Scored Anonymous Credentials

Sherman S. M. Chow[1]*, Jack P. K. Ma[1], and Tsz Hon Yuen[2]

[1] Department of Information Engineering
The Chinese University of Hong Kong,
Shatin, N.T., Hong Kong
[2] Department of Computer Science
The University of Hong Kong,
Pokfulam, Hong Kong
{smchow,mpk016}@ie.cuhk.edu.hk,thyuen@cs.hku.hk

Abstract. Securely maintaining "credits" of users judging their behavior in past authenticated sessions is vital to encourage user participation, but doing it over anonymous credentials is non-trivial, especially when users would avoid claiming negative credit and escape from blocklisting. Prevalent designs impose an authentication cost linear in the blocklist size or a stringent requirement of *sequential and timely* judgment of each session without retrospective adjustment, as a single unjudged session curbs the authentication of *all* users. We propose scored anonymous credentials, a new design storing a number of *active* sessions with volatile scores downgradable before finalized. Sessions can be judged in any order and at varying times without affecting all users. Any backlog of unjudged sessions only affects the users behind them. We achieve efficiency and flexibility using verifiable shuffle, which is hardly used in existing anonymous blocklisting/reputation enforcement systems.

1 Introduction

Anonymity has always been an important issue on the Internet. In the early days, most people (wrongly) assumed anonymity on the Internet by using pseudonyms, but the service provider (SP) might use IP addresses to identify the users. Anonymity networks [29,36] were then developed to make tracing IP addresses more difficult. However, this may not be directly applicable to services that require users to register and authenticate. Using the same registered pseudonym to access the service each time links user behavior across sessions. It is possible to uniquely profile a user across different services by collecting this information [41].

Anonymous credentials [18] ensure unlinkable authentications. However, users may "misbehave" in some sessions exploiting their anonymity. To illustrate,

* Sherman Chow is supported in part by the General Research Funds (CUHK 14210621 and 14209918), University Grants Committee, Hong Kong, and is also grateful for their recognition in granting the Early Career Award on "Accountable Privacy in Online Communication" (CUHK 439713). We thank Kin-Ying Yu for inspiration from his Ph.D. thesis, and Russell Lai and reviewers for their helpful comments.

Wikipedia users (contributors/reviewers) may prefer their identities to be masked so that their views would not represent any belonging organizations or communities (e.g., a company/political party) and avoid their online behavior causing any consequence in their "real" life, especially in places with restricted freedom of speech. On the other hand, Wikipedia has its own set of "subjective" policies and guidelines (e.g., concerning copyright) that users must follow. Users who violate such policies repetitively and deliberately should be penalized.

It is challenging to support *subjective* revocation (cf., legal contract) in anonymous authentication without relying on any *trusted third party* (TTP). Some systems only support *objective* revocation based on mathematically or algorithmically evaluable claims of misbehavior (cf., smart contract), such as double spending in e-cash [37,23] or double voting [24]. Revocable anonymous credentials [19,17,3,2,28] often assume users to be revoked had been deanonymized by some external means. TTP-based approaches, such as group/traceable signatures, require a revocation authority to use a non-public database gathered when the user first joins the system [22,7] or a secret key to recover the signer identity [26,7] or create a tracing token [1,48]. The TTP is assumed not to violate user privacy unnecessarily. Simply, "off-the-shelf" credentials themselves do not support TTP-free subjective revocation [44].

Generalizing revocation, reputation can be used as a basis for granting privileges to special services or imposing limitations on users based on their behaviors. It is important to encourage participation, especially when participation may potentially incur a revocation or real-life consequences. A large-scale online anonymous community has already been developed. For example, Bernstein *et al.* [10] showed that over 90% of posts are anonymous on `4chan.org`, which has a million posts per day[3] and 22+ million unique monthly users. However, registration is not needed for posting anonymously in these forums currently. So, reputation cannot be maintained. We refer to [35] for a survey on reputation management.

Updatable anonymous credential is useful for adding reputation. Each user is associated with a *reputation score*, an aggregated sum of all individual reputations decided by the SP (possibly after hearing from other users) one gains from each contribution. A high reputation could mean the user has successfully helped many others and can be considered more trustworthy. Reputation can also somewhat mitigate *sybil attacks* since an attacker would need to give up any previously earned reputation for registering anew.

1.1 A Critical Review of Existing Systems

Existing (updatable) anonymous credential systems with subjective revocation, however, are designed for applications with lower traffic and anonymity needs (e.g., assuming anonymity is needed for 20% of all Wikipedia posts and is meant to handle about $100,000$ anonymous posts per day [4,5]). They are not suitable for a large-scale community with millions of posts per day, e.g., Reddit.

[3] Statistic at `https://4stats.io`

All works reviewed below assume one SP serving many clients. There is no separate (trusted) entity for revocation. For simplicity, this line of works assumes the SP is both the credential issuer and verifier, and has the final say in the score for each session, which can be published on a public bulletin board.

After a successful user authentication, the SP will assign a publicly-known session identifier. The user credential will be attached to a ticket associated to the session identifier. These two terms are often used interchangeably in this paper.

Table 1 compares known systems in terms of their desired properties.

Blocklisting/Reputation with Linear (Proving) Cost. BLAC [44] and EPID [13] are two early TTP-free systems. Their blocklist consists of *"deterministic tags"* directly available from the authentication transcripts of sessions that should be blocklisted. Revocation is done by proving that the user secret in the credential could not have generated any tag in the blocklist. This takes $O(L)$ time, where L is the blocklist size. Moreover, the blocklist *never shrinks*. It can become very long quickly since malicious users are motivated to keep damaging (e.g., spamming or committing vandalism) in a short period before getting caught. The *whole* system keeps slowing down as times go by, affecting all (honest) users.

BLACR [5] extends BLAC to also support reputation with three lists, $\mathbb{L}_+$, $\mathbb{L}_-$, and $\mathbb{L}_{\mathtt{B}}$, storing the tags of positively scored, negatively scored, and blocked sessions, respectively. Authentication now runs in $O(|\mathbb{L}_+| + |\mathbb{L}_-| + |\mathbb{L}_{\mathtt{B}}|)$ time, which is even worse than BLAC's $O(L)$ time complexity, where $L = |\mathbb{L}_{\mathtt{B}}|$.

SNARKBlock [43] improves BLAC with an optimized zkSNARK-based proof protocol that allows reusing proofs against chunks of unchanged blocklist. If a session is later unblocked, all must recompute the proof w.r.t. the affected chunk.

Discouraging Infrequent Users. To tackle the linear complexity, BLACR was extended into BLACR-Express [5] with *checkpoints*. Suppose a user has passed an authentication after the i-th checkpoint in the blocklist; the SP marks the value i on the credential. Any future blocklist checking of this *updated* credential only needs to be done with respect to ΔL entries put into the blocklist after the i-th checkpoint. This discourages infrequent users from logging in since their authentication request before catching up with the latest checkpoint will be slower and stand out from the crowd, degrading their anonymity. Users are motivated to rush to get their express passes, forcing the SP to handle a burst of requests at each checkpoint, which might result in a denial of service.

Stringent Timelines for Blocklisting. PEREA [45] changes the authentication semantic for reducing the $O(L)$ complexity to $O(K)$, where K is the fixed number of *tickets* in a credential. Each authentication *replaces* the oldest of the K tickets with a new one, so misbehavior must be caught within the "revocation window" of K authentications. Since malicious users are motivated to wash off their bad records, it imposes a *stringent requirement* for the SP to evaluate each session *sequentially and timely.* The same applies to FAUST [38] and PERM [4]. In Table 1, PERM is marked to have no score consistency since the washing-out problem makes the credential score inconsistent with the judgment intention.

Table 1. Summary of Related Work: Halt-free systems should have complexity independent on any global list, without halting authentication due to a single hard-to-judge session; SNARKBlock is like BLAC, and PEREA is like PERM in this table.

Scheme	Halt-free	Score Consistency	Rate Limit	Cred. Update	Dynamic Judgment
BLAC	✓✗	✗	✗	✗	Full Flexibility
BLACR	✓✗	✓	✗	✓✗	Non-“Checkpointed”
PERM	✗	✗	✗	✓	Block-then-Forgive
PE(AR)^2	✓	✓✗	✗	✓	Block-then-Forgive
FARB	✗	✓	✓	✓	Block-then-Forgive
SAC	✓	✓	✓	✓	Before Finalization

Block First, Forgive Later. PERM and PE(AR)^2 [47], which do not prove w.r.t. the whole global list, only support score upgrades – an easy option as a user is motivated to claim them. Indeed, PE(AR)^2 does not force redeeming of negative scores by itself. Also, the SP might take strict “block-first-forgive-later” measures.

Unavoidably Halting Innocent Users. PERM [4] takes the revocation-window model further, which forces the session identifiers to increase sequentially. The goal is to remove the use of accumulator in PEREA [45] for reducing user computation (albeit with a slightly higher server-side cost) via using a *global* judgment pointer pointing to the latest authenticated session that has been *judged* (e.g., free from misbehavior). The eldest unjudged session of every credential should not be “too far away” from the global pointer for successful authentication. FARB [46] further exploits the sequential order assumption, which replaces K disjunctive proofs (judged or unjudged) with simply one range proof.

Such an authentication semantic forbids any user from redeeming the judged session in an arbitrary order (or it violates the increasing-identifier invariant). The existence of one hard-to-judge session will affect “innocent” users who have nothing to do with that session to an even greater extent than PEREA (which essentially maintains a local judgment pointer compared by counting).

Moreover, even when the global halting is “lifted,” *i.e.*, that controversial session is eventually finalized, all users are motivated to redeem all redeemable sessions in a rush, similar to the checkpoint design of BLACR-Express [5].

We stress that this is a design oversight of all these systems under the revocation-window model, creating a technical restriction incompatible with operational characteristics. In other words, ideally, the fact that some sessions take longer to judge should only affect their originators but not *all system users.*

1.2 Our Contribution

We propose scored anonymous credentials (SAC) with a number of features.

Avoiding Queue Structure or Disjunctive Proof. All K tickets a user holds are assigned with an initial score, e.g., 0, at their creation time. These tickets are marked as “active” and can only be removed from the credential when the SP finalizes them. This is in contrast to PERM/PEREA, which always removes the oldest ticket and adds a new one to the credential upon authentication.

To assure the SP of the ticket statuses in a credential, authentication requires proving that each ticket is either an *active* one with a volatile score, a judged one with a *finalized* score, or a dummy ticket. Instead of the apparent need for a disjunctive proof (extensively used in PERM/PEREA), we employ a signature-based set membership proof for each ticket (with its optimization and complexity to be described shortly) against a global ticket list with volatile/finalized scores.

Shuffling and Summing Up All Tickets. The user can redeem (and remove) any finalized or dummy tickets. To highlight a novelty of our new design, we do not use complicated zero-knowledge proof (ZKP) to explicitly force a user to redeem/claim a session that is finalized or with its volatile score downgraded. Instead, we use the same signature-based set membership proof to force the inclusion of all tickets (of any status) via a summation of all (volatile/finalized) scores. To privately prove/update this metadata, we employ verifiable shuffle and dummy tickets so the SP/user can judge/redeem tickets *in an arbitrary order.* To the best of our knowledge, no related systems so far use verifiable shuffle.

Solving the Global-halting Problem. As a *major innovation*, SAC deviates from the two prevalent designs – reliance on a *global* judgment pointer (e.g., PERM) or *global* checkpoints (e.g., BLACR), while still being efficient. The SP can keep a "problematic" session pending prolonged judgment "active" with a volatile score. A credential is only blocked from further authentication if it has K unjudged sessions. Users who are not involved with the problematic cases just authenticate as usual. This leads to "intelligent" rate-limiting *tracing back to originating users.*

Flexibility of Scoring. Supporting active sessions allows an assignment of volatile scores before finalization, making SAC scoring mechanism highly flexible and free from any chronological order of judgment. The SP could temporarily assign a negative score to the session in question, leading to a more natural and benign authentication semantic in which a user with a higher reputation (many prior positive scores) is less likely to be blocked abruptly due to one potential wrongdoing. Yet, once the wrongdoing is confirmed, the session status can be changed from active to finalized, and the user could still be (permanently) banned. Moreover, our efficient ZKP over the status of *all* sessions in a credential also enables *downgradable* scores, while prior works might support only block-then-forgive.

Flexible Anonymity-Efficiency Trade-off. The use of verifiable shuffle ensures an updated credential remains unlinkable to its old version sharing some old tickets. SAC thus features a *dynamic buffer size*[4] K. Users with fewer/more than K tickets can pad/remove dummy tickets during each authentication. Frequent users are more willing to choose a larger K, either actually storing many active sessions or having some of them being *dummy* sessions of score 0. Less frequent

[4] It is non-trivial to reduce the queue size of PEREA/PERM while preserving privacy.

users can opt for a smaller K, which leads to a faster authentication time. This is similar to ring signatures or related techniques [23], in which one can choose which user groups (cf., those using a particular suggested buffer size in SAC) to hide inside. As the SP knows the number of tickets to be removed, the users can agree to remove one (dummy or judged) ticket per authentication for anonymity.

High Efficiency via Optimized Cryptographic Techniques. As explained, a key idea to support redeeming of judged sessions in an *arbitrary order* is to leverage verifiable shuffle, a building block yet to be explored by the research line of TTP-free anonymous credentials with blocklist and reputation. This ensures that the set of sessions is correctly shuffled across the old credential and an updated one. A breakthrough of Bayer and Groth [9] features a "minimal" communication bandwidth of $O(\sqrt{K})$ for shuffling K commitments with almost linear complexity in proving and verification. Recent advances in (succinct) zero-knowledge argument of knowledge (e.g., Bulletproofs [14]) reduce the communication costs to $O(\log K)$ with higher computational costs for the prover and verifier.

Our construction also includes some other optimization for better efficiency. Recall that each authentication ZK-proves that some of its past tickets are in the active ticket list. Like PERM, we do not use pairing-based accumulators [2]; but instead of enforcing a sequential ticket order in a credential to simplify the membership proof, we use a constant-size signature-based set membership proof, albeit without incurring global halting caused by one controversial session.

To claim the score of judged tickets, PERM uses ZKP of signatures, which results in $O(K)$ pairings and exponentiations. We equip our signatures with *an efficient batch verification algorithm accompanied by its zero-knowledge version.* It takes $O(K)$ exponentiation and $O(1)$ pairing only, which greatly improves the efficiency. The use of disjunctive proofs in PERM prevents batching operations.

Finally, we empirically show that SAC outperforms existing related systems, given the simplicity of our design and the cryptographic techniques we employed.

Concurrent Work. SMART [39] extends FARB [46] into a multi-queue design to alleviate the time pressure of finalizing a hard-to-judge session. Each queue keeps tickets that have undergone the same number of transient judgments. A transient judgment takes its effect or is "redeemed" in each authentication, which moves the corresponding ticket from one queue to another. As a credential can be seen as multiple credentials (using the implicit queue design of FARB) bundled together, sequential judging is still enforced per queue but does not affect other queues. This reduces the global-halting effect of sequential judging with a linear cost in the number of queues. However, it requires a more complex ZKP to hide the queue number during redemption. On the other hand, SMART marks the "version" of each transient judgment independently, while enforcing score updates in SAC involves expiring all previously signed judgments and issuing new ones (Section 3.5). In Table 1, SMART shares the same characteristics as SAC.

2 Definitions

2.1 Syntax of Scored (or Blocklistable) Anonymous Credentials

We first formulate an abstract definition for anonymous credentials, modeling the two kinds of interactions between *users* and the *service provider* (SP). After the SP sets up the whole system, each user *registers* with the SP anonymously to obtain a credential. A user can then use it to *authenticate* to the SP. For SAC, the SP *updates* the public information to reflect the new score or status assigned to each session according to the action carried out by its user.

We use an existing notation [47] of $\mathsf{2PC}(\mathcal{U}(\mathsf{in}_{\mathcal{U}}), \mathcal{S}(\mathsf{in}_{\mathcal{S}})) \to \{\mathcal{U}(\mathsf{out}_{\mathcal{U}}), \mathcal{S}(\mathsf{out}_{\mathcal{S}})\}$ to represent parties $\mathcal{U}$ and $\mathcal{S}$ interact via the respective $\mathcal{U}$ and $\mathcal{S}$ portion of the 2-party computation protocol $\mathsf{2PC}$ taking in_i and outputting out_i for $i \in \{\mathcal{U}, \mathcal{S}\}$.

Definition 1. *Anonymous credentials involve the algorithms/protocols below.*

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{sk})$*: The SP inputs in a security parameter λ, and outputs a secret key* sk *for the SP itself and the public parameter* pp*.*

$\mathsf{Reg}(\mathcal{U}(\mathsf{pp}), \mathcal{S}(\mathsf{pp}, \mathsf{sk})) \to \{\mathcal{U}(\mathsf{cred}, \mathsf{A}), \mathcal{S}(\mathsf{pp}')\}$*: The user $\mathcal{U}$ inputs the public parameter* pp*, while the SP $\mathcal{S}$ takes* pp *and the secret key* sk *as input. Upon completion, the user outputs a credential* cred *and an attribute set* A*, while the SP outputs an updated public parameter* pp'*.*

$\mathsf{Auth}(\mathcal{U}(\mathsf{pp}, \mathsf{cred}, \mathsf{A}, f), \mathcal{S}(\mathsf{pp}, \mathsf{sk}, f)) \to \{\mathcal{U}(b, \mathsf{cred}', \mathsf{A}'), \mathcal{S}(b, \mathsf{pp}')\}$*: The user $\mathcal{U}$ inputs the public parameter* pp*, its credential* cred*, its attributes* A*, and the same access policy f, while the SP $\mathcal{S}$ inputs the public parameter* pp*, the secret key* sk*, and an access policy f. Upon completion, $\mathcal{U}$ outputs a bit b indicating whether authentication is successful, an updated credential* cred'*, and updated attributes* A'*, while $\mathcal{S}$ outputs the same bit b and an updated public parameter* pp'*.*

$\mathsf{Update}(\mathsf{pp}, \mathsf{sk}, \mathsf{aux}) \to \mathsf{pp}'$*: The SP inputs the public parameter* pp*, the secret key* sk*, and auxiliary information* aux*, capturing the score assigned to a certain session as recorded by* Auth *in* pp'*. It outputs an updated public parameter* pp'*.*

Prior Formulations. Earlier works [45,38,5,4] never explicitly give a syntactic framework of blockable anonymous credentials or those also supporting reputation. A notable exception is PE(AR)^2 [47], which features a standalone redemption protocol that updates a credential according to the SP-side update given by two separate algorithms for revocation and scoring (a session) run by the SP.

Our SAC formulation forces the redemption of any negative finalized score (not supported by PE(AR)^2) and hence encapsulates the redemption protocol in Auth, the authentication protocol. For a generic treatment, we also encapsulate all kinds of assignments of score/status into one Update algorithm run by the SP.

In contrast to the different designs of prior works, our definition is generic for an abstract authentication semantic purely based on an attribute set A as a user secret input and an authentication policy f as a common public input.

In what follows, we will also supplement details specific to SAC, e.g., what are in pp, with discussions of alternatives in some prior works.

Credential Attributes. In SAC (and systems like PERM/PEREA), the credential encoding attributes A stores session identifiers $\mathbb{U} = \{t_1, \ldots, t_K\}$ originated from each session (e.g., a forum post or a Wikipedia edit) and score(s) s to be checked against the access policies, e.g., whether they exceed certain thresholds.

Session Judgment. Public session judgments made by the SP are reflected in the updatable public parameter pp. In SAC, pp contains a list $\mathbb{L}_{\mathtt{A}} = \{(t_j, s_j, \sigma_j)\}$ of active judgments and a list $\mathbb{L}_{\mathtt{J}} = \{(t_j, \hat{\sigma_j})\}$ of finalized, judged tickets, where t_j is a ticket unique to each session and s_j is the session score. PERM maintains only $\mathbb{L}_{\mathtt{A}}$; while PEREA also maintains $\mathbb{L}_{\mathtt{J}}$ as a list of blocked tickets.

All entries are accompanied by σ_j or $\hat{\sigma_j}$, possibly different forms of signatures issued by the SP. The use of signature varies according to instantiations, which we omit for brevity in parts of our later discussion. Particularly, for SAC, the signatures in $\mathbb{L}_{\mathtt{J}}$ are for set membership proof. We often use a "compact" form $\mathbb{L} = \{(t_i, s_i, \sigma_i, \emptyset/\hat{\sigma_i}\}$ that combines $\mathbb{L}_{\{\mathtt{A},\mathtt{D},\mathtt{J}\}}$. For $t_i \in \mathbb{L}_{\mathtt{A}} \setminus \mathbb{L}_{\mathtt{J}}$, the last entry is $\emptyset$.

In alternative formulation (e.g., PEREA), $\mathbb{L}_{\mathtt{J}}$ could contain a single (public) cryptographic object, e.g., accumulator, instead of many signatures. $\mathbb{L}_{\mathtt{A}}$ might also be a cryptographic object storing key-value pairs.

Workflow. Below supplements details of SAC calling the (abstract) algorithms:

1. ($\mathsf{SAC.Setup}$) The service provider (SP) setups the credential system.
   – It runs $\mathsf{KGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{sk})$, where sk is the secret credential-issuance key.
   – For public parameters pp, its static part contains the cryptographic parameters and the public (verification) key corresponding to sk. It also defines default values like the list $\mathbb{L}_{\mathtt{D}}$ of dummy tickets, and the base score s_0 for the initial attributes A_0. Its dynamic part contains initially empty lists $\mathbb{L}_{\mathtt{A}}$ and $\mathbb{L}_{\mathtt{J}}$ of active and judged tickets with scores, and a set for recording the nonce revealed by the user during authentication.
2. ($\mathsf{SAC.Reg}^{\mathcal{U}} \leftrightarrow \mathsf{SAC.Reg}^{\mathcal{S}}$) A user registers to the SP to obtain a credential.
   – The user and SP run $\mathsf{Reg}(\mathcal{U}(\mathsf{pp}), \mathcal{S}(\mathsf{pp}, \mathsf{sk})) \to \{\mathcal{U}(\mathsf{cred}, \mathsf{A}_0), \mathcal{S}(\mathsf{pp}')\}$.
   – The initial attributes $\mathsf{A}_0 = (x, q, s_0, \mathbb{U})$ contain user secret x, a nonce q (both kept secret from the SP), an initial score s_0, and a user ticket set $\mathbb{U}$, which will be initialized with a set of dummy tickets $\mathbb{U}_{\mathtt{D}}$ (as defined by pp).
3. ($\mathsf{SAC.Auth}^{\mathcal{U}} \leftrightarrow \mathsf{SAC.Auth}^{\mathcal{S}}$) A user who holds a valid credential from $\mathsf{SAC.Reg}$ authenticates via $\mathsf{Auth}(\mathcal{U}(\mathsf{pp}, \mathsf{cred}, \mathsf{A}, f), \mathcal{S}(\mathsf{pp}, \mathsf{sk}, f))$, where policy f checks:
   – Nonce q has not been previously used by any authentication.
   – Judged tickets in $\mathbb{U}_{\mathtt{J}} = (\mathbb{U} \cap \mathbb{L}_{\mathtt{J}})$ are (required[5] to be) cleared out.
   – The credential score s plus the score of all tickets in $\mathbb{U}$ satisfies f.
   It outputs $\perp$ if the check fails; otherwise, it updates the credential attributes:
   – Judged or dummy ticket(s) are removed from $\mathbb{U}$. (Users can redeem any number of dummy tickets, *i.e.*, the set $\mathbb{T}$ containing tickets from $(\mathbb{U}_{\mathtt{D}} \cup \mathbb{U}_{\mathtt{J}})$.)
   – Nonce q is replaced by a new one (unknown to the SP) chosen by the user.
   – Their scores are accumulated to s.
   – A new ticket t, chosen by the SP, is appended to the user's credential.

[5] In PERM/PEREA, the oldest ticket would be removed even if it is not yet judged.

– Dummy tickets are padded to maintain $|\mathbb{U}| = K$ (if needed).
The SP assigns t with an initial score 0 and updates $\mathbb{L}_{\mathtt{A}}$ with (t, s, σ) applied as an active (and unjudged) session. The SP also records q in pp'.

4. ($\mathsf{SAC.Update}$) The SP can judge the ticket t with a score s using its key by running $\mathsf{Update}(\mathsf{pp}, \mathsf{sk}, (t, s)) \to \mathsf{pp}'$ that appends (t, s) (with a signature) to list $\mathbb{L}_{\mathtt{A}}$. Alternatively, it can also finalize t by adding it to list $\mathbb{L}_{\mathtt{J}}$. The list of signatures on (t, s) (and t) is made public and is managed by the SP.

2.2 Security Requirements

We consider the requirements of blocklistable anonymous credentials [45,38,5,4]. At a high level, SAC should satisfy the following properties:

- **Completeness.** An honest user can be authenticated by an honest SP if its credential satisfies the access policy, *i.e.*, $f_{\mathsf{pp}}(\mathsf{A}) = 1$.
- **Soundness.** A user must hold a valid credential encoding A, *i.e.*, $f_{\mathsf{pp}}(\mathsf{A}) = 1$, to authenticate. The updated attribute A' follows specifications in Auth.
- **Anonymity.** The SP can only learn whether an authenticating user satisfies the authentication policy. The SP cannot distinguish the authentication requested by user $i \in \{0, 1\}$ with attributes A_i if $f_{\mathsf{pp}}(\mathsf{A}_0) = f_{\mathsf{pp}}(\mathsf{A}_1)$.

Soundness. We consider soundness against malicious users similar to and largely relies on the soundness of the underlying ZKP. It covers scoring consistency, *i.e.*, a malicious user cannot claim a ticket owned by others or a wrong score.

In SAC, although a user can choose to "procrastinate" in claiming the score of judged sessions $\mathbb{U} \cap \mathbb{L}_{\mathtt{J}}$, the scores of all tickets in $\mathbb{U}$ are still counted toward the authentication policy. All tickets are initially in an active state, e.g., $(t_i, s_{t_i}, \sigma_{t_i}) \in \mathbb{L}_{\mathtt{A}}$ for all tickets t_i in the system. Scoring consistency requires the score s' computed as a sum of users' current score s (taking one attribute slot in the credential) and those of the past tickets $\sum_{t \in \mathbb{U}} s_t$ is consistent with $\mathbb{L}_{\mathtt{A}}$ and $\mathbb{L}_{\mathtt{J}}$.

Anonymity. Anonymity is defined against a passive SP (strictly stronger than eavesdroppers) trying to deanonymize a user who is invoking an authentication instance. Due to the correct functionality, the authentication policy f can distinguish whether (the credential of) a user satisfies the required score threshold. The best anonymity guarantee only holds modulo what is inferrable from f.

Active attacks manipulating f or the score of a session are excluded. For example, the SP, leveraging its role, could put a session of question to the blocklist for identifying if the user being authenticated has originated the now blocked session. Such manipulations are noticeable publicly and leave evidence. A user can refuse to carry out the authentication and possibly complain against the SP.

In more detail, SAC is anonymous if any adversary can only win the anonymity game below with probability negligibly better than a random guess:

1. Adversary $\mathcal{A}$ runs Setup and outputs the public parameter pp.
2. $\mathcal{A}$ can instruct a user controlled by challenger $\mathcal{C}$ to run Reg with $\mathcal{A}$. If the protocol outputs a valid credential, $\mathcal{C}$ stores it with a unique user identifier.

3. $\mathcal{A}$ can instruct registered users controlled by $\mathcal{C}$ to run Auth over a policy f with $\mathcal{A}$ as the SP. If the user's credential does not pass f, $\mathcal{C}$ outputs $\perp$ to $\mathcal{A}$.
4. $\mathcal{A}$ picks two users u_0, u_1, and sends them to $\mathcal{C}$.
5. $\mathcal{C}$ checks if both u_0, u_1 pass the policy check f. If so, it flips a coin $b \in \{0,1\}$ and runs Auth using u_b's credential, else it outputs $\perp$.
6. $\mathcal{A}$ wins if it guesses b correctly and $\mathcal{C}$ does not output $\perp$ in Step 5.

One might consider unlinkability, in which the SP cannot tell whether the same user is authenticating. The unlinkability game can be captured by having the adversary pick a user u in Step 4, and the challenger randomly authenticates with a random valid user or u. Intuitively, it is captured by anonymity since the SP only learns if $f_{\mathsf{pp}}(\mathsf{A})$ returns 1 during authentication. This is similar to the equivalence between "left-or-right" and "real-or-random" formulations.

Alternatively, one could formulate a simulation-based definition, requiring the transcript of different Auth instances to be uncorrelated from those of the same user or the Reg instance. More formally, it asks for a probabilistic polynomial-time simulator that can simulate the view of a corrupted SP in Auth. The ideal functionality is in Appendix D.

3 Proposed System

3.1 Building Blocks

Zero-Knowledge Proof-of-Knowledge (ZKPoK). We use a ZKPoK system with correctness, soundness, knowledge extraction, and zero-knowledgeness. It involves a three-move commit-challenge-response Σ-protocol. In the random oracle model, it can be converted into non-interactive signatures/proofs of knowledge.

We use the notation from Camenisch and Stadler [20], like $\mathsf{PoK}\{(\alpha, \rho) \colon z = g^{\alpha} h^{\rho}\}$, to denote such proof of (α, ρ) where $z = g^{\alpha} h^{\rho}$ holds. Multiple ZKPoK protocols could be chained into a bigger one for multiple conditions. Compared to ZKPoK, a zero-knowledge argument of knowledge (ZKAoK) is a proof system that satisfies soundness property against any computationally-bounded prover.

Set Membership Proof. Given a commitment $C = g^i h^{\rho}$ with $g, h \in \mathbb{G}_1$ to a value i and randomness ρ, a set membership proof is a ZKPoK that i belongs to some discrete set Φ. The proof $\mathbb{P}_{\mathsf{Set}}$ of Camenisch *et al.* [16] uses the selectively-secure Boneh–Boyen (BB) signature [11]. The SP with the signature key pair $(x, w = f^{\beta}) \in \mathbb{Z}_p \times \mathbb{G}_2$ alongside with $(g', g'^{\beta}) \in \mathbb{G}_1^2$ (required by the underlying simulator) publishes signatures $\hat{\sigma}_i = g^{\frac{1}{\beta+i}} \in \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$ on values i in set Φ. The prover picks $r \in_R \mathbb{Z}_p$, and computes $V = \hat{\sigma}_i^r$ and $V' = V^{-i} g^r$. The proof consists of $(V \neq 1_{\mathbb{G}_1}, V')$ and the PoK: $\mathsf{PoK}\{(i, r, \rho) : C = g^i h^{\rho};\ V' = V^{-i} g^r\}$ as follows.

1. The prover picks $s, t, u \in_R \mathbb{Z}_p$ and sends $a = V^{-s} g^t, D = g^s h^u$ to the verifier.
2. The verifier returns a random challenge $c \in_R \mathbb{Z}_p$.
3. The prover sends $z_i = s - ic$, $z_r = t - rc$, and $z_\rho = u - \rho c$.
4. The verifier checks if $a = V'^c V^{-z_i} g^{z_r}$, $D = C^c g^{z_i} h^{z_\rho}$, and $\hat{e}(V, w) = \hat{e}(V', f)$.

In SAC, we also use BB signature to certify the judged status of a session.

Range Proof. A range proof can be viewed as a special case of set membership proof [15] by defining the set as the range, say, integers in $[A, B]$. This imposes an upper bound value, so we do not need to handle the wrap-around issue in $\mathbb{Z}_p$ even though p is public.This range-proof system is also simple and efficient.

Bulletproofs. Bulletproofs proposed by Bünz *et al.*. [14] is a non-interactive zero-knowledge proof protocol without a trusted setup, featuring a proof size only logarithmic in the witness size. Bulletproofs are well suited for proofs for general arithmetic circuits and inner-product relations. Range proof over the interval $[0, 2^n)$ can be done using inner product arguments over a committed value v, *i.e.*, $\mathsf{PoK}\{(v, \rho) : C = g^v h^\rho \wedge (0 \leq v < 2^n)\}$. Two interval proofs can be combined as a range proof on the arbitrary range $[A, B]$ using a standard trick [15]. At a high level, for $2^{b-1} < B < 2^b$, the prover proves $v - A$, $v - B + 2^b$ are in $[0, 2^b)$. For SAC, we also set the upper limit (say, $[0, 2^{64} - 1]$) to cover possible scores.

Zero-knowledge Argument of a Shuffle. A shuffle of commitments $\{C_1, \ldots, C_N\}$ of messages $\{a_1, \ldots, a_N\}$ is a set of commitments $\{C'_1, \ldots C'_N\}$ of $\{b_1, \ldots, b_N\}$ committing to the same set of messages but in a permuted order, *i.e.*, $b_i = a_{\pi(i)}$ for some permutation $\pi : [N] \to [N]$. If we treat the messages as the roots of two polynomials of degree N, one can test for a random z (can be picked by the verifier) if $\prod_{i=1}^{N}(a_i - z) = \prod_{i=1}^{N}(b_i - z)$ holds for AoK of permutation. As an arithmetic circuit, this requires $2(N - 1)$ multiplication gates and is readily supported by the Bulletproofs. The prover and verifier computation are both linear in N but with logarithmic proof size (excluding the commitments). Appendix B reviews the ZK shuffle argument by Bayer and Groth [9] as an alternative.

Verifiable shuffle is used in SAC authentication after the client's list of session identifiers is updated. Its purpose is to provide anonymity to the user.

BBS+ Signatures. BBS+ signature of Au, Susilo, Mu, and Chow [6] extends the BBS signature of Boneh, Boyen, and Shacham [12] with multiple message blocks and efficient ZK protocols for signing and verification. It is existentially unforgeable against adaptive chosen message attacks under the q-SDH assumption over pairing groups [16] with no efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Let $h_0, h_1, \ldots, h_{\ell+1}$ be generators of $\mathbb{G}_1$ and f be a generator of $\mathbb{G}_2$. The signer's secret key is $\gamma \in \mathbb{Z}_p$ and the public key is $w = f^\gamma$. To sign on blocks of messages $(m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$, the signer randomly picks $e, y \in \mathbb{Z}_p$ and computes $A = (h_0 h_1^{m_1} \cdots h_\ell^{m_\ell} h_{\ell+1}^y)^{\frac{1}{\gamma+e}}$. The signature is (A, e, y), and one can verify it by checking if $\hat{e}(A, wf^e) = \hat{e}(h_0 h_1^{m_1} \cdots h_\ell^{m_\ell} h_{\ell+1}^y, f)$ holds.

BBS+ signatures mostly serve as credentials in many privacy-preserving systems. In SAC, they also link the ticket with its score.

Protocol $\mathbb{P}_{\mathsf{Iss}}$. It allows a user to obtain a signature from the signer on a block of values $(m_1, \ldots, m_\ell)$ without revealing them. It is used in SAC registration and credential update during authentication.

Table 2. Major Notations for User-side and SP-side Data Structures

Notation	Description
x, q, s	user long-term secret key, nonce used for authentication, credential score
cred, σ	credential over $\mathsf{A} = (x, q, s, \mathbb{U})$, signature from the SP, e.g., on A
$\mathbb{U}$	list of session identifiers (or tickets) kept by a user's credential
$\mathbb{L}$	list of sessions ($(t_i, s_i, \sigma_i, \mathtt{I})$ tuples) maintained and published by the SP
$\mathbb{L}_{\mathtt{I}}$, $\mathbb{U}_{\mathtt{I}}$	sub-lists of $\mathbb{L}$ or $\mathbb{U}$ containing sessions of a specific status $\mathtt{I} \in \{\mathtt{J}, \mathtt{A}, \mathtt{D}\}$, meaning `Judged`, `Active`, or `Dummy`, respectively

1. The user computes $C_M = h_1^{m_1} h_2^{m_2} \cdots h_\ell^{m_\ell} h_{\ell+1}^{y'}$ for some randomly generated $y' \in \mathbb{Z}_p$, and sends C_M to the signer with the following proof:

$$\mathsf{PoK}\left\{(\{m_i\}_{i\in[1,\ell]}, y') : \quad C_M = h_1^{m_1} h_2^{m_2} \cdots h_\ell^{m_\ell} h_{\ell+1}^{y'}\right\}.$$

2. The signer aborts if the proof does not verify; otherwise, randomly picks $e, y'' \in \mathbb{Z}_p$, computes $A = (h_0 C_M h_{\ell+1}^{y''})^{\frac{1}{e+\gamma}}$, and returns (A, e, y'') to the user.
3. The user aborts if $\hat{e}(A, wf^e) \neq \hat{e}(h_0 h_1^{m_1} \cdots h_\ell^{m_\ell} h_{\ell+1}^{y'+y''}, f)$. Otherwise, the user outputs σ as $(A, e, y = y' + y'')$.

Protocol $\mathbb{P}_{\mathsf{Sig}}$. It enables proving the knowledge of a signature $\sigma = (A, e, y)$ on message blocks $(m_1, \ldots, m_\ell)$ without revealing the signature nor the messages. The prover can also disclose messages contained in $\mathbb{D} \subset \{m_1, \ldots, m_\ell\}$. The prover randomizes the signature with $\rho_1 \in \mathbb{Z}_p^*$ by setting $A' = A^{\rho_1}$. It also computes $b = h_0 h_{\ell+1}^s \prod_{i=1}^{\ell} h_i^{m_i} = A^{\gamma+e}$, $\bar{A} = A'^{-e} \cdot b^{\rho_1}$, and $\rho_3 = 1/\rho_1$. It then picks $r_2 \in \mathbb{Z}_p$ and sets $d = b^{\rho_1} \cdot h_{\ell+1}^{-\rho_2}$ and $s' = s - \rho_2 \cdot \rho_3$. The prover computes:

$$\mathsf{PoK}\{(\{m_j\}, e, \rho_1, \rho_2, \rho_3, s') : \bar{A}/d = h_{\ell+1}^{\rho_2}/A'^e \wedge h_0 \prod_{m_i\in\mathbb{D}} h_i^{m_i} \prod_{m_j\notin\mathbb{D}} h_i^{m_j} = d^{\rho_3} h_{\ell+1}^{-s'}\}.$$

The above proof is performed as follows:

1. The prover picks $r_e, r_1, r_2, r_3, r_{s'}, r_{m_1}, \ldots, r_{m_\ell} \in \mathbb{Z}_p$ and sends to the verifier $R_1 = A'^{-r_e} h_{\ell+1}^{r_2}$ and $R_2 = d^{r_3} h_{\ell+1}^{-r_{s'}} \prod_{m_j\notin\mathbb{D}} h_i^{-r_{m_j}}$.
2. The verifier returns a random challenge $c \in_R \mathbb{Z}_p$.
3. The prover sends $z_e = r_e - ce$, $z_{s'} = r_{s'} - cs'$, $z_i = r_i - c\rho_i$ for $i \in [1, 3]$, and $z_{m_j} = r_{m_j} - cm_j$ for j such that $m_j \notin \mathbb{D}$.
4. The verifier checks $\frac{\bar{A}}{d} = R_1^c A^{-z_e} h_{\ell+1}^{z_2}$, $h_0 \prod_{m_i\in\mathbb{D}} h_i^{m_i} = R_2^c h_{\ell+1}^{-z_{s'}} / \prod_{m_j\notin\mathbb{D}} h_i^{z_{m_j}}$.

The proof consists of $(A', \bar{A}, d, \pi)$ and can be verified by checking $A' \neq 1_{\mathbb{G}_1}$ and $\hat{e}(A', w) = \hat{e}(\bar{A}, f)$. The signer needs to publish $(\bar{g}, \bar{g}^\gamma)$ for $\bar{g} \neq 1_{\mathbb{G}_1}$ (for the zero-knowledge simulator). It is used in SAC authentication for proving the credential on attribute sets and proving the score of each authenticated session.

3.2 Key Ideas

Setup. In Setup, the SP runs the key generation of all underlying signature schemes (the choices will be specified in Section 3.3). All the signing keys are put to the secret key sk, and the public keys are put to the public parameter pp, along with the list $\mathbb{L}$, and system parameters including the threshold score s_{th}, score $-S_{\mathsf{max}}$ for blocklisting, and the buffer sizes $\mathbb{K}$ with K_{max} being the maximum.

List $\mathbb{L}$ contains tuples for the sessions, which includes:

– a set $\mathbb{L}_{\mathtt{D}}$ of dummy sessions, each with a score of 0,
– a set $\mathbb{L}_{\mathtt{A}}$ for storing active sessions with (dynamically) rated scores, and
– a set $\mathbb{L}_{\mathtt{J}}$ for judged sessions with finalized, judged scores.

Table 2 lists the major notations. Only $\mathbb{L}$ in the table is public. Each entry of $\mathbb{L}$ is of the form $(t_i, s_i, \sigma_{t_i,s_i}, \mathtt{state})$, storing a session identifier t_i, its score s_i, a signature σ_{t_i,s_i} (for the integrity of $\mathbb{L}$), and its state information $\mathtt{state}$, which is

– an empty string if the state is *active*, or
– a signature on t_i for *judged* or *dummy* (so later authentication can redeem it).

Suppose $s_{\mathsf{th}} = 0$. Initially, the SP puts K_{max} dummy sessions of score 0, each in the form of $(t_i, 0, \sigma, \hat{\sigma})$, into $\mathbb{L}_{\mathtt{D}}$, where σ signs on $(t_i, 0)$ and $\hat{\sigma}$ signs on t_i.

Registration. In Reg, each eligible user chooses a credential size $K \in \mathbb{K}$, randomly chooses two $\mathbb{Z}_p$ values as the long-term secret x and the (first) nonce q for showing the freshness of the credential, and prepares an attribute set of size $(K+3)$ as $\mathsf{A} = (x, q, s = 0, t_1, \ldots, t_K)$, where s is the (initial) score of the user, and $\mathbb{U}_{\mathtt{D}} = (t_1, \ldots, t_K)$ is a subset of dummy session identifiers from $\mathbb{L}_{\mathtt{D}}$ in pp.

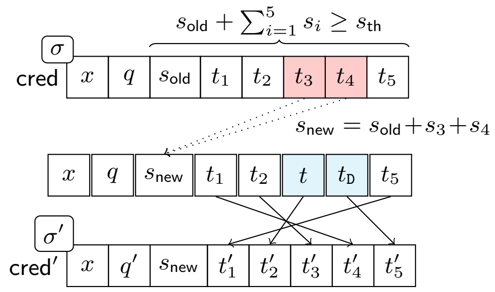
Let $\mathbb{U}$ denote the set of session identifiers kept by a credential. Just after registration, $\mathbb{U} = \mathbb{U}_{\mathtt{D}}$. This will make a newborn credential indistinguishable (in size) from ones with the same number of sessions while some of them are active or judged. The user proves in ZK that attribute set A is well-formed:

– Knowledge of x, q, which are hidden from the SP;
– $s = 0$ (or any base score agreed between the user and the SP);
– $(t_1, \ldots, t_K)$ are dummy tickets, by showing $t_i \in \mathbb{L}_{\mathtt{D}}$ (via PoK of signatures).

The SP then completes the signature issuance via the ZK signature issuance protocol $\mathbb{P}_{\mathsf{Iss}}$, which produces a signature σ_{A} on A, serving as a user credential cred.

Authentication. In Auth, the user ZK-proves that credential cred on attribute set $\mathsf{A} = (x, q, s, t_1, \ldots, t_K)$ satisfies the access policy (to be made explicit in Section 3.3), except q is revealed in clear. Let $\mathbb{U} = (t_1, \ldots, t_K)$. The proof consists of:

– *Credential Validation.* The user ZK-proves via $\mathbb{P}_{\mathsf{Sig}}$ that it has a signature σ_{A} on A, but reveals nonce q to show that it is not a replay of past credentials.
– *Score Satisfaction.* For set $\mathbb{U}$ of size K in A certified by cred, the user proves in ZK, for each $t_a \in \mathbb{U}$, that the session t_a has score s_{t_a} as specified in list $\mathbb{L}$ by proving via $\mathbb{P}_{\mathsf{Sig}}$ $\sigma_{t_a,s_{t_a}}$ is a signature on (t_a, s_{t_a}). This proof can be done in a batch for efficiency. The user can then prove in ZK that $s + \sum_{t \in \mathbb{U}} s_t > s_{\mathsf{th}}$ for some agreed threshold s_{th}. The SP cannot learn $\mathbb{U} = \{t_a\}$.



$s_{\mathsf{old}} + \sum_{i=1}^{5} s_i \geq s_{\mathsf{th}}$
σ
cred
x q s_{old} t_1 t_2 t_3 t_4 t_5
$s_{\mathsf{new}} = s_{\mathsf{old}} + s_3 + s_4$
x q s_{new} t_1 t_2 t t_{D} t_5
σ'
cred′
x q' s_{new} t'_1 t'_2 t'_3 t'_4 t'_5

Fig. 1. Suppose t_3 is judged and t_4 is dummy. The user authentication now redeems $\mathbb{T} = \{t_3, t_4\}$ to get a new score s_{new}. A ticket t for this session is added to the credential. The user (with help from the SP) adds a dummy ticket t_{D} to maintain the credential size. The user proves that $(t_1, t_2, t, t_{\mathsf{D}}, t_5)$ is shuffled to $(t'_1, t'_2, t'_3, t'_4, t'_5)$ where the SP remains oblivious to the session type of all the t_i's and information like whether they are old.

After some authentications and before any judgment is finalized, $\mathbb{U}$ transforms to $\mathbb{U}_{\mathtt{A}} \cup \mathbb{U}_{\mathtt{D}}$, with some active tickets added. Over time, the SP will mark some sessions as judged. $\mathbb{U}$ then transforms to $\mathbb{U}_{\mathtt{J}} \cup \mathbb{U}_{\mathtt{A}} \cup \mathbb{U}_{\mathtt{D}}$ (similar to the SP-side public session list $\mathbb{L} = \mathbb{L}_{\mathtt{J}} \cup \mathbb{L}_{\mathtt{A}} \cup \mathbb{L}_{\mathtt{D}}$). Also, a user may adjust $\mathbb{U}_{\mathtt{D}}$ to make the size of $\mathbb{U}$ equal to some other allowed value $K' \in \mathbb{K}$.

The protocol should also ensure that the credential is updated faithfully:

– *Credential Update.*
  1. The SP chooses and publishes t' as the current session identifier.
  2. The user picks a set of tickets $\mathbb{T} \subseteq \mathbb{U}$ and uses the set membership proof $\mathbb{P}_{\mathsf{Set}}$ to ZK-prove that they are either judged or dummy, e.g., $\mathbb{T} \subset \mathbb{L}_{\mathtt{J}} \cup \mathbb{L}_{\mathtt{D}}$.
  3. The user updates $\mathbb{U}$ to $\mathbb{U}' = (\mathbb{U} \setminus \mathbb{T}) \cup \mathbb{U}'_{\mathtt{D}} \cup \{t'\}$, where $\mathbb{U}'_{\mathtt{D}} \subset \mathbb{L}_{\mathtt{D}}$ pads[6] the number of sessions of $\mathbb{U}'$ to K', for a possibly new size K' (for $K' \in \mathbb{K}$).
  4. The user picks a fresh random nonce q' and proves in ZK that the shuffled $\mathbb{U}'$ and the new aggregated score s' are well-formed w.r.t. sessions in $\mathbb{T}$.
  5. The SP and user engage in credential issuance protocol $\mathbb{P}_{\mathsf{Iss}}$ for a new credential cred' on the new attributes $\mathsf{A}' = (x, q', s', \mathbb{U}')$.

  For the new $\mathbb{U}'$ with $\mathbb{U}'_{\mathtt{D}}$ and t' added and $\mathbb{T}$ removed, verifiable shuffle is applied to permute their order, *i.e.*, the positions of $\mathbb{T}$ in $\mathbb{U}$. Figure 1 depicts an example where the updated ticket set $(t_1, t_2, t, t_{\mathtt{D}}, t_5)$ is verifiably shuffled.

Finally, the SP updates pp by adding an entry $(t', 0, \sigma_{t',0}, \emptyset)$ into list $\mathbb{L}$, meaning that session t' has an initial score 0 (or any default) and an active state.

Session Score Update. In Update, the SP rates a session t by updating an entry $(t, s_{\mathsf{old}}, \sigma_{t,s_{\mathsf{old}}}, \cdot)$ in list $\mathbb{L}$ into $(t, s_{\mathsf{new}}, \sigma_{t,s_{\mathsf{new}}}, \cdot)$, meaning that the new score of the session t is s_{new}, which can be less than s_{old}, negative, or even $-(S_{\mathsf{max}} + 1)$

[6] Dummy sessions are judged with a default (zero) score, allowing users to pad $\mathbb{U}$ with some dummy tickets from $\mathbb{U}$ to $\mathbb{T}$ to hide the judged ones among them.

for revoking a credential, where S_{max} is the maximum possible score of any credential[7] (and the threshold for maintaining unrevoked status is 0). To finalize a session t, the SP updates $(t, \cdot, \cdot, \emptyset)$ to $(t, \cdot, \cdot, \hat{\sigma})$ in $\mathbb{L}$, where $\hat{\sigma}$ is a signature on t.

A user can keep a ticket t in its credential as long as it can provide a valid membership proof of t against $\mathbb{L}_{\mathtt{A}}$. If t is finalized, the score for t in both $\mathbb{L}_{\mathtt{A}}$ and $\mathbb{L}_{\mathtt{J}}$ should be consistent. This way, the user credential score is correctly accounted for during authentication, and keeping tickets would not harm the system.

3.3 Instantiation

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{sk})$: On input of the security parameter λ, the SP generates public parameter pp and the secret key sk as follows:

1. The SP chooses the bilinear map context over groups of prime order p (a $\mathsf{poly}(\lambda)$-bit prime) with pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
2. The SP creates the key for BBS+ signature by randomly picking $\gamma \in \mathbb{Z}_p$, generators $h_0, h_1, \ldots, h_{K_{\mathsf{max}}+4} \in \mathbb{G}_1$ and $f_0 \in \mathbb{G}_2$, and computing $w_0 = f_0^\gamma$.
3. The SP creates the keys for set membership proofs by randomly picking $\beta \in \mathbb{Z}_p$, generators $g \in \mathbb{G}_1$, $f_1 \in \mathbb{G}_2$, and computing $w_1 = f_1^\beta$.
4. The SP maintains a list of sessions $\mathbb{L} = \{(t, s, \sigma, \mathtt{state})\}$, where t is the session identifier, s is the score, σ is the BBS+ signature on (t, s), and $\mathtt{state}$ is the status of the session (either an empty string for *active*, or a Boneh-Boyen signature on t, for *judged* or *dummy*). The SP initializes $\mathbb{L}$ by adding K_{max} dummy sessions $(t, 0, \sigma, \hat{\sigma})$ with a different identifier t, where:
   - $\sigma = (A = (h_0 h_1^t h_3^y)^{\frac{1}{\gamma+e}}, e, y)$ is a BBS+ signature for random $e, y \in \mathbb{Z}_p$.
   - $\hat{\sigma} = g^{\frac{1}{\beta+t}}$ denotes a Boneh-Boyen signature on t.
5. The SP runs $\mathcal{K}(1^\lambda)$, the algorithm for generating the common reference string crs for the ZKPoK protocol PoK (to be defined below). In particular, it contains the set of Boneh-Boyen signatures $\{\hat{\sigma}_i\}$ needed by the signature-based membership proof $\mathbb{P}_{\mathsf{Set}}$.
6. The SP sets secret key $\mathsf{sk} = (\gamma, \beta)$ and public parameters
$$\mathsf{pp} = (\mathsf{crs}, g, h_0, \ldots, h_{K_{\mathsf{max}}+4}, f_0, f_1, w_0, w_1, \mathbb{K}, S_{\mathsf{max}}, \mathbb{L}).$$

$\mathsf{Reg}(\mathcal{U}(\mathsf{pp}), \mathcal{S}(\mathsf{pp}, \mathsf{sk})) \to \{\mathcal{U}(\mathsf{cred}, \mathsf{A}), \mathcal{S}(\mathsf{pp}')\}$: The user obtains a credential from the SP via an authenticated channel as follows:

1. (Preparation:) The user randomly picks $x', q \in \mathbb{Z}_p$, and selects $K \in \mathbb{K}$. The user prepares attributes of size $K+3$ as $\mathsf{A} = (x', q, s = 0, t_1, \ldots, t_K)$, where $t_i \in \mathbb{L}_{\mathtt{D}}$ for $i \in [1, K]$. The user generates the commitment $C'_M = h_1^{x'} h_2^q h_4^{t_1} \cdots h_{K+3}^{t_K} h_{K+4}^{y'}$ for some random $y' \in \mathbb{Z}_p$ and sends C'_M to the SP.
2. (Signing:) The SP randomly picks $x'' \in \mathbb{Z}_p$, computes $C_M = C'_M \cdot h_1^{x''}$, and sends x'' to the user. The user secret is then computed by $x = x' + x''$. They then engage in the protocol $\mathbb{P}_{\mathsf{Iss}}$ of the BBS+ signature for the commitment C_M using the public key $(h_0, \ldots, h_{K+4}, f_0, w_0)$. In the end, the user obtains a BBS+ signature σ_{A}. The public parameter $\mathsf{pp}' = \mathsf{pp}$ remains unchanged.

[7] The SP can derive it from K_{max} with an appropriate upper limit for each session.

3. (Credential Generation:) The user stores the credential $\mathsf{cred} = \sigma_{\mathsf{A}}$ and the attributes $\mathsf{A} = (x, q, s = 0, \mathbb{U} = \{t_1, \ldots, t_K\})$.

$\mathsf{Auth}(\mathcal{U}(\mathsf{pp}, \mathsf{cred}, \mathsf{A}, f), \mathcal{S}(\mathsf{pp}, \mathsf{sk}, f)) \rightarrow \{\mathcal{U}(b, \mathsf{cred}', \mathsf{A}'), \ \mathcal{S}(b, \mathsf{pp}')\}$: With cred on attributes $\mathsf{A} = (x, q, s, \mathbb{U} := \{t_1, \ldots, t_K\})$, the user attempts to prove that a credential with a score above s_{th} given in the access policy f as follows:

1. (Nonce revelation:) The user reveals q privately to the SP. If q is fresh, the SP picks t' and publishes it as the identifier of the *new (active) session* for this user authentication. Otherwise, the SP aborts as it is a replay.
2. (Proof about authentication requirements:) To prove to the SP, the user runs a combined ZKPoK PoK, which is a conjunction of several ZKPoK's:
   (a) $\mathbb{P}_{\mathsf{Sig}}$: knowing a signature σ_{A} on the attributes $\mathsf{A} = (x, q, s, \{t_1, \ldots, t_K\})$;
   (b) $\mathbb{P}_{\mathsf{Sig}}$: knowing $s_{t_i}, \sigma_{t_i, s_{t_i}}$ where $\sigma_{t_i, s_{t_i}}$ is a BBS+ signature on the ticket t_i and the score s_{t_i}, for $t_i \in \mathbb{U}$;
   (c) Range proof[8]: $s_{\mathsf{th}} \leq s + \sum_{i=1}^{K} s_{t_i} \leq S_{\mathsf{max}}$.

   Note that the range proof considers the *current* scores for all tickets irrespective of their status, even if a downgraded score has not been redeemed.
3. (Updated credential:) The combined ZKPoK PoK also proves about knowing:
   (a) (many copies of) $\mathbb{P}_{\mathsf{Set}}$: a set of *judged* or *dummy* session identifiers $\mathbb{T}$, via a signature-based membership proof that for all $t_i \in \mathbb{T}$, there exists a Boneh-Boyen signature $\hat{\sigma}_i$ (only presents for a judged or dummy session);
   (b) commitment on a new nonce $q' \in \mathbb{Z}_p$;
   (c) $\hat{C}_i = g^{t_i} h_0^{r_i} \ \forall t_i \in \mathbb{U}'$, where $(\mathbb{U}', s') \leftarrow \mathsf{Redeem}(\mathbb{U} \cup \{t'\}, \mathbb{T}, K')$ to be defined shortly, and $K' \in \mathbb{K}$, which the user can choose to have $K' = K$.
   (d) $\hat{C}'_i$ is a shuffle π of $\hat{C}_i$, which means $\hat{C}'_i = g^{t_{\pi(i)}} h_0^{r'_i}$ for all $t_i \in \mathbb{U}'$;
   (e) C'_M is a new commitment on $(x, q', s{+}s', \hat{\mathbb{U}})$, where $\hat{\mathbb{U}} = \{t_{\pi(1)}, \ldots, t_{\pi(K')}\}$.

   $\mathsf{Redeem}(\mathbb{U}, \mathbb{T}, K)$ is for redeeming scores in $\mathbb{T}$ within $\mathbb{U}$ and puts dummy sessions into $\mathbb{U}'$ if needed to make $|\mathbb{U}'| = K'$. Note that $\mathbb{U}' = \mathbb{U} \cup \{t'\} \setminus \mathbb{T}$ is of size $|\mathbb{U}'| = K + 1 - |\mathbb{T}|$. If it is less than K', $\mathbb{U}'$ will be padded with dummy sessions $\{d\} \subseteq \mathbb{L}_{\mathsf{D}}$. It also outputs a new score $s' = \sum_{t_j \in \mathbb{T}} s_{t_j}$.
4. (New credential generation:) The SP issues a BBS+ signature cred' on $\mathsf{A}' = (x, q', s + s', \hat{\mathbb{U}})$ by using the protocol $\mathbb{P}_{\mathsf{Iss}}$ on C'_M.
5. (List update:) The SP adds the entry $(t', 0, \sigma, \emptyset)$ to list $\mathbb{T}$, where σ is a BBS+ signature on the message $(t', 0)$.

Appendix C provides a concrete instantiation using signature-based range proof $\mathbb{P}_{\mathsf{Set}}$, verifiable shuffle [9], and batch BBS+ signatures (in Appendix A).

$\mathsf{Update}(\mathsf{pp}, \mathsf{sk}, \mathsf{aux}) \rightarrow \mathsf{pp}'$: To assign a new score s to a session t given in aux, the SP issues a new BBS+ signature σ on (t, s) for the new or updated entry (t, s) in $\mathbb{L}$. (See Section 3.5 for the signature timestamping issue.) To block, the SP assigns the lowest possible negative score $-(S_{\mathsf{max}} + 1)$.

When no more change is needed for a session t, the SP changes the status of the session to "judged" by computing $\hat{\sigma}_t = g^{\frac{1}{\beta + t}}$ and putting $(t, \cdot, \cdot, \hat{\sigma}_t)$ on $\mathbb{L}$.

[8] It involves commitments of s and s_{t_i} for $t_i \in \mathbb{U}$ if it is instantiated by Bulletproofs.

3.4 Efficiency and Flexibility Highlights

Achieving Efficiency for Many Active Sessions. The user needs to store all its active session identifiers and construct proof about them for claiming the scores of all the sessions, *i.e.*, the computation complexity grows with the size of user-side storage. We use a few interesting techniques to improve user-side efficiency:

1. Users can remove session identifiers from the credential once they are judged. Their scores will be permanently aggregated to the finalized score field s of the credential. Low-usage users can choose to keep a small buffer size K.
2. The score of each session is signed by BBS+ signature, which support efficient ZK proof (without proving pairing relation) and batch verification.
3. The same proof asserts the ticket status without disjunctive proof and the involved commitments needed by PERM/PEREA for proving unjudged sessions ($t_i - \mathsf{jp} < N$ or knowledge of a judgment on t_i). This saves approximately K range proofs without requiring sequential judging.

Fair Rate-Limiting. SAC supports a (set of) maximum buffer size K ($\mathbb{K}$). The SP is required to finalize the judgment on at least one of these sessions when all K active session slots are used up by the user. Any user is only rate-limited by its own previously unjudged tickets. Besides, the number of unjudged tickets sets an upper bound to the cardinality of the to-redeem ticket set $\mathbb{T}$. PERM/PEREA can be treated as a special case where $|\mathbb{T}| = 1$ ("redeeming" the first ticket).

Removal of Old Judgments. Signatures that were too old could be removed. The SP can still keep only the session identifier and score. If an infrequent user eventually claims those sessions, it can still be done without affecting correctness but just anonymity. This is more flexible than the existing blocklist-based approach, in which truncating the blocklist may forgive some bad users for free.

3.5 Discussion on Privacy and Security Issues

Variations of User Authentication/Redemption Behavior. The SP and the users can judge and redeem scores from the sessions, respectively, in an arbitrary order. From an anonymity perspective, suppose a user always redeems the first few sessions in its credential during authentication while another user always redeems the last few; their difference in behavior may compromise anonymity. Verifiable shuffle is thus used to ensure obliviousness to any pattern of redemption. Anonymity holds among users using the same size parameters, specifically, K (the number of sessions in the credential) and $|\mathbb{T}|$ (the size of the ticket to redeem) with the system-wide choices of size $\mathbb{K}$ and many dummy tickets $\mathbb{L}_{\mathsf{D}}$.

Retrieval of Signatures. Similar to redemption in PERM, users should also retrieve the signatures corresponding to others' sessions to hide in the crowd. The download can be amortized, such as getting a random subset after each interaction (e.g., browsing on a forum), or can be done via private information retrieval, like what is necessary for other privacy-preserving applications (e.g., getting the list of bridges for Tor [40]). Compared with BLAC/EPID, which performs expensive cryptographic operations against each blocklist entry, our "allowlist" does not need to be downloaded during the authentication.

Credential "Hijacking" Prevention. A user can freely choose the nonce for each credential update. If a user picks a used nonce, it cannot be used to authenticate. This brings a subtle issue if a malicious user somehow learns the victim's credential nonce. A malicious user can block a victim user by using the victim's nonce as the nonce of a new credential and authenticate using it before the victim.

The issue can be prevented by slightly extending the protocol so the SP can contribute a part of the nonce randomness. The SP and user respectively pick q', q''. The new nonce will be set to $q' + q''$ during credential generation. The probability of hitting a used nonce is negligible if the nonce space is exponentially large, which is our case. In the rare event that results in a used nonce, the user can reveal q'' and rerun the credential generation part with the SP.

Signatures Timestamping/Expiry. SAC needs to expire old signatures when the score is updated. This is a general problem of timestamping signatures, which has been studied thoroughly in different contexts with multiple solutions, e.g., outsourced and authenticated data structure which supports membership query and update [42]. Solutions for timestamping signatures can be plugged into SAC generically. We describe below a simple (but not necessarily optimal) solution.

The SP issues a new set of signatures for each session in predefined intervals. The interval identifier can be signed together with the updated score via the multi-block feature of BBS+ signature. The proof will require proving the interval to be the current one, as how proofs are made on other messages certified.

This simple solution requires the SP to generate multiple signatures for multiple updates. Nevertheless, the SP is supposed to be more resourceful and can use online/offline signatures with preparation done offline. The SP would not update indefinitely as the scores of active sessions will eventually be finalized. Only the most-updated signatures or the finalized ones should be kept.

Expiry of signatures is only needed for volatile scores. If a session can only be changed from unjudged to finalized, signature expiry is not needed at all.

3.6 Integration with Other System Components

Precondition for Registration. To prevent unauthorized sharing of credentials, the standard practice is to embed valuable secrets to credentials such that sharing them means sharing the secret too. The SP can require the user to post the public key of a cryptocurrency account, prove the knowledge of its secret key to the SP, and use it as the user secret in the SAC credential.

Multi-SP and Decentralization. A 1-out-of-n disjunctive (or membership) proof can be used to prove the validity of cred and judgments under a set of authorized SPs' public keys. With homomorphic commitments, judgments from different

SPs can be combined arithmetically. However, one cheating SP can ruin the system by issuing malformed credentials. Threshold BBS+ signature [30] with blind signing protocol can be applied to achieve t-out-of-n threshold signing, where t-out-of-n signing parties (SPs) are needed to issue/update user credentials.

Enforcement vs. Voting. SAC is for privacy-preserving reputation *enforcement* (PPRE). The basic setting assumes the SP decides the score. Alternatively, privacy-preserving reputation *voting* (PPRV), often called "privacy-preserving reputation" [33], aims for the privacy of (peer) voters who cast votes of a score for other users. They may use cryptographic techniques such as linkable ring signatures [25,32] for double-voting detection [24]. Some require a TTP, or users can request votes from many voters via secure multi-party computation.

Among many PPRV approaches, "reputation transfer" [33] overlaps with PPRE. A user can use a claiming mechanism to link scores cast by voters to a new pseudonym. Nevertheless, users are not forced to claim every (negative) rating. Some work discusses the usage of ZKP, which should cover all ratings and brings us back to linearly processing a global list we strive to avoid. In general, incorporating PPRV with PPRE may not be straightforward.

A concurrent work [23] outlined a "decentralized anonymous social networks" construction supporting both PPRE and PPRV, with a focus on sustainability.

Keyed-Verification Anonymous Credentials (KVAC). In many scenarios, the SP also acts as the verifier without the need for public verifiability offered by SAC. In KVAC [21], verifying the proof of possession of a credential requires the issuer's secret (issuance) key. KVAC [21,8,27] does not consider credential updates based on previously committed (and authenticated) messages. Moreover, revocable KVAC [34] assumes a TTP revocation authority using traceable-tag techniques [22,6,1]. Finally, KVAC [21,8,27] could still use public-key algebraic-group operations. We leave more efficient constructions using KVAC as future work.

4 Performance

4.1 Computation and Communication Complexities

We select PERM with one score category ($\ell = 1$) [4] as the representative to compare. We set $|\mathbb{T}| = 10$ (the set of tickets to be redeemed). New dummy sessions will be appended and shuffled with currently active sessions to maintain the credential buffer size. The size for the shuffle proof via Bulletproofs consists of the input and output commitments and the logarithmic size (inner-product) proof (1601 bytes for $K = 200$). An authentication needs to prove $(K+3)$ BBS+ signatures for the accumulated score. PERM needs extra K (possibly simulated) range proofs for its ZKP of partial knowledge. Since we utilize succinct (sublinear size) ZKAoK, the communication overhead is relatively small compared to the ZKP on the BBS+ signatures (a few KBs versus hundreds of KBs).

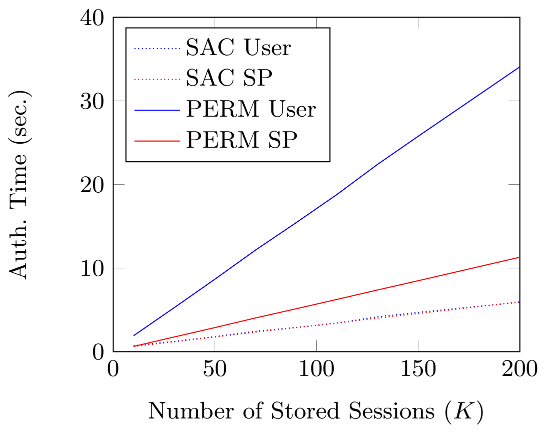


Fig. 2. Computation time of the service provider and user with $|\mathbb{T}| = 10$: (i) PERM only supports redeeming the first ticket, *i.e.*, $|\mathbb{T}| = 1$. (ii) BBS+ signatures over Type-3 curves feature batch ZKP verification, which our implementation does not employ.

4.2 Empirical Results

We run SAC and PERM [4] over a desktop PC equipped with Ryzen 7 3700X and 96GB RAM running Ubuntu 20.04 (on Windows Subsystem for Linux 2). We modified the Rust libraries[9] that implemented the BBS+ signature over the pairing-friendly curve BLS12-381, and the Bulletproofs implementation[10] for BLS12-381. For SAC, the set of possible buffer sizes is $\mathbb{K} = \{10, 30, \ldots, 200\}$. We compare SAC with running PERM on different window sizes $K_{\mathsf{max}} = K$.

Figure 2 outlines the authentication time. For the user side, SAC takes 0.714s/5.91s when $K = 10, 200$. Meanwhile, PERM takes 1.90s and 34.0s when $K = 10, 200$. For $K = 50$, SAC still performs better than PERM at $K = 10$.

For the SP side, SAC takes 0.612s/5.97s when $K = 10/200$. Comparatively, PERM takes 0.625s and 11.3s when $K = 10, 200$. For $K \geq 90$, the SP computation cost for PERM is like twice of SAC for the same number of stored sessions.

For data transfer, we suppose each session score and the maximum reputation score of a user are in a 64-bit range. Furthermore, the total number of authentications is less than 2^{64}. The serialized size of a $\mathbb{Z}_p$, $\mathbb{G}_1$, and $\mathbb{G}_2$ element is 32, 48, and 96 bytes, respectively. Each entry in our list $\mathbb{L}$ is 176 bytes for active sessions and 80 bytes for judged or dummy sessions. The total downlink complexity is $176|\mathbb{L}_{\mathtt{A}}| + 80(|\mathbb{L}_{\mathtt{D}}| + |\mathbb{L}_{\mathtt{J}}|)$ bytes. Here, the signature of the SP used in the range proof is put in the public parameters as in PERM. Dummy sessions in SAC never change. Users can download them during registration.

To compare with PERM, suppose the number of anonymous authentications per day is 20,000, and the same number of authentications is judged (which only

[9] `https://github.com/docknetwork/crypto`
[10] `https://github.com/dalek-cryptography/bulletproofs`

adds an extra 80 bytes to $\mathbb{L}$). A user of PERM and SAC would need to download respectively 3.5MB and 5MB of data a day to keep up-to-date without decoys.

For each authentication, a user of both PERM and SAC proves possession of $O(K)$ BBS+ signature on stored sessions with the scores accumulated, which is about 368 KBytes for $K = 200$. PERM runs ZKP for K disjunctive statements, which consist of signature possession and range proof. We instantiated the range proof with Bulletproofs; the extra communication overhead of PERM is around 3676 bytes per session. On the other hand, the proof of shuffle using Bulletproofs in SAC is 1601 bytes (for $K = 200$) and the total size with commitments is $96 \cdot K$ bytes. The extra ZKP on a BB signature has around $240 \cdot |\mathbb{T}|$ bytes ($|\mathbb{T}| \leq K$). Thus, we have a lower authentication communication cost than PERM.

5 Conclusion

We propose scored anonymous credentials (SAC), a new and intuitive credential design supporting revocation and reputation. Unlike the two existing designs of either checking every session that ever happened (e.g., BLAC(R) [44,5]) or assuming sequential judgments (e.g., PEREA [45], PERM), we directly deal with the user sessions via verifiable shuffle and other optimized cryptographic techniques. We also support downgrading the score of a session until it is finalized, while existing updatable anonymous credentials (e.g., PE(AR)^2 [47], PERM) only support score upgrade. We evaluate the efficiency of our proposed system SAC and show that it outperforms existing related systems, given the simplicity of our design and cryptographic techniques. We thus resolved the open problem of devising an anonymous credential with (reputation and) revocation mechanism that does not halt the entire system due to just one misbehaving user.

References

1. Abe, M., Chow, S.S.M., Haralambiev, K., Ohkubo, M.: Double-trapdoor anonymous tags for traceable signatures. Int. J. Inf. Sec. **12**(1), 19–31 (2013)
2. Acar, T., Chow, S.S.M., Nguyen, L.: Accumulators and U-Prove revocation. In: Financial Cryptography and Data Security. pp. 189–196 (2013)
3. Acar, T., Nguyen, L.: Revocation for delegatable anonymous credentials. In: Public Key Cryptography. pp. 423–440 (2011)
4. Au, M.H., Kapadia, A.: PERM: Practical reputation-based blacklisting without TTPs. In: CCS. pp. 929–940 (2012)
5. Au, M.H., Kapadia, A., Susilo, W.: BLACR: TTP-free blacklistable anonymous credentials with reputation. In: NDSS (2012)
6. Au, M.H., Susilo, W., Mu, Y., Chow, S.S.M.: Constant-size dynamic k-times anonymous authentication. IEEE Systems Journal **7**(2), 249–261 (2013)
7. Backes, M., Hanzlik, L., Schneider-Bensch, J.: Membership privacy for fully dynamic group signatures. In: CCS. pp. 2181–2198 (2019)
8. Barki, A., Brunet, S., Desmoulins, N., Traoré, J.: Improved algebraic MACs and practical keyed-verification anonymous credentials. In: Selected Areas in Cryptography (SAC). pp. 360–380 (2016)

9. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: EUROCRYPT. pp. 263–280 (2012)
10. Bernstein, M.S., Monroy-Hernández, A., Harry, D., André, P., Panovich, K., Vargas, G.G.: 4chan and /b/: An analysis of anonymity and ephemerality in a large online community. In: AAAI Conf. on Web & Social Media (ICWSM) (2011)
11. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptology **21**(2), 149–177 (2008)
12. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO. pp. 41–55 (2004)
13. Brickell, E., Li, J.: Enhanced Privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In: WPES. pp. 21–30 (2007)
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: S&P. pp. 315–334 (2018)
15. Camenisch, J., Chaabouni, R., abhi shelat: Efficient protocols for set membership and range proofs. In: ASIACRYPT. pp. 234–252 (2008)
16. Camenisch, J., Drijvers, M., Hajny, J.: Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In: WPES. pp. 123–133 (2016)
17. Camenisch, J., Kohlweiss, M., Soriente, C.: Solving revocation with efficient update of anonymous credentials. In: SCN. pp. 454–471 (2010)
18. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: EUROCRYPT. pp. 93–118 (2001)
19. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO. pp. 61–76 (2002)
20. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: CRYPTO. pp. 410–424 (1997)
21. Chase, M., Meiklejohn, S., Zaverucha, G.: Algebraic MACs and keyed-verification anonymous credentials. In: CCS. pp. 1205–1216 (2014)
22. Chow, S.S.M.: Real traceable signatures. In: Selected Areas in Cryptography (SAC). pp. 92–107 (2009)
23. Chow, S.S.M., Egger, C., Lai, R.W.F., Ronge, V., Woo, I.K.Y.: On sustainable ring-based anonymous systems. In: IEEE Computer Security Foundations (CSF) Symposium (2023), to appear.
24. Chow, S.S.M., Liu, J.K., Wong, D.S.: Robust receipt-free election system with ballot secrecy and verifiability. In: NDSS (2008)
25. Chow, S.S.M., Susilo, W., Yuen, T.H.: Escrowed linkability of ring signatures and its applications. In: VIETCRYPT. pp. 175–192 (2006)
26. Chow, S.S.M., Zhang, H., Zhang, T.: Real hidden identity-based signatures. In: Financial Cryptography and Data Security. pp. 21–38 (2017)
27. Couteau, G., Reichle, M.: Non-interactive keyed-verification anonymous credentials. In: Public Key Cryptography. pp. 66–96 (2019)
28. Derler, D., Hanser, C., Slamanig, D.: A new approach to efficient revocable attribute-based anonymous credentials. In: IMACC. pp. 57–74 (2015)
29. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: the second-generation onion router. In: USENIX Security Symposium. pp. 303–320 (2004)
30. Doerner, J., Kondi, Y., Lee, E., abhi shelat, Tyner, L.: Threshold BBS+ signatures for distributed anonymous credential issuance. In: S&P. pp. 2095–2111 (2023)
31. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical short signature batch verification. In: Crypto. Track at RSA (CT-RSA). pp. 309–324 (2009)
32. Fiore, D., Garms, L., Kolonelos, D., Soriente, C., Tucker, I.: Ring signatures with user-controlled linkability. In: ESORICS Part II. pp. 405–426 (2022)

33. Gurtler, S., Goldberg, I.: SoK: privacy-preserving reputation systems. Proc. Priv. Enhancing Technol. **2021**(1), 107–127 (2021)
34. Hajny, J., Dzurenda, P., Marques, R.C., Malina, L.: Privacy ABCs: Now ready for your wallets! In: PerCom Workshops. pp. 686–691 (2021)
35. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. **43**(2), 618–644 (2007)
36. Lai, R.W.F., Cheung, K., Chow, S.S.M., So, A.M.: Another look at anonymous communication. IEEE Trans. Dependable Secur. Comput. **16**(5), 731–742 (2019)
37. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling private payments without trusted setup. In: CCS. pp. 31–48 (2019)
38. Lofgren, P., Hopper, N.: FAUST: efficient, TTP-free abuse prevention by anonymous whitelisting. In: WPES. pp. 125–130 (2011)
39. Ma, J.P.K., Chow, S.S.M.: SMART credentials in the multi-queue of slackness (or Secure management of anonymous reputation traits without global halting). In: IEEE European Symposium on Security and Privacy (EuroS&P) (2023), to appear.
40. Mittal, P., Olumofin, F.G., Troncoso, C., Borisov, N., Goldberg, I.: PIR-Tor: scalable anonymous communication using private information retrieval. In: USENIX Security Symposium (2011)
41. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: IEEE Symposium on Security and Privacy. pp. 173–187 (2009)
42. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: CRYPTO. pp. 91–110 (2011)
43. Rosenberg, M., Maller, M., Miers, I.: SNARKBlock: Federated anonymous blocklisting from hidden common input aggregate proofs. In: IEEE Symposium on Security and Privacy (S&P). pp. 1290–1307 (2022)
44. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: Blacklistable anonymous credentials: Blocking misbehaving users without TTPs. In: CCS. pp. 72–81 (2007)
45. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: PEREA: Towards practical TTP-free revocation in anonymous authentication. In: CCS. pp. 333–344 (2008)
46. Xi, L., Feng, D.: FARB: Fast anonymous reputation-based blacklisting without TTPs. In: WPES. pp. 139–148 (2014)
47. Yu, K.Y., Yuen, T.H., Chow, S.S.M., Yiu, S.M., Hui, L.C.K.: PE(AR)^2: Privacy-enhanced anonymous authentication with reputation and revocation. In: European Symposium on Research in Computer Security (ESORICS). pp. 679–696 (2012)
48. Zhang, T., Wu, H., Chow, S.S.M.: Structure-preserving certificateless encryption and its application. In: Crypto. Track at the RSA Conf. (CT-RSA). pp. 1–22 (2019)

A Batch BBS+ Signature

The original ZKP for a BBS+ signature [6] is performed as follows.

Protocol $\mathbb{P}_{\mathsf{Sig}}$. It allows a prover to prove that it knows a signature $\sigma = (A, e, y)$ on blocks of messages $(x_1, \ldots, x_\ell)$ without revealing the signature nor the messages.

1. The prover randomly generates $r_A \in \mathbb{Z}_p$, sets $\beta = r_A e$, and sends $A_1 = A\hat{h}^{r_A}, A_2 = h_1^{r_A}$ to the verifier along with the following proof Π:

$$\mathsf{PoK}\left\{\begin{array}{l}(\{x_i\}_{i\in[1,\ell]}, e, y, r_A, \beta) : \\ (A_2 = h_1^{r_A}) \wedge (1 = A_2^{-e} h_1^{\beta}) \wedge \\ \quad \dfrac{\hat{e}(A_1, w)}{\hat{e}(h_0, f)} = \hat{e}(h_1, f)^{x_1} \cdots \hat{e}(h_\ell, f)^{x_\ell} \cdot \hat{e}(h_{\ell+1}, f)^{y} \cdot \\ \qquad\qquad\qquad \hat{e}(\hat{h}, w)^{r_A} \cdot \hat{e}(\hat{h}, f)^{\beta} / \hat{e}(A_1, f)^{e}\end{array}\right\}.$$

2. Upon receiving (A_1, A_2, Π), the verifier outputs 1 if proof Π is valid.

Details such as its construction can be found in [6].

Batch BBS+ Signatures. We construct batch BBS+ (B-BBS+) featuring a batch verification algorithm for BBS+ signatures and its zero-knowledge version. Without loss of generality, we illustrate with two-message blocks $(s_1, m_1) \in \mathbb{Z}_p^2$ (suffices for SAC). The signature $(A_1 = (h_0 h_1^{s_1} h_2^{m_1} h_3^{y_1})^{\frac{1}{\gamma+e_1}}, e_1, y_1)$ can be verified by:

$$\hat{e}(A_1, w f^{e_1}) = \hat{e}(h_0 h_1^{s_1} h_2^{m_1} h_3^{y_1}, f).$$

For $i \in [1, K]$, let (A_i, e_i, y_i) be the signature on (s_i, m_i). To batch verify, pick $(\delta_1, \ldots, \delta_K)$ as a random vector of ℓ_b-bit elements from $\mathbb{Z}_p$ and test if:

$$\hat{e}(\prod_{i=1}^{K} A_i^{\delta_i}, w) \cdot \hat{e}(\prod_{i=1}^{K} A_i^{e_i \delta_i}, f) = \hat{e}(h_0^{\sum_{i=1}^{K} \delta_i} \cdot h_1^{\sum_{i=1}^{K} s_i \delta_i} \cdot h_2^{\sum_{i=1}^{K} m_i \delta_i} \cdot h_3^{\sum_{i=1}^{K} y_i \delta_i}, f).$$

which takes 2 pairings, $2K+4$ exponentiations in $\mathbb{G}_1$, and $2K+2$ multiplications in $\mathbb{G}_1$. Since 1 pairing takes roughly the time of 5 exponentiations, and non-batch verifications take $2K$ pairings, $4K$ exponentiations in $\mathbb{G}_1$, and $4K$ multiplications in $\mathbb{G}_1$ for K signatures, the batch verification speeds up by $4.5\times$ for large K.

Theorem 1. *For security level ℓ_b, the above algorithm is a batch verifier for BBS+ signatures with the probability of accepting an invalid signature being $2^{-\ell_b}$.*

The batching technique for BBS+ basically follows from [31][Theorem 3.2]. BBS+ signature is weakly secure in the standard model under the q-SDH assumption [6]. Below is the batched zero-knowledge proof $\mathbb{P}_{\mathsf{Bvfy}}$ for B-BBS+.

Protocol $\mathbb{P}_{\mathsf{Bvfy}}$. It allows proving the knowledge of K signatures $\sigma_i = (A_i, e_i, y_i)$ on messages (s_i, m_i) for $i \in [1, K]$.

1. Let δ_i be an ℓ_b-bit element picked by the verifier in $\mathbb{Z}_p$. The prover randomly generates $\mu_i \in \mathbb{Z}_p$, computes $B_i = h_1^{\mu_i}, D_i = A_i \hat{h}^{\mu_i}$, $\iota_i = \mu_i e_i$, and sends

B_i, D_i to the verifier along with the proof Π:

$$\mathsf{PoK}\left\{\begin{array}{l}(\{s_i, m_i, \mu_i, e_i, y_i, \iota_i\}_{i\in[1,K]}) : \\ \bigwedge_{i=1}^{K} \left(B_i = h_1^{\mu_i} \wedge 1 = B_i^{-e_i} h_1^{\iota_i}\right) \wedge \\ \dfrac{\hat{e}(\prod_{i=1}^{K} D_i^{\delta_i}, w)}{\hat{e}(h_0^{\sum_{i=1}^{K}\delta_i}, f)} = \hat{e}(h_1, f)^{\sum_{i=1}^{K}\delta_i s_i} \cdot \hat{e}(h_2, f)^{\sum_{i=1}^{K}\delta_i m_i} \cdot \\ \qquad\qquad\qquad \hat{e}(h_3, f)^{\sum_{i=1}^{K}\delta_i y_i} \cdot \hat{e}(\hat{h}, w)^{\sum_{i=1}^{K}\delta_i \mu_i} \cdot \\ \qquad\qquad\qquad \hat{e}(\hat{h}, f)^{\sum_{i=1}^{K}\delta_i \iota_i} / \hat{e}(\prod_{i=1}^{K} D_i^{\delta_i e_i}, f) \end{array}\right\}.$$

2. Upon $((B_1, D_1), \ldots, (B_K, D_K), \Pi)$, the verifier outputs 1 if proof Π is valid.

B Alternative for Zero-knowledge Argument of a Shuffle

Let $C_i = \mathcal{E}_{\mathsf{pk}}(M_i; \rho_i)$ for $i \in [1, N]$, a ciphertext created from encrypting message M_i under public key pk using randomness ρ_i. Homomorphic encryption allows simple multiplications of encrypted M_i and M_j without decrypting it first via simple multiplication of ciphertexts: $\mathcal{E}_{\mathsf{pk}}(M_i; \rho_i)\mathcal{E}_{\mathsf{pk}}(M_j; \rho_j) = \mathcal{E}_{\mathsf{pk}}(M_i M_j; \rho_i + \rho_j)$. There are efficient zero-knowledge arguments for showing a shuffling of ciphertexts produced by homomorphic encryption, *i.e.*, for all $i \in [N]$, decrypting C'_i is the same as decrypting $C_{\pi(i)}$, where $\pi : [N] \to [N]$ is the permutation.

We review a recent scheme by Bayer and Groth [9]. The first step is to commit to permutation π. The prover receives a challenge x and commits to $x^{\pi(1)}, \ldots x^{\pi(N)}$. The prover will give an argument of knowledge of openings of the commitments to permutations of $1, \ldots, N$ and $x^1, \ldots, x^N$. The prover demonstrates that the same permutation has been used in both cases using random challenges y and z. By using the homomorphic properties of the commitment, the prover can compute commitments to $d_1 - z = y\pi(1) + x^{\pi(1)} - z, \ldots, d_N - z = y\pi(N) + x^{\pi(N)} - z$, in a verifiable manner, then uses a product argument to show:

$$\prod_{i=1}^{N}(d_i - z) = \prod_{i=1}^{N}(yi + x^i - z).$$

These are two identical degree-N polynomials in z (with the roots permuted). By Schwartz-Zippel lemma, the prover has a negligible chance over the choice of z of making a convincing argument unless there is a permutation π such that $d_i = y\pi(i) + x^{\pi(i)}$ for $i \in [1, N]$. Furthermore, there is negligible probability over the choice of y of this being true unless the first commitment contains $\pi(i)$ and the second commitment contains $x^{\pi(i)}$ for $i \in [1, N]$.

The prover has commitments to $x^{\pi(i)}$ and uses the multi-exponentiation argument to show there exists a ρ such that

$$\prod_{i=1}^{N} C_i^{x^i} = \mathcal{E}_{\mathsf{pk}}(1;\rho) \prod_{i=1}^{N} C_i^{x^{\pi(i)}}.$$

Since the encryption $\mathcal{E}$ is homomorphic, the verifier can deduce that $\prod_{i=1}^{N} M_i^{x^i} = \prod_{i=1}^{N} M_i^{x^{\pi(i)}}$ for some permutation π. Since x is a random challenge chosen by the verifier, we have a correct shuffle with overwhelming probability.

Depending on the implementation, there is a trade-off between the round complexity, communication complexity, and the computation time of the users and the SP [9]. In principle, we can apply any zero-knowledge arguments of a shuffle. For efficiency, we will require the shuffle and the commitments (to session identifiers) to be in the same group. For our construction, we can use the homomorphic ElGamal encryption of a message M with $\mathsf{pk} = (h, u)$ is (Mu^ρ, h^ρ), which fits with Bayer-Groth's argument [9]. Since decryption is not needed in our case, we can encode a ticket t by $M = h^t$, and only include the first part of the ciphertext. It is easy to see that it preserves the homomorphic property.

C Authentication using Batch-BBS+ and Range Proof

The signature-based range proof can be constant-size if the threshold score s_{th} and the maximum score s_{max} is short, e.g., ℓ_b-bit integers for $\ell_b = 10$. Let $g \in \mathbb{G}_1$, $u \in \mathbb{G}_1$ (for the shuffle), and $f_2 \in \mathbb{G}_2$ be generators. The SP picks a random $\alpha \in \mathbb{Z}_p$, and computes $w_2 = f_2^{\alpha}$. The SP puts BB signatures $Y_j = g^{\frac{1}{\alpha+j}}$ for all $j \in [s_{\mathsf{th}}, s_{\mathsf{max}}]$ in the public parameters. (One can instead apply Bulletproofs [14].)

1. Let $\sigma_{\mathsf{A}} = (A, e, y)$ be the credential on attributes $\mathsf{A} = (x, q, s, \{t_1, \ldots, t_K\})$. Let $\mathbb{J}$ be a set of $M = |\mathbb{J}|$ indexes where $t_j \in \mathbb{T}$ for all $j \in \mathbb{J}$. The user sends the parameters K, M, and q to the SP. The SP returns a new identifier t'.
2. The SP randomly picks $\delta_1, \ldots, \delta_K$, ζ_i, ι_i for $i \in [1, K - M]$ and ℓ_b-bits numbers $\{\theta_j\}_{j\in\mathbb{J}}$ and sends to the user.
3. The user picks $r_A, r_t, r_e, r_\beta, r_x, r_s, r_1, \ldots, r_K$ in $\mathbb{Z}_p$, computes $\beta = r_A e$ and:

$$\begin{aligned} &A_1 = A\hat{h}^{r_A},\ A_2 = h_1^{r_A},\ T_1 = h_1^{r_t},\ \ T_2 = A_2^{-r_e} h_1^{r_\beta}, \\ &R_x = g^{r_x},\quad R_s = g^{r_s},\quad R_1 = g^{r_1}, \ldots, R_K = g^{r_K}, \\ &\ R = \hat{e}(h_1, f)^{r_x} \cdot \hat{e}(h_3, f)^{r_s} \cdot \hat{e}(h_4, f)^{r_1} \cdots \hat{e}(h_\ell, f)^{r_K} \cdot \\ &\qquad \hat{e}(h_{\ell+1}, f)^{r_y} \cdot \hat{e}(\hat{h}, w)^{r_t} \cdot \hat{e}(\hat{h}, f)^{r_\beta} / \hat{e}(A_1, f)^{r_e}. \end{aligned}$$

For $i \in [1, K]$, let (A_i, e_i, y_i) be the B-BBS+ signature for ticket t_i and score s_i in $\mathbb{L}$. The user randomly picks $\mu_i, r_{s_i}, r_{\mu_i}, r_{\beta_i}$ in $\mathbb{Z}_p$, and computes

$$\beta_i = \mu_i e_i,\ D_i = A_i \hat{h}^{\mu_i},\ B_i = h_1^{\mu_i}, \quad S_i = g^{r_{s_i}}, \bar{T}_i = h_1^{r_{\mu_i}}, \quad \bar{W}_i = B_i^{-r_{e_i}} h_1^{r_{\beta_i}},$$
$$R_{\mathbb{L}} = \hat{e}(h_1, f)^{\sum_{i=1}^K \delta_i r_i} \cdot \hat{e}(h_2, f)^{\sum_{i=1}^K \delta_i r_{s_i}} \hat{e}(h_3, f)^{\sum_{i=1}^K \delta_i r_{y_i}} \cdot \hat{e}(\hat{h}, w_0)^{\sum_{i=1}^K \delta_i r_{\mu_i}}$$
$$\hat{e}(\hat{h}, f)^{\sum_{i=1}^K \delta_i r_{\beta_i}} / \hat{e}(\prod_{i=1}^K D_i^{\delta_i r_{e_i}}, f).$$

Let $s^* = s + \sum_{i=1}^K s_i$. The user computes a range proof of s^* with $v, r_v \in \mathbb{Z}_p$:

$$V = Y_{s^*}^v, \quad R^* = \hat{e}(V, f_2)^{-r_s - \sum_{i=1}^K r_{s_i}} \cdot \hat{e}(g, f_2)^{r_v}.$$

For all $j \in \mathbb{J}$, the user randomly picks $\mathtt{v}_j, r_{\mathtt{v}_j}$ in $\mathbb{Z}_p$ and computes:

$$\mathtt{V}_j = \hat{\sigma}_j^{\mathtt{v}_j}, \quad \mathtt{R}_{\mathbb{J}} = \hat{e}(\prod_{j \in \mathbb{J}} \mathtt{V}_j^{-\theta_j r_j}, f) \cdot \hat{e}(g, f)^{\sum_{j \in \mathbb{J}} \theta_j r_{\mathtt{v}_j}}.$$

The user runs $\mathsf{Redeem}(\mathbb{U} \cup \{t'\}, \mathbb{T}, K')$ as follows. Suppose w.l.o.g. $\mathbb{U} = \{t_1, \ldots, t_K\}$ and $\mathbb{T} = \{t_{K-M+1}, \ldots, t_K\}$. So $\mathbb{J} = \{K - M + 1, \ldots, K\}$. Suppose $\mathbb{U}' = \{t'_1, \ldots, t'_{K'-1}, t'\}$ is selected where $t'_i = t_i$ for $i \in [1, K - M]$, $t'_{K-M+1} = t'$, and t'_i is a random element in $\mathbb{L}_{\mathtt{D}}$, $i \in [K-M+2, K']$. The user computes $s' = \sum_{j \in \mathbb{J}} s_j$, and the encryption for tickets $t'_1, \ldots t'_{K-M+1}$, by randomly picking $\rho_i \in \mathbb{Z}_p$ for $i \in [1, K - M + 1]$ and setting $\hat{C}_{i,1} = g^{t'_i} u^{\rho_i}, \hat{C}_2 = g^{\sum_{i=1}^{K-M+1} \zeta_i \rho_i}$. The user picks a permutation $\pi : [K'] \to [K']$. Let $\hat{t}_i = t'_{\pi(i)}$ for $i \in [1, K - M + 1]$. The user also computes the homomorphic encryption by randomly picking $\rho'_i \in \mathbb{Z}_p$ and setting $\hat{C}'_{i,1} = g^{\hat{t}_i} u^{\rho'_i}, \hat{C}'_2 = g^{\sum_{i=1}^{K-M+1} \iota_i \rho'_i}$. For updating the credential, the user randomly picks $\hat{r}_1, \ldots, \hat{r}_{K-M}$ and sets:

$$C'_M = h_1^x h_2^{q'} h_3^{s+s'} h_4^{\hat{t}_1} \cdots h_{K-M+4}^{\hat{t}_{K-M+1}} h_{K'+4}^{y'},$$
$$R'_M = h_1^{r_x} h_2^{r_{q'}} h_3^{r_s + \sum_{j \in \mathbb{J}} r_{s_j}} h_4^{\hat{r}_1} \cdots h_{K-M+4}^{\hat{r}_{K-M+1}} h_{K'+4}^{r_{y'}},$$
$$\hat{R}_1 = g^{\hat{r}_1}, \quad \ldots, \quad \hat{R}_{K-M+1} = g^{\hat{r}_{K-M+1}}$$

and sends SP set $\mathbb{J}$, dummies $(t'_{K-M+2}, \ldots, t'_{K'-1})$, and the commitments: $A_1, A_2, T_1, T_2, R_x, R_s, R_1, \ldots, R_K, R$, $\{D_i, B_i, S_i, \bar{T}_i, \bar{W}_i\}_{i \in [1,K]}, R_{\mathbb{L}}, V, R^*$, $\{\mathtt{V}_j\}_{j \in \mathbb{J}}, \mathtt{R}_{\mathbb{J}}$, $\{\hat{C}_{i,1}, \hat{C}'_{i,1}, \hat{R}_i\}_{i \in [1, K-M+1]}, \hat{C}_2, \hat{C}'_2$, C'_M, R'_M.

The user can now perform the ZK argument for shuffling $\hat{C}_{i,1}, \hat{C}_2$ to $\hat{C}'_{i,1}, \hat{C}'_2$. Details can be found in [9].

4. If $(t'_{K-M+2}, \ldots, t'_{K'-1})$ are dummies, the SP returns challenge $c \in \mathbb{Z}_p$.

5. The user computes and sends the following with q to the SP:

$$\begin{array}{lll} z_x = r_x + cx, & z_s = r_s + cs, & z_A = r_t + cr_A, \\ z_e = r_e + ce, & z_v = r_v + cv, & z_{q'} = r_{q'} + cq', \\ z_\beta = r_\beta + cr_A e, & z_{y'} = r_{y'} + cy', & z_y = r_y + cy. \\ \text{For } i \in [1, K], & & \\ z_i = r_i + ct_i, & z_{s_i} = r_{s_i} + cs_i, & z_{y_i} = r_{y_i} + cy_i, \\ z_{e_i} = r_{e_i} + ce_i, & z_{\mu_i} = r_{\mu_i} + c\mu_i, & z_{\beta_i} = r_{\beta_i} + c\mu_i\beta_i. \\ \text{For } j \in \mathbb{J}, & z_{\mathtt{v}_j} = r_{\mathtt{v}_j} + c\mathtt{v}_j. & \\ \text{For } l \in [1, K-M], & \hat{z}_l = \hat{r}_l + c\hat{t}_l. & \end{array}$$

6. The SP checks if $T_1 A_2^c = h_1^{z_A}$, $T_2 = A_2^{-z_e} h_1^{z_\beta}$,

$$\begin{aligned} R\Big(\frac{\hat{e}(A_1, w)}{\hat{e}(h_0, f)\hat{e}(h_2, f)^q}\Big)^c &= \hat{e}(h_4, f)^{z_1} \hat{e}(h_5, f)^{z_2} \cdots \hat{e}(h_{K+3}, f)^{z_K} \hat{e}(h_1, f)^{z_x} \cdot \\ &\quad \hat{e}(h_3, f)^{z_s} \hat{e}(h_{K+4}, f)^{z_y} \hat{e}(\hat{h}, w)^{z_A} \hat{e}(\hat{h}, f)^{z_\beta} / \hat{e}(A_1, f)^{z_e}, \\ R_{\mathbb{L}}\Big(\frac{\hat{e}(\prod_{i=1}^K D_i^{\delta_i}, w_0)}{\hat{e}(h_0^{\sum_{i=1}^K \delta_i}, f)}\Big)^c &= \hat{e}(h_1, f)^{\sum_{i=1}^K \delta_i z_i} \hat{e}(h_2, f)^{\sum_{i=1}^K \delta_i z_{s_i}} \cdot \hat{e}(h_3, f)^{\sum_{i=1}^K \delta_i z_{y_i}} \cdot \\ &\quad \hat{e}(\hat{h}, w_0)^{\sum_{i=1}^K \delta_i z_{\mu_i}} \cdot \hat{e}(\hat{h}, f)^{\sum_{i=1}^K \delta_i z_{\beta_i}} / \hat{e}(\prod_{i=1}^K D_i^{\delta_i z_{e_i}}, f), \\ R^* \cdot \hat{e}(V, w_2)^c &= \hat{e}(V, f_2)^{-z_s - \sum_{i=1}^K z_{s_i}} \cdot \hat{e}(g, f_2)^{z_v}, \\ R'_M {C'_M}^c &= h_1^{z_x} h_2^{z_{q'}} h_3^{z_s + \sum_{j \in \mathbb{J}} z_{s_j}} \prod_{i=1}^{K-M+1} h_{i+3}^{\hat{z}_i} h_{K'+4}^{z_{y'}}, \\ \mathtt{R}_{\mathbb{J}} \cdot \hat{e}(\prod_{j \in \mathbb{J}} \mathtt{V}_j^{\theta_j}, w_1)^c &= \hat{e}(\prod_{j \in \mathbb{J}} \mathtt{V}_j^{-\theta_j z_j}, f) \hat{e}(g, f)^{\sum_{j \in \mathbb{J}} \theta_j z_{\mathtt{v}_j}}. \end{aligned}$$

It checks $\bar{T}_i B_i^c = h_1^{z_{\mu_i}}$, $\bar{W}_i = B_i^{-z_{e_i}} h_1^{z_{\beta_i}}$ for $i \in [1, K]$ and the zero-knowledge argument for the shuffling [9].

The SP computes: $A' = (h_0 C'_M h_{K-M+5}^{t'_{K-M+2}} \cdots h_{K'+2}^{t'_{K'-1}} h_{K'+3}^{t'_{K'}} h_{K'+4}^{y''})^{\frac{1}{e'+\gamma}}$ for random $e', y'' \in \mathbb{Z}_p$, and sends (A', e', y'') to the user.

The SP then adds the entry $(t', 0, \sigma, \emptyset)$ to list $\mathbb{T}$, where $\sigma = (A, e, y)$ is the B-BBS+ signature with $A = (h_0 h_1^{t'} h_3^{y})^{\frac{1}{\gamma+e}}$ for some random $e, y \in \mathbb{Z}_p$.

7. The user gets $\sigma'_{\mathsf{A}} = (A', e', y' + y'')$ and updates its attributes to $\mathsf{A}' = (x, q', s + s', \{\hat{t}_1, \ldots, \hat{t}_{K-M+1}, t'_{K-M+2}, \ldots, t'_{K'}\})$.

D Security

D.1 Simulation-based Model

We use the simulation-based security definition following the literature [45,4]. We consider a security game where an environment $\mathcal{E}$, which can schedule the invocation of the functionalities of SAC at its wish, is asked whether it is interacting with the real world or the ideal world. In the real world, all honest players communicate as specified in the protocol description. In the ideal world, the same players also follow the protocol except that they interact via a trusted party $\mathcal{T}$, responsible for handling all the inputs and outputs for them. The adversary $\mathcal{A}$ in the real world takes control of some of the players and can communicate arbitrarily with environment $\mathcal{E}$. But $\mathcal{A}$ does not know the communications between honest parties and the origin of messages received by $\mathcal{A}$.

Roughly, SAC is secure if, for any probabilistic polynomial time (PPT) algorithms $\mathcal{A}$ and $\mathcal{E}$, there exists another algorithm $\mathcal{S}$, which has black-box access to $\mathcal{A}$, controlling the same players in the ideal world as $\mathcal{A}$ does in the real world, such that $\mathcal{E}$ cannot distinguish if it is interacting with $\mathcal{A}$ or $\mathcal{S}$. In other words, it also cannot distinguish between the real world and the ideal world. We first specify the functionalities of SAC in the real world and the ideal world, respectively:

1. Setup. The system starts when $\mathcal{E}$ specifies the set of honest and dishonest users and the SP (static model).
   - *Real World.* The SP generates $(\mathsf{pp}, \mathsf{sk})$ and gives pp to all players.
   - *Ideal World.* The trusted party $\mathcal{T}$ initializes a database that stores the registration and authentication transcripts of all users. It also keeps track of the attributes of each user, and the public parameter pp, which contains the status/score of each session.
2. $\mathsf{Registration}$. $\mathcal{E}$ asks user i to register with the SP.
   - *Real World.* User i registers with the SP, and both parties output individually the output of this interaction to $\mathcal{E}$. If user i has already been registered, the honest SP will reject the request. Similarly, an honest user discards the credential from the SP if it has successfully registered before.
   - *Ideal World.* User i sends a registration request to $\mathcal{T}$, who informs the SP about the request and whether user i has obtained a credential before. $\mathcal{T}$ forwards the decision of the SP to user i. The user and the SP individually send the output of this interaction to $\mathcal{E}$. If the SP accepts the request and user i has not registered before, $\mathcal{T}$ stores this transcript in its database.
3. $\mathsf{Authentication}$. $\mathcal{E}$ asks user i to authenticate and redeem some sessions.
   - *Real World.* User i authenticates with the SP w.r.t. the access policy f like a threshold score s_{th} and redeems the scores of sessions specified by $\mathcal{E}$. Both user i and the SP pass the local output of this interaction to $\mathcal{E}$.
   - *Ideal World.* User i sends an authentication request to $\mathcal{T}$, who checks according to pp whether the user i satisfies the authentication condition. In more detail, $\mathcal{T}$ maintains a database of the current and past tickets for each user, where the score of a user should match with the current pp. $\mathcal{T}$ informs the SP that some anonymous user wants to authenticate and whether the user satisfies the authentication condition. The SP replies with a new session identifier t or reject to $\mathcal{T}$, and $\mathcal{T}$ forwards it to user i. If the authentication is successful, $\mathcal{T}$ removes the redeemed sessions

($\mathbb{T}$ specified by the user) from the user's attributes and updates its score. It also adds an entry for the active session t with score 0 to the database, and stores t as one of the user's session identifiers. The user and the SP individually send the output of this interaction and t (if not reject) to $\mathcal{E}$.
4. Update. $\mathcal{E}$ asks the SP to update the score s to, or finalize, a session t.
   – *Real World*. The SP runs the update algorithm as instructed by $\mathcal{E}$.
   – *Ideal World*. $\mathcal{T}$ updates its database accordingly as instructed by $\mathcal{E}$.

In the ideal world, all sessions are anonymous and unlinkable from the SP's view, and $\mathcal{T}$ verifies whether the authenticating user satisfies the authentication condition. These capture completeness, anonymity, and soundness.

Definition 2. *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 it runs in the real world (resp. ideal world) with adversary* $\mathcal{A}$ *(resp.* $\mathcal{S}$ *having black-box accesses to* $\mathcal{A}$*). SAC is secure if, for all PPT algorithms,* $|\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Ideal}_{\mathcal{E},\mathcal{S}}(\lambda)|$ *is negligible in* λ*.*

D.2 Proof

Theorem 2. *Our scheme is secure if the BBS+ and weak BB signatures are existentially unforgeable, and the range proof, set membership proof, zero-knowledge argument of shuffling,* $\mathbb{P}_{\mathsf{Iss}}$, $\mathbb{P}_{\mathsf{Sig}}$, $\mathbb{P}_{\mathsf{Set}}$, *and ZKPoK protocols are secure.*

We describe how to construct $\mathcal{S}$ in detail, except with the details of the underlying building blocks omitted (e.g., how to extract what $\mathcal{S}$ needs from the ZKPoK). Firstly, $\mathcal{S}$ maintains a list of credentials issued to $\mathcal{A}$ during the life span of the system. $\mathcal{S}$ also acts as an ideal-world adversary to the trusted party $\mathcal{T}$. $\mathcal{S}$ simply forwards any messages between $\mathcal{E}$ and $\mathcal{A}$. We consider two cases for $\mathcal{S}$:

Case 1: The SP is honest:

1. Setup. $\mathcal{S}$ generates $(\mathsf{pp}, \mathsf{sk})$ and sends pp to $\mathcal{A}$.
2. $\mathsf{Registration}$. $\mathcal{S}$ acts as a dishonest user i (in the ideal world) to $\mathcal{T}$ and an honest SP to $\mathcal{A}$ as a dishonest user in the real world. Using the knowledge extractor of the ZKPoK protocol, $\mathcal{S}$ extracts the value of x from $\mathcal{A}$. This value will be used to identify the dishonest user i. $\mathcal{S}$ sends the request to $\mathcal{T}$ on behalf of the user i. If $\mathcal{T}$ replies accept, $\mathcal{S}$ issues the credential to $\mathcal{A}$ and also stores that credential.
3. $\mathsf{Authentication}$. $\mathcal{S}$ acts as a dishonest user i to $\mathcal{T}$ and an honest SP to $\mathcal{A}$. $\mathcal{S}$ extracts and uses the value x during authentication to determine user i. Two worlds are indistinguishable except in the rare events below:
   – During registration, $\mathcal{S}$ fails to extract x from $\mathcal{A}$. This happens with negligible probability by the soundness property of $\mathbb{P}_{\mathsf{Iss}}$.
   – During a successful authentication, $\mathcal{S}$ fails to extract x from $\mathcal{A}$. This happens with negligible probability by the soundness property of $\mathbb{P}_{\mathsf{Sig}}$.
   – There exists a successful authentication from $\mathcal{A}$ such that $\mathcal{S}$ outputs accept on behalf of an honest SP, but $\mathcal{T}$ indicates the authenticating user does not satisfy the policy.

The last case implies that $\mathcal{A}$ successfully did one of the following:
- forged a credential on attributes that has never been issued,
- obtained a credential on attributes with a ticket that is neither originated from the past version of the credential nor any dummy sessions,
- created one fake proof in authentication.

All these happen with negligible probability due to the following:
- BBS+ signatures are existentially unforgeable and $\mathbb{P}_{\mathsf{Sig}}$ is sound,
- the set membership proof is sound,
- the argument of shuffling is (computationally) sound, and
- the zero-knowledge proof is sound.

Since $\mathcal{S}$ may need to run the extractor of the ZKPoK protocol, we require that the registration and authentications are run sequentially. A similar restriction also applies to PEREA, BLACR, and PERM.

Case 2: The SP is dishonest:

1. Setup. $\mathcal{S}$ is given pp by $\mathcal{A}$.
2. Registration. $\mathcal{S}$ acts as a dishonest SP to $\mathcal{T}$ (in the ideal world) and an honest user i to $\mathcal{A}$. When $\mathcal{T}$ requests registration for user i, $\mathcal{S}$ runs the registration protocol with $\mathcal{A}$ using the simulator of $\mathbb{P}_{\mathsf{Iss}}$. If $\mathcal{S}$ does not obtain a valid credential from $\mathcal{A}$, then $\mathcal{S}$ replies reject to $\mathcal{T}$.
3. Authentication. $\mathcal{S}$ acts as a dishonest SP to $\mathcal{T}$ and an honest user to $\mathcal{A}$. When $\mathcal{T}$ requests authentication for an anonymous user, $\mathcal{S}$ runs the authentication protocol with $\mathcal{A}$. If $\mathcal{T}$ proceeds and satisfies the authentication policy, $\mathcal{S}$ uses the simulator of the ZKPoKs and shuffling protocol to simulate the view of $\mathcal{A}$ using the random number q. If $\mathcal{A}$ rejects, $\mathcal{S}$ replies reject to $\mathcal{T}$.

The simulation provided to $\mathcal{A}$ is correct due to the zero-knowledge property of the ZKPoK protocols and shuffling protocol and the hiding property of the commitment scheme. The behavior of $\mathcal{S}$ in the ideal world is the same as that of $\mathcal{A}$ in the real world.