, during the two probing phases, REMOVE-FROM-BL$(j, \text{Extract}(\pi_{k_0}))$ or REMOVE-FROM-BL$(j, \text{Extract}(\pi_{k_1}))$.

3.3.3 Non-frameability

The follow game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ formally defines Non-frameability.

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

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

End Game Phase. $\mathcal{A}$ outputs $i \in \mathcal{U}_B$ and $j \in \mathcal{S}_P$ such that if k was the output of a P-AUTH(i, j) or B-AUTH(i, j) query there was no ADD-TO-BL(j, k) query. $\mathcal{C}$ then runs P-AUTH(i, j). $\mathcal{A}$ wins the game if that P-AUTH is unsuccessful.

4 Preliminaries

In this section we outline the assumptions and cryptographic tools that we use as building blocks in our construction of the BLAC system.

4.1 Pairings

A *pairing* is a bilinear mapping from a pair of group elements to a group element. Specifically, let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of order p. Suppose P and Q are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. A function $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is said to be a pairing if it satisfies the following properties:

- *(Bilinearity.)* $\hat{e}(A^x, B^y) = \hat{e}(A, B)^{xy}$ for all $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_p$.
- *(Non-degeneracy.)* $\hat{e}(P, Q) \neq 1$, where 1 is the identity element in $\mathbb{G}_T$.
- *(Efficient Computability.)* $\hat{e}(A, B)$ can be computed efficiently (i.e. in polynomial time) for all $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

4.2 Mathematical Assumptions

The security of our construction of the BLAC system requires the following two assumptions:

Assumption 1 (DDH) The *Decisional Diffie-Hellman (DDH)* problem in group $\mathbb{G}$ is defined as follows: On input of a quadruple $(g, g^a, g^b, g^c) \in \mathbb{G}^4$, output 1 if $c = ab$ and 0 otherwise. We say that the DDH assumption holds in group $\mathbb{G}$ if no probabilistic polynomial time (PPT) algorithm has non-negligible advantage over random guessing in solving the DDH problem in $\mathbb{G}$.

Assumption 2 (q-SDH) The *q-Strong Diffie-Hellman (q-SDH)* problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: On input of a $(q+2)$-tuple $(g_0,\ h_0,\ h_0^x,\ h_0^{x^2},\ \ldots,\ h_0^{x^q}) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$, output a pair $(A, c) \in \mathbb{G}_1 \times \mathbb{Z}_p$ such that $A^{(x+c)} = g_0$ where $|\mathbb{G}_1| = p$. We say that the q-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no PPT algorithm has non-negligible advantage in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.

4.3 Proofs of Knowledge

In a *Zero-Knowledge Proof of Knowledge (ZKPoK)* protocol [GMR89], a prover convinces a verifier that some statement is true without the verifier learning anything except the validity of the statement. Σ-protocols are a special type of three-move ZKPoK protocols, which can be converted into non-interactive *Signature Proof of Knowledge (SPK)* schemes, or simply signature schemes [GMR88], that are secure under the *Random Oracle (RO)* Model [BR93].

In the following, we review several Σ-protocols that will be needed as building blocks in our construction. We follow the notation introduced by Camenisch and Stadler [CS97]. For instance, $PK\{(x) : y = g^x\}$ denotes a Σ-protocol that proves the knowledge of $x \in \mathbb{Z}_p$ such that $y = g^x$ for some $y \in \mathbb{G}$. The corresponding signature scheme resulting from the application of the Fiat-Shamir heuristic to the above Σ-protocol is denoted by $SPK\{(x) : y = g^x\}(M)$.

4.3.1 Knowledge and Inequalities of Discrete Logarithms

Let $g, b \in \mathbb{G}$ and $b_i \in \mathbb{G}$ for all i be generators of some group $\mathbb{G}$ of prime order p such that their relative discrete logarithms are unknown. One can prove in zero-knowledge the knowledge of the discrete logarithm $x \in \mathbb{Z}_p$ of $y \in \mathbb{G}$ in base g by using the Σ-protocol:

$$PK\left\{(x) : y = g^x\right\},$$

the construction of which first appeared in Schnorr identification [Sch91]. As we shall see, our BLAC construction requires the SPK of this protocol to prove the correctness of tickets.

One can further prove in zero-knowledge that x does *not* equal $\log_b t$, the discrete log of $t \in \mathbb{G}$ in base b, using the Σ-protocol:

$$PK\left\{(x) : y = g^x \wedge t \neq b^x\right\},$$

the most efficient construction of which is due to Camenisch and Shoup [CS03, §5].

In our BLAC system construction we will need a generalized version of the above Σ-protocol to prove that a user is not currently on the blacklist. In particular, we need a protocol that allows one to prove in zero-knowledge that, for some $n > 1$ and for all $i = 1$ to n, $x \neq \log_{b_i} t_i$, where $t_i \in \mathbb{G}$. That is,

$$PK\left\{(x) : y = g^x \wedge \left(\bigwedge_{i=1}^{n} t_i \neq b_i^x\right)\right\}.$$

Such a Σ-protocol can be constructed by applying a technique due to Cramer et al. [CDS94] to Camenisch and Shoup's construction mentioned above.[7]

4.3.2 BBS+ Signatures

Let $g_0, g_1, g_2 \in \mathbb{G}1$ and $h_0 \in \mathbb{G}_2$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively such that $g_0 = \psi(h_0)$ and their relative discrete logarithms are unknown, where ψ is a computable isomorphism and $(\mathbb{G}_1, \mathbb{G}_2)$ is a pair of groups of prime order p in which the q-SDH assumption holds. Let e be a pairing defined over the pair of groups. One can prove possession of a tuple $(A, e, x, y) \in \mathbb{G}_1 \times \mathbb{Z}_p^3$ such that $A^{e+\gamma} = g_0 g_1^x g_2^y$, or equivalently, $\hat{e}(A, w h_0^e) = \hat{e}(g_0 g_1^x g_2^y, h_0)$, where $w = h_0^\gamma$, by the Σ-protocol:

$$PK\{(A, e, x, y) : A^{e+\gamma} = g_0 g_1^x g_2^y\}.$$

[7] The technique describes a general method of constructing proofs of disjunction or conjunction of any of the two statements about knowledge of discrete logarithms.

The construction of this protocol can be found in [BBS04, §4], which is secure under the Decision-linear Diffie-Hellman assumption. Au et al. [ASM06] provide a modified construction that does not need to rely on such an assumption. As first pointed out in [CL04], the protocol's corresponding SPK is actually the SDH-variant of CL signatures [CL02b], which is referred to as BBS+ Signatures in [ASM06]. Our BLAC construction will need this protocol as a building block for users to prove that they are legitimate in the system. We will employ the construction given in [ASM06] to avoid the need of less standard assumptions.

5 System Construction

In this section, we detail our cryptographic construction and assess its security and efficiency.

5.1 Description

5.1.1 Parameters

Let λ, ℓ be sufficiently large security parameters. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with computable isomorphism ψ as discussed such that $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p of λ bits. Also let $\mathbb{G}$ be a group of order p where DDH is intractable. Let $g_0, g_1, g_2 \in \mathbb{G}_1$ and $h_0 \in \mathbb{G}_2$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively such that $g_0 = \psi(h_0)$ and the relative discrete logarithm of the generators are unknown.[8] Let $H_0 : \{0,1\}^* \to \mathbb{G}$ and $H : \{0,1\}^* \to \mathbb{Z}_p$ be secure cryptographic hash functions.

5.1.2 Setup

The GM randomly chooses $\gamma \in_R \mathbb{Z}_p$ and computes $w = h_0^\gamma$. The group secret key is $gsk = (\gamma)$ and the group public key is $gpk = (w)$.

5.1.3 Registration

At the successful termination of this protocol between a user Alice and the GM, Alice obtains a credential in the form of (A, e, x, y) such that $A^{e+\gamma} = g_0 g_1^x g_2^y$, and (A, e, x, y) is known only to the user. The private input to the GM is the group secret key gsk.

1. The GM sends m to Alice, where $m \in_R \{0,1\}^\ell$ is a random challenge.
2. Alice sends a pair (C, Π_1) to the GM, where $C = g_1^x g_2^{y'} \in \mathbb{G}_1$ is a commitment of $(x, y') \in_R \mathbb{Z}_p^2$ and Π_1 is a signature proof of knowledge of
$$SPK_1 \left\{ (x, y') : C = g_1^x g_2^{y'} \right\} (m) \tag{1}$$
on challenge m, which proves that C is correctly formed.
3. The GM returns as `failure` if the verification of Π_1 returns `invalid`. Otherwise the GM sends to Alice a tuple (A, e, y''), where $e, y'' \in_R \mathbb{Z}_p$ and $A = (g_0 C g_2^{y''})^{\frac{1}{e+\gamma}} \in \mathbb{G}_1$.
4. Alice computes $y = y' + y''$. She returns as `failure` if $\hat{e}(A, w h_0^e) \neq \hat{e}(g_0 g_1^x g_2^y, h_0)$. Otherwise she outputs $cred = (A, e, x, y)$ as her credential.

To prevent the possibility of a concurrent attack [Dam00], we require that users must be registered one after the other, as opposed to concurrently.

[8] This can be done by setting the generators to be the output of a cryptographic hash function of some publicly known seeds.

5.1.4 Authentication

During an execution of this protocol between a user Alice and an SP Bob, Alice's private input is her credential $cred = (A, e, x, y)$. Let $\texttt{Bob} \in \{0,1\}^*$ be the string that uniquely identifies Bob. When the protocol terminates, Bob outputs `success` or `failure`, indicating whether Bob should consider the authentication attempt successful.

1. *(Challenge.)* Bob sends to Alice a pair (BL, m), where $m \in_R \{0,1\}^\ell$ is a random challenge and $BL = \langle \tau_1, \ldots, \tau_n \rangle$ is Bob's current blacklist and $\tau_i = (s_i, t_i) \in \{0,1\}^\ell \times \mathbb{G}$, for $i = 1$ to n, is the i-th ticket in the blacklist.

2. *(Blacklist Inspection.)* Alice computes, for $i = 1$ to n, the *bases* $b_i = H_0(s_i || \texttt{Bob})$. She returns as `failure` if tag $t_i = b_i^x$ for some i (indicating that she is blacklisted). She proceeds otherwise.

3. *(Proof Generation.)* Alice returns to Bob a pair (τ, Π_2), where $\tau = (s, t) \in \{0,1\}^\ell \times \mathbb{G}$ is a ticket generated by randomly choosing a *serial* $s \in_R \{0,1\}^\ell$ and computing the base b as $H_0(s||\texttt{Bob})$ and then the tag t as b^x, and Π_2 is a signature proof of knowledge of:

$$SPK_2 \left\{ (A, e, x, y) : A^{e+\gamma} = g_0 g_1^x g_2^y \ \wedge \left(\bigwedge_{i=1}^{n} t_i \neq b_i^x \right) \wedge t = b^x \right\} (m) \tag{2}$$

   on the challenge m, which proves:

   (a) $A^{e+\gamma} = g_0 g_1^x g_2^y$, i.e., Alice is a group member,

   (b) $\bigwedge_{i=1}^{n} t_i \neq H_0(s_i||\texttt{Bob})^x$, i.e., Alice is not currently on Bob's blacklist, and

   (c) $t = H_0(s||\texttt{Bob})^x$, i.e., the ticket τ is correctly formed.

4. *(Proof Verification.)* Bob returns as `failure` if the verification of Π_2 returns `invalid`.[9] Otherwise Bob returns `success`.

The protocol transcript of a successful authentication at Bob is thus $\texttt{trans} = \langle \texttt{Bob}, BL, m, \tau, \Pi_2 \rangle$. As discussed, Bob stores ticket τ extracted from the transcript, along with information logging Alice's activity within the authenticated session.

5.1.5 Blacklist Management

The three algorithms are all very simple and efficient. Extract(`trans`) returns ticket τ in the input transcript $\texttt{trans} = \langle BL, m, \tau, \Pi_2 \rangle$. Add$(BL, \tau)$ returns blacklist BL', which is the same as the input blacklist BL, except with the input ticket τ appended to it. Remove(BL, τ) returns blacklist BL', which is the same as the input blacklist BL, except with all entries equal to the input ticket τ dropped.

5.2 SPK Instantiation

Both SPK_1 and SPK_2 presented above require instantiation. We omit spelling out the relatively trivial instantiation of SPK_1. Now we instantiate SPK_2 as follows.

[9] Bob also terminates with `failure` if the blacklist is being updated concurrently. This behavior ensures that if a user is blacklisted at time t, she cannot authenticate to the SP after t or until she is unblacklisted.

5.2.1 SPK_2 Signing

To produce a proof Π_2 for SPK_2 on message m, do the following.

1. Produce auxiliary commitments $(A_1, A_2, A_3, \tilde{A}_1, \ldots, \tilde{A}_n)$ by randomly picking $\rho_1, \rho_2, \rho_3, \rho_4 \in_R \mathbb{Z}_p$ and computing $A_1 = g_1^{\rho_1} g_2^{\rho_2}$, $A_2 = A g_2^{\rho_1}$, $A_3 = g_1^{\rho_3} g_2^{\rho_4}$ and, for all $i = 1$ to n, $\tilde{A}_i = (b_i^x / t_i)^{\rho_3}$.

2. Return Π_2 as $(A_1, A_2, A_3, \tilde{A}_1, \ldots, \tilde{A}_n, \Pi_3)$, where Π_3 is a signature proof of knowledge of:

$$SPK_3 \Bigg\{ (e, x, y, \rho_1, \rho_2, \rho_3, \rho_4, \alpha_1, \alpha_2, \beta_3, \beta_4) :$$

$$\begin{gathered} A_1 = g_1^{\rho_1} g_2^{\rho_2} \ \wedge \ 1 = A_1^{-e} g_1^{\alpha_1} g_2^{\alpha_2} \ \wedge \\ A_3 = g_1^{\rho_3} g_2^{\rho_4} \ \wedge \ 1 = A_3^{-x} g_1^{\beta_3} g_2^{\beta_4} \ \wedge \\ \frac{\hat{e}(A_2, w)}{\hat{e}_0} = \hat{e}(A_2, h_0)^{-e} \hat{e}_1^x \hat{e}_2^{y+\alpha_1} \hat{e}(g_2, w)^{\rho_1} \ \wedge \\ \left(\bigwedge_{i=1}^{n} \tilde{A}_i = b_i^{\beta_3} t_i^{-\rho_3} \right) \ \wedge \ 1 = b^{\beta_3} t^{-\rho_3} \Bigg\} (m) \end{gathered} \tag{3}$$

on message m, which can be computed using the knowledge of e, x, y, ρ_1, ρ_2, ρ_3, ρ_4, α_1, α_2, β_3 and β_4, where $\alpha_1 = \rho_1 e$, $\alpha_2 = \rho_2 e$, $\beta_3 = \rho_3 x$ and $\beta_4 = \rho_4 x$. In the above, we denoted $\hat{e}(g_i, h_0)$ as $\hat{e}_i$ for $i = 0$ to 2.

5.2.2 SPK_2 Verification

To verify a proof $\Pi_2 = (A_1, A_2, A_3, \tilde{A}_1, \ldots, \tilde{A}_n, \Pi_3)$ for SPK_2 on message m, return `valid` if the verification of Π_3 on m returns `valid` and $\tilde{A}_i \neq 1$ for all $i = 1$ to n. Return `invalid` otherwise.

The instantiation of SPK_3 itself is enumerated below.

5.2.3 SPK_3 Signing

To produce a proof Π_3 for SPK_3 on message $m \in \{0,1\}^*$, do the following:

1. *(Commit.)* Pick r_e, r_x, r_y, r_{ρ_1}, r_{ρ_2}, r_{ρ_3}, r_{ρ_4}, r_{α_1}, r_{α_2}, r_{β_3}, $r_{\beta_4} \in_R \mathbb{Z}_p^*$ uniformly at random. Compute $T_1 = g_1^{r_{\rho_1}} g_2^{r_{\rho_2}}$, $T_2 = A_1^{-r_e} g_1^{r_{\alpha_1}} g_2^{r_{\alpha_2}}$, $T_3 = g_1^{r_{\rho_3}} g_2^{r_{\rho_4}}$, $T_4 = A_3^{-r_x} g_1^{r_{\beta_3}} g_2^{r_{\beta_4}}$, and

$$T_5 = \hat{e}(A_2, h_0)^{-r_e} \cdot \hat{e}(g_1, h_0)^{r_x} \cdot \hat{e}(g_2, h_0)^{r_y} \cdot \hat{e}(g_2, w)^{r_{\rho_1}} \cdot \hat{e}(g_2, h_0)^{r_{\alpha_1}}.$$

   Also compute: $\tilde{T}_i = b_i^{r_{\beta_3}} t_i^{-r_{\rho_3}}$, for all $i = 1$ to n, and $T = b^{r_{\beta_3}} t^{-r_{\rho_3}}$.

2. *(Challenge.)* Compute c as: $H(A_1, A_2, A_3, \tilde{A}_1, \ldots, \tilde{A}_n, T_1, \ldots, T_5, \tilde{T}_1, \ldots, \tilde{T}_n, T, m)$.

3. *(Response.)* Compute $s_e = r_e - ce$, $s_x = r_x - cx$, $s_y = r_y - cy$, $s_{\rho_i} = r_{\rho_i} - c\rho_i$ for $i = 1$ to 4. $s_{\alpha_i} = r_{\alpha_i} - c\rho_i e$ for $i = 1, 2$, and $s_{\beta_i} = r_{\beta_i} - c\rho_i x$ for $i = 3, 4$.

4. *(Output.)* The signature proof of knowledge Π' on m is

$$\Pi' = (c, s_e, s_x, s_y, s_{\rho_1}, s_{\rho_2}, s_{\rho_3}, s_{\rho_4}, s_{\alpha_1}, s_{\alpha_2}, s_{\beta_3}, s_{\beta_4}).$$

5.2.4 SPK_3 Verification

To verify a proof Π_3 for SPK_3 on message m, do the following:

1. Compute $T_1' = g_1^{s_{\rho_1}} g_2^{s_{\rho_2}} A_1^c$, $T_2' = A_1^{-s_e} g_1^{s_{\alpha_1}} g_2^{s_{\alpha_2}}$, $T_3' = g_1^{s_{\rho_3}} g_2^{s_{\rho_4}} A_3^c$, $T_4' = A_3^{-s_x} g_1^{s_{\beta_3}} g_2^{s_{\beta_4}}$, and
$$T_5' = \hat{e}(A_2, h_0)^{-s_e} \cdot \hat{e}(g_1, h_0)^{s_x} \cdot \hat{e}(g_2, h_0)^{s_y} \cdot \hat{e}(g_2, w)^{s_{\rho_1}} \cdot \hat{e}(g_2, h_0)^{s_{\alpha_1}} \cdot \left(\frac{\hat{e}(A_2, w)}{\hat{e}(g_0, h_0)}\right)^c.$$
2. Compute $\tilde{T}_i' = b_i^{s_{\beta_3}} t_i^{-s_{\rho_3}} \tilde{A}_i^c$ for all $i = 1$ to n, and $T' = b^{s_{\beta_3}} t^{-s_{\rho_3}}$.
3. Return `valid` if c equals: $H(A_1, A_2, A_3, \tilde{A}_1, \ldots, \tilde{A}_n, T_1', \ldots, T_5', \tilde{T}_1', \ldots, \tilde{T}_n', T', m)$. Return `invalid` otherwise.

Note that among the 5 pairings needed to compute T_5 above, 4 of them are constant and are assumed to be included in the system's parameters. The signer thus only needs to compute one pairing, namely $e(A_2, h_0)$. This pairing does not depend on the blacklist and the message and can thus be precomputed. Similarly, the SP needs to compute two pairings during verification, namely $e(A_2, h_0)$ and $e(A_2, w)$. The above explains the last row in Table 1.

5.3 Analysis

5.3.1 Security

The correctness of the construction mostly stems from the correctness of SPK's. Its proof is thus relatively straightforward. We claim that our construction has correctness without proof for the sake of conciseness.

We now state the following theorem about the security of our construction. Its proof can be found in Appendix A.

Theorem 1 (Security) *Our construction of the* BLAC *system is secure if the q-SDH problem is hard in* $(\mathbb{G}_1, \mathbb{G}_2)$ *and the DDH problem is hard in* $\mathbb{G}$ *under the Random Oracle Model.*

5.3.2 Complexity

We analyze the efficiency of our construction in terms of both time and space/communication complexities. First we emphasize that both complexities are independent of the number of users and SPs in the system. Thus our system scales well with respect to these two quantities. Both complexities, however, are dependent on the size of the blacklist. In particular, the time it takes for both a user and a SP to execute the **authentication** protocol, as well as communication overhead for the same protocol, grow linearly with the current size of the SP's blacklist.

More specifically, a blacklist of size n contains n tickets, each consisting of an ℓ-bit string and an element of $\mathbb{G}$. A proof Π_2 of SPK_2 consists of 3 $\mathbb{G}_1$ elements, n $\mathbb{G}$ elements and 12 $\mathbb{Z}_p$ elements. The total communication complexity for an authentication is thus $(n+2)$ ℓ-bit strings, 3 $\mathbb{G}_1$ elements, $(2n+1)$ $\mathbb{G}$ elements and 12 $\mathbb{Z}_p$ elements. SPs need to store a ticket for every successful authentication.

A breakdown of time complexity of the **authentication** protocol into the number of *multi-exponentiations (multi-EXPs)*[10] in various groups and pairings is shown in Table 1. Other operations such as $\mathbb{G}$ addition and hashing are neglected as they take negligible time. Some preprocessing

[10] A multi-EXP computes the product of exponentiations faster than performing the exponentiations separately. We assume that one multi-EXP operation multiplies up to 3 exponentiations.

Table 1: Number of operations during an authentication with a blacklist of size n.

Operation	User		SP
	w/o Preproc.	w/ Preproc.	
$\mathbb{G}_1$ multi-EXP	7	0	4
$\mathbb{G}_T$ multi-EXP	2	0	2
$\mathbb{G}$ multi-EXP	$2n+1$	$2n$	$n+1$
Pairing	1	0	2

is possible at the user before the knowledge of the challenge message and the blacklist. In fact, all but $2n$ multi-EXPs in $\mathbb{G}$ can be precomputed by the user.

6 Performance Evaluation

We implemented our construction of the BLAC system in C and packaged the code into a software library to allow for easy adoption by different potential applications. We used the Pairing-Based Cryptography (PBC) Library.[11] (version 0.4.7) for the underlying elliptic-curve and pairing operations, which is built on the GNU MP Bignum (GMP) Library.[12] We also made use of several routines in OpenSSL,[13] such as its SHA-1 hash function for instantiating the cryptographic hash functions needed by our construction.

The choice of curve parameters can have a significant effect on the performance of an implementation. We used pairings over Type-A curves as defined in the PBC library. A curve of such type has the form of $E : y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime q. Both $\mathbb{G}_1$ and $\mathbb{G}_2$ are the group of points $E(\mathbb{F}_q)$ of order p for some prime p such that p is a factor of $q+1$. The pairing is symmetric and has an embedding degree k of 2. Thus $\mathbb{G}_T$ is a subgroup of $\mathbb{F}_{q^2}$. In our implementation, q and p are respectively 512-bit and 160-bit. We also used $\mathbb{G}_T$ for $\mathbb{G}$, the group wherein the tickets reside.

The interface to the library we implemented is defined by a list of C functions. Some of the more important ones are as follows. `setup()` is a function that implements the **Setup** algorithm. The functions `register_gm()` and `register_user()`, executed by the GM and the user respectively, together implement the **Registration** protocol. Similarly `authen_sp()` and `authen_user()` together implement the **Authentication** protocol.

6.1 Prototyping

Using our library, we prototyped a proof-of-concept application that allows users to post text messages at a web forum. This can be thought of as users editing Wikipedia pages. We did not prototype the user registration part of the system because our major interest was to study the performance of the **Authentication** protocol.

In our prototype, the authentication is carried out as follows. The SP first creates a listening socket. Upon the arrival of a connection request from a user, the SP sets up an SSL socket with the user using OpenSSL.[14] This means that a confidential and server-authenticated channel is set up between the user and the SP. From within this channel, the user and the server respectively

[11] `http://crypto.stanford.edu/pbc/`
[12] `http://gmplib.org/`
[13] `http://www.openssl.org/`
[14] For simplicity's sake, the SP uses a self-signed key-pair to authenticate himself.

execute `authen_user()` and `authen_sp()`. If `authen_sp` returns `failure`, then the SP closes the SSL connection, thereby refusing to serve the user. Otherwise, SP serves the user using the same channel by recording the text message sent by the user, along with the ticket extracted from the authentication transcript. The SP may then manually inspect the text message and add the associated ticket to its blacklist.

Alternatively, by integrating it with SSL server-authentication, BLAC authentication can be turned into a mutual authentication, in which the user authenticates the server's identity but the server is ensured that and only that the user is some well-behaving user.

6.2 Experimental Results and Analysis

For our experiments, we used a Dell GX745 desktop machine with an Intel dual-core 2.16 GHz CPU and 2GB of RAM, running Linux/Ubuntu 6.10. All the timings reported below are averaged over 10 randomized runs.

We measured two time quantities related to the execution of the **Authentication** protocol: (1) the time it took for an SP to verify the authentication (i.e., step 4 of the protocol), and (2) the time it took for a user to inspect the blacklist and produce a proof (i.e., steps 2 and 3 of the protocol), with preprocessing enabled. The sum of these two quantities roughly represents the total latency incurred by the protocol as perceived by the user if we ignore the network I/O delay, which is network-dependent.

When the blacklist was empty, it took the SP 0.06s to verify the authentication. When the blacklist had 400 entries instead, it took the server 0.46s to do the same. On the other hand, when the blacklist size was 0 and 400, the user spent 0.09ms and 0.73s respectively to inspect the blacklist and produce a proof. The estimated protocol latencies are thus 0.06s and 1.19s respectively. The total communication overhead due to the authentication protocol is roughly 0.27KB per blacklist entry. Table 2 shows experimental figures collected with different blacklist sizes. Please see our discussion in Section 7 that elaborates on the feasibility of our construction in real applications.

Note that our authentication protocol scales well with the number of cores in CPUs because virtually all computation that grows linearly with the blacklist size is parallelizable.[15] As evidence, on our dual-core machine, all the timings we collected using our original single-threaded implementation almost doubled the figures we just reported above. In our current multi-threaded implementation, the library interface includes a bootstrapping function that takes the number of threads as an input.

7 Discussion

Efficiency In our cryptographic construction, blacklist verification requires $O(n)$ computations, where n is the number of entries in the blacklist. As indicated by Section 6, our scheme would support 1,600 blacklist entries with 2 authentications per second on an 8-core machine.[16] Since anonymous authentications will be used at SPs such as Wikipedia only for certain operations such as editing webpages, we believe this performance is reasonable. Consider two extreme examples. In March 2007, Wikipedia averaged about two edits per second to its set of English webpages.[17] Likewise, YouTube reported less than one video upload per second on average in July 2006.[18]

[15] The only exception is the two calls to SHA-1, but they take comparably negligible time.

[16] An 8-core Mac Pro with two 3.0GHz Quad-Core Intel Xeon processors was available for under $4,000 at the time of writing.

[17] `http://stats.wikimedia.org/EN/PlotsPngDatabaseEdits.htm`

[18] `http://technology.guardian.co.uk/weekly/story/0,,1823959,00.html`

Table 2: Performance of our authentication protocol with respect to different blacklist sizes.

Blacklist Size (#Entries)	0	100	200	400	800	1600
Time (in s) for User to inspect the blacklist and generate a proof (steps 2 & 3)	0.00	0.18	0.36	0.73	1.45	2.85
Time (in s) for SP to verify the proof (step 4)	0.06	0.16	0.26	0.46	0.87	1.68
Estimated Protocol Latency (in s) perceived by User	0.06	0.34	0.62	1.19	2.32	4.53
Communication Overhead (in KB)	0.8	27.7	54.7	108.6	216.4	431.8

The communication complexity required to sustain one or two authentications per second with 1,600 blacklist entries would be about 3.5 to 7 Mbps for the SP. Such a data rate would be high for an individual server, but would be reasonable for large SPs such as YouTube and Wikipedia, which may have distributed servers across the nation for handling large bandwidth. Based on these calculations, SPs with much lower authentication rates than Wikipedia or YouTube (e.g., one authentication every few seconds) can easily be served on commodity hardware and T-1 lines. We reiterate that our construction is the first to allow anonymous blacklisting without TTPs, and more efficient blacklist checking, perhaps in $O(\log n)$ or $O(1)$ time, is an open problem that deserves further research. Faster verification will allow much higher rates of authentication while supporting extremely large blacklists, and this problem is, therefore, worthy of further study.

Interleaving Authentications One concern is that an individual user may attempt to interleave multiple authentications and take up several hundreds of entries in the blacklist by misbehaving several times in a short span of time. Such an attack is possible because users can parallelize several anonymous sessions with an SP. A promising approach would be to use a scheme such as Camenisch et al.'s periodic n-times anonymous authentication [CHK$^+$06] to rate-limit the number of anonymous accesses from users. In such a scheme, an anonymous user would be able to access the SP anonymously at most n times within a time period. For example, for $n = 10$ and a time period of 1 day, a single user would be able to contribute at most 10 entries to the blacklist in a given day.

Remark. Since concurrent sessions are preempted while an entry is added (atomically) to a blacklist, our system guarantees that once an entry is added to the blacklist at time t, the blacklisted user will not be able to access the service after time t (or until unblacklisted at a later time).

Enrollment Issues We assume that the Group Manager issues only one credential per legitimate user and assume it is difficult to perform "Sybil" attacks [Dou02], where users are able to obtain multiple credentials by posing as different identities. The Sybil attack, however, is a challenging problem that any credential system is vulnerable to, and we do not attempt to solve this problem here.

In a real deployment of a BLAC system, users may eventually misplace their credentials, or have

them compromised. Since that credential may be blacklisted by an SP, issuing a new credential to a user can help that user circumvent anonymous blacklisting. As a trade-off, we suggest that if a user misplaces his or her credential, that user is issued a pseudonymous credential for a certain amount of time called the "linkability window." If a user repeatedly attempts to acquire new credentials, the linkability window of that user can be increased to curb misbehavior.

Allowing the Sharing of (Entries in) Blacklists We have presented our construction of the BLAC system in which an SP Bob cannot use an entry from another SP's blacklist (corresponding to Alice) to prevent Alice from successfully authenticating to Bob. Nevertheless, in some applications, a group of SPs may desire to block users misbehaving at any one of the SPs.

Our system can be modified to allow such sharing—instead of computing the tag as $t = H(s||\mathtt{Bob})^x$, one computes the tag as $t = H(s)^x$ regardless of the SP for which the ticket is meant. Tickets with tags computed this way are sharable between SPs because adding a user's ticket borrowed from another SP is no different from the SP obtaining the ticket directly from the user. Such a change in construction, however, makes it necessary to redefine security notions. For example, Wikipedia may decide to add only YouTube's tickets to its blacklist. If a user's authentication fails, Wikipedia knows that the user has previously visited YouTube. Even though the user is anonymous, an SP can learn some information about the user's behavior at another SP.

Revoking Compromised TPMs Concurrently and independently, Brickell and Li [BL07] have proposed a method to unlinkably revoke compromised Trusted Platform Modules (TPMs) [TPM06]. While they focus on revoking compromised hardware, as opposed to blacklisting misbehaving users, their construction is similar to ours. Both solutions use a protocol for proving the inequality of multiple discrete logarithms to prove that a user is not revoked/blacklisted. Nevertheless, signatures in their solution are not bound to the verifier's identity and authenticating even once could result in the global revocation of the prover. Our solution provides more privacy by allowing sharing and non-sharing of blacklist entries among verifiers. Finally, their solution is RSA-based while ours is pairing-based.

8 Conclusion

We motivated the need for anonymous credential systems that support anonymous blacklisting and subjective judging without relying on trusted third parties that are capable of deanonymizing (or linking) users. All previous solutions rely on either trusted third parties or restricted formulations of misbehavior. We provide the first cryptographic construction that simultaneously provides *anonymous blacklisting*, *subjective judging*, and *eliminates the reliance on trusted third parties* capable of revoking the privacy of users.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, volume 1880 of *LNCS*, pages 255–270. Springer, 2000.

[ACS05] Man Ho Au, Sherman S. M. Chow, and Willy Susilo. Short e-cash. In *INDOCRYPT*, volume 3797 of *LNCS*, pages 332–346. Springer, 2005.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In *SCN*, volume 4116 of *LNCS*, pages 111–125. Springer, 2006.

[AST02] Giuseppe Ateniese, Dawn Xiaodong Song, and Gene Tsudik. Quasi-efficient revocation in group signatures. In *Financial Cryptography*, volume 2357 of *LNCS*, pages 183–197. Springer, 2002.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.

[BL07] Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. Cryptology ePrint Archive, Report 2007/194, 2007. `http://eprint.iacr.org/`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM Press, 1993.

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *ACM Conference on Computer and Communications Security*, pages 168–177. ACM, 2004.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira 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.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 302–321. Springer, 2005.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *SCN*, volume 4116 of *LNCS*, pages 141–155. Springer, 2006.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[CL02a] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[CL02b] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, volume 3152 of *LNCS*, pages 56–72. Springer, 2004.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.

[CvH91] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT*, pages 418–430, 2000.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[Dou02] John R. Douceur. The sybil attack. In *IPTPS*, volume 2429 of *LNCS*, pages 251–260. Springer, 2002.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[JKTS07] Peter C. Johnson, Apu Kapadia, Patrick P. Tsang, and Sean W. Smith. Nymble: Anonymous IP-address blocking. In *Privacy Enhancing Technologies Symposium (PET '07), Ottawa, Canada*, June 2007. To appear.

[KY05] Aggelos Kiayias and Moti Yung. Group signatures with efficient concurrent join. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 198–214. Springer, 2005.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In *ACISP*, volume 3108 of *LNCS*, pages 325–335. Springer, 2004.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *LNCS*, pages 275–292. Springer, 2005.

[NSN05] Lan Nguyen and Reihaneh Safavi-Naini. Dynamic k-times anonymous authentication. In *ACNS*, volume 3531 of *LNCS*, pages 318–333. Springer, 2005.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[SSG97] Paul F. Syverson, Stuart G. Stubblebine, and David M. Goldschlag. Unlinkable serial transactions. In *Financial Cryptography*, volume 1318 of *LNCS*, pages 39–56. Springer, 1997.

[TAKS07] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: Blocking misbehaving users without TTPs. In *ACM Conference on Computer and Communications Security*. ACM, 2007. To Appear.

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-times anonymous authentication (extended abstract). In *ASIACRYPT*, volume 3329 of *LNCS*, pages 308–322. Springer, 2004.

[TPM06] TPM Work Group. TCG TPM specification version 1.2 revision 94. Technical report, Trusted Computing Group, 2006.

[TS06] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In *Public Key Cryptography*, volume 3958 of *LNCS*, pages 525–542. Springer, 2006.

[TW05] Patrick P. Tsang and Victor K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In *ISPEC*, volume 3439 of *LNCS*, pages 48–60. Springer, 2005.

[TWC+04] Patrick P. Tsang, Victor K. Wei, Tony K. Chan, Man Ho Au, Joseph K. Liu, and Duncan S. Wong. Separable linkable threshold ring signatures. In *INDOCRYPT*, volume 3348 of *LNCS*, pages 384–398. Springer, 2004.

A Proof Sketches

We sketch the proof for Theorem 1 in the following three subsections, one for each security requirement.

A.1 Blacklistability

Suppose there exists a PPT adversary $\mathcal{A}$ who can win in game *Blacklistability* with non-negligible probability, we show how to construct a PPT simulator $\mathcal{C}$ that solves the q-SDH problem with non-negligible probability.

On input of an instance of the q-SDH problem $(g_0', h_0', {h_0'}^{\gamma}, \ldots, {h_0'}^{\gamma^q})$, $\mathcal{C}$'s task is to output a pair $(\bar{A}, \bar{e})$ such that $\hat{e}(\bar{A}, {h_0'}^{\gamma} {h_0'}^{\bar{e}}) = \hat{e}(g_0', h_0')$. Let q be the number of times $\mathcal{A}$ queries the A-Reg oracle and the Corrupt-U oracle. $\mathcal{C}$ uses the problem instance to generate the public parameters so that it can answer these q queries, using the same technique as in [ASM06]. Specifically, $\mathcal{C}$ randomly generates a degree $(q-1)$ polynomial f such that $f(x) = \prod_{i=1}^{q-1}(x+e_i)$. It computes $h_0 = {h_0'}^{f(\gamma)}$ and $w = h_0^{\gamma} = {h_0'}^{\gamma f(\gamma)}$. It also computes $h_1 = [(w h_0^{e^*})^{k^*} h_0^{-1}]^{1/a^*}$ and $h_2 = h_1^{\mu}$ for some $e^*, k^*, a^*, \mu \in \mathbb{Z}_p^*$ generated uniformly at random. Next, it computes $g_i = \psi(h_i)$ for $i = 0$ to 2. Finally, $\mathcal{C}$ gives $(h_0, h_1, h_2, g_0, g_1, g_2, w)$ to $\mathcal{A}$ as the system parameters. Let $\mathcal{K}$ be the set $\{1, \ldots, q-1\} \cup \{*\}$.

$\mathcal{C}$ keeps track of every user in the system. For a user $j \in \mathcal{U}_P$, $\mathcal{C}$ simulates the P-Reg oracle by first selecting $x_j \in \mathbb{Z}_p^*$ uniformly at random and uses it to simulate the Registration protocol. This is possible since the Registration protocol has Honest-Verifier Zero-Knowledgeness (HVZK).

To simulate P-Auth, B-Auth for user $j \in \mathcal{U}_P$, $\mathcal{C}$ computes the tag as $t = b^{x_j}$ and simulates the other steps using the HVZK property of the Authentication protocol.

When user $j \in \mathcal{U}_P$ is to be corrupted, $\mathcal{C}$ chooses i in the set $\mathcal{K}$ uniformly at random. If $i = *$, $\mathcal{C}$ sets $y_j = (a^* - x_j)/\mu$, $A_j = g^{k^*}$ and $e_j = e^*$, and returns (A_j, e_j, x_j, y_j) as the credential of user j. Otherwise, $\mathcal{C}$ chooses $y_j \in \mathbb{Z}_p^*$ uniformly at random, sets $e_j = e_i$, computes A_j as

$$
\begin{aligned}
A_j = \left(g_0 g_1^{x_j+\mu y_j}\right)^{\frac{1}{e_j+\gamma}} &= {g_0'}^{\frac{f(\gamma)}{e_j+\gamma}} g_1^{\frac{x_j+\mu y_j}{\gamma+e_i}} \\
&= {g_0'}^{\frac{f(\gamma)}{e_j+\gamma}} \left(g_0^{\frac{(x_j+\mu y_j)k^*(e^*+\gamma)-(x_j+\mu y_j)}{(e_j+\gamma)a^*}}\right) \\
&= {g_0'}^{\frac{f(\gamma)}{e_j+\gamma}\left(1-\frac{x_j+\mu y_j}{a^*}\right)} \left(g_0^{\frac{(x_j+\mu y_j)k^*}{a^*}}\right)^{\left(1-\frac{e_j-e^*}{e_j+\gamma}\right)} \\
&= {g_0'}^{\frac{f(\gamma)}{e_j+\gamma}\left(1-\frac{x_j+\mu y_j}{a^*}-\frac{(e_j-e^*)(x_j+\mu y_j)k^*}{a^*}\right)} g_0^{\frac{(x_j+\mu y_j)k^*}{a^*}},
\end{aligned}
$$

and finally returns (A_j, e_j, x_j, y_j) as the credential of user j. In both cases, $\mathcal{C}$ removes i from $\mathcal{K}$.

The simulation of A-Reg is similar. Upon receiving C for user j (to be added to $\mathcal{U}_A$), $\mathcal{C}$ first extracts the pair (x_j, y_j') by rewinding the adversary and selects i from $\mathcal{K}$ uniformly at random. If $i = *$, $\mathcal{C}$ chooses y_j'' such that $x_j + \mu(y_j' + y_j'') = a^*$, sets $A_j = g^{k^*}$, $e_j = e^*$ and finally returns (A_j, e_j, y_j''). Otherwise, $\mathcal{C}$ chooses y_j'' uniformly at random, sets $e_j = e_i$ and $y_j = y_j' + y_j''$, computes A_j as

$$
A_j = {g_0'}^{\frac{f(\gamma)}{e_j+\gamma}\left(1-\frac{x_j+\mu y_j}{a^*}-\frac{(e_j-e^*)(x_j+\mu y_j)k^*}{a^*}\right)} g_0^{\frac{(x_j+\mu y_j)k^*}{a^*}},
$$

and finally returns (A_j, e_j, y_j''). $\mathcal{C}$ removes i from $\mathcal{K}$ in both cases.

$\mathcal{C}$ stores all the credentials issued to $\mathcal{A}$ in a set $\mathcal{U}_{KI}$.

For each A-AUTH query, $\mathcal{C}$ extracts the underlying credential (A_n, e_n, x_n, y_n) by rewind simulation and stores it in the set $\mathcal{U}_{KA}$.

During the *End Game* phase, $\mathcal{A}$ has produced $k+1$ authentications in the A-AUTH Oracle. Let $\{H_0(s_i, U), t_i\}$ be the set of tickets associated with these $k+1$ authentications. Let x_i denote $\log_{H_0(s_i,U)}(t_i)$ for $i = 1$ to $k+1$. Due to the soundness of SPK_2, there exists $q+1$ x's such that $x_i \neq x_j$ if $i \neq j$ for all $i, j \in \{1, \ldots, q+1\}$. It implies that there exists at least one tuple in $\mathcal{U}_{KA}$ and is not in $\mathcal{U}_{KI}$.

Let $(A', e', x', y') \in \mathcal{U}_{KA} \wedge \notin \mathcal{U}_{KI}$. There are three possibilities:

- Case I: $e' \notin \{e_i, e^*\}$. Denote $z = x' + \mu y'$. We have:

$$\begin{aligned} A'^{e'+\gamma} &= g_0 g_1^z \\ A'^{e'+\gamma} &= g_0^{\frac{k^* z(e^*+\gamma)-z}{a^*}} \\ A' &= g_0^{\frac{a^*-z}{a^*(e'+\gamma)}} \left(g_0^{\frac{k^* z}{a^*}}\right)^{\left(1-\frac{e'-e^*}{e'+\gamma}\right)} \\ g_0^{\frac{1}{e'+\gamma}} &= \left(A' g_0^{\frac{-k^* z}{a^*}}\right)^{\frac{a^*}{a^*-z-k^* z(e'-e^*)}}. \end{aligned}$$

  Denote $B' = g_0^{\frac{1}{e'+\gamma}} = g_0'^{\frac{f(\gamma)}{e'+\gamma}}$. Using long division, there exists a degree $(q-2)$ polynomial f_q such that $\frac{f(\gamma)}{(e'+\gamma)} = f_q(\gamma)(e'+\gamma) + f_1$ for some $f_1 \in \mathbb{Z}_p^* \backslash \{0\}$. Thus $B' = g_0'^{\frac{f_1}{e'+\gamma} + f_q(\gamma)}$. Finally, $\mathcal{A}$ computes $\bar{A} = \left(B' g_0'^{-f_q(\gamma)}\right)^{1/f_1}$ and sets $\bar{e} = e'$. $(\bar{A}, \bar{e})$ is the solution to the q-SDH problem.

- Case II: $(e' = e_i \wedge A' = A_i)$ or $(e' = e^* \wedge A' = A^*)$. This case happens with negligible probability unless $\mathcal{F}$ solves the relative discrete logarithm of h_2 to base h_1.

- Case III: $e' \in \{e_i, e^*\}$ and $(A' \neq A_i \vee A' \neq A^*)$. With probability $1/q$, $e' = e^*$. Denote $z = x' + \mu y'$. We have:

$$\begin{aligned} A'^{e^*+\gamma} &= g_0 g_1^z \\ A' &= g_0^{\frac{a^*-z}{a^*(e^*+\gamma)}} g_0^{\frac{k^* z}{a^*}} \\ g_0^{\frac{1}{e^*+\gamma}} &= \left(A' g_0^{\frac{-k^* z}{a^*}}\right)^{\frac{a^*}{a^*-z}}. \end{aligned}$$

  Denote $B' = g_0^{\frac{1}{e'+\gamma}} = g_0'^{\frac{f(\gamma)}{e'+\gamma}}$. $\mathcal{C}$ uses the same method as in Case I to solve the q-SDH problem.

A.2 Anonymity

Suppose there exists a PPT adversary $\mathcal{A}$ who can win in game Anonymity with non-negligible probability, we show how to construct a PPT simulator $\mathcal{C}$ that solves the DDH problem with non-negligible probability. On input of a DDH tuple $(g', g'^{u'}, g'^{v'}, T')$, $\mathcal{C}$ is required to decide if $T' = g'^{u'v'}$. $\mathcal{C}$ sets $G = \langle g' \rangle$ and generates all other parameters honestly. The parameters and the master key of the GM are given to $\mathcal{A}$.

$\mathcal{C}$ keeps track of every user in the system. $\mathcal{C}$ chooses one user, denoted as i^*. For all oracle queries (except the Hash oracle) not related to i^*, $\mathcal{C}$ follows the protocol honestly.

Queries related to user i^* are handled as follows. For B-Reg, $\mathcal{C}$ simulates the protocol as if (u', y') is an opening of the commitment C. The distribution is perfect since for any u' there exists an y' such that $C = g_1^{u'} g_2^{y'}$. Upon receiving (A, e, y'') from $\mathcal{A}$, $\mathcal{C}$ records the credential for i^* as $(A, e, \perp, \perp)$. The credential for i^* is (A, e, u', y) such that $y = y' + y''$. This credential, however, is unknown to $\mathcal{C}$. (P-Auth or B-Auth queries involving user i^* is discussed below.)

For P-Auth or B-Auth queries related to user i^*, $\mathcal{C}$ chooses s uniformly at random and sets $H_0(s||SP_j) = g'^R$ for some R generated uniformly at random. $\mathcal{C}$ then computes $t = g'^{u'R}$ and simulates the protocols with $\tau = (s, t)$.

Finally, if i^* is chosen to be one of the two challenge users, $\mathcal{C}$ embeds the problem instance into the authentication transcript by choosing s_{i^*} uniformly at random and setting $H_0(s_{i^*}||SP_j) = g'^{v'}$. $\mathcal{C}$ computes the ticket $\tau = (H_0(s_{i^*}||SP_j), t_{i^*}) = (g'^{v'}, T')$ and simulates the Authentication protocol. If T' is a DDH tuple, the simulation is perfect while if T' is a random element, the authentication transcript is not related to either of the challenge users. Thus, if $\mathcal{A}$ can answer the challenge correctly, $\mathcal{C}$ concludes that $(g', g'^{u'}, g'^{v'}, T')$ is a DDH-tuple.

A.3 Non-Frameability

Suppose there exists a PPT adversary $\mathcal{A}$ who can win in game *Non-Frameability* with non-negligible probability, we show how to construct a PPT simulator $\mathcal{C}$ that solves the discrete logarithm problem in $\mathbb{G}$.

On input of a DL problem instance (T', g'), $\mathcal{C}$ is required to compute u' such that $g'^{u'} = T'$. $\mathcal{C}$ sets $\mathbb{G} = \langle g' \rangle$ and all other parameters are generated honestly. The parameters and the master key of GM are given to $\mathcal{A}$.

$\mathcal{C}$ keeps track of every user present in the system. $\mathcal{C}$ chooses one user, denoted as i^*. For all oracle queries (except Hash oracle) not related to i^*, $\mathcal{C}$ follows the protocol honestly. Let $\mathcal{K}$ be the set of credentials $\mathcal{C}$ has obtained from $\mathcal{A}$ in the B-Reg query.

Queries related to user i^* are handled as follows. For B-Reg, $\mathcal{C}$ simulates the protocol as if u', y' is an opening of the commitment C. The distribution is perfect since for any u' there exists a y' such that $C = g_1^{u'} g_2^{y'}$. Upon receiving (A, e, y'') from $\mathcal{A}$, $\mathcal{C}$ adds $(A, e, \perp, \perp)$ to $\mathcal{K}$. For P-Auth or B-Auth query related to user i^*, $\mathcal{C}$ chooses s uniformly at random and sets $H_0(s||SP_j) = g'^R$ for some R generated uniformly at random. $\mathcal{C}$ then computes $t = g'^{u'R}$ and simulates the protocols with $\tau = (s, t)$.

To win the game, $\mathcal{A}$ must have produced an authentication transcript with ticket $\tau = (s, t)$ such that $\hat{x}$, defined as $\log_{H_0(s||id)}(t)$, is in $\mathcal{K}$. With probability $1/|\mathcal{K}|$, $\hat{x} = u'$. Using rewind simulation, $\mathcal{C}$ obtains u', which is the discrete logarithm of T'.