Decentralized Blacklistable Anonymous Credentials with Reputation

Rupeng Yang[1,2] *, Man Ho Au[2] **, Qiuliang Xu[1] **, and Zuoxia Yu[2]

[1] School of Computer Science and Technology, Shandong University, Jinan, 250101, China
orbbyrp@gmail.com, xql@sdu.edu.cn
[2] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
csallen@comp.polyu.edu.hk, zuoxia.yu@gmail.com

Abstract. Blacklistable anonymous credential systems provide service providers with a way to authenticate users according to their historical behaviors, while guaranteeing that all users can access services in an anonymous and unlinkable manner, thus are potentially useful in practice. Traditionally, to protect services from illegal access, the credential issuer, which completes the registration with users, must be trusted by the service provider. However, in practice, this trust assumption is usually unsatisfied. Besides, to better evaluate users, it is desired to use blacklists, which record historical behaviors of users, of other service providers, but currently, this will threaten the security unless a strong trust assumption is made. Another potential security issue in current blacklistable anonymous credential systems is the blacklist gaming attack, where the service provider attempt to compromise the privacy of users via generating blacklist maliciously.

In this paper, we solve these problems and present the decentralized blacklistable anonymous credential system with reputation, which inherits nearly all features of the BLACR system presented in Au et.al. (NDSS'12). However, in our new system, no trusted party is needed to register users. Moreover, blacklists from other service providers can be used safely in the new system assuming a minimal trust assumption holds. Besides, the new system is also partially resilient to the blacklist gaming attack. Technically, the main approach to solving these problems is a novel use of the blockchain technique, which serve as a public append-only ledger and are used to store credentials and blacklists. To simplify the construction, we also present a generic framework for constructing our new system. The general framework can be instantiated from three different types of cryptographic systems, including the RSA system, the classical DL system, and the pairing based system, and all these three types of instantiations can be supported simultaneously in the framework. To demonstrate the practicability of our system, we also give a proof of concept implementation for the instantiation under the

* This work was mainly done when doing the internship at The Hong Kong Polytechnic University.

** Corresponding author.

RSA system. The experiment results indicate that when authenticating with blacklists of reasonable size, our implementation can fulfill practical efficiency demands, and when authenticating with empty blacklists, it is more efficient than that of Garman et al. (NDSS'14), which presents a decentralized anonymous credential system without considering revocation.

1 Introduction

There always exists a conflict between users and service providers (SP) on the Internet. On the one hand, the SPs need to protect their services from illegal users and users with misbehaviors, thus hope to know the exact identity and historical behaviors of each user. On the other hand, the users would like to protect their privacy, and thus hope to access services in an anonymous and unlikable manner.

The blacklistable anonymous credential system [11, 39] is a good attempt to address this conflict. In this system, each SP maintains a blacklist to record users with misbehaviors, and a user attempting to access services of a SP is required to prove that he is legitimately registered and that he is not in the blacklist of the SP. Both the authentications and the maintenance of the blacklists are conducted in an anonymous and unlinkable fashion, thus privacy of users are well protected. Compared to traditional anonymous credential systems [9, 12–15, 18, 20], the blacklistable anonymous credential system supports revocation of users, thus can protect SPs from users with misbehaviors. Moreover, compared to some other revocable anonymous credential systems [12,13], this is achieved without relying on a trusted third party, so in practice the blacklistable anonymous credential system is preferable.

Subsequently, there are a series of works following this line of research. Some of them consider how to improve the efficiency [32, 40, 45], and some others consider how to utilize historical behaviors of users in a cleverer way [5,6,42,44]. In particular, in [6], an anonymous credential system supporting fine-grained "blacklist" is proposed. In this system, instead of merely putting misbehaved users into the blacklist, the SP will rate behaviors of users in using the services. The rated scores can be either positive or negative for good and bad behaviors respectively, and belong to different categories based on types of behaviors rated. When authenticating, SPs can set complex policies about these scores, and a user attempting to access services of a SP needs to prove that he is legitimately registered and that his scores satisfy the policy of the SP. Likewise, all those operations are conducted in an anonymous and unlinkable fashion. For simplicity of notation, in this section, we still use the word "blacklist" to denote this fine-grained type of "blacklists", and will not distinguish them from the normal ones if not necessary.

To better explain how these blacklistable anonymous credential systems work, we illustrate the workflow for them in Figure 1.a. Generally speaking, a user who wants to access services of a SP first registers himself to the credential issuer

and gets a credential back. Then he requests a policy from the SP and proves to the SP that he has a valid credential and that he satisfies the policy of the SP each time he wants to access the services of a SP. Behaviors of the user will be rated by the SP after he finishes using the services. Note that to protect services from illegal access, the credential issuer must be trusted by the SP. Therefore, it is usually suggested that the credential issuer should be acted by the SP itself. However, in practice, this suggestion is often contradicted. Considering a SP who runs a forum about alcohol abuse, anyone who registers for this service runs the risk of revealing his drinking problem to the SP. So, at worst, no one would register for using this forum. As a result, the SP faces the dilemma of either trusting a third party credential issuer and suffering potential attacks or insisting on issuing credentials all by itself and suffering a loss of potential users. A similar dilemma occurs when we consider the blacklist management. More precisely, services will be better protected if the SP can refer to blacklists of other SPs and further evaluate a user according to his historical behaviors when using other services, but it may bring additional security issues if the shared blacklists are fake. Besides these two problems, current blacklistable anonymous credential systems are also vulnerable to the blacklist gaming attack, where a malicious SP attempts to learn the identity of the user via providing a maliciously generated blacklist during the authentication.

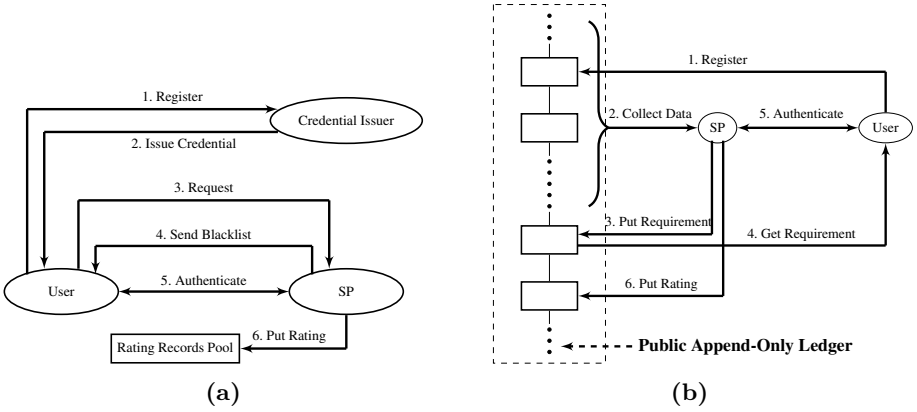


Fig. 1 Workflows of the traditional blacklistable anonymous credential systems (left) and our new decentralized blacklistable anonymous credential system with reputation (right).

The first problem, namely the requirement of a trusted credential issuer, is partially solved in [27], in which a decentralized anonymous credential system is constructed. In particular, in [27], a blockchain based public append-only ledger is employed to replace the credential issuer, and to register in the system, a user just needs to put his personal information attached with his credential

to the ledger. When authenticating, a user needs to prove that his credential belongs to a set, which is selected by the SP from credentials of all registered users. However, in [27], the authors have not considered revocation of users, and it is unknown whether their techniques can be applied to decentralize current blacklistable anonymous credential systems. Besides, the other two problems, namely the blacklist management problem and the blacklist gaming attack, are still open.

1.1 Our Results

In this paper, we solve these open problems by defining and constructing the decentralized blacklistable anonymous credential system with reputation (DBLACR), whose workflow is illustrated in Figure 1.b. More precisely, similar to that in [27], in our new system, there is no central credential issuer, and a user registers via uploading his credential together with his personal information to the public append-only ledger, which can be instantiated with the blockchain technique. Each SP collects data from the ledger automatically and put its requirement, including the selected candidate users set and the blacklist, to the ledger regularly. When a user wants to access a service of a SP, he first gets the latest requirement of the SP from the ledger, then he checks its validity and whether he satisfies it. If both tests are passed, he then proves to the SP that he satisfies its requirement. The user can access the service if the proof is valid, and scores for his behavior in using the service will be rated and put on the ledger by the SP then.

The DBLACR system can achieve an enhanced security guarantee, which is summarized in Table 1. Roughly speaking, its security is superior to that of existing blacklistable anonymous credential systems in the following three aspects.

- *The registration is decentralized.* In our new system, no trusted credential issuer is needed, and each SP can select candidate users by itself. Thus, security for the SPs is improved. Note that the user does not need to indicate which service he would like to access when registering and only the fact that he wants to access at least one service in the system is revealed. Thus, the real purpose of the user is well hidden if there are some common and insensitive services in the system. Therefore, our solution will not compromise the privacy of users.
- *There is a consistency between the used blacklist and the shared blacklist for any SP.* This is because a SP will put his own used blacklist in the public append-only ledger, thus cannot share a fake blacklist without being caught. The property implies that to refer to blacklists of other SPs, a SP only needs to trust that they will not *use* a fake blacklist when conducting their own authentication protocols instead of trusting that they will not *share* a fake blacklist. So, to a great extent, the SP can employ blacklists of other SPs safely and makes better evaluations for users.

– *The system is partially resilient to the blacklist gaming attacks, thus provides a better protection for the privacy of users during the authentication.* This is achieved in two aspects. First, as in our system SPs update their blacklists regularly, a malicious SP can only make a less powerful passive blacklist gaming attack in each time period, where it fixes a blacklist in the beginning. Besides, in our system, a user can learn whether he could pass the verification in advance and will not attempt to launch an authentication if he does not satisfy the requirement, thus less information is leaked from authentication results. We give a more detailed discussion on how these two modifications could boost the security in Sec. 3.

Table 1: The Comparison.

	Decentralized Registration	Blacklist Supporting	Blacklist Sharing	Blacklist-Gaming Resilience
BLAC[39]	✗	†	‡	✗
EPID[11]	✗	†	‡	✗
PEREA[40]	✗	†	✗	✗
PE(AR)^2[45]	✗	†	✗	✗
FAUST[32]	✗	†	✗	✗
BLACR[6]	✗	✓	‡	✗
EXBLACR[42]	✗	✓	‡	✗
PERM[5]	✗	✓	✗	✗
FARB[44]	✗	✓	✗	✗
DAC[27]	✓	✗	-	-
Ours	✓	✓	✓	✓*

We compare the security of our constructed DBLACR system with that of current blacklistable anonymous credential systems and that of the decentralized anonymous credential system in this table. The column "Decentralized Registration" indicates whether the user registration can be completed without a trusted credential issuer. The column "Blacklist Supporting" indicates whether the system supports authentication based on historical behaviors of users, and the symbol † suggests that only a basic blacklist is supported. The column "Blacklist Sharing" indicates whether blacklists can be shared among different SPs, and the symbol ‡ suggests that blacklists can be shared if SPs trust each other. The column "Blacklist-Gaming Resilience" indicates whether the system is secure against blacklist gaming attacks, and the symbol ✓* suggests that our system is partially resilient to the blacklist gaming attacks.

It seems that our newly constructed system is just natural extensions of the system in [27], and those enhanced security properties can be achieved trivially via the introduction of the blockchain technique. However, there still exists many technical obstacles. One obstacle is that in the construction in [27], double discrete logarithm proof, which is fairly expensive in both computation and communication, is desired, thus their system is not very efficient. Things get even worse in our scenario since proving that one is not in a blacklist is also

very costly. Therefore, we need to develop new construction techniques to circumvent the employment of the double discrete logarithm proof and improve the efficiency. Another obstacle is that the introduction of the blockchain technique only guarantee that SPs are not likely to share fake blacklist items, but in some systems [5, 32, 40, 44, 45] blacklist sharing is unavailable even every SP shares correct blacklist. So, we need to construct our systems carefully to ensure that they are compatible with blacklist sharing.

Here, we overcome those obstacles and give a general framework to construct our DBLACR system, which can greatly reduce the complexity of the construction. To guarantee that the construction is compatible with blacklist sharing, the framework is similar to the constructions in [6, 39], which can support blacklist sharing when SPs trust each other. To avoid the use of the double discrete logarithm proof, we give an instantiation of our framework based on the RSA system, which can achieve a much better efficiency compared to the construction in [27]. Thus, our system is also preferable even no user revocation is needed. Beyond the RSA based instantiation, we also provide a classical DL system based instantiation and a pairing system based instantiation, which provide more choices for practical applications. In fact, our general framework can support multiple instantiations simultaneously, i.e. users with credentials generated under different instantiations are authenticated in the same way, and even the type of the credential used is hidden. To demonstrate the feasibility and practicability of our system, we also give a proof of concept implementation of the RSA system based instantiation, whose result indicates that our implementation is fairly practical and can be deployed in many real world scenarios.

2 Preliminaries

In this section, we review the notations, cryptographic assumptions, and cryptographic primitives used in this paper.

Notations. For a set $\mathcal{S}$, we write $x \xleftarrow{\$} \mathcal{S}$ to indicate that x is sampled uniformly from $\mathcal{S}$. We write $negl(\cdot)$ to denote a negligible function. For two random variables $\mathcal{X}$ and $\mathcal{Y}$, we write $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$ to denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

2.1 Cryptographic Assumptions.

Strong RSA Assumption ([7, 26]). Given a randomly generated RSA modules n that is a product of two safe primes, and a random element $y \in Z_n^*$, it is hard to compute $x \in Z_n^*$ and interger exponent $e > 1$ such that $x^e \equiv y \bmod n$.

LD-RSA Assumption [41]. Let $N = (2p'+1)(2q'+1)$ be a product of two large safe primes and g is a generator of QR_N. Let p_0, q_0, p_1, q_1 be four sufficiently large and distinct primes, and $n_0 = p_0 q_0$, $n_1 = p_1 q_1$. Then we have $(N, g, n_0, n_1, g^{p_0+q_0}) \overset{c}{\approx} (N, g, n_0, n_1, g^{p_1+q_1})$.

Discrete Logarithm (DL) Assumption [21]. Let G be a cyclic group with generator g. Given $h \xleftarrow{\$} G$, it is hard to compute x such that $g^x = h$.

Decisional Diffie-Hellman (DDH) Assumption. Let G be a cyclic group of prime order p, and g be a generator. Then we have $(G,\ g, g^a, g^b,\ g^{ab}) \overset{c}{\approx} (G,\ g, g^a, g^b,\ g^c)$, where $a,\ b,\ c \xleftarrow{\$} Z_p$.

DDH-II Assumption [17]. The original DDH-II assumption works in a prime order sub-group of a group $\mathbb{Z}_p^*$ for prime p, here we extend it to the quadratic residue group QR_N, where N is the product of two safe primes. Let g be a generator of QR_N, then for any distribution $\mathcal{X}$ with a super-logarithmic min-entropy, we have $(g, g^x, g^y, g^{xy}) \overset{c}{\approx} (g, g^x, g^y, g^z)$, where $x \leftarrow \mathcal{X}$ and $y, z \xleftarrow{\$} \mathbb{Z}_N$.

2.2 Cryptographic Primitives.

Zero-knowledge proof of knowledge. In a zero-knowledge proof of knowledge [29] system, a prover proves to a verifier that he possesses the witness for a statement without revealing any additional information. In this paper, we will in fact use non-interactive zero knowledge proof of knowledge, which works in a non-interactive manner, and requires that the proof system has the following properties.

1. *Completeness.* This property states that if a proof is generated by an honest prover that indeed possesses the witness for a true statement, then an honest verifier will accept this proof.
2. *Soundness.* This property states that for any (malicious) prover, if a statement is not true, then the prover cannot generate a valid proof with non-negligible probability.
3. *Zero Knowledge.* This property states that a valid proof of a true statement can be generated by a simulator who is only given the description of the statement (but may control the random oracle or the generation of the common reference string).
4. *Extractability.* This property states that there exists a simulator (who may control the random oracle or the generation of the common reference string) that can extract the correct witnesses from any valid proof.
5. *Simulation Soundness.* This property states that the soundness property holds even the (malicious) prover sees fake proofs of false statements generated by a zero knowledge simulator.
6. *Simulation Extractability.* This property states that the extractability property holds even the (malicious) prover sees fake proofs of false statements generated by a zero knowledge simulator.

To construct a proof system satisfying those properties, one can apply the well-known Fiat-Shamir heuristic [24] to transform a three round public coin interactive zero-knowledge proof of knowledge (sigma protocol) into a non-interactive one. One advantage led by the Fiat-Shamir transform is that the transformed

non-interactive proof system can in addition admit a message as input, thus it is also called signature proof of knowledge (SPK), and is usually written as $SPK\{(w) : \mathcal{S}\}[m]$, where $\mathcal{S}$ is the statement to be proved, w is the witness, and m is the message. For convenience, the message m can be omitted if it is not explicitly indicated in context. When constructing our system, we may need to construct SPK systems for complicated statements that are combined by some simple components via logic compositions. To handle them, we can apply the technique presented in [19], which allows one to prove that indexes of true statements in a set of statements satisfy some monotone function.

Commitment Scheme. A commitment scheme allows a user to generate a commitment on a value, which ensures that the user cannot change his choice after committing, and that no one can learn anything about the value from the commitment. We will use the Strong-RSA assumption based commitment scheme in [26], which works in a quadratic residue group QR_N with generators g, h, where N is the product of two safe prime. To commit an integer a, one samples r uniformly at random from $\mathbb{Z}_{N/4}$ and computes the commitment as $c = g^a h^r$; to verify whether an opening (a, r) of a commitment c is correct, one just needs to check if $g^a h^r = c$. Note that the commitment scheme is additive homomorphic, i.e. given c_1 and c_2, which are commitments of a_1, a_2 respectively, one can obtain the commitment of $a_1 + a_2$ simply by computing $c = c_1 \cdot c_2$. We will also use the DL assumption based commitment schemes [35], which works similar to the strong-RSA assumption based one and is also additive homomorphic.

Dynamic Accumulator. An accumulator [10], can accumulate a large set of inputs into a small value (the accumulator). It also provides a way to efficiently compute a short witness for any element in the accumulated set, which can help verify whether the element is indeed incorporated into the accumulator. Security of accumulator requires that no one can generate a valid witness for an element not in the accumulated set. In this paper, we will use the strong-RSA assumption based accumulator [13], which works as follows:

- $\mathsf{Setup}(1^n) \rightarrow (N, u)$. On input a security parameter, this algorithm samples two prime numners p and q, and compute $N{=}pq$. It also picks a seed value $u \in QR_N$ that $u{\neq}1$, and outputs the public parameter (N, u).
- $\mathsf{Acc}\ ((N, u), R) \rightarrow A$. On input the public parameter (N, u), this algorithm accumulates a set of prime numbers $R = \{r_1, r_2, \ldots, r_n\}$ into the accumulator $A = u^{r_1 r_2 \cdots r_n} \mod N$.
- $\mathsf{Witness}\ ((N, u), r, R) \rightarrow w$. On input the public parameter (N, u), a set R and a value $r \in R$, this algorithm outputs the accumulation of all the other value in R besides r. Namely, $w = \mathsf{Acc}\ ((N, u), R - \{r\})$.
- $\mathsf{Verify}\ ((N, u), r, w, A) \rightarrow \{0, 1\}$. On input the public parameter (N, u), a value r, a witness w and an accumulator A, the algorithm outputs 1 if and only if $w^r = A \mod N$.

One property of this accumulator scheme is that the accumulator can be incrementally updated when new elements are added to the set. Besides, as stated in

[13], the accumulator admits an efficient SPK system proving that a committed value is in an accumulator.

CL Signature Schemes. A CL signature scheme [12,14] is a signature scheme that allows a signer to sign on the commitment of a message, and allows one to prove the possession of a valid message signature pair in a zero knowledge manner. Thus it can provide both the authentication and the privacy of message. We will use the strong-RSA assumption based CL signature presented in [14], which works as follows. As in our construction, the protocol of signing on a commitment will not be applied, we just omit this protocol here.

- **Key Generaton.** On input a security parameter 1^λ, the key generation algorithm first samples two safe primes p and q, and computes $n = pq$ of length ℓ_n. Then it samples random generators a, b, c of QR_n, and outputs $pk = (n, a, b, c)$ and $sk = p$.
- **Signing.** The signing algorithm signs messages with length at most ℓ_m. On input a message $m \in [0, 2^{\ell_m})$ and a secret key p, the signing algorithm samples a random prime number e of length $\ell_e = \ell_m + 2$ and a random number s of length $\ell_s = \ell_n + \ell_m + \ell$ where ℓ is the security parameter, and computes the value v that $v^e = a^m b^s c \mod n$. The signature is just (e, s, v).
- **Verification.** On input a signature tuple (e, s, v) and a message m, the verification algorithm outputs 1 if $v^e = a^m b^s c \mod n$ and $2^{\ell_e - 1} < e < 2^{\ell_e}$.
- **Proof of Knowledge of Signature.** The prover's secret input includes a message x, a randomness r_x, and a signature (s, e, v) on x. The public input includes the public key $(g, h) \in QR_n$ of a commitment scheme and a commitment $C_x = g^x h^{r_x}$ on x with randomness r_x. The prover samples w, r_w as random values of length ℓ_n, computes $z = we$, $r'_w = r_w e$, $C_v = vg^w$, $C_w = g^w h^{r_w}$, and $\Pi = SPK\{(s, v, e, x, r_x, w, z, r_w, r'_w) : C_x = g^x h^{r_x} \wedge C_v^e = a^x b^s c g^z \wedge C_w = g^w h^{r_w} \wedge C_w^e = g^z h^{r'_w} \wedge 2^{\ell_e - 1} < e < 2^{\ell_e} \wedge x < 2^{\ell_m}\}$, and outputs $(g, h, C_x, C_v, C_w, \Pi)$.

It is worth noting that via merely modification, the scheme can in fact sign multiple messages simultaneously. To achieve this, we only need to include multiple 'a's in the public key and use one for each message in the signing algorithm.

2.3 The Public Append-Only Ledger

We will also use a public append-only ledger, which is formally described in Fig. 2, in constructing our DBLACR system. Roughly speaking, a trusted party is employed to maintain a list of public information, and will protect the integrity of data put in the ledger and guarantee a consistent view of the ledger for every party. Every participant can add new information to this list, and once the data are uploaded, nobody, including the data owner himself, can delete or modify them. Also, as the trusted party will check the correctness of the pseudonym, no one can impersonate another participant to publish information. Besides, the trusted party can provide everyone with the latest data list, thus ensuring that the views of the public ledger for all participants are consistent.

Functionality $\mathcal{F}_{BB}^{\star}$

The functionality $\mathcal{F}_{BB}^{\star}$ running with parties $P_1, \ldots, P_n$ and an ideal adversary $\mathcal{S}$ proceeds as follows:

- **Initialize.** Initialize with an empty list $\mathcal{L}$ in the beginning.
- **Store.** Upon input (**Store**, P_i, Nym, M), checks that Nym is the correct pseudonym for P_i, then stores the tuple (Nym, M) to $\mathcal{L}$ and notify $\mathcal{S}$ that a new item is added to the public list $\mathcal{L}$.
- **Retrieve.** Upon input (**Retrieve**, P_i), sends the public list $\mathcal{L}$ to P_i.

Fig. 2 The ideal functionality $\mathcal{F}_{BB}^{\star}$ for public append-only ledger.

It is a folklore that the public append-only ledger can be instantiated via the blockchain technique, and there are already some works constructing advanced applications based on this assumption [25, 27, 34, 36]. In this paper, we will also assume this assumption holds, and build our system on the blockchain technique.

3 The Definition of The Decentralized Blacklistable Anonymous Credential System with Reputation

3.1 The Syntax

There are two types of entities, namely the users and the service providers, and a public ledger in the DBLACR system, and the system consists of the following protocols:

- **Setup.** To setup the system, a trusted party is employed to generate the public parameter of the system. Note that this party is only used in the setup phase and we only need to trust that it will generate the public parameter honestly and will erase all the internal states of the generation process.
- **Registration.** In this protocol, a user registers himself to the system. To complete this task, a user just needs to put some information to a public ledger, which should include some auxiliary proof data and his attributes to aid the SPs in deciding whether to accept the user as a valid candidate user for accessing their services.
- **Authentication.** This protocol is executed between a user and a SP. The user attempts to access services of the SP in an anonymous and unlinkable fashion, and the SP will accept the user if and only if the user fulfills its requirement, which will be explained later in this section.
- **Interaction with The Ledger.** The public ledger in this system is public and accessible to every participant, including the users and the SPs. In addition, the SPs can put data to the ledger. In particular, it can upload its requirement to the ledger regularly. Besides, it can submit a rating for the

anonymous user in an authentication event (or the rating for this authentication event for short) and submit a revocation of a rating record submitted by itself.

The Requirement. The requirement for users to access services of a SP includes three parts, namely the candidate users set $\mathcal{C}$, the policy $\mathcal{P}_R$ and the rating records list $\mathcal{L}$. The set $\mathcal{C}$ consists of candidate users for accessing the services and each item in $\mathcal{C}$ is a unique string for a candidate user (in practice, this is the user's credential). The list $\mathcal{L}$ consists of rating records used for evaluating users, and each item in $\mathcal{L}$ is a unique string for a rating record. In more detail, the main part of each rating record is a tuple (sid, tid, s) where sid is the unique string of the SP submitting this rating, tid is the unique string for the rated authentication event, and s is the rating for tid, which is a set of scores in different categories. In particular, each item in s is a tuple (c, ς) where c represents a category and ς is the score for the category c. The policy $\mathcal{P}_R$ is identical to that in [6]. More precisely, it consists of two parts, namely a DNF boolean formula $\mathcal{F}$ and a set of adjusting factor lists. The formula $\mathcal{F}$ takes a set of scores in different categories as input, and each variable in $\mathcal{F}$ is a tuple (c, d), which will be evaluated to 1 iff the given score in category c is not less than the threshold d. The set of adjusting factor lists is used to aid calculating the scores of a user, and consists of two weights lists for each category, where each weight is a positive number.

A user uid is said to satisfy a requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ if $uid \in \mathcal{C}$ and the scores of uid calculated according to $\mathcal{L}$ and $\mathcal{P}_R$ satisfy $\mathcal{P}_R$. More precisely, to calculate the scores for a user uid, one first filters all rating records for authentication events launched by uid in $\mathcal{L}$. Then it sums up scores from these rating records to get scores of uid in each category. To better reward (or penalize) users with repeated good (or bad) behaviors, positive scores and negative scores in each category will be summed up with weights from respective adjusting factor lists defined in $\mathcal{P}_R$. These calculated scores are then employed to evaluate each variable of the DNF boolean formula $\mathcal{F}$ in $\mathcal{P}_R$ and finally the result of $\mathcal{F}$. The scores is said to satisfy $\mathcal{P}_R$ if $\mathcal{F}$ is evaluated to 1 according to them.

To generate a valid requirement, a SP needs to generate a valid candidate users set, a well-formed policy and a valid rating records list. A candidate users set $\mathcal{C}$ is said valid if every user in $\mathcal{C}$ has been registered to the system. Besides, the SP also needs to ensure that for each candidate user, the attached auxiliary proof data are valid, and the attributes fulfill its terms of services. A policy is said well-formed if it contains a well-formed DNF boolean formula and a set of $2k$ positive number lists, where k is the number of categories. A rating records list $\mathcal{L}$ is said valid if each rating record with main part (sid, tid, s) in $\mathcal{L}$ is not revoked by sid and the authentication event tid is acceptable and is authenticated with the SP sid. Besides, to avoid including maliciously rated scores in the requirement, the SP also needs to ensure that each selected rating record submitted by a SP sid has been put in the requirement of sid.

3.2 The Security

To formally define the security of our system, we first present an ideal DBLACR system that is instantiated via a trusted party $\mathcal{T}_P$ with a public ledger:

- **Registration.** To register himself to the system, a user with a unique identity string *uid* first logs into the trusted party $\mathcal{T}_P$, then he sends his personal information, namely, *attr* and *aux*, which are his attributes and proof of his identity respectively, to $\mathcal{T}_P$. Then $\mathcal{T}_P$ stores $(uid, attr, aux)$ in its public ledger.
- **Authentication.** A user with a unique identity string *uid* would like to authenticate to a SP with a unique identity string *sid*. First, the user *uid* logs into the trusted party $\mathcal{T}_P$ and sends a request $(\text{'}request\text{'}, sid)$ to $\mathcal{T}_P$. Then $\mathcal{T}_P$ checks the validity of the latest requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ of *sid*. If the requirement is legal, then $\mathcal{T}_P$ checks if *uid* satisfies the requirement of *sid*, and sends the result *result* to *uid*, who needs to decide if to proceed the protocol then. If *uid* proceeds, then $\mathcal{T}_P$ assigns a unique identifier *tid* for this authentication event, stores $(tid, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ in its private ledger and sends a tuple $(\text{'}request\text{'}, sid, tid, result)$ to *sid*. $\mathcal{T}_P$ also stores (tid, sid) in its private ledger if the result is '*valid*'. Upon receiving the request (and the recommended result), *sid* needs to decide if it would like to proceed the protocol, and sends a response, which is either '*accept*' or '*reject*', back if it proceeds. Finally, $\mathcal{T}_p$ forwards the response along with *tid* to *uid*.
- **Interaction with The Ledger.** Every participant can obtain all data in the public ledger of the trusted party via submitting a "retrieve" request to $\mathcal{T}_P$. Besides, a SP *sid* can also upload its new requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ to $\mathcal{T}_P$, and a record $(sid, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ will be recorded. Also, it can submit a rating s for an authentication event *tid*. Then a tuple $(rid, sid, tid, s, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ will be recorded in the public ledger if *tid* has occurred, where *rid* is the identifier for this record, and $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ is the requirement for the authentication event *tid* and has been recorded by $\mathcal{T}_P$ in its private ledger. In addition, it can also publish a statement revoking a rating record *rid* that he has submitted, and a record $(\text{'}revoke\text{'}, rid, sid)$ will be recorded in the public ledger. We remark that the validity of those submitted data will not be checked when the data are uploaded, instead, it will be checked when the data are used.

Security of DBLACR system requires that the protocols can emulate the ideal protocols. Formally, we require that:

Definition 1. *Let* $\boldsymbol{Real}_{\mathcal{A}}(1^{\lambda})$ *be the joint distribution of the outputs of all parties of the real world protocols and the output of the real world adversary* $\mathcal{A}$*. Let* $\boldsymbol{Ideal}_{\mathcal{S}}(1^{\lambda})$ *be the joint distribution of the outputs of all parties of the ideal world protocols and the output of the ideal world adversary* $\mathcal{S}$*. A DBLACR system is secure if for all PPT real world adversary* $\mathcal{A}$*, there exists a PPT ideal world adversary* $\mathcal{S}$*, who controls the ideal-world players corresponding to the real-world players controlled by* $\mathcal{A}$ *and has black box access to* $\mathcal{A}$*, such that:* $\boldsymbol{Real}_{\mathcal{A}}(1^{\lambda}) \overset{c}{\approx} \boldsymbol{Ideal}_{\mathcal{S}^{\mathcal{A}}}(1^{\lambda})$.

Definition 1 can imply a variety of security properties, here we highlight a few that are most concerned in practice:

- **Authenticity.** The authenticity property guarantees that SPs are assured to accept authentication events only from users satisfying their requirements.
- **Anonymity.** The anonymity property guarantees that all a SP learns from an authentication event is if the authenticating user satisfies its current requirement.
- **Non-Frameability.** The non-frameability property guarantees that if a SP is honest, then users satisfying the current requirement of this SP can always successfully authenticate to it.
- **Sybil-Attack Resilience.** The Sybil attack [23] allows users to get new credentials after their current credentials are blacklisted, thus may expose services to users with misbehaviors. In our new system, since users register to the system via uploading their identities to the public ledger, the Sybil attack can be prevented if SPs only select users whose identities have not been uploaded previously as candidate users.
- **Authenticity of Registration.** This property guarantees that SPs can decide which users are legitimate directly and do not have to resort to a third party to complete this task. The property can provide a better protection for SPs.
- **Privacy of Registration.** This property guarantees that only the fact that the registered user hopes to access at least one service supported by the system can be learned from a registration event. As personal information is usually required in registration, this property is significant in protecting the privacy of users.
- **Consistency of Blacklists.** This property guarantees that each rating record selected by a SP will be honestly assessed unless there exist SPs hoping to expose their services to possible malicious users. The property can greatly reduce the requirement of trust when using rating records from other SPs.
- **Blacklist-Gaming Attack Resilience.** The blacklist gaming attack [40] allows a SP to compromise the privacy of users via generating blacklists (requirements) maliciously. Our new system is partially resilient to the blacklist gaming attack and this is achieved in the following two aspects. First, in our new system, the SPs can only update their requirements regularly, thus in each time period, the requirement used in authentication protocols is fixed. Compared to that in previous systems, where the malicious SP can use an adaptively chosen blacklist during each authentication event, the privacy of users is better protected now. To demonstrate this, we consider the following scenario. Via Some auxiliary information, a SP conjectures that the next authentication event is launched by the same user who launches a previous authentication event with identifier "t". In current blacklistable anonymous credential systems, the SP can definitely verify its conjecture via providing a blacklist with merely "t" in it. However, this attack is not applicable in our system since no SP is able to use a temporary blacklist in an authentication.

Next, in our new system, as a user could obtain the latest requirement of a SP from the public ledger, he can check whether he is able to pass the verification in advance and will not attempt to launch the authentication protocol if he does not satisfy the requirement. To see why this can better protect the privacy of users, we consider the following scenario. Again, via some auxiliary information, a SP learns that the following authentication events will be launched by one of two lists of users. It also learns whether each user in these two lists satsifies a pre-defined requirement. Previously, even restricting the malicious SP to the pre-defined requirement, it can still determine the list of users in use if there exists an index i that the ith users in the two lists are different in satisfying the requirement. In contrast, in our new system, the malicious SP can learn nothing if the numbers of users satisfying the requirement in these two lists are identical.

We remark that the first four properties are already achieved in current blacklistable anonymous credential systems. The property "authenticity of registration" and the property "privacy of registration" have also been achieved previously, but no system has these two properties simultaneously, and our system is the first one that can protect both the security of the SPs and that of the users in the registration. The last two properties are new security properties that are only available in our new system.

4 A General Framework for Constructing Decentralized Blacklistable Anonymous Credential System with Reputation

In this section, we provide a general framework for constructing the decentralized blacklistable anonymous credential system with reputation. The general construction is built from several components, so we first introduce these basic algorithms and protocols. Then we describe how to combine these components to complete the construction. Finally, we give a security proof to illustrate the security of our construction.

4.1 Building Blocks

Our DBLACR system can be instantiated from various public key systems, and for each public key system, we need the following sub-protocols to help build our system:

A Key Generation Algorithm. On input a security parameter 1^λ, the key generation algorithm returns a public key/secret key pair, namely, $(pk, sk) \leftarrow KeyGen(1^\lambda)$. In our system, the public key is the credential of a user, and the secret key is the witness for it. So we require that the key generation algorithm has the following properties:

- *Verifiability.* There exists a polynomial-time algorithm T s.t. $T(pk, sk) = 1$ iff the pair (pk, sk) is a legal key pair of the public key system.

– *Onewayness.* Given the public key pk, it is computationally hard to compute a secret key sk such that $T(pk, sk) = 1$.
– *Collision Resistance.* It is computationally hard to find a pair of different secret keys (sk_1, sk_2) and a public key pk such that $T(pk, sk_1) = T(pk, sk_2) = 1$.

A Ticket Generation Algorithm. On input a secret key sk, the ticket generation algorithm generates a ticket for sk, namely, $\tau \leftarrow TicketGen(sk)$. In our system, each ticket will be the representation of an authentication event, and an authentication event with a ticket τ will be regarded as launched by the owner of a secret key sk iff $S(sk, \tau) = 1$. So we require that the ticket generation algorithm has the following properties:

– *Verifiability.* There exists a polynomial-time algorithm S s.t. $S(sk, \tau) = 1$ iff τ is a valid ticket of sk.
– *Indistinguishability.* Let $(pk, sk) \leftarrow KeyGen(1^\lambda)$, then for any probabilistic polynomial time adversary $\mathcal{A}$, $\Pr[b \xleftarrow{\$} \{0,1\}; b \leftarrow \mathcal{A}^{\mathcal{O}_b}(pk)] \leq 1/2 + negl(\lambda)$, where $\mathcal{O}_0$ outputs a ticket of sk each time invoked, and $\mathcal{O}_1$ outputs a random element in the range of the ticket generation algorithm each time.
– *Verifying Consistency.* For any secret keys sk_1, sk_2, if there exists a τ s.t. $S(sk_1, \tau) = S(sk_2, \tau) = 1$, then for any τ' in the range of the ticket generation algorithm, we have $S(sk_1, \tau') = S(sk_2, \tau')$.
– *Connectivity.* Let $(pk, sk) \leftarrow KeyGen(1^\lambda)$, $\tau \leftarrow TicketGen(sk)$, and sk' be a secret key s.t. $S(sk', \tau) = 1$, then given (pk, sk'), one can efficiently compute sk.

An SPK System Proving The Possession of The Secret Key. We need an SPK system to prove that the prover possesses the secret key sk of a given public key pk. Formally, the prover needs to prove $SPK\{(sk) : T(pk, sk) = 1\}$. Here, we require the SPK system (and SPK systems below) to have the security properties described in Sec. 2.

An SPK System Proving the Validity of A Public Key and A Ticket. We need an SPK system proving that the prover possesses a secret key sk for a given ticket τ and the secret key is associated with a public key in a given set $\mathcal{C}$. Formally, the prover needs to prove $SPK\{(sk, pk) : S(sk, \tau) = 1 \wedge T(pk, sk) = 1 \wedge pk \in \mathcal{C}\}$.

An SPK System Proving The Fulfilment of A Policy. We also need an SPK system proving that the prover possesses a secret key sk for a given ticket τ and the secret key represents a user whose scores evaluated according to a policy $\mathcal{P}_R$ and a rating records list $\mathcal{L}$ satisfies $\mathcal{R}_R$. For simplicity of description, in this section, we define a boolean function $\mathcal{E}$ that outputs 1 iff the latter condition is satisfied. Then, the prover needs to prove $SPK\{(sk) : S(sk, \tau) = 1 \wedge \mathcal{E}(\mathcal{P}_R, \mathcal{L}, sk) = 1\}$.

4.2 The Construction

Now, we present the general construction of our DBLACR system, which is built on the sub-protocols shown in Sec.4.1 and a public append-only ledger with ideal functionality $\mathcal{F}_{BB}^{\star}$. Recall that the construction supports multiple public key systems simultaneously. Formally, we have:

Setup. On input a security parameter 1^{λ}, a trusted party runs the setup algorithm for each sub-protocol of each public key system and outputs all those generated public parameters as the public parameter for the DBLACR system.

Registration. To register himself to the system, a user with auxiliary proof data aux and attributes $attr$ first generates his public key/secret key pair $(pk, sk) \leftarrow KeyGen(1^{\lambda})$ for one of the supported public key systems. Then he computes $\Pi_R \leftarrow SPK\{(sk) : T(pk, sk) = 1\}[aux \| attr]$. Finally, he stores the tuple $(Nym, pk, \Pi_R, attr, aux)$ to the public ledger via $\mathcal{F}_{BB}^{\star}$, where Nym is his pseudonym in the public ledger. We remark that here the user can use a temporary pseudonym and not a permanent one.

Authentication. In this protocol, a user uid attempts to authenticate with a service provider sid. Interactions between these two parties are summarized in Fig. 3. For the clarity of presentation, here we assume that there are k public key systems employed in our system, and denote them as $\Psi_1, \ldots, \Psi_k$ respectively. All algorithms in Ψ_i will be labeled with a superscript "(i)", and w.l.o.g. we assume that the user uid chooses the first public key system when registering.

In more detail, in this protocol, the user uid first downloads the requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ for accessing services of sid from the public ledger. Then he verifies the validity of this requirement.[3] If the requirement is valid, the user then checks whether he satisfies the requirement. If not, he aborts the protocol even without communicating with sid. Otherwise, uid sends a request to sid and gets a challenge $m \| sid'$ back, where m is a randomly chosen bit string whose length is polynomial in the security parameter. Then, uid checks whether $sid = sid'$ and if so he generates a ticket $\mathcal{T}$ and a proof Π_A, and sends $(\mathcal{T}, \Pi_A)$ to sid. More precisely, to generate the ticket $\mathcal{T}$, the user computes $\tau_1 \leftarrow TicketGen^{(1)}(sk)$, randomly samples τ_i in the range of $TicketGen^{(i)}(\cdot)$ for $i \in [2, k]$, and sets $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$. To generate the proof Π_A, the user computes $\Pi_A = SPK\{(sk, pk) : \bigvee_{i=1}^{k}(T^{(i)}(pk, sk) = 1 \wedge pk \in \mathcal{C}_i \wedge S^{(i)}(sk, \tau_i) = 1 \wedge \mathcal{E}^{(i)}(\mathcal{P}_R, \mathcal{L}^{(i)}, sk) = 1)\}[m \| sid]$, which is constructed by employing the technique in [19] to combine the proof of "validity of a public key and a ticket" and the proof of "fulfillment of a policy" for each public key system, where $\mathcal{C}_i$ consists of all public keys of Ψ_i that are in $\mathcal{C}$, and $\mathcal{L}^{(i)}$ consists of all rating records in $\mathcal{L}$ but for each record the ticket $\mathcal{T}' = (\tau_1', \ldots, \tau_k')$ is replaced with τ_i'. Upon receiving

[3] Checking the validity of $\mathcal{C}$ and $\mathcal{L}$ may be difficult and time-consuming for a normal user, so he could skip this step or only verify a fraction of them. This will not degrade the security too much since the requirement is public to everyone and an invalid requirement in the public ledger may be discovered by an honest participant soon.

the response $(\mathcal{T}, \Pi_A)$, *sid* verifies the proof and sends the result, which will be "accept" iff the proof is valid, back to *uid*.

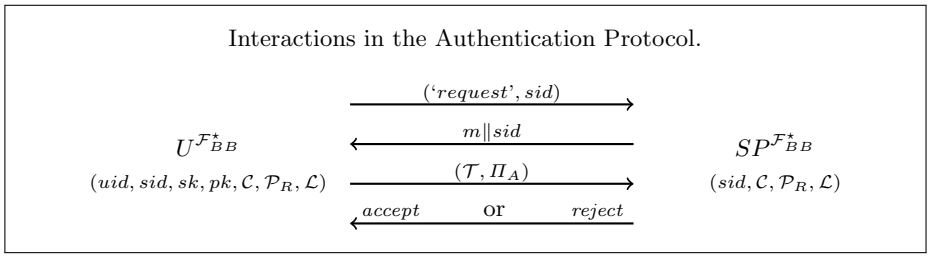


Fig. 3 Interactions in the authentication protocol. Here, we use "U" to denote the user, and "SP" to denote the service provider.

Interaction with The Ledger. To obtain data from the public ledger, a participant just needs to submit a "retrieve" request to $\mathcal{F}^{\star}_{BB}$. To put data to the public ledger, a SP just needs to submit a "store" request together with its permanent pseudonym and its data to $\mathcal{F}^{\star}_{BB}$. The submitted data vary depending on the purpose of the SP. In particular, when a SP would like to submit a rating s, it needs to put a tuple $(rid, \mathcal{T}, s, \Gamma)$ to the public ledger, where *rid* is a unique string identifying this rating record, $\mathcal{T}$ is the ticket for the rated authentication event, and Γ is the transcript of this authentication event, which is used to prove that the rated authentication event can be accepted by this SP. When a SP would like to submit a revocation of a rating record *rid*, it needs to put a tuple $('revoke', rid)$ to the public ledger. When a SP would like to publish a new requirement, it first generates a valid requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$, then puts it to the public ledger. To generate a valid requirement, apart from meeting those demands listed in Sec. 3.1, the SP should further ensure that each selected user in $\mathcal{C}$ is attached with a valid proof Π_R. We remark that all those data uploaded to the public ledger will not be verified in this phase, instead, the verification will be postponed until the data are used.

The Security. Security of our system is guaranteed by Theorem 1 stated as following, whose proof is put in Appendix A.2.

Theorem 1. *The system presented in Sec. 4.2 is a secure DBLACR system if each sub-protocol has the properties demanded in Sec. 4.1.*

5 Instantiations from Different Public Key Cryptographic Systems

To demonstrate the utility of our general framework, in this section, we instantiate sub-protocols defined in Sec. 4.1 under three different types of public key

systems, namely, the classical DL system, the pairing based system, and the RSA system.

The system built from the classical DL system based instantiations can be viewed as an extension of the decentralized anonymous credential system presented in [27]. Similar to the system in [27], our system also works in a q-order subgroup $(\mathbb{G}, q, \mathfrak{g})$ of the group $\mathbb{Z}_p^*$ for large primes p and q. Moreover, secret keys of users are also random numbers in $\mathbb{Z}_q$, and we will also employ the technique in [27] to conduct the proof of "the possession of the secret key" and the proof of "the validity of a public key and a ticket". However, in our instantiations, the credential (public key) of a user with a secret key x will be $\mathfrak{g}^x$ instead of a commitment of x. Besides, compared to the system in [27], here we further need a ticket generation algorithm and a proof of "the fulfilment of a policy", which are constructed by applying the method in [6]. One obstacle in extending the construction in [6], which is based on the pairing system, to our classical DL setting is that we cannot find a proper classical DL based CL signature scheme. To solve this problem, we use the dynamic accumulator scheme employed in constructing the proof of "the validity of a public key and a ticket" instead.

The system built from the pairing system based instantiations looks very similar to the one instantiated from the classical DL system, due to resemblances between the two public key systems in the abstract level. However, in a classical DL system, group elements are numbers, while in a pairing system, group elements are points in an elliptical curve. As a result, the accumulator, which is employed to accelerate the membership proof used in the proof of "the validity of a public key and a ticket" in the classical DL setting, cannot be applied here in the pairing setting. Therefore, we can only employ the "or proof" directly to complete the task.

Note that the proof of "the validity of a public key and a ticket" is constructed inefficiently in both the classical DL setting and the pairing setting, due to the inefficiency of the membership proof in these two constructions. More precisely, in the classical DL setting, similar to that in [27], one needs to prove that he possesses the secret key of a public key accumulated in an accumulator. As the secret key is the discrete logarithm of the public key on $\mathfrak{g}$, double discrete logarithm proof should be employed, which implies that the prover needs to perform 80 to 128 iterations of the protocol. Things get even worse in the pairing setting, since here one have to use the "or proof" directly to prove that his public key is in a specific set, which will lead to a communication and computation cost that grow linearly with the size of the set. To solve these problems, we propose the new RSA system based instantiations, where the secret key and the public key are in a polynomial relation (instead of an exponent relation), which can better fit the accumulator related proofs and lead to an improved efficiency.

Next, we will provide detailed construction of sub-protocols from the RSA system in Sec. 5.1, and construction of sub-protocols from the classical DL system in Sec. 5.2. We omit the pairing system based instantiation here since it is too similar to the classical DL system based instantiation.

5.1 Instantiations of Sub-Protocols from The RSA System

Our RSA based sub-protocols works in a quadratic residue group QR_N with a generator $\mathfrak{g}$, where N is the product of two big safe prime numbers.

The Key Genaration Algorithm. Upon input a security parameter 1^λ, the key generation algorithm generates the secret key $sk = (p, q)$ and the public key $pk = n$, where p and q are two safe primes and $n = 2pq + 1$ is also a prime.

To check the validity of a key pair given $pk = n$ and $sk = (p, q)$, one just needs to check that p and q are safe prime numbers with identical lengths, $n = 2pq + 1$, and n is a prime. Onewayness of the key generation algorithm comes from the factoring assumption directly, and the key generation algorithm is collision resistant because the map from the secret key to the public key is injective.

The Ticket Genaration Algorithm. Upon input a secret key $sk = (p, q)$, the ticket generation algorithm generates the ticket $\tau = (b, t)$ by first sampling $r \xleftarrow{\$} \mathbb{Z}_N$, then computing $b = \mathfrak{g}^r \mod N$ and $t = b^{p+q} \mod N$.

Given a secret key $sk = (p, q)$ and a ticket $\tau = (b, t)$, to verify the validity of the ticket, one just needs to check whether $t = b^{p+q} \mod N$. Indistinguishability of the ticket generation algorithm comes from the LD-RSA assumption and the DDH-II assumption. Formally, we describe this in Lemma 1 and prove this in Sec. A.1.

Lemma 1. *Assuming both the LD-RSA assumption and the DDH-II assumption hold in the quadratic residue group QR_N, then the RSA system based ticket generation algorithm has the indistinguishability property.*

Given two secret keys $sk = (p, q)$ and $sk' = (p', q')$ that $S(sk, \tau) = S(sk', \tau) = 1$ for some $\tau = (b, t)$, we can conclude that $(p + q) = (p' + q') = \log_b t \mod \varphi(N)/4$. In fact, in our system, we also restrict that valid secret keys cannot be too large, namely, cannot exceed $\varphi(N)/4$, and implicitly require the prover to prove this in their proof, so we actually have $p + q = p' + q'$. As a result, the set of valid tickets for sk is identical to that for sk', i.e. for any τ' in the range of the ticket generation algorithm, $S(sk, \tau') = S(sk', \tau')$. Also, to compute a secret key sk given the public key $pk = n$ for sk and sk', one just needs to compute $\mu = p' + q'$ and $\nu = (n - 1)/2$, and solve the equation $x^2 - \mu x + \nu = 0$.

An SPK System Proving The Possession of The Secret Key. In this proof system, one needs to prove that he possesses a secret key $sk = (p, q)$ for a properly generated public key $pk = n$. Here, we divide the task into two parts. First, the prover proves that $(n - 1)/2$ is a product of two *primes*. This is accomplished by employing the proof system proposed in [28], and actually, we only need the first two stages of the proof.[4] Then, the prover needs to prove that he knows two numbers p, q with identical lengths that satisfy $2pq + 1 = n$. To instantiate this proof system, we apply the framework presented in [31], which provides a simple

[4] To ensure an honest user can generate the proof, we further require that the secret key $sk = (p, q)$ satisfies that $p \neq q \mod 8$ in the key generation algorithm.

method to prove knowledge of discrete logs that are in an interval and fulfil a set of equations over groups of unknown order. More precisely, let g, h be random generators of the group QR_N and $n' = (n-1)/2$, the prover samples $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_N$, computes $a = qr_1$, $C_0 = g^{r_1}$, $C_1 = g^p h^{r_1}$, $C_2 = g^q h^{r_2}$, and proves $SPK\{(p, q, r_1, r_2, a) : C_0 = g^{r_1} \wedge C_1 = g^p h^{r_1} \wedge C_2 = g^q h^{r_2} \wedge C_0^q = g^a \wedge g^{n'} h^a = C_1^q \wedge p \in [2^{\ell_n/2} - \Delta, 2^{\ell_n/2} + \Delta] \wedge q \in [2^{\ell_n/2} - \Delta, 2^{\ell_n/2} + \Delta]\}$, where Δ is chosen according to the secret key space of the concrete key generation algorithm employed and we use ℓ_n to denote the length of n in this section. Finally, the generated proof is a simple concatenation of these two parts. As it has been proved in the first part that $(n-1)/2$ is a product of two primes, and in the second part that $pq = (n-1)/2$ and p and q are of identical lengths, the proof can ensure that (p, q) is exactly the secret key for the properly generated pk.

An SPK System Proving the Validity of A Public Key and A Ticket. In this proof system, one needs to prove that he possesses a secret key $sk = (p, q)$ associated with a public key $pk = n$ in a given set $\mathcal{C}$ and that a given ticket $\tau = (b, t)$ is generated from this secret key. To prove the first part of the statement, we can apply the approach presented in [22], which also builds on the framework of [31]. More precisely, the prover first accumulates public keys in $\mathcal{C}$ with a dynamic accumulator, then proves in zero-knowledge the possession of the secret key of a public key in the accumulator. To further prove that the given ticket is also generated from the same secret key, he can just plug the equation $t = b^p b^q$ into existing proofs. Formally, the prover needs to prove $SPK\{(p, q, n, r, a_1, a_2) : T_1 = g^r \wedge T_2 = h^r g^n \wedge T_1^n = g^{a_1} \wedge T_3 = s^r g^q \wedge T_1^q = g^{a_2} \wedge T_4^n = v y^{a_1} \wedge T_5^q g = z^{a_2} g^n \wedge t = b^{p+q} \wedge n \in [2^{\ell_n} - \Delta_1, 2^{\ell_n} + \Delta_1] \wedge q \in [2^{\ell_n/2} - \Delta_2, 2^{\ell_n/2} + \Delta_2]\}$, where Δ_1, Δ_2 are chosen according to the concrete key generation algorithm used, $r \xleftarrow{\$} \mathbb{Z}_N$, $a_1 = rn$, $a_2 = rq$, g, h, y, z, s are random generators of the group QR_N, $T_1 = g^r$, $T_2 = h^r g^n$, $T_3 = s^r g^q$, $T_4 = w y^r$, $T_5 = z^r g^{2p}$, $v = \mathfrak{g}^{\prod_{n' \in \mathcal{C}} n'}$ is the accumulator for the set $\mathcal{C}$ and $w = \mathfrak{g}^{\prod_{n' \in \mathcal{C} - \{n\}} n'}$ is the witness for $n \in \mathcal{C}$. We remark that as in our system, all selected public keys are valid, the prover does not need to reprove this here.

An SPK System Proving The Fulfilment of A Policy. In this proof system, given a ticket $\tau = (b, t)$, a policy $\mathcal{P}_R$ and a rating records list $\mathcal{L}$, one needs to prove that he possesses a secret key $sk = (p, q)$ that represents a user whose scores evaluated according to $\mathcal{P}_R$ and $\mathcal{L}$ satisfies $\mathcal{P}_R$, and that τ is generated from sk. We exploit the idea in [6] to construct the proof system, but will employ RSA-based cryptographic primitives instead of those pairing-based ones. In particular, we will apply strong-RSA assumption based additive homomorphic commitment scheme [26] and CL signature scheme [14], and we will also apply the framework in [31] to construct our proof system.

More precisely, the proof system is constructed in two tiers. First, as the policy $\mathcal{P}_R$ is evaluated by checking whether scores of a user in different categories satisfy a given DNF formula, we can first construct proof systems for each literal separately and combine them by the general framework in [19] then. To construct the proof system for each literal, which proves that the score of the user for a

category is not less than (or 'less than' if a negation symbol appears) a given threshold, we first filter data related to the target category. In more detail, we regard each rating record as a term (b, t, ς), where (b, t) is the ticket for an authentication event and ς is the rated score in the target category for this authentication event. According to the value of ς, we divide the whole rating records list into two parts, namely the meritlist, which contains terms with $\varsigma \geq 0$, and the blacklist, which contains the rest terms. Also, we only consider the two adjusting factor lists for the target category. Before constructing the proof system, we still need to solve two problems, one of which is that we need a way to prove that correct adjusting factor is used, and the other one is that we need an accurate range proof. Both problems can be solved by introducing the CL signature scheme. In more detail, to ensure that the correct adjusting factor is used, each adjusting factor list will be attached with a public key of the CL signature scheme, and each term (an adjusting factor) in the list will also be attached with a signature (signed by the service provider), which is signed on the adjusting factor together with its index, and is used to bind them. Besides, to build an accurate range proof, which is used to prove that the score of the user is not less than (or less than) the threshold θ for the target category, we also require the service provider in our system to attach the threshold θ with a public key of the CL signature scheme as well as signatures for integers in $[\theta, \theta_{max}]$ (or $[\theta_{min}, \theta]$), where θ_{max} and θ_{min} are the upper bound and the lower bound of possible scores in the target category respectively. Now, let $x = p + q$, l_m be the size of the meritlist, l_b be the size of the blacklist, the proof consists of the following parts:

1. A set of auxiliary commitments $aux^+ = (C_1^+, \widetilde{C}_1^+, \ldots, C_{l_m}^+, \widetilde{C}_{l_m}^+)$ together with a proof $\Pi^+ = (\Pi_1^+, \ldots, \Pi_{l_m}^+)$. Let $(\mathfrak{n}, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{b}, \mathfrak{c})$ be the public key of the CL signature scheme for the adjusting factor list of the meritlist. Also, for each $i \in [1, l_m]$, let (b_i, t_i, ς_i) be the ith element in the meritlist, $\kappa_i = \|\{j \mid j \in [1, i] \wedge b_j^x = t_j \mod N\}\|$, and $(\Delta_{\kappa_i}, (e_{\kappa_i}, s_{\kappa_i}, v_{\kappa_i}))$ be the κ_ith element in the adjusting factor list for the meritlist. Then, for $i \in [1, l_m]$: 1) if $b_i^x \neq t_i \mod N$, then both C_i^+ and $\widetilde{C}_i^+$ are commitments of 0; 2) otherwise, $\widetilde{C}_i^+$ is a commitment of 1, and C_i^+ is a commitment of $\Delta_{\kappa_i}\varsigma_i$. Also, for each $i \in [1, l_m]$, Π_i^+ proves that C_i^+ and $\widetilde{C}_i^+$ are properly generated, and is an "or proof" of two parts:
   The first part is for the case that $b_i^x \neq t_i \mod N$, and is the proof $SPK\{(x, \alpha_{i,1}, \alpha_{i,2}, \beta_{i,1}, \beta_{i,2}, \gamma_i, \widetilde{\gamma}_i) : t = b^x \wedge U_{i,2} = g_1^{\alpha_{i,1}} g_2^{\alpha_{i,2}} \wedge 1 = U_{i,2}^{-x} g_1^{\beta_{i,1}} g_2^{\beta_{i,2}} \wedge U_{i,1} = t_i^{\alpha_{i,1}} b_i^{-\beta_{i,1}} \wedge C_i^+ = g_2^{\gamma_i} \wedge \widetilde{C}_i^+ = g_2^{\widetilde{\gamma}_i} \wedge x < 2^{\ell_n/2+2}\}$, where $\alpha_{i,1}$, $\alpha_{i,2}$, γ_i and $\widetilde{\gamma}_i$ are sampled uniformly at random from $\mathbb{Z}_N$, $\beta_{i,1} = \alpha_{i,1}x$, $\beta_{i,2} = \alpha_{i,2}x$, g_1, g_2 are generators of QR_N, $U_{i,1} = t_i^{\alpha_{i,1}} b_i^{-\beta_{i,1}}$, and $U_{i,2} = g_1^{\alpha_{i,1}} g_2^{\alpha_{i,2}}$. We remark that to verify the validity of this part, the verifier should also reject the proof if $U_{i,1} = 1$.
   The second part is a proof for the case $b_i^x = t_i \mod N$, and is a concatenation of three proofs $\Pi_{1,i}^+, \Pi_{2,i}^+$ and $\Pi_{3,i}^+$. The first proof $\Pi_{1,i}^+$ is the main body of this part, and we have $\Pi_{1,i}^+ = SPK\{(x, \gamma_i, \widetilde{\gamma}_i, \kappa_i, \beta_{i,3}, \Delta_{\kappa_i},$

$\alpha_{i,3}, \alpha_{i,4}) : t = b^x \wedge t_i = b_i^x \wedge \widetilde{C}_i^+ = g_1 g_2^{\widetilde{\gamma}_i} \wedge \widetilde{D}_i = g_1^{\kappa_i} g_2^{\beta_{i,3}} \wedge C_i^+ = U_{i,3}^{\Delta_{\kappa_i}} g_2^{\gamma_i} \wedge U_{i,4} = g_1^{\kappa_i} g_2^{\alpha_{i,3}} \wedge U_{i,5} = g_1^{\Delta_{\kappa_i}} g_2^{\alpha_{i,4}} \wedge x < 2^{\ell_n/2+2}\}$, where $\alpha_{i,3}$, $\alpha_{i,4}$, γ_i and $\widetilde{\gamma}_i$ are sampled uniformly at random from $\mathbb{Z}_N$, $\beta_{i,3} = \sum_{j=1}^{i} \widetilde{\gamma}_j$, g_1, g_2 are generators of QR_N, $\widetilde{D}_i = \prod_{j=1}^{i} \widetilde{C}_j$, $U_{i,3} = g_1^{\varsigma_i}$, and $U_{i,4}$ and $U_{i,5}$ are commitments of κ_i and Δ_{κ_i} respectively. The second proof $\Pi_{2,i}^+$, which proves the possession of a signature on κ_i and Δ_{κ_i}, works in the group $QN_{\mathfrak{n}}$ and we have $\Pi_{2,i}^+ = SPK\{(s_{\kappa_i}, v_{\kappa_i}, e_{\kappa_i}, \kappa_i, \Delta_{\kappa_i}, w_i, z_i, r_{w,i}, r'_{w,i}, r_{1,i}, r_{2,i}) : T_{v,i}^{e_{\kappa_i}} = \mathfrak{a}_1^{\kappa_i} \mathfrak{a}_2^{\Delta_{\kappa_i}} \mathfrak{b}^{s_{\kappa_i}} \mathfrak{c} g^{z_i} \wedge T_{w,i} = g^{w_i} h^{r_{w,i}} \wedge T_{w,i}^{e_{\kappa_i}} = g^{z_i} h^{r'_{w,i}} \wedge T_{1,i} = g^{\kappa_i} h^{r_{1,i}} \wedge T_{2,i} = g^{\Delta_{\kappa_i}} h^{r_{2,i}} \wedge 2^{\ell_e - 1} < e_{\kappa_i} < 2^{\ell_e} \wedge \kappa_i < 2^{\ell_m} \wedge \Delta_{\kappa_i} < 2^{\ell_m}\}$, where $w_i, r_{w,i}, r_{1,i}, r_{2,i}$ are sampled uniformly at random from $\mathbb{Z}_{\mathfrak{n}}$, $z_i = w_i e_{\kappa_i}$, $r'_{w,i} = r_{w,i} e_{\kappa_i}$, g, h are generators of $QR_{\mathfrak{n}}$, $T_{v,i} = v_{\kappa_i} g^{w_i}$, $T_{w,i}, T_{1,i}, T_{2,i}$ are commitments of $w_i, \kappa_i, \Delta_{\kappa_i}$ respectively, and ℓ_e, ℓ_m are defined in the signature scheme. Since $\Pi_{1,i}^+$ and $\Pi_{2,i}^+$ are generated in different groups, we also need the third proof $\Pi_{3,i}^+$ to connect them, which proves that $U_{i,4}$ and $T_{1,i}$ are commitments of the same value, and $U_{i,5}$ and $T_{2,i}$ are commitments of the same value. Here, we apply the technique in [16] to accomplish it.

2. A set of auxiliary commitments $aux^- = (C_1^-, \widetilde{C}_1^-, \ldots, C_{l_b}^-, \widetilde{C}_{l_b}^-)$ together with a proof Π^- proving that each items in aux^- is properly generated, which is generated in the same way as the generation of aux^+ and Π^+.
3. A proof Π_S proving that the final score, which is comitted in the commitment $C = \prod_{i=1}^{l_m} C_i^+ \prod_{i=1}^{l_b} C_i^-$, is above (or below) the threshold θ. It is sufficient to prove that one of the signature associated with the threshold θ is on the committed value of C, and we can prove this statement in the same way as we have used when generating $\Pi_{2,i}^+$ in Π^+.

Finally, the proof for each literal is $\Pi = (aux^+, aux^-, \Pi^+, \Pi^-, \Pi_S)$.

5.2 Instantiations of Sub-Protocols from The Classical DL System

Our classical DL system based sub-protocols works in a q-order subgroup ($\mathbb{G}$, $q, \mathfrak{g}$) of a group $\mathbb{Z}_p^*$, where p and q are both primes, and in a quadratic residue group QR_N, where N is the product of two big safe prime numbers.

The Key Genaration Algorithm. Upon input a security parameter 1^λ, the key generation algorithm generates the secret key $sk = x$ and the public key $pk = \mathfrak{g}^x$, where $x \xleftarrow{\$} \mathbb{Z}_q$.

To check the validity of a key pair given $pk = h$ and $sk = x$, one just needs to check whether $h = \mathfrak{g}^x$. Onewayness of the key generation algorithm comes from the DL assumption directly, and the key generation algorithm is collision resistant because the map from the secret key to the public key is injective.

The Ticket Genaration Algorithm. Upon input a secret key $sk = x$, the ticket generation algorithm generates the ticket $\tau = (b, t)$ by first sampling $r \xleftarrow{\$} \mathbb{Z}_q$, then computing $b = \mathfrak{g}^r$ and $t = b^x$.

Given a secret key $sk = x$ and a ticket $\tau = (b, t)$, to verify the validity of the ticket, one just needs to check whether $t = b^x$. Indistinguishability of the ticket generation algorithm comes from the DDH assumption directly. The verifying consistency and the connectivity hold since for any ticket τ, there exists exactly one secret key sk satisfying $S(sk, \tau) = 1$.

An SPK System Proving The Possession of The Secret Key. In this proof system, one needs to prove that he possesses a secret key $sk = x$ for a properly generated public key $pk = h$. Here we can use the protocol presented in [37], which proves knowledge of discrete logarithms, to complete this task.

An SPK System Proving the Validity of A Public Key and A Ticket. In this proof system, one needs to prove that he possesses a secret key $sk = x$ associated with a public key $pk = h$ in a given set $\mathcal{C}$ and that a given ticket $\tau = (b, t)$ is generated from this secret key. To prove the first part of the statement, we can apply the same technique used in [27]. More precisely, the prover first uses the Stron-RSA assumption based dynamic accumulator, which works in the quadratic residue group QR_N, to accumulate all public keys in $\mathcal{C}$. Then he uses the double discrete logarithm technique [38] to prove the possession of the secret key of a public key in the accumulator. To further prove that the given ticket is also generated from the same secret key, he can just plug the equation $t = b^x$ into existing proofs.

An SPK System Proving The Fulfilment of A Policy. In this proof system, given a ticket $\tau = (b, t)$, a policy $\mathcal{P}_R$ and a rating records list $\mathcal{L}$, one needs to prove that he possesses a secret key $sk = x$ that represents a user whose scores evaluated according to $\mathcal{P}_R$ and $\mathcal{L}$ satisfies $\mathcal{P}_R$, and that τ is generated from sk. We exploit the idea in [6] to construct the proof system, but will employ classical DL system based cryptographic primitives instead of those pairing-based ones. One problem is that we cannot find a proper classical DL based CL signature scheme to help prove that correct adjusting factor is used and that a score is in the required interval. To solve this problem, we use the strong-RSA based dynamic accumulator. More precisely, for an adjusting factor list with m elements, let $l = 2^{\ell_l}$ be a proper number greater than the upper bound of weights in the adjusting factor list; let L be a proper number much larger than $(m+1)l$, and $L' = 2^{\ell_L}$ satisfying $(m+1)l < L' < L$; given C_1 and C_2, which are commitments of an index i and a weight Δ, let k be the minimal natural number satisfying $i \cdot l + \Delta + k \cdot L$ is prime; to prove that Δ is exactly the ith element in an adjusting factor list, the prover first computes C_k as a commitment of k, $C_3 = C_1^l \cdot C_2$, and $C = C_3 \cdot C_k^L$. Then he computes $z_j = j \cdot l + \Delta_j + k_j \cdot L$ for $j \in [1, m]$, where Δ_j is the jth element in the adjusting factor list and k_j is the minimal natural number satisfying z_j is a prime, and accumulates all those z_j. Finally, he proves that the committed value in C is in the accumulator, the committed value in C_3 is in $[0, L' - 1]$, and the commitment value in C_2 is in $[0, l - 1]$, where the latter two statements can be proved via a range proof [33]. To prove that a score s is in an interval $[l, u]$ given a commitment C of s, let L be a proper number much larger than u, $L' = 2^{\ell_L}$ satisfying $u < L' < L$,

and k be the minimal natural number satisfying $s + kL$ is a prime, the prover first computes C_k as a commitment of k and $C' = C \cdot C_k^L$. Then he computes $z_i = i + k_i L$ for $i \in [l, u]$, where k_i is the minimal natural number satisfying z_i is a prime, and accumulates all those z_i. Finally, he proves that the committed value in C' is in the accumulator and the committed value in C is in $[0, L' - 1]$.

6 The Implementation

To demonstrate the practicability of our system, in this section, we provide a proof of concept implementation for it. The implementation includes two relatively independent parts, namely, the public ledger part and the credential system part, and we describe the results for them in Sec. 6.1 and in Sec. 6.2 respectively.

6.1 The Public Ledger

First, we explore how the public ledger could be realized. As have discussed in Sec. 2, the public ledger can be instantiated via the blockchain technique. So, we choose the Bitcoin and the Ethereum, which are the two most popular blockchain technique instantiations currently, as the test object. The test is conducted on a personal computer with a 3.16GHz Intel(R) Core(TM)2 Duo Processor E8500, 8GB RAM and 500GB disk, running ARCHLinux version 4.10.6. The Bitcoin client run in the experiment is Bitcoin Core Version 0.14.0 and the Ethereum client is go-ethereum 1.5.9. The result is summarized in Table 2.

Table 2: Comparison of Public Ledger Instantiations.

	Bitcoin	Ethereum
Market Cap	19257718797 USD	4376127411 USD
Initial Data Size	118GB	15 GB
Initial Sync Time	9h	5h
Ease of Use	Difficult	Easy
Data Size Limit	80bytes	*
Cost	0.5342 USD	0.0225 USD
Confirmation Time	6min / 70min	a few seconds / 3min

The row "Market Cap" indicates the market capitalizations of each instantiation, and the data come from [1]. This can reflect the robustness of the blockchain to some extent. The row "Initial Data Size" and the row "Initial Sync Time" indicates the disk space and time needed before one could employ the public ledger. We remark that both results are tested in the "fast" mode: the latest Bitcoin core client introduce a default "assumevalid" option, which will hardwire the latest

block in the software and skip the verification of content in preceding blocks; the Ethereum blockchain is synced with the "fast" option, with which each transaction in blocks old enough will be partially downloaded (only the transaction outcomes are downloaded) and will not be verified. The row "Ease of Use" and the row "Data Size Limit" indicates the accessibility of using blockchain as a public ledger. For Bitcoin, in each transaction, there exists a field OP_RETURN allowing one to put up to 80 bytes arbitrary data [3] on it, but it seems that the Bitcoin community do not hope people to use this field, and the client Bitcoin Core also does not provide a convenient way to implement this functionality. Thus, we test this facility via a third party open source project on GitHub [30]. For Ethereum, putting data in a transaction is natively supported. There is also no explicit limits on the size of data put in a transaction, but for each block, there is a block gas limit, which is about 4 millions for current blocks. As it will consume gas to attach data to a transaction, one could only put dozens to hundreds kilobytes data in one transaction now according to the content of the data. The row "Cost" indicates the amount of money cost to put data on the blockchain. For Bitcoin, this is the transaction fee for rewarding the miners. One can send a transaction with no transaction fee, but this transaction may be not handled by miners in time and even be regarded as a spam transaction. According to statistics (data from [4]), to hope miners to deal with the transaction immediately, the transaction fee should be above 1.8×10^{-6} BTC per byte, and for our purpose, which will send a transaction of about 250 bytes (about 200 bytes for the basic transaction and about 50 bytes for the attached data), the transaction fee should be 0.00045 BTC, which is about 0.5342 USD according to the price of 1 BTC at April 14th, 2017. For Ethereum, the cost comes from the gas consumed. Currently, each gas is about 2×10^{-8} Ether, and according to the yellow paper of Ethereum [43], a transaction will cost 21000 gas for itself, and each non-zero byte put in the data field will cost 68 gas, so it could cost one about 0.000556 Ether, or about 0.0266 USD according to the price of 1 Ether at April 14th, 2017, if he would like to put 100 bytes on the Ethereum blockchain. In our experiment, we put 32 bytes in a transaction and this cost us 0.00047 Ether (0.0225 USD). The row "Confirmation Time" indicates the time needed to wait for the transactions and the data to be confirmed. For Bitcoin, on average, it will take 10 minutes to generate a new block, so on average, it will take about 5 minutes to see the data appear on the blockchain, and about 1 hour to confirm that the data are put in the blockchain (6 confirmation). For Ethereum, the new block appears every a few seconds, so the data will appear on the blockchain immediately. As claimed by the Ethereum Blog [2], 10 confirmation in Ethereum is enough to achieve a similar degree of security as that of 6 confirmation in Bitcoin, so it may take about 3 minutes to wait for the confirmation of the transaction/data.

From the experiment result, we observe that neither the Bitcoin nor the Ethereum can support large data storage. So in practice, to use them as a public ledger, one should first upload the data to some public cloud, then put the link (40 to 60 bytes for a dropbox link and 10 to 20 bytes if google url shorten service is

used) and hash value of the data (32 bytes if SHA-256 is used) to the blockchain. In this way, the functionality of the public ledger still reserves. Another problem is that while it is quite easy for a service provider to sync and maintain a Bitcoin blockchain or an Ethereum blockchain in its server, this is not the case for a normal user. To tackle this problem, we suggest users with constrained devices to use a lightweight client or refer to an online service to complete interactions with the public ledger (they could exploit multiple approaches to retrieve data to boost the security), and this will not harm the security as long as there exists services providing correct Bitcoin or Ethereum blockchain information. When comparing the Bitcoin and the Ehtereum, it seems that the Bitcoin blockchain is more robust, while the Ethereum is also very secure and is much more convenient to use. Thus, in practice, Ethereum seems a better choice. There are also some other blockchain instantiations, e.g. the Namecoin, the Emercoin, the Nxt etc., providing convenient ways to putting data on the blockchain, but the market capitalizations for them is much smaller (a few thousandth) compared to the Bitcoin and the Ethereum, and it seems that they do not provide much better features compared to Ethereum. Therefore, we prefer to employ Ethereum to realize our system.

6.2 The Credential System

Then we examine the practicality of the credential system part of our system. The implementation is for the RSA-based instantiation, which is the most efficient one among all three instantiations. To simplify the criterion for evaluating the experiment result, we only consider a simple policy with a single category, threshold 0, and no adjusting factor, and a rating records list with one blacklist. The experiment is conducted on a Macbook Pro with 8GB of 1866MHz LPDDR3 onboard memory and a 2.7GHz dual-core Intel Core i5 processor, running OSX 10.12.4. The test code is written in C based on the OPENSSL library (version 1.0.2).

There are two main operations, namely the registration and the authentication, in the system, thus our experiment also focuses on the performance of these two protocols. First, we test the performance of the registration protocol, including the time for a user to generate a credential, the time for a service provider to verify a credential, and the credential size. As the user may already have a key pair when joining the system, the time consumption for generating a credential is tested in two modes, namely the normal mode, where the user needs to generate both the key pair and the proof, and the pre-computation mode, where credential is generated on a given public key/secret key pair. Then, we test the performance of the authentication protocol, including the time for generating a proof, the time for verifying a proof, and the size of the proof. Since the user can access the requirement in advance and precompute some parts, we will test the times for generating a proof both with and without pre-computation.

The experiment performance is measured under different parameters, including the security parameter, the candidate users set size, and the blacklist size. In more detail, we will consider security parameters of 1024 bits, 2048 bits, and

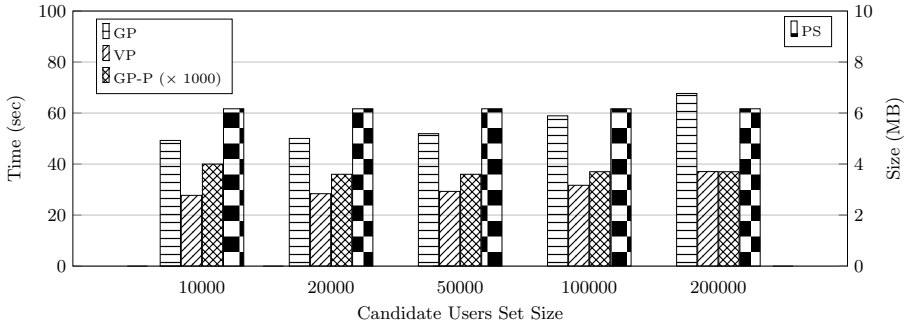
3072 bits, which can achieve a security strength of about 80 bits, 112 bits, and 128 bits respectively (according to [8]), and summarize the performance of our system under different security parameters in Table 3. we will consider candidate users set size of 10000, 20000, 50000 100000, and 200000, and blacklist size of 1000, 2000, 3000, 4000, and 5000, and summarize the performance of the authentication protocol under these parameters in Figure 4. When analyzing the relation between the performance and one particular parameter, the other two parameters will be set as default, and the default values of the security parameter, the candidate users set size, and the blacklist size are 2048 bits, 50000, 3000 respectively. Besides, we also test the performance for the setting with an empty blacklist, which is exactly the scenario considered in [27], and compare our results with theirs in Figure 5.

Table 3: The performance of the registration protocol and the authentication protocol under different security parameters with 50000 users and 3000 blacklist records.

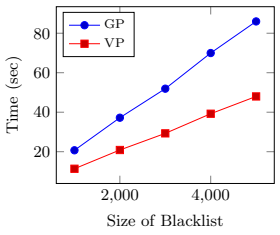
	GC	GC-P	VC	CS	GP	GP-P	VP	PS
1024 bits	1.316s	0.153s	0.047s	70.1 KB	10.878s	0.021s	5.686s	3.1 MB
2048 bits	19.296s	0.932s	0.295s	139.9 KB	51.917s	0.036s	29.289	6.2 MB
3072 bits	69.578s	2.959s	0.910s	209.8 KB	142.123s	0.047s	84.872s	9.3 MB

Here, we use GC, GC-P, and VC to denote time consumed in generating a credential, generating a credential with pre-computation, and verifying the validity of a credential respectively; we use GP, GP-P, and VP to denote time consumed in generating a proof, generating a proof with pre-computation, and verifying a proof respectively; and we use CS and PS to denote the size of a credential and an authentication proof respectively.

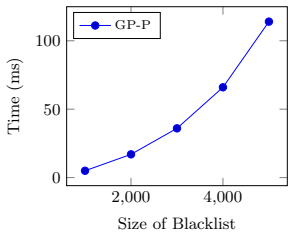
From the experiment results, we can conclude that our system is quite practical when deployed in practice. First, at the user side, the time consumption in the registration phase is acceptable even no pre-computation is made, and it is extremely fast to generate a proof in the authentication phase if the user could pre-compute. At the service provider side, it is also fairly fast to verify the validity of a credential, but it seems time-consuming to verify the validity of a proof. Nonetheless, the service provider often controls more computation resources, and it can also further reduce the time consumption via parallelizing the verification algorithm, so it will take less time to wait for the verification in real world applications. Besides, the size of the credential and the proof is also not very large, thus the communication cost of our system is also quite low. One advantage of our system is that the proof size is independent of the size of the candidate users set, and the time costs at both the user side and the service provider side hardly increase with the increasing of the candidate users set size, thus our system is scalable in the number of supported users. It worth noting that this is important for the usefulness of our system, since a large number of registered users is always desired to protect the privacy of particular users. However, this is not the case for the blacklist size, as both the communication cost and the computation cost grow linearly with the size of the blacklist. So,



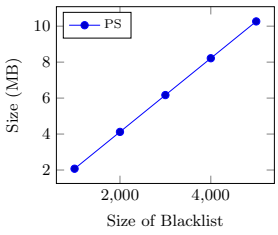
(a) Performance for the authentication protocol under different candidate users set size with security parameter 2048 bits and 3000 blacklist records. GP, GP-P and VP are times for generating a proof without pre-computation, generating *1000* proofs with pre-computation, and verifying a proof respectively, and PS is the size of the authentication proof.



(b) Time for generating and verifying a proof under different blacklist size with security parameter 2048 bits and 50000 users. GP and VP are times for generating (without pre-computation) and verifying a proof respectively.



(c) Time (in milliseconds) for generating a proof with pre-computation under different blacklist size with security parameter 2048 bits and 50000 users.



(d) Size (in Megabyte) of an authentication proof under different blacklist size with security parameter 2048 bits and 50000 users.

Fig. 4 Performance of our system under different candidate users set size and blacklist size.

it is better to employ our system in settings with a small blacklist. We leave how to upgrade the system to being scalable in the size of the blacklist as an open problem. When comparing the efficiency of our system with that in [27], we observe that our efficiency is much better than theirs, thus our system is preferable even no revocation is considered.

References

[1] Cryptocurrency market capitalizations. `https://coinmarketcap.com/`. Accessed: 2017-04-15.

[2] On slow and fast block times. `https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/`.

[3] Op_return. `https://en.bitcoin.it/wiki/OP_RETURN`. Accessed: 2017-04-15.

[4] Predicting bitcoin fees for transactions. `https://bitcoinfees.21.co/`. Accessed: 2017-04-14.

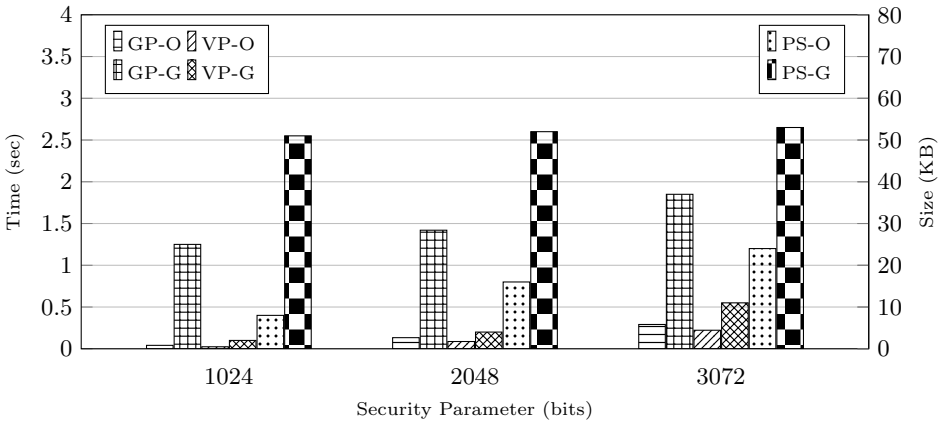


Fig. 5 Comparison between the performance of our authentication protocol with empty blacklist and the performance of the authentication protocol in [27]. Since in their experiment, accumulator is computed separately, we also do not count time consumed by this part in the test. Here, GP-O and VP-O are times for generating an authentication proof without pre-computation and verifying an authentication proof in our system respectively; GP-G and VP-G are respective times in [27]; and PS-O and PS-G are our authentication proof size and theirs respectively.

[5] M. H. Au and A. Kapadia. Perm: Practical reputation-based blacklisting without ttps. In *CCS*, pages 929–940. ACM, 2012.

[6] M. H. Au, A. Kapadia, and W. Susilo. Blacr: Ttp-free blacklistable anonymous credentials with reputation. In *NDSS*, 2012.

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

[8] E. Barker. Recommendation for key management–part 1: General (revision 4). 2015.

[9] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC*, pages 356–374. Springer, 2008.

[10] J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT*, pages 274–285. Springer, 1993.

[11] E. Brickell and J. Li. Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30. ACM, 2007.

[12] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118. Springer, 2001.

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

[14] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks*, pages 268–289. Springer, 2002.

[15] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72. Springer, 2004.

[16] J. Camenisch, M. Michels, et al. Separability and efficiency for generic group signature schemes. In *Annual International Cryptology Conference*, pages 413–430. Springer, 1999.
[17] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, page 455. Springer, 1997.
[18] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
[19] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187. Springer, 1994.
[20] I. B. Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In *CRYPTO*, pages 328–335. Springer, 1988.
[21] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
[22] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in ad hoc groups. In *EUROCRYPT*, pages 609–626. Springer, 2004.
[23] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
[24] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. Springer, 1986.
[25] C. Fromknecht, D. Velicanu, and S. Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptology ePrint Archive*, 2014:803, 2014.
[26] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30. Springer, 1997.
[27] C. Garman, M. Green, and I. Miers. Decentralized anonymous credentials. In *NDSS*, 2014.
[28] R. Gennaro, D. Micciancio, and T. Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *CCS*, pages 67–72. ACM, 1998.
[29] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.
[30] G. Greenspan. Project php-op_return. `https://github.com/coinspark/php-OP_RETURN`. Accessed: 2017-04-15.
[31] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT*, pages 571–589. Springer, 2004.
[32] P. Lofgren and N. Hopper. Faust: Efficient, ttp-free abuse prevention by anonymous whitelisting. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 125–130. ACM, 2011.
[33] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Public Key Cryptography*, pages 60–71. Springer, 1998.
[34] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *S&P*, pages 397–411. IEEE, 2013.
[35] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.
[36] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *S&P*, pages 459–474. IEEE, 2014.
[37] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[38] M. Stadler. Publicly verifiable secret sharing. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 190–199. Springer, 1996.
[39] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without ttps. In *CCS*, pages 72–81. ACM, 2007.
[40] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Perea: Towards practical ttp-free revocation in anonymous authentication. In *CCS*, pages 333–344. ACM, 2008.
[41] P. P. Tsang and V. K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In *International Conference on Information Security Practice and Experience*, pages 48–60. Springer, 2005.
[42] W. Wang, D. Feng, Y. Qin, J. Shao, L. Xi, and X. Chu. Exblacr: Extending blacr system. In *ACISP*, pages 397–412. Springer, 2014.
[43] G. Wood. Ethereum yellow paper, 2014.
[44] L. Xi and D. Feng. Farb: fast anonymous reputation-based blacklisting without ttps. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 139–148. ACM, 2014.
[45] K. Y. Yu, T. H. Yuen, S. S. Chow, S. M. Yiu, and L. C. Hui. Pe (ar) 2: Privacy-enhanced anonymous authentication with reputation and revocation. In *ESORICS*, pages 679–696. Springer, 2012.

A Omitted Proofs

A.1 Proof of Lemma 1

Proof. To prove Lemma 1, we define the following games, between whom the indistinguishabilities can be reduced to either the LD-RSA assumption or the DDH-II assumption.

- **Game$_0$.** This is the original game where the oracle is answered honestly. More precisely, when the challenge b is 0, then tickets of the secret key sk of the given public key pk are responded; when the challenge b is 1, then tickets sampled uniformly at random from the range of the tickets generation algorithm, namely QR_N^2, are responded.
- **Game$_1$.** This is identical to Game$_0$ except that when $b = 0$, tickets of a randomly generated secret key sk' are responded. Indistinguishability between Game$_0$ and Game$_1$ comes from the LD-RSA assumption directly. Note that given a ticket of a particular secret key (either sk or sk'), one can easily computes random tickets of the same secret key via rerandomizing the given ticket.
- **Game$_2$.** This is identical to Game$_1$ except that when $b = 0$, the challenger will first samples $r^* \xleftarrow{\$} \mathbb{Z}_N$, then answer each query with a tuple (h, h^{r^*}), where h is sampled freshly and uniformly from QR_N each time. Since the public key for sk' is not given, sk' still has a high min-entropy. Thus, indistinguishability between Game$_1$ and Game$_2$ can be reduced to the DDH-II assumption.

– **Game$_3$.** This is identical to Game$_2$ except that when $b = 0$, the challenger answers each query with a tuple (h_1, h_2), where both h_1 and h_2 are sampled freshly and uniformly from QR_N each time. Indistinguishability between Game$_3$ and Game$_2$ comes from the standard DDH assumption, which can be implied by the DDH-II assumption, directly.

Note that in Game$_3$, the oracle is answered identically for both the case $b = 0$ and the case $b = 1$, and that completes the proof. □

A.2 Proof of Theorem 1

Proof. We first define the ideal world adversary $\mathcal{S}$ based on a real world adversary $\mathcal{A}$, which proceeds as follows:
The Ideal World Adversary $\mathcal{S}$. In the beginning, the ideal world adversary $\mathcal{S}$ runs the setup protocol of the decentralized blacklistable anonymous credential system with reputation, and give the generated public parameter $params$ to $\mathcal{A}$. It also initializes the public ledger functionality $\mathcal{F}^{\star}_{BB}$. Besides, it initializes severarl private lists aiding its simulating. More precisely, it create two empty lists $\mathfrak{U}_A$ and $\mathfrak{U}_H$ for recording corrupted users and honest users respectively. Each item in $\mathfrak{U}_H$ will be a tuple (uid, sk, pk) where uid is the identifier of a user in the ideal world and (sk, pk) is the key pair for him in the simulated world, and each item in $\mathfrak{U}_A$ will be a tuple $(uid, \mathcal{K})$ where uid is the identifier of a user in the ideal world and $\mathcal{K}$ consists of key pairs for him in the simulated world. It also create lists $\mathfrak{S}_A$ and $\mathfrak{S}_H$ for recording corrupted service providers and honest service providers respectively, and this time, it fill these two lists with items in the form of "(sid, Nym)" instead of creating two empty lists, where sid is the identifier of a service provider in the ideal world and Nym is the pseudonym of a service provider for storing data to the public ledger in the simulated world. Here, $\mathcal{S}$ is able to store data via $\mathcal{F}^{\star}_{BB}$ under the pseudonym of any honest service provider. Besides, It also creates an empty list $\mathfrak{AL}$ for recording authentication events and an empty list $\mathfrak{RL}$ for recording the correspondence between rating records in the ideal world and that in the simulated world. Here, each item in $\mathfrak{AL}$ is a tuple $(tid, sid, \mathcal{T}, \Gamma)$, where tid and sid are the ideal world identifiers for the authentication event and for the invloved service provider respectively, and $\mathcal{T}$ and Γ are the ticket and the transcript for the authentication event in the simulated world respectively. Each item in $\mathcal{RL}$ is a tuple (rid, rid'), where rid and rid' are the ideal world identifier and the simulated world identifier for rating records respectively.

During the simulation, $\mathcal{S}$ will monitor the public ledger of $\mathcal{T}_P$ and that of the simulated world, thus can capture the latest modifications to them. For clarity of notation, we denote the public ledger of $\mathcal{T}_P$ as "public ledger" and denote that in the simulated world world as "bulletin board".

Once $\mathcal{A}$ registers a new user to the system via storing a tuple $(Nym, pk, \Pi_R, attr, aux)$ to the bulletion board, $\mathcal{S}$ will register this user in the ideal world via the following process. First, $\mathcal{S}$ verifies the validity of Π_R and attempts to extract a secret key from Π_R. If either the Π_R is invalid or the extraction fails,

$\mathcal{S}$ stops the registeration process. Assuming the extracted secret key is sk, then $\mathcal{S}$ computes $\tau \leftarrow TicketGen(sk)$. Next, for each item (uid', sk', pk') in $\mathfrak{U}_H$, $\mathcal{S}$ checks if $S(sk', \tau) = 1$, and stops the registeration if so. Also, for each item $(uid', \mathcal{K}) \in \mathfrak{U}_A$ and each $(sk', pk') \in \mathcal{K}$, $\mathcal{S}$ checks if $S(sk', \tau) = 1$ and appends (sk, pk) to the set $\mathcal{K}$ if so. If neither checks passes, then $\mathcal{S}$ creates a new user "uid" in the ideal world, register it in the ideal world by sending a registeration request with personal information $(attr, aux)$ to the trusted party $\mathcal{T}_P$, and adds a new item $(uid, \{(sk, pk)\})$ in $\mathfrak{U}_A$.

Once $\mathcal{T}_P$ writes a tuple $(uid, attr, aux)$ to the public ledger after an honest user registers himself in the ideal world, $\mathcal{S}$ also registers this user in the simulated world. More precisely, $\mathcal{S}$ first chooses one type of supported public key system and runs its key generation algorithm to generate the public key/secret key pair (pk, sk). Then it simulate a fake proof $\Pi_R = SPK\{(sk) : T(pk, sk) = 1\}$ via the simulation algorithm of the SPK system. Finally, it stores $(Nym, pk, \Pi_R, attr, aux)$ to the bulletin board where Nym is a newly generated pseudonym, and appends (uid, sk, pk) to $\mathfrak{U}_H$.

Once the adversary stores a new rating record $(Nym, rid, \mathcal{T}, s, \Gamma)$ to the bulletin board, $\mathcal{S}$ dumps this rating record to the ideal world via the following process. First, $\mathcal{S}$ queries Nym in $\mathfrak{S}_A$, and stops the dump process if no record with Nym is found in $\mathfrak{S}_A$. Note that this will not affect the usage of the rating record since a rating record is regarded as illegal if it is stored with a pseudonym not in $\mathfrak{S}_H \cup \mathfrak{S}_A$ and $\mathcal{A}$ cannot stores data to the bulletin board with a pseudonym of an honest service provider. $\mathcal{S}$ continues the dump process after it finds an item (sid, Nym) in $\mathfrak{S}_A$, and queries the tuple $(\mathcal{T}, \Gamma)$ in $\mathfrak{AL}$. If a record $(tid, sid', \mathcal{T}, \Gamma)$ is found in $\mathfrak{AL}$, $\mathcal{S}$ further checks whether sid is identical to sid'. It stops if $sid \neq sid'$, and submits (tid, s) to $\mathcal{T}_P$ if $sid = sid'$. If no record with $(\mathcal{T}, \Gamma)$ is found in $\mathfrak{AL}$, which means that this authentication event occurs between two parites controlled by the adversary, $\mathcal{S}$ will attempts to duplicate this authentication event in the ideal world. More precisely, it first verifies whether the authentication event is accepted by the service provider sid, and stops if the answer is no. Then it attempts to extract a secret key from the proof in Γ and query the extracted secret key in $\mathfrak{U}_A$. It stops if it fails in conducting these two steps. Assuming a secret key sk is extracted and a user uid in $\mathfrak{U}_A$ is located to possess the secret key sk, $\mathcal{S}$ then runs the authentication protocol in the ideal world on behalf of the user uid and the service provider sid. Note that to guarantee the consistency of the transcript Γ and the ideal world authentication event, $\mathcal{S}$ will also evaluate the requirement $(\mathcal{C}, \mathcal{P}_R, \mathcal{L})$ from Γ, submits it to the public ledger before starting the authentication, and restore the orginal one after the authentication event. After receiving a response (no matter whether it is '$valid$' or '$invalid$') along with a string tid from $\mathcal{T}_P$, $\mathcal{S}$ submits (tid, s) to $\mathcal{T}_P$ on behalf of sid, and adds a tuple $(tid, sid, \mathcal{T}, \Gamma)$ in $\mathfrak{AL}$. We reamrk that whenever $\mathcal{S}$ succeeds in submitting a rating to $\mathcal{T}_P$, it also put the tuple (rid', rid) into $\mathfrak{RL}$, where rid' is the identifier for the rating record in the ideal world.

Once $\mathcal{T}_P$ writes a tuple $(rid, sid, tid, s, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ to the public ledger after an honest service provider submits his rating to the trusted party, $\mathcal{S}$ also dumps this

rating record to the simulated world. In more details, $\mathcal{S}$ first queries sid in $\mathfrak{S}_H$ and gets a tuple (sid, Nym) back. Then it queries tid in $\mathfrak{AL}$. If it finds a tuple $(tid, sid, \mathcal{T}, \Gamma)$ in $\mathfrak{AL}$, it then stroes the tuple $(Nym, rid', \mathcal{T}, s, \Gamma)$ to the bulletin board, where rid' is chosen uniformly at random. If it fails in finding a tuple with tid in $\mathfrak{AL}$, which means the authentication event tid occurs between two honest parties, $\mathcal{S}$ will generates a fake transcript for this authentication event. To complete this, $\mathcal{S}$ first transforms $\mathcal{C}$ and $\mathcal{L}$ into their counterparts $\mathcal{C}'$ and $\mathcal{L}'$ in the real world. Then it generates a fake ticket $\mathcal{T}$ by randomly sampling each part, and generates a fake proof Π_A via the simulation algorithms of the SPK systems. The fake transcript is $\Gamma = (sid, \mathcal{C}', \mathcal{P}_R, \mathcal{L}'; \text{`}request\text{'} \| sid; m \| sid, ; (\mathcal{T}, \Pi_A))$, where m is chosen uniformly at random. Next, $\mathcal{S}$ stores the tuple $(Nym, rid', \mathcal{T}, s, \Gamma)$ to the bulletin board, where rid' is chosen uniformly at random, and adds a tuple $(tid, sid, \mathcal{T}, \Gamma)$ in $\mathfrak{AL}$. We reamrk that whenever $\mathcal{S}$ succeeds in storing a rating to the bulletin board, it also put the tuple (rid, rid') into $\mathfrak{RL}$, where rid' is the identifier for the rating record in the simulated world.

Once the adversary stores a revocation statement $(\text{`}revoke\text{'}, Nym, rid)$ to the bulletin board, $\mathcal{S}$ dumps this revocation statement to the ideal world via the following process. First, $\mathcal{S}$ queries Nym in $\mathfrak{S}_A$ and queries rid in $\mathfrak{RL}$. It stops the dump process if either no record with Nym is found in $\mathfrak{S}_A$ or no record with rid is found in $\mathfrak{RL}$. Next, $\mathcal{S}$ submits $(\text{`}revoke\text{'}, rid')$ to $\mathcal{T}_P$ on behalf of sid after it finds an item (sid, Nym) in $\mathfrak{S}_A$ and an item (rid', rid) in $\mathfrak{RL}$.

Once $\mathcal{T}_P$ writes a tuple $(\text{`}revoke\text{'}, rid, sid)$ to the public ledger after an honest service provider submits his revocation statement to the trusted party, $\mathcal{S}$ also dumps this revocation statement to the simulated world. To complete this, $\mathcal{S}$ queries sid in $\mathfrak{S}_H$ and gets an item (sid, Nym). Also it queries rid in $\mathfrak{RL}$ and gets an item (rid, rid'). Then it stores $(\text{`}revoke\text{'}, Nym, rid')$ to the bulletin board.

Once the adversary stores a new requirement $(Nym, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ to the bulletin board, $\mathcal{S}$ dumps this requirement to the ideal world via the following process. First, $\mathcal{S}$ queries Nym in $\mathfrak{S}_A$, and stops the dump process if no record with Nym is found in $\mathfrak{S}_A$. Next, after finding an item (sid, Nym) in $\mathfrak{S}_A$, $\mathcal{S}$ attempts to generate the corresponding requirement $(\mathcal{C}', \mathcal{P}_R, \mathcal{L}')$ in the ideal world. More precisely, for each public key pk in $\mathcal{C}$, $\mathcal{S}$ queries pk in $\mathfrak{U}_A \cup \mathfrak{U}_H$, and puts the uid to $\mathcal{C}'$ if a user uid is located to possess the public key pk; otherwise, it assigns a temporary uid s.t. $(uid, *) \notin \mathfrak{U}_A \cup \mathfrak{U}_H$ to pk and stores this uid to $\mathcal{C}'$. Also, for each identifier rid in $\mathcal{L}$, $\mathcal{S}$ queries rid in $\mathfrak{RL}$, and puts rid' to $\mathcal{L}'$ if it finds a tuple (rid', rid); otherwise, it assigns a temporary rid' s.t. $(rid', *) \notin \mathfrak{RL}$ to rid and stores this rid' to $\mathcal{L}'$. Finally, $\mathcal{S}$ deduplicate the set $\mathcal{C}'$ and uploads the requirement $(\mathcal{C}', \mathcal{P}_R, \mathcal{L}')$ to $\mathcal{T}_P$ on behalf of sid.

Once $\mathcal{T}_P$ writes a tuple $(sid, \mathcal{C}, \mathcal{P}_R, \mathcal{L})$ to the public ledger after an honest service provider uploads his policy to the trusted party, $\mathcal{S}$ also dumps this policy to the simulated world. To complete this, $\mathcal{S}$ queries sid in $\mathfrak{S}_H$ and gets an item (sid, Nym). Then it generates the corresponding requirement $(\mathcal{C}', \mathcal{P}_R, \mathcal{L}')$ in the simulated world. To generate $\mathcal{C}'$, $\mathcal{S}$ queries each item uid of $\mathcal{C}$ in $\mathfrak{U}_H$, and puts pk in $\mathcal{C}'$ after it finds a tuple (uid, sk, pk); it also queries uid in $\mathfrak{U}_A$, and puts all public keys from $\mathcal{K}$ into $\mathcal{C}'$ after it finds a tuple $(uid, \mathcal{K})$. To generate $\mathcal{L}'$,

$\mathcal{S}$ queries each item rid of $\mathcal{L}$ in $\mathfrak{RL}$, and puts rid' in $\mathcal{L}'$ after it finds a tuple (rid, rid'). Finally it stores $(Nym, \mathcal{C}', \mathcal{P}_R, \mathcal{L}')$ to the bulletin board.

Besides monitoring the public ledger in each world, $\mathcal{S}$ should also handle authentication events between a corrupted party and an honest party.

Upon receiving an authentication request $(\text{'}request\text{'}, sid, tid, result)$ from $\mathcal{T}_P$, where sid is a service provider controlled by $\mathcal{A}$, tid is the identifier for this authentication event, and $result$ is the recommended result from $\mathcal{T}_P$, which is either '$valid$' or '$invalid$', $\mathcal{S}$ stops if the result is '$invalid$'. Otherwise, it sends a tuple $(\text{'}request\text{'}, sid)$ to $\mathcal{A}$. If $\mathcal{A}$ aborts, $\mathcal{S}$ also stops this authentication event. Otherwise, $\mathcal{S}$ will get a challenge $m \| sid'$ back. If $sid \neq sid'$, $\mathcal{S}$ sends '$abort$' to $\mathcal{A}$ and stops this authentication event. Otherwise, $\mathcal{S}$ sends a tuple $(\mathcal{T}, \Pi_A)$ to $\mathcal{A}$, where $\mathcal{T}$ is a fake ticket generated by randomly sampling each part, and Π_A is a fake proof generated via the simulation algorithm of the SPK systems, and adds $(tid, sid, \mathcal{T}, \Gamma)$ in $\mathfrak{AL}$, where $\Gamma = (sid, \mathcal{C}, \mathcal{P}_R, \mathcal{L}; \text{'}request\text{'} \| sid; m \| sid; (\mathcal{T}, \Pi_A))$ is the transcript of this authentication event. If $\mathcal{A}$ aborts, $\mathcal{S}$ also stops this authentication event. Otherwise, $\mathcal{A}$ will sends a response, which is either '$accept$' or '$reject$', to $\mathcal{S}$, and $\mathcal{S}$ forwards this response to $\mathcal{T}_P$.

Upon receiving an authentication request $(\text{'}request\text{'}, sid)$ from $\mathcal{A}$, $\mathcal{S}$ first checks if sid is an honest service provider and stops the authentication event if sid is not an honest service provider. Then $\mathcal{S}$ samples m uniformly at random and sends $m \| sid$ to $\mathcal{A}$. If $\mathcal{A}$ aborts, $\mathcal{S}$ stops the authentication event. If the response is a tuple $(\mathcal{T}, \Pi_A)$, $\mathcal{S}$ will continue the authentication event in the ideal world. More precisely, it first verifies the proof Π_A. If Π_A is invalid, $\mathcal{S}$ sends '$reject$' to $\mathcal{A}$ and stops the authentication event. Then it attempts to extract a secret key from Π_A and query the extracted secret key in $\mathfrak{U}_A$. If it fails in conducting these two steps, $\mathcal{S}$ sends '$reject$' to $\mathcal{A}$ and stops the authentication event. Assuming a secret key sk is extracted and a user uid in $\mathfrak{U}_A$ is located to possess the secret key sk, $\mathcal{S}$ then logs into $\mathcal{T}_P$ on behalf of uid and sends the request $(\text{'}request\text{'}, sid)$ to $\mathcal{T}_P$. Then, $\mathcal{S}$ receives a result from $\mathcal{T}_P$ and choose to proceeds. Then, it receives the final result together with an identifier tid back, and forwards the result back to $\mathcal{A}$. Finally, $\mathcal{S}$ records the tuple $(tid, sid, \mathcal{T}, \Gamma)$ in $\mathfrak{AL}$, where $\Gamma = (sid, \mathcal{C}, \mathcal{P}_R, \mathcal{L}; \text{'}request\text{'} \| sid; m \| sid; (\mathcal{T}, \Pi_A))$ is the transcript of this authentication event.

At the end of the simulation, $\mathcal{S}$ will output what $\mathcal{A}$ outputs.

The Indistinguishability. Next, we argue that the joint distribution of the output of $\mathcal{S}$ and the outputs of all parties in the ideal world is indistinguishable from the joint distribution of the output of $\mathcal{A}$ and outputs of all parties in a real world system. It is sufficient to prove that the view of $\mathcal{A}$ in the real world and that in the world simualted by $\mathcal{S}$ is computationally indistinguishable. To argue this, we define the following games, between whom the indistinguishabilities can be reduced to properties of building blocks straightforwardly:

- **Game**$_{real}$**.** This is the real world, namely, the system constructed in this work is run between several honest parties and some parties controlled by the adversary $\mathcal{A}$.

- **Game$_1$.** This is identical to Game$_{real}$ except that every proof generated by honest users are replaced with a simulated one. Indistinguishability between Game$_{real}$ and Game$_1$ comes from the zero knowledge property of those employed SPK systems.
- **Game$_2$.** This is identical to Game$_1$ except that every ticket generated by honest users is generated by randomly sampling each part. Indistinguishability between Game$_1$ and Game$_2$ comes from the indistinguishability of the ticket generation algorithm.
- **Game$_3$.** This is identical to Game$_2$ except that a proof Π_R or Π_A generated by the adversary $\mathcal{A}$ will be regarded as invalid if the proof cannot be extracted with an extractor. Indistinguishability between Game$_2$ and Game$_3$ comes from the simulation extractbility of the employed SPK systems.
- **Game$_4$.** This is identical to Game$_3$ except that a proof Π_R or Π_A generated by the adversary $\mathcal{A}$ will be regarded as invalid if the extracted secret key is the secret key of an honest user. Indistinguishability between Game$_3$ and Game$_4$ comes from the onewayness of the key generation algorithm.
- **Game$_5$.** This is identical to Game$_4$ except that a proof Π_A generated by the adversary $\mathcal{A}$ will be regarded as invalid if the extracted secret key is not a secret key of a dishonest user. Indistinguishability between Game$_4$ and Game$_5$ comes from the collision resistance of the key generation algorithm and the simulation soundness of the employed SPK systems.
- **Game$_6$.** This is identical to Game$_5$ except that a proof Π_R generated by the adversary $\mathcal{A}$ will be regarded as invalid if $S(sk', \tau) = 1$ where sk' is the secret key of an honest user, and τ is a ticket generated by the extracted secret key. Indistinguishability between Game$_5$ and Game$_6$ comes from the connectivity of the ticket generation algortihm and the onewayness of the key generation algorithm.
- **Game$_7$.** This is identical to Game$_6$ except that an authentication response (τ, Π_A) generated by the adversary $\mathcal{A}$ will be rejected if the extracted secret key does not fulfils the requirement (according to the context). Indistinguishability between Game$_6$ and Game$_7$ comes from the verifying consistency of the ticket generation algorithm and the simulation soundness of the employed SPK systems.
- **Game$_{ideal}$.** This is the world simulated by $\mathcal{S}$. It is not hard to check that Game$_{ideal}$ is identical to Game$_7$.

That completes the proof. □