BLACKLISTING BASED ANONYMOUS AUTHENTICATION SCHEME FOR SHARING ECONOMY

by
CAVİT ÖZBAY

Submitted to the Faculty of Natural Sciences and Engineering
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July, 2022

ABSTRACT

BLACKLISTING BASED ANONYMOUS AUTHENTICATION SCHEME FOR SHARING ECONOMY

CAVİT ÖZBAY

COMPUTER SCIENCE AND ENGINEERING M.Sci. THESIS, JULY 2022

Thesis Supervisor: Prof. Albert Levi

Keywords: anonymous credential, blacklisting, sharing economy, conditional anonymity

Authentication and blacklisting mechanisms have a key role for service providers to deliver the service to correct users through digital channels. On the other hand, a user may reveal private data through the service, and an authentication/blacklisting mechanism that identifies the user may be used to link such private data to the user herself. Thus, there always have been concerns about privacy of the users. While there are previous works in the literature that provide *unconditional anonymity* to users, these schemes prevent service providers from blacklisting misbehaving users. At this point, the *conditional anonymity* concept is proposed as a remedy. A recent approach in the literature for conditional anonymity is *blacklistable anonymous credentials*, which allows service providers to blacklist users for an authentication session without identifying the user. In this thesis, we improve user anonymity in conditionally anonymous schemes using two complementary mechanisms. First, we define a property, *whitelisting property*, for blacklistable anonymous credentials and give a construction of this scheme. The whitelisting property can be used to unlink an honestly behaved authentication session from the user. Secondly, we propose an extension of this scheme for a more specific use case, *sharing economy services*. This scheme allows a service provider to blacklist a user only if the user have not returned the shared asset in due time. We benchmark the performance of our schemes by comparing them with the rival schemes. For communication and computation metrics, our experiments show that our first scheme has comparable performance to previous works, and our second scheme is advantageous compared to a rival one.

ÖZET

PAYLAŞIM EKONOMİSİ İÇİN KARA LİSTE TABANLI BİR ANONİM KİMLİK DOĞRULAMA ŞEMASI

CAVİT ÖZBAY

BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, HAZİRAN 2022

Tez Danışmanı: Prof. Dr. Albert Levi

Anahtar Kelimeler: gizli kimlik, kara liste, paylaşım ekonomisi, koşullu gizlilik

Kimlik doğrulama ve kara liste mekanizmaları dijitaller kanallar üzerinde çalışan servis sağlayıcılarının doğru kullanıcılara hizmet ulaştırmasında anahtar bir rol sahibidir. Öte yandan, bir kullanıcı servis süresince mahrem verilerini açığa çıkarabilir ve kullanıcıyı gerçek hayattaki kimliğini ortaya çıkaran bir kimlik doğrulama/kara liste mekanizması bu verilerin kullanıcıyla ilişkilendirilmesine sebep olabilir. Bu nedenle, bu sistemlerde kullanıcı mahremiyeti üzerine kaygılar bulunmaktadır. Literatürdeki çalışmalar kullanıcılara *koşulsuz gizlilik* sağlayan sistemler önerse de bu şemalar kötü niyetli kullanıcıları kara listeye alma imkanı tanımaz. *Koşullu gizlilik* konsepti bu noktada çözüm olarak önerilmektedir. Koşullu gizlilik için literatürdeki önerilen konseptlerden biri *kara listelenebilir gizli kimliklerdir*. Bu şema servis sağlayıcıların kullanıcıyı bir kimlik doğrulama oturumu için kullanıcı hakkında bilgi edinmeden kara listeye almasına imkan sağlar. Bu tezde, koşullu gizli şemalardaki kullanıcı gizliliği iki mekanizma önererek geliştirilmektedir. İlk olarak kara listelenebilir gizli kimlikler için *beyaz liste* özelliği tanımlanmakta ve bu özelliğe sahip bir şema sağlanmaktadır. Beyaz liste özelliği bir kullanıcının dürüst davranış sergilediği bir kimlik doğrulama oturumuyla ilişkisini kaldırır. İkinci olarak, bu şemanın daha özel bir kullanım senaryosu olan *paylaşım ekonomisi* için bir uzantısı önerilmektedir. Bu şemada, bir servis sağlayıcısı bir kullanıcıyı yalnızca paylaşılan bir varlığı teslim zamanında getirmezse kara listeye alabilir. İki şemanın da performansı literatürdeki çalışmalarla karşılaştırılarak değerlendirilmiştir. Bu değerlendirmeler hesaplama ve iletişim maliyetleri açısından ilk şemanın literatürdeki çalışmalarla kıyaslanabilir performansa sahip olduğunu, ikinci şemanınsa daha avantajlı performansa sahip olduğunu göstermiştir.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Prof. Albert Levi for his guidance and encouragement throughout my study. I am grateful to jury members Prof. Erkay Savaş and Assoc. Prof. Alptekin Küpçü who accepted to review my thesis and be a part of my thesis jury by spending their valuable time. I also want to thank my friends and roommates to make my life more enjoyable during my masters study. Last but not least, I would like to thank my family, especially my mother and sister, for their endless support in my life.

TABLE OF CONTENTS

LIST OF TABLES .... **ix**

LIST OF FIGURES .... **x**

1. INTRODUCTION **1**
- 1.1. Contribution 3
- 1.2. Outline 4

2. BACKGROUND AND RELATED WORK **5**
- 2.1. Background 5
  - 2.1.1. Assumptions 6
  - 2.1.2. Structure-Preserving Signatures on Equivalence Classes 6
  - 2.1.3. Accumulators 8
  - 2.1.4. Time-Lock Puzzles 9
  - 2.1.5. Zero-Knowledge Proofs 10
- 2.2. Related Work 11

3. BLACKLISTABLE ANONYMOUS CREDENTIALS WITH WHITELISTING PROPERTY **14**
- 3.1. Definition of Our Blacklistable Anonymous Credential with Whitelisting Property Scheme 14
  - 3.1.1. Formal Definition of Algorithms 14
  - 3.1.2. Security Definitions 15
    - 3.1.2.1. Anonymity Experiment 17
    - 3.1.2.2. Experiment of Soundness 17
- 3.2. Construction and Security Analysis of Our BLACW Scheme 18
  - 3.2.1. Construction 19
  - 3.2.2. Security Proofs 23

4. PERFORMANCE ANALYSIS OF OUR BLACW SCHEME **29**

5. PRIVACY PRESERVING BORROWING SCHEME **32**

5.1. Definition of Our Privacy Preserving Borrowing Scheme 32
5.1.1. Formal Definition of Algorithms 34
5.1.2. Security Definitions 35
5.1.2.1. Anonymity Experiment 37
5.1.2.2. Backward-Unlinkability Experiment 37
5.1.2.3. Experiment of Soundness 38
5.2. Construction and Security Analysis of Our PPB Scheme 39
5.2.1. Construction 39
5.2.2. Time-Lock Puzzle Construction 44
5.2.2.1. Efficient Proof of Equality to Prime Order Discrete-Logarithm 45
5.2.3. Security Proofs 49
5.3. Practical Applications 54

6. PERFORMANCE EVALUATION OF OUR PPB SCHEME 57
6.1. Computational Cost of Our PPB Scheme 57
6.2. Comparative Evaluation with DAPA 58
6.2.1. Comparison of Computational Cost 60
6.2.2. Precomputation for PPB 61
6.2.3. Comparison of Communication Costs 61
6.3. Discussion on Further Optimizations 62
6.4. Performance Evaluation for Real-World Suitability 63

7. CONCLUSION 67

BIBLIOGRAPHY 68

LIST OF TABLES

Table 4.1. Credential Issuance Computation Cost Comparison 31

Table 4.2. Communication Cost Comparison 31

Table 6.1. Communication Cost Comparison 62

Table 6.2. Specifications for experiment environments 63

Table 6.3. Issuance Time-Cost 65

Table 6.4. Borrowing Time-Cost for blacklist size, 3900. 65

Table 6.5. Repayment Time-Cost for blacklist size 3900 and $|BIDS_u| = 4$. 66

LIST OF FIGURES

Figure 1.1. A blacklistable anonymous credential system 2

Figure 3.1. Setup, IssuerKeyGen, and UserKeyGen algorithms 19
Figure 3.2. IssueCred and ReceiveCred algorithms of BLACW 20
Figure 3.3. VerifyAuth and ProveAuth algorithms of BLACW 22
Figure 3.4. VerifyWhitelist and ProveWhitelist algorithms 24
Figure 3.5. Blacklist algorithm 24

Figure 4.1. Authentication cost of users for blacklist size 2500 30
Figure 4.2. Authentication cost of users for blacklist size 4000 30

Figure 5.1. System Architecture. Step 1 corresponds to credential issuance. Step 2 corresponds to a borrowing-lending operation. Step 3 and Step 4 are alternatives of each other. If Step 3, repayment-collecting, is performed successfully, than blacklisting, Step4, the user is not possible. Otherwise, the user can be efficiently blacklisted as in Step 4. 33
Figure 5.2. Setup, IssuerKeyGen, and UserKeyGen algorithms of PPB 40
Figure 5.3. IssueCred and ReceiveCred algorithms of PPB 41
Figure 5.4. Lend and Borrow algorithms 42
Figure 5.5. Collect and Repay algorithms 43
Figure 5.6. ExtractBid and Blacklist algorithms 43
Figure 5.7. Distributions for Simulator Responses 47
Figure 5.8. Car Sharing Architecture 55

Figure 6.1. Borrow Cost at User for Blacklisted BID's 58
Figure 6.2. Borrow Cost at User Side for Debt Number 58
Figure 6.3. Comparison of borrowing-lending cost on the user side between PPB and DAPA 59
Figure 6.4. Comparison of borrowing-lending cost on the verifiable server and issuer sides between PPB and DAPA 59
Figure 6.5. Comparison of cumulative cost for debt number at user side between PPB and DAPA 60

Figure 6.6. Architecture of the experiment environment for mobile benchmarks 64

1. INTRODUCTION

Identity management systems are fundamental parts of almost all services which are provided through digital channels. The main task of identity management systems is to provide an authentication mechanism that allows service providers to deliver the service to the correct parties. Blacklisting mechanisms, on the other hand, play a complementary role to the authentication mechanism. Blacklisting mechanisms are used to prevent a certain party from authenticating herself anymore. While these mechanisms are mainly built upon the service providers' concerns, users of these applications may have concerns on their privacy. If the underlying authentication mechanism reveal the identity of a user to service providers, then service providers may learn various private information about the user which is irrelevant to the provided service. On the other hand, an unconditionally anonymous authentication mechanism cannot provide a blacklisting mechanism for service providers.

Solutions based on identity management systems to these problems rely on the notion of conditional anonymity. In the literature, there are two main approaches to such problems. The first approach relies on a trusted authority (TA) (also named *opening authority* or *deanonymization authority* in different works) who can trace all users in the system, and the TA recovers the identity of the misbehaved user to report to service providers. As a reasonable extension to this system, it is also proposed to provide a mechanism to blacklist the users based on the recovered identity information. Considering the ability of TA, any vulnerability in TA's system may lead to major privacy leaks. Hence, it is obvious that employing a TA with such an excessive power is detrimental to users' privacy. At this point, one may think that the vulnerability of a central TA may be mitigated by dividing its responsibility among multiple TAs using secret-sharing methods. While this method seems like a partial solution, it comes with the burden of communication complexity, as multiple TAs must operate with consensus.

The second line of work is built on the scheme which is called *blacklistable anonymous credential*. Blacklistable anonymous credential schemes mainly propose a system in which service providers can blacklist a user based on the specific session in which the user got the service. The approach of this system eliminates the role of TA who

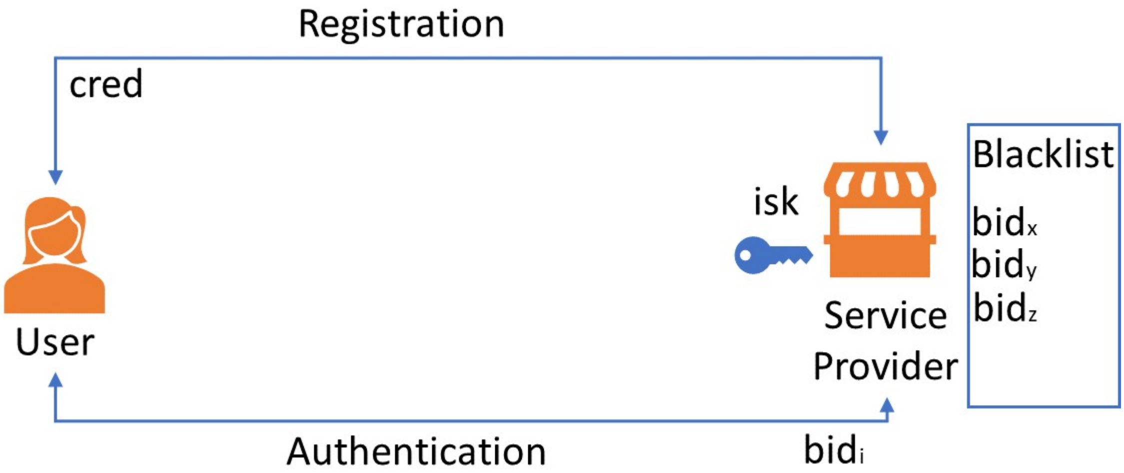

Figure 1.1 A blacklistable anonymous credential system

recovers users' identities, since it allows one to blacklist a user without learning the identity of the user. Figure 1.1 depicts the environment of a blacklistable anonymous credential scheme. Users are registered through a credential issuance phase. In each authentication session, the user provides additional information (blacklist id bid_i) which can be used to blacklist the user for the session. Unlike the opening authority based blacklisting mechanism, there is no way to link an authentication session to the real world identity of a user. Commonly, blacklistable anonymous credential schemes assume that service providers are honest-but-curious. This assumption ensures that the service provider runs the registration and authentication protocols as defined. However, blacklistable anonymous credential schemes apply the mechanism *subjective blacklisting*. It means that there is no determined policy about the behavior of the service provider about blacklisting a user. In more detail, there is no defined policy on choosing authentication sessions to be blacklisted, and the service provider can choose sessions to be blacklisted as he wishes. Hence, the service provider may blacklist a user just to link two authentication sessions to each other by still keeping his honest-but-curious behavior. This kind of blacklists are called malicious blacklists and they are the main potential threat against user privacy in a blacklistable anonymous credential scheme. In short, blacklistable anonymous credentials may give some room to link authentication sessions of the same user to each other in a malicious way, even though they do not allow service providers to link these authentication sessions to the real-world identity of the user.

When we turn our attention to a more specific type of service, sharing-economy services, we see that the problems we have mentioned in this chapter occur again. Sharing economy concepts are utilized for diversified use cases, and lots of these use cases occur through digital channels. While these applications aim to optimize the use of various resources as common resources, they come with concerns on privacy of users. To be specific, sharing economy applications may leak various kind of private information of users to service providers, which is not actually related to the provided service. For instance, tracked location of a rented vehicle may leak the residence of the user or the location of a private meeting. On the other hand, service providers have the following legitimate reason to reject the existence of unconditionally anonymous users. Service providers, lend their assets to users, so unconditionally anonymous users are not suitable for their business model. For instance, a user who rented a car may not return back the vehicle in time. Beyond these two opposing examples, we may consider further scenarios to express the trade-off between user privacy and accountability. Suppose a user did not return the rented car because he had an accident, the service provider's reason is still valid. However, the user did not misbehave intentionally. Hence, the user may demand that at least this unintentional misbehaviour should not reveal his entire car rental history. In other words, the user may demand *backward-unlinkability*.

1.1 Contribution

In this thesis, we focus on blacklisting mechanisms with improved anonymity features for users. The contribution of this thesis can be presented in a nutshell as follows.

1) We propose an extension to blacklistable anonymous credentials, which we call *whitelisting property*. Traditionally, blacklistable anonymous credential schemes have an *authentication* operation which a user proves that he is not blacklisted, and he also provides a *blacklist id* to the service provider which can be used to blacklist the user for the authenticated session. These schemes assume that the service provider checks the user behavior for each session and that the service provider can blacklist the corresponding user if there is misuse. In the case that the user behaved honestly, the service provider does not take an action. However, it is possible to blacklist corresponding user of the session at any time in the future. This flexibility provides a room for malicious blacklists as service providers can blacklist honest users just to link them to the future sessions. We define a blacklistable anonymous credential scheme with whitelisting property such that when the service provider detects a misuse in a session, the corresponding user can be blacklisted. However, if there is no misbehavior, the user and the service provider can perform the whitelisting operation, and the user can no longer be linked to the related session.

2) We provide a scheme which targets more specific scenarios in which users demand the usage of a resource owned by service providers and worry about their privacy against service providers and vulnerable TA's. This scheme is built upon the blacklistable anonymous credential scheme with whitelisting property that we define. Using the whitelisting property, a user can unlink the relationship between his credential and a session. However, the time of this operation is not fixed. In the sharing economy scenarios, users already interact with the service provider to end a session when they return the shared asset. Hence, we can design a dedicated protocol for it, for which the whitelisting operation can be performed regularly at the end of each session. On top of this idea, we provide the following extension. Instead of providing the *blacklist id* to the service provider in the authentication operation, users provide a tag which reveals the *blacklist id*, but only after an amount of time, which is the time for the user to return shared resource. If the user returns the shared asset in time, the user and the service provider can perform the whitelisting operation. By doing that, an honest user has a session such that the corresponding *blacklist id* is not revealed before it is whitelisted. On the other hand, if the resource has not returned in time, the service provider can blacklist the user by using the revealed *blacklist id*.

3) We also provide an efficient zero-knowledge proof-of-knowledge protocol to prove the knowledge of a secret value in a time-lock puzzle that can be opened to a discrete logarithm value from $\mathbb{Z}_p$ for some prime p. Even though we propose to use it in our scheme, we believe it may be useful independently from this work as will be explained further in this thesis.

4) We provide a detailed performance analysis for both of our schemes.

1.2 Outline

The rest of the thesis is organized as follows. In Chapter 2, we present the necessary background for the rest of the thesis and an overview of previous related works in the literature. Chapter 2 defines our blacklistable anonymous credentials with whitelisting property scheme, provides a construction of it, ends with the security analysis of the construction. Chapter 5 contains the formal definition of the proposed privacy preserving borrowing scheme, the construction of our privacy preserving borrowing scheme, and its security analysis. Lastly, Chapter 6 includes a comprehensive performance analysis on the privacy-preserving borrowing scheme, and Chapter 7 concludes the thesis.

2. BACKGROUND AND RELATED WORK

In this chapter, we first present the necessary technical background for our protocols in Section 2.1. To be specific, we provide the formal definitions of cryptographic tools that will be used throughout this thesis. Then, in Section 2.2, we provide current related work to the thesis topic and used cryptographic tools in the literature.

2.1 Background

This section presents preliminary information on the cryptographic concepts necessary for the rest of the thesis. Mainly, we provide the formal definitions of underlying cryptographic concepts (and used constructions for some of them).

We first start with the definition of *collision-resistant hash functions*.

Definition 1 (Collision-resistant Hash Functions)**.** *A set of functions* $H = \{h_i : D_i \to R_i\}_{i \in I}$ *is a family of collision resistant hash functions if:*

- *There is a p.p.t sampling algorithm* $\mathsf{Gen}(1^\lambda) \in I$
- *For* $i \in I$ *and* $x \in D_i$*,* $h_i(x)$ *can be computed in p.p.t.*
- *For all p.p.t. non-uniform adversaries* $\mathcal{A}$ *and* $\lambda \in \mathbb{N}$*, there exists a negligible function* μ *such that*

$$\Pr\Big[i \leftarrow \mathsf{Gen}(1^\lambda); x, x' \leftarrow \mathcal{A}(1^\lambda, i) : h_i(x) = h_i(x') \wedge x \neq x'\Big] \leq \mu(\lambda)$$

Next, we present the definitions related to *bilinear pairings* for completeness.

Definition 2 (Bilinear Pairing)**.** *For* $\langle g \rangle = \mathbb{G}_1$*,* $\langle \hat{g} \rangle = \mathbb{G}_2$ *and* $\mathbb{G}_T$ *which groups of prime order* p*,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is a bilinear pairing if it is efficiently computable and:*

- ***Bilinear****:* $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab} = e(g^b, \hat{g}^a)\ \forall a, b \in \mathbb{Z}_p$

- ***Non-degenerate***: $\langle e(g,\hat{g})\rangle = \mathbb{G}_T$, *so* $e(g,\hat{g}) \neq 1_{\mathbb{G}_T}$

Throughout the thesis, we use Type-3 pairings, so there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 3 (Bilinear-group generator)**.** *A bilinear-group generator* BGGen *is a p.p.t. algorithm which takes a security parameter* 1^λ *and outputs a bilinear pairing description* $\mathcal{BG} = (e, \mathbb{G}_1, \mathbb{G}_2, g, \hat{g}, p)$ *such that* $\langle g\rangle = \mathbb{G}_1$, $\langle \hat{g}\rangle = \mathbb{G}_2$, *and* $\mathbb{G}_T$ *are groups of prime order* p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is a bilinear pairing, and* $\lceil \log_2 p \rceil = \lambda$.

2.1.1 Assumptions

We use two traditional assumptions which are well-studied in the literature. The first one is based on a decisional problem, *decisional Diffie-Hellman problem.*

Definition 4 (DDH Assumption)**.** *For a group of prime order* $\mathbb{G}$ *with order* p *and* $g \leftarrow \mathbb{G}$, *there exists a negligible function* $\mu(\lambda)$ *such that*

$$\left| \frac{1}{2} - \Pr\left[\begin{array}{c} r,s,t \leftarrow \mathbb{Z}_p, \\ b \leftarrow \{0,1\}, \\ b' \leftarrow \mathcal{A}(g^r, g^s, g^{(1-b)t+brs}) \end{array} : b' = b \right] \right| \leq \mu(\lambda)$$

The second assumption is based on *strong RSA problem.* We do not use this assumption directly, but we use Theorem 1 which can be reduced to the strong RSA assumption. For the proof of Theorem 1, we refer the reader to (Camenisch & Shoup, 2003).

Definition 5 (Strong RSA Assumption)**.** *Given the RSA modulus* N *and a random element* $g \in \mathbb{Z}_N^*$, *it is hard to compute* $h \in \mathbb{Z}_N^*$ *and* $e > 1$ *such that* $g = h^e$ ***mod*** N.

Theorem 1. *Under the strong RSA assumption, given an RSA modulus* N *and two elements* $\mathfrak{g}, \mathfrak{h} \leftarrow \mathbb{Z}_N^*$ *it is hard to compute* $w \in \mathbb{Z}_N^*$ *and integers a, b, c s.t.:*

$$w^c = \mathfrak{g}^a \mathfrak{h}^b \wedge (c \nmid a \vee c \nmid b)$$

2.1.2 Structure-Preserving Signatures on Equivalence Classes

We take the definition and instantation from (Fuchsbauer, Hanser & Slamanig, 2019) for structure-preserving signatures on equivalence classes. The equivalence class relation $\mathcal{R}$ for this scheme is as follows. It worths to point out that this relationship is an equivalence relation only if $\mathbb{G}$ is a prime order group.

$$\mathcal{R} = \{(\overrightarrow{M}, \overrightarrow{N}) \in (\mathbb{G}^*)^\ell \times (\mathbb{G}^*)^\ell \mid \exists s \in \mathbb{Z}_p^* : \overrightarrow{N} = \overrightarrow{M}^s\}$$

Definition 6 (SPS-EQ)**.** *A structure-preserving signature on equivalence classes scheme is a tuple of algorithms* $(\mathsf{BGGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{ChgRep}, \mathsf{Verify}, \mathsf{VKey})$ *such that:*

- $\mathsf{BGGen}(1^\lambda)$*: Probabilisticly generates a prime-order bilinear group* $\mathcal{BG}$ *for security parameter* 1^λ *and outputs it.*
- $\mathsf{KeyGen}(\mathcal{BG}, 1^\ell)$*: Probabilistic algorithm which generates and outputs a key pair* (sk, pk) *for vector length* $\ell > 1$*.*
- $\mathsf{Sign}(sk, \overrightarrow{M})$*: For* $\overrightarrow{M} \in (\mathbb{G}_i^*)^\ell$ *from equivalence class* $[\overrightarrow{M}]$*, created and outputs signature* σ *using secret key sk.*
- $\mathsf{ChgRep}(pk, \overrightarrow{M}, \sigma, \mu)$*: Probabilistic algorithm which outputs a valid signature* σ' *for* $\overrightarrow{M}' = \mu \cdot \overrightarrow{M}$ *which is a representative of* $\overrightarrow{M} \in (\mathbb{G}_i^*)^\ell$ *from* $[\overrightarrow{M}]$*.*
- $\mathsf{Verify}(pk, \overrightarrow{M}, \sigma)$*: Deterministic algorithm outputs* 1 *if* σ *is a valid signature for* $\overrightarrow{M} \in (\mathbb{G}_i^*)^\ell$*. Outputs 0, otherwise.*
- $\mathsf{VKey}(sk, pk)$*: Outputs 1 if* $(\hat{g}^{x_i} = \hat{X}_i)_{i=1}^\ell$ *holds, for* $sk = (x_i)_{i=1}^\ell$ *and* $pk = (\hat{X}_i)_{i=1}^\ell$*. Outputs 0, otherwise.*

Definition 7 (EUF-CMA)**.** *An* SPS-EQ *over* $\mathbb{G}_i$ *is existentially unforgeable under chosen message attack (EUF-CMA) if for all* $\ell > 1$ *and all PPT algorithms* $\mathcal{A}$*, there is a negligible function* $\mu(\cdot)$ *s.t.:*

$$Pr\left[\begin{array}{c} \mathcal{BG} \leftarrow \mathsf{BGGen}(1^\lambda), \\ (sk, pk) \leftarrow \mathsf{KeyGen}(\mathcal{BG}, 1^\ell), \\ (\overrightarrow{M}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk) : b = 1 \\ \forall \overrightarrow{M} \in Q : [\overrightarrow{M}^*] \neq [\overrightarrow{M}], \\ b = \mathsf{Verify}(pk, \overrightarrow{M}^*, \sigma^*) \end{array}\right] \leq \mu(\lambda)$$

for the set of queries issued to the signing oracle by $\mathcal{A}$*,* Q*.*

Definition 8 (Perfect Adaptation)**.** *For* $\ell > 1$*, an* SPS-EQ *on* $(\mathbb{G}_i^*)^\ell$ *perfectly adapts signatures if for all* $(sk, pk, \overrightarrow{M}, \sigma, \mu)$ *satisfying* $\overrightarrow{M} \in (\mathbb{G}_i^*)^\ell$*,* $\mathsf{Verify}(pk, \overrightarrow{M}, \sigma) = 1$ *and* $\mu \in \mathbb{Z}_p^*$*,* $\mathsf{ChgRep}(pk, \overrightarrow{M}, \sigma, \mu)$ *has an identical distribution to* $\mathsf{Sign}(sk, \mu\overrightarrow{M})$*.*

Our scheme uses the following SPS-EQ instantiation from (Fuchsbauer et al., 2019).

- $\mathsf{BGGen}(1^\lambda)$: Creates and outputs a type-3 bilinear pairing $\mathcal{BG}$ according to the Definition 3.

- $\mathsf{KeyGen}(\mathcal{BG},\ell)$: Choose $(x_i)_{i\in[\ell]} \leftarrow (\mathbb{Z}_p^*)^\ell$. Return (sk,pk) which is $sk \leftarrow (x_i)_{i\in[\ell]}$ and $(pk \leftarrow (\hat{X}_i)_{i\in[\ell]} = (\hat{g}^{x_i})_{i\in[\ell]})$.

- $\mathsf{Sign}(sk,\overrightarrow{M})$: Choose $y \leftarrow \mathbb{Z}_p^*$. Return $\sigma \leftarrow (Z,Y,\hat{Y})$ s.t. $Z \leftarrow (\prod_{i\in[\ell]} M_i^{x_i})^y$, $Y \leftarrow g^{\frac{1}{y}}$, $\hat{Y} \leftarrow \hat{g}^{\frac{1}{y}}$.

- $\mathsf{Verify}(pk,\overrightarrow{M},\sigma)$: Return 1 iff $e(Y,\hat{g}) \stackrel{?}{=} e(g,\hat{Y}) \wedge \prod_{i\in[\ell]} e(M_i,\hat{X}_i) \stackrel{?}{=} e(Z,\hat{Y})$. Otherwise, return 0.

- $\mathsf{ChgRep}(pk,\overrightarrow{M},\sigma,\mu)$: Choose $\psi \leftarrow \mathbb{Z}_p^*$. Parse σ as $(Z,Y,\hat{Y})$ and return $\sigma' \leftarrow (Z^{\mu\psi},Y^{\frac{1}{\mu}},\hat{Y}^{\frac{1}{\mu}})$.

2.1.3 Accumulators

Accumulators are widely used cryptographic building blocks for performing membership and non-membership checks. Following accumulator may be seen as a more primitive version of (Thakur, 2019) or (Ghosh, Ohrimenko, Papadopoulos, Tamassia & Triandopoulos, 2016). We also benefit from the ideas used to hide the set of accumulated elements in our protocol from (Ghosh et al., 2016) which is also proposed in (Fuchsbauer et al., 2019) while constructing hiding set commitments. Following definitions just follow the aforementioned constructions and notation from (Badimtsi, Canetti & Yakoubov, 2020).

Definition 9 (Acc)**.** *An accumulator scheme* Acc *is a tuple of 4 algorithms* $(\mathsf{Gen},\mathsf{Add},\mathsf{NonMemWitCreate},\mathsf{VerifyNonMem})$ *where*

- $\mathsf{Gen}(1^\lambda,L_0)$*: For security parameter λ and initial set L_0, outputs public key pk, accumulator secret key sk, initial accumulator value acc, and auxiliary value m_0. It stores the set $L = L_0$ as the set of accumulated elements.*

- $\mathsf{Add}(pk,a_t,m_t,V)$*: Adds set of elements V to the accumulated set and outputs a_{t+1}, m_{t+1} and $upmsg_{t+1}$.*

- $\mathsf{NonMemWitCreate}(pk,a_t,X,\{upmsg_i\}_{i=1}^t)$*: It creates and outputs a non-membership witness w_t^X for a_t and V.*

- $\mathsf{VerifyNonMem}(a_t,X,w_t^V)$*: It outputs 1 if all elements of X are not a member of a_t. Outputs 0, otherwise.*

Definition 10 (Collision Freeness)**.** *An accumulator is collision-free if for all ppt adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that:*

$$\Pr\left[\begin{array}{c} L_0 \leftarrow \mathcal{A}() \\ (pk, acc_0) \leftarrow \mathsf{Gen}(1^\lambda, L_0), \\ X, w_t^X \leftarrow \mathcal{A}^{\mathsf{Add}()}(pk, acc_0), \\ b = \mathsf{VerNonMem}(a_t, X, w_t^X) \end{array} : \begin{array}{c} b = 1 \wedge \\ X \cap L_t \neq \emptyset \end{array}\right] \leq \mu(\lambda)$$

Using Definition 9 above, we use the following construction which relies on (Thakur, 2019) and (Ghosh et al., 2016). For simplicity, we assume that q is a constant parameter that represents the maximum number of elements that can be accumulated at the same time. We exclude a_t from the input parameters of Add and sk from output parameters of Gen, since they are not necessary for the construction.

- $\mathsf{Gen}(1^\lambda, \mathcal{BG}, L_0)$: Choose $\alpha \leftarrow \mathbb{Z}_p$. Initialize $sk \leftarrow \alpha$ and $pk \leftarrow \left((g^{\alpha^i})_{i\in[q]}, (\hat{g}^{\alpha^i})_{i\in[q]}\right)$. Return (pk, a_0, m_0) for $a_0 = \hat{g}$ and $m_0 = L_0$.

- $\mathsf{Add}(pk, m_t, V)$: Return $a_{t+1} = \hat{g}^{f(\alpha)}$ and $m_{t+1} = m_t \cup V$ for $f(X) = \prod_{v\in m_{t+1}}(X+v)$.

- $\mathsf{NonMemWitCreate}(pk, m_t, V)$: For $f(X) = \prod_{v\in m_t}(X+v)$ and $f_0(X) = \prod_{v\in V}(X+v)$, find $h(X)$ and $h_0(X)$ s.t. $f(X)h(X) - f_0(X)h_0(X) = 1$. Return $w_t^V = (\hat{w}_1, w_2) = (\hat{g}^{h_0(\alpha)}, g^{h(\alpha)})$.

- $\mathsf{VerifyNonMem}(a_t, V, w_t^V)$: Parse w as $(\hat{w}_1, w_2)$. For $f_0(X) = \prod_{v\in V}(X+v)$, return 1 iff $e(g^{f_0(\alpha)}, \hat{w}_1) = e(w_2, \hat{A})e(g, \hat{g})$.

2.1.4 Time-Lock Puzzles

For time-lock puzzles, we rely on the definitions and a variation of a construction from (Malavolta & Thyagarajan, 2019).

Definition 11 (TLP)**.** *A time-lock puzzle scheme* TLP *is a tuple of algorithms* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve})$ *which has following structure.*

- $\mathsf{PSetup}(\lambda, T)$*: A probabilistic algorithm to output public parameters pp for security parameter λ and time hardness parameter T.*

- $\mathsf{PGen}(pp, s)$*: A probabilistic algorithm which outputs puzzle Z for public parameters pp and solution $s \in \mathcal{S}$.*

- $\mathsf{PSolve}(pp, Z)$: *A deterministic algorithm which computes the solution $s \in \mathcal{S}$ for puzzle Z and public parameters pp.*

Definition 12 (TLP Security). *A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve})$ *is reusable secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $T^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\left| \frac{1}{2} \text{-} \Pr\left[\begin{array}{c} pp \leftarrow \mathsf{TLP.PSetup}(1^\lambda, T(\lambda)), \\ \tau, s_0, s_1 \leftarrow \mathcal{A}_1(pp), \\ b \leftarrow \{0,1\}, \\ Z \leftarrow \mathsf{TLP.PGen}(pp, s_b), \\ b' \leftarrow \mathcal{A}_2(Z, \tau) \end{array} : b' = b \right] \right| \leq \mu(\lambda)$$

Definition 13 (TLP Correctness). *A TLP scheme* $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve})$ *is correct if for all $\lambda \in \mathbb{N}$, all polynomials T in λ, all $s \in \mathcal{S}$, $pp \leftarrow \mathsf{PSetup}(1^\lambda, \mathsf{T})$, and all $Z \leftarrow \mathsf{PGen}(pp, s)$ there exists a negligible function μ such that*

$$\Pr[\mathsf{PSolve}(pp, Z) \neq s] \leq \mu(\lambda)$$

2.1.5 Zero-Knowledge Proofs

Definitions below for zero-knowledge proof of knowledge follow the related definitions from (Boneh & Shoup, 2020).

Definition 14 (Existential Soundness). *Let $\Pi = (P, V)$ be a Sigma protocol for $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$. If there is a negligible function $\mu(\cdot)$ s.t. $\mathsf{Adv}_{\mathcal{A}}^{\Pi} \leq \mu(\cdot)$ for all ppt adversaries $\mathcal{A}$ runs the protocol Π as prover P, and verifier $V(y)$ outputs 1 for $y \notin L_{\mathcal{R}}$ chosen by $\mathcal{A}$, then Π existentially sound.*

Definition 15 (Zero-Knowledge). *Let $\Pi = (P, V)$ be a Sigma protocol for $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$ with challenge space $\mathcal{C}$. If there is an efficient probabilistic algorithm* Sim *and a negligible function $\mu(\cdot)$ for all $(x, y) \in \mathcal{R}$ s.t.*

$$\left| \frac{1}{2} - \Pr\left[\begin{array}{c} b \leftarrow \{0,1\}, c_0 \leftarrow \mathcal{C} \\ (t_0, z_0) \leftarrow \mathsf{Sim}(y, c_0) \\ (t_1, c_1, z_1) \leftarrow \langle P(x, y), V(Y) \rangle \\ b' \leftarrow \mathcal{A}(t_b, c_b, z_b) \end{array} : b = b' \right] \right| \leq \mu(\cdot)$$

then Π *is zero-knowledge.*

Through out the thesis, we need zero-knowledge proof protocols which can be instantiated as *sigma protocols*, and we implement all of them by converting them to non-interactive protocols by applying Fiat-Shamir Heuristic (Fiat & Shamir, 1987).

2.2 Related Work

As far as our knowledge, the common concept of conditional anonymity for anonymous authentication first appeared in (Kilian & Petrank, 1998), which is called *identity escrow*. According to this scheme, a user gives his identity to only a trusted third party in the system, which needs to be online only when some users identity must be revealed. Famous anonymous credential system Idemix ((Camenisch & Van Herreweghen, 2002)) also implements an anonymity revocation mechanism on top of the same idea using verifiable encryption scheme of (Camenisch & Shoup, 2003). Verifier-local group signatures are group signatures which signatures can be revoked only managing a list at verifier side. (Bringer & Patey, 2012; Nakanishi & Funabiki, 2005) constructs backward-unlinkable VLR group signatures. Their unlinkability model assumes epochs in the system and provides backward-unlinkability only for next epochs. (Verheul, 2016) also proposes an approach similar to verifier-local revocation for identity management systems including Idemix, which aims to provide blacklisting in 24 hours.

Limited lifetime concept which credentials enclose a validity period is a widely used concept in digital certificates such as X.509 certificates. (Camenisch, Kohlweiss & Soriente, 2010) gives a construction on top of this idea for anonymous credentials by developing methods to perform updates on credential attributes. However, our scheme differs from this approach as our scheme allows to relate credentials with pseudonyms corresponding to different events having different deadlines. Unlike our scheme, time-based credential schemes allow a unique deadline for the credential itself. (Blömer, Bobolz, Diemert & Eidens, 2019; Bobolz, Eidens, Krenn, Slamanig & Striecks, 2020) formalize *updatable anonymous credentials*, and they design incentive systems to provide privacy-preserving point collection schemes. However, the concept of point collection is not sufficient in our case. Lending schemes require a valid repayment for each borrowing. Unlike lending schemes, there are no such dependence between operations in point collection schemes.

When we turn our attention to privacy preserving protocols concerning on sharing economy applications, we may see following works. (Hu, Zhao, Zheng & Wang,

2020) employs a protocol to prevent cross-bank over loans. There is a similar intention behind this work and ours, as both try to provide some kind of accountability on borrowers without harming their privacy concerns at the same time. However, they use blockchain as a public ledger among banks, and each borrower has a unique registered pseudonym on blockchain. For this reason, operation times of a specific pseudonym can easily be identified. (Xie, Holmes & Dagher, 2020) proposes a platform to perform transfers for a peer-to-peer lending system, called ZeroLender. It assumes that borrowers do not have privacy in the real world, and they only aim to achieve anonymity in Bitcoin network. In this manner, our work takes a complementary role, and focuses on finding a way to provide both borrower privacy and lending system security. (Huang, Lu, Ni & Shen, 2020) propose a car sharing mechanism that relies on a trusted party to remove renter anonymity in case of need. They propose to decentralize identity escrow by first distributing it among multiple authorities, and second by changing those authorities by time. A line of work containing (Symeonidis, Aly, Mustafa, Mennink, Dhooghe & Preneel, 2017), and recently finalized by (Symeonidis, Rotaru, Mustafa, Mennink, Preneel & Papadimitratos, 2022) constructs a system to enable car-sharing in the case of multi-car owners. In this scenario, individual car owners intend to rent their cars, and the paper suggests a comprehensive system that covers key management of cars with several privacy concerns. Nevertheless, their system model does not provide privacy between car owners and rentee. Thus, our proposal, again, has an integrant role for this line of work. Lastly, (Liu, Xue, He, Wei & Guizani, 2020) designs a general framework for IoT device usage in sharing economy. Their scheme also includes access control mechanism to IoT devices. Futhermore, similar to our system model, multiple lenders (service providers) can serve using a single issuer (central server) in their system model. However, the users have no privacy against the central server. For time-based services, which we focus on, they propose a mechanism called "service termination" looks similar to our repayment concept, but they do not provide accountability on users.

(Tsang, Au, Kapadia & Smith, 2007) defines the blacklistable anonymous credentials such that a service provider is able to blacklist a user whenever it is wished without relying on any third party. Unlike the solutions relying on trusted third parties, (Tsang et al., 2007) does not deanonymize a user completely for blacklisting purposes. (Tsang, Au & Kapadia, 2008) is another blacklistable anonymous credential scheme that checks whether a user has blacklisted in her last k authentications for some number k. They also provide another variation of their scheme such that users can be scored for their negative behaviors and can be blacklisted if their score is above a threshold. (Nakanishi & Kanatani, 2018) is one of recent blacklistable anonymous credential schemes that allows negative and positive scores on user reputation together. (Yang, Au, Xu & Yu, 2019) designs a blacklistable anonymous credential system where there is no central credential issuer. All these schemes allow service providers to blacklist users as they wish which creates another privacy concern, as service providers may use blacklisting features maliciously to trace or deanonymize users.

(Malavolta & Thyagarajan, 2019) proposes homomorphic time-lock puzzles which can be used to evaluate functions on time-lock puzzles without solving them. Homomorphic time-lock puzzles are also utilized to solve more than one puzzles at once to optimize the puzzle-solving phase. (Abadi & Kiayias, 2021) constructs a time-lock puzzle scheme that can generate a single puzzle for multiple secret values with different opening so that single sequential computation can evaluate a puzzle to multiple secret values at different points of the computation. (Chvojka, Jager, Slamanig & Striecks, 2021) further improves this design. Their design allows to determine evaluation times first, and puzzles can be generated with arbitrary secret values to the determined times at any other time. While these works focus on optimizing puzzle solving phase, recent implementation areas of time-lock puzzles arose two main problems on the verifiability of time-lock puzzles: verifiability and non-malleability. For the verifiability property, an early work of Boneh and Naor (Boneh & Naor, 2000) proposed timed commitments which allows committers to prove that the timed commitment opens to a certain secret value. Recently, Manevich and Akavia presented the concept of attribute verifiable timed commitmens, which allows users to prove that the commitment opens to a secret which holds a predicate for any predicate (Manevich & Akavia, 2022). (Katz, Loss & Xu, 2020) defines CCA-Secure timed commitments which brings CCA-security notion to timed commitments to provide non-malleability features. Another recent work on non-malleability discusses different definitions of non-malleability on time-lock puzzles in detail and gives constructions for those definitions (Freitag, Komargodski, Pass & Sirkin, 2021).

3. BLACKLISTABLE ANONYMOUS CREDENTIALS WITH WHITELISTING PROPERTY

This chapter is dedicated to blacklistable anonymous credentials with whitelisting property (BLACW). Section 3.1 defines BLACW and its properties. Subsequently, Section 3.2 provides a construction of BLACW and the security analysis of this construction.

3.1 Definition of Our Blacklistable Anonymous Credential with Whitelisting Property Scheme

In this section, we present the formal definitions of our *blacklistable anonymous credential with whitelisting property* scheme and related security definitions. As in the previous blacklistable anonymous credential schemes, we also assume that the service providers are honest-but-curious. As a naming convention, in this chapter and in the following chapter, we call the service provider also as credential issuer since the service provider issues credentials to users to register them. Informally, we add a property to the concept of blacklistable anonymous credentials, which allows users and service providers to unlink a session from a user if the user behaved honestly, so that the session can no longer be linked to the user. To make our motivation clear, it is convenient to compare it with the previous constructions. (Tsang et al., 2007) defines a blacklistable anonymous credential scheme which a user can be blacklisted only for last K authentication sessions. While this approach seems similar to ours, our notion is more flexible, which allows us to decide the situation of each authentication session regardless with its order of occurrence.

3.1.1 Formal Definition of Algorithms

Formally, a blacklistable anonymous credential with whitelisting property scheme BLACW is a tuple of algorithms ($\mathsf{Setup}, \mathsf{IssueCred}, \mathsf{ReceiveCred}, \mathsf{VerifyAuth}, \mathsf{ProveAuth},$

$\mathsf{VerifyWhitelist}, \mathsf{ProveWhitelist}, \mathsf{Blacklist}, \mathsf{IssuerKeyGen}, \mathsf{UserKeyGen})$ such that:

- $\mathsf{Setup}(1^\lambda)$: Generates public parameters pp, blacklist public parameters, bpk, for the blacklist and an empty blacklist $L^{(\emptyset)}$. Finally, outputs $(pp, bpk, L^{(\emptyset)})$. For the remaining algorithms, we assume that pp is known by all algorithms, and we do not express it explicitly.

- $\mathsf{IssuerKeyGen}()$: Generates and outputs a key pair (sk, pk) for an issuer.

- $\mathsf{UserKeyGen}(pk)$: Generates and outputs a user key pair (usk, upk).

- $\mathsf{IssueCred}(sk, pk, upk) \leftrightarrow \mathsf{ReceiveCred}(pk, usk, upk)$: Generates a credential *cred* for the user on the double-spent identifier *dsid* (not known by the issuer) and outputs *cred* to the user. Initially, the credential does not include any blacklist identifier *bid*, so the set of *bid*'s the user has, $BIDS_u = \emptyset$.

- $\mathsf{VerifyAuth}(sk, pk, L) \leftrightarrow \mathsf{ProveAuth}(pk, usk, cred, L)$: User generates a fresh blacklist id *bid* and reveals *dsid* from *cred* to the issuer. The issuer checks if *dsid* belongs to a previously used credential and checks if *bid* value is a previously used value. The issuer also checks if *cred* includes any blacklisted blacklist id using the blacklist L. Subsequently, generates a credential $cred^*$ for the user on blacklist id set $BIDS_u = BIDS_u \cup \{bid\}$, and a fresh double-spent identifier $dsid^*$ (not known by the issuer). Afterwards, the issuer sends them to the user. If the issuer fails during the process, returns $(0, \perp, \perp)$ and returns $(1, dsid, bid)$, otherwise. If the user fails during the process, returns $\perp$ and returns $cred^*$, otherwise.

- $\mathsf{VerifyWhiteList}(sk, pk, bid) \leftrightarrow \mathsf{ProveWhitelist}(pk, usk, cred, bid)$: Performs whitelisting operation for $bid \in BIDS_u$. Generates a credential $cred^*$ for the user on the set of blacklist id's $BIDS_u^* = BIDS_u \setminus \{bid\}$ and the double-spent identifier *dsid* from the credential *cred*. The issuer then sends all these values to the user. The issuer returns $\perp$ if there is a failure during the process and returns *bid*, otherwise. The user returns $\perp$ if there is a failure during the process and returns $cred^*$, otherwise.

- $\mathsf{Blacklist}(bpk, bid, L)$: Adds *bid* to L and computes L^*. Outputs L^*.

3.1.2 Security Definitions

We define the security experiments for the properties *anonymity* and *soundness* of our scheme. These experiments are built on top of the oracles below. Both security experiments we have starts by running Setup algorithm. For notational simplicity, we assume that all oracles have access to the output of this Setup call.

- $\mathcal{O}^{\texttt{RegisterIssuerKey}(pk)}$: It stores the issuer key for the user oracles. The oracle must be called before calling one of the oracles $\mathcal{O}^{\texttt{UserKeyGen}}$, $\mathcal{O}^{\texttt{ProveAuth}}$, or $\mathcal{O}^{\texttt{ProveWhitelist}}$. It also sets the set of blacklist id's which there is no corresponding $\mathcal{O}^{\texttt{ProveWhitelist}}$ call yet, $\mathcal{B} = \emptyset$.

The following oracles are used to model an honest user in anonymity experiment.

- $\mathcal{O}^{\texttt{UserKeyGen}()}$: Generates a user handle u and a key pair $(usk_u, upk_u) \leftarrow \mathsf{UserKeyGen}(pk)$ and stores $(upk_u, usk_u, cred_u) \leftarrow (upk_u, usk_u, \perp)$. Finally, it outputs (u, upk_u).
- $\mathcal{O}^{\texttt{ReceiveCred}(u)}$: Runs $cred^* \leftarrow \mathsf{ReceiveCred}(pk, usk_u, upk_u)$. If $cred^* =\perp$, it outputs $\perp$. Otherwise, it updates $cred_u$ as $cred^*$. This oracle cannot be called more than once for a user handle u, and must be called before any calls to $\mathcal{O}^{ProveAuth}$ $\mathcal{O}^{ProveWhitelist}$ with the handle u.
- $\mathcal{O}^{\texttt{ProveAuth}(u,L)}$: Outputs $\perp$ if $cred_u =\perp$. Otherwise, it runs $(cred^*, bid) \leftarrow \mathsf{ProveAuth}(pk, usk_u, cred_u, L)$ with the issuer. If $cred^* =\perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$. Lastly, the oracle updates $\mathcal{B}$ as $\mathcal{B}' = \mathcal{B} \cup \{(u, bid)\}$ and outputs bid.
- $\mathcal{O}^{\texttt{ProveWhitelist}(u,bid)}$: Outputs $\perp$ if $cred_u =\perp$ or if $(u, bid) \notin \mathcal{B}$. Otherwise, it runs $cred^* \leftarrow \mathsf{ProveWhitelist}(pk, usk_u, cred_u, bid)$ with the issuer. If $cred^* =\perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$, $\mathcal{B}$ as $\mathcal{B}' \leftarrow \mathcal{B} \setminus \{(u, bid)\}$, and outputs bid (just as a dummy value which is not equal to $\perp$).

The following oracles are used to model an honest issuer in a soundness experiment.

- $\mathcal{O}^{\texttt{IssuerKeyGen}()}$: Generates and stores $(sk, pk) \leftarrow \mathsf{IssuerKeyGen}()$, and outputs only pk. Sets the set of users $\mathcal{U} \leftarrow \emptyset$. Oracle also sets the set of different blacklists $\mathcal{L} = \emptyset$. Adversary is allowed to create arbitrary number of blacklists using $\mathcal{O}^{\texttt{RegisterBlacklist}}$ and $\mathcal{L}$ is used to keep track of these blacklists. Lastly, it sets $bid_c =\perp$ which is the value of challenge bid in the soundness game.
- $\mathcal{O}^{\texttt{IssueCred}(upk)}$: If $upk \notin \mathcal{U}$, then tries to run $\mathsf{IssueCred}(sk, pk, upk)$. If it runs successfully, the oracle updates $\mathcal{U} \leftarrow \mathcal{U} \cup \{upk\}$.
- $\mathcal{O}^{\texttt{VerifyAuthChl}(j)}$: Runs $(b, dsid, bid) \leftarrow \mathsf{VerifyAuth}(sk, pk, , L_j)$. If $b = 0$, it outputs $(b, \perp, \perp)$. Otherwise, it stores $(dsid, bid)$, sets $bid_c := bid$ and outputs $(b, dsid, bid)$.

- $\mathcal{O}^{\texttt{VerifyAuth}(j)}$: Runs $(b, dsid, bid) \leftarrow \mathsf{VerifyAuth}(sk, pk, , L_j)$. If $b = 0$, it outputs $(b, \perp, \perp)$. Otherwise, it stores $(dsid, bid)$ and outputs $(b, dsid, bid)$.
- $\mathcal{O}^{\texttt{VerifyWhitelist}(bid)}$: If $bid = bid_c$, then outputs $\perp$. Otherwise, the oracle runs $\mathsf{VerifyWhitelist}(sk, bid)$ with the user and outputs bid.
- $\mathcal{O}^{\texttt{RegisterBlacklist}()}$: This oracle just adds another empty blacklist to $\mathcal{L}$. For $j = |\mathcal{L}|$, it sets $L_j = L^{(\emptyset)}$, adds j to $\mathcal{L}$, and outputs j.
- $\mathcal{O}^{\texttt{Blacklist}(j,bid)}$: If $L_j \notin \mathcal{L}$, it outputs $\perp$. Otherwise, this oracle runs $L^* \leftarrow \mathsf{Blacklist}(L_j, bid)$, sets $L_j = L^*$, and outputs L^*.

3.1.2.1 Anonymity Experiment

Definition 16 ensures anonymity of users who do not yet have any authentication or users who have run ProveWhitelist for all their authentications. The intuition behind the anonymity experiment is as follows. While steps 1, 2 and 3 perform the setup steps for two users, u_0 and u_1, adversary $\mathcal{A}$ can make arbitrary calls to oracles which are denoted by $\mathcal{O}^{\texttt{Query}}$ in step 4. In step 5, $\mathcal{A}$ makes a single authentication oracle call to u_b, which is the main challenge phase of the experiment.

Definition 16 (Anonymity)**.** *A blacklistable anonymous credential with whitelisting scheme is anonymous if for all p.p.t adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr\left[\begin{array}{c} (pp, bpk, L^{(\emptyset)}) \leftarrow \mathsf{Setup}(1^\lambda), b \leftarrow \{0,1\} \\ (pk, st) \leftarrow \mathcal{A}(pp), \mathcal{O}^{\texttt{RegisterIssuerKey}(pk)} \\ \left((u_i, upk_i) \leftarrow \mathcal{O}^{\texttt{UserKeyGen}()}\right)_{i \in \{0,1\}} \\ st \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{Query}()}}(st, u_0, upk_0, u_1, upk_1) \\ \mathcal{B}^* := \mathcal{B}, b' \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{ProveAuth}(u_b,\cdot)}}(st) \end{array} : \begin{array}{c} \perp \notin \{cred_{u_0}, cred_{u_1}\} \\ \forall i \in \{0,1\}, (\nexists bid, (u_i, bid) \in \mathcal{B}^*) \\ b = b' \end{array} \right] - \frac{1}{2} \right| \leq \mu(\lambda)$$

where $\mathcal{O}^{\texttt{Query}()}$ is $\{\mathcal{O}^{\texttt{ReceiveCred}(\cdot)}, \mathcal{O}^{\texttt{ProveAuth}(\cdot,\cdot)}, \mathcal{O}^{\texttt{UserKeyGen}()}, \mathcal{O}^{\texttt{ProveWhitelist}(\cdot,\cdot)}\}$, and the adversary $\mathcal{A}$ can make arbitrary number of oracle queries in step 4.

3.1.2.2 Experiment of Soundness

Definition 17 provides our definition for soundness. It has a similar notion as the concept of misauthentication resistance in blacklistable anonymous credentials (Tsang et al., 2007). Before explaining this experiment, we can determine the properties of soundness for our scheme intuitively. First, as a general property of anonymous credential schemes, credential forgeries must be infeasible. Second, since we provide a blacklisting property, authentication using a blacklisted credential must be infeasible. If $upk_u \notin \mathcal{U}$ and the lending query is successful ($b_1 = 1$), it means that $\mathcal{A}$ is able to forge a credential. Otherwise, the experiment blacklists bid related to this lending operation and challenges the adversary to perform another lending query for the same public key upk_u. If the adversary can perform this operation successfully ($b_2 = 1$), it means that the adversary can authenticate even with a blacklisted credential. If the adversary $\mathcal{A}$ can achieve one of these, then $\mathcal{A}$ wins and loses otherwise.

Definition 17 (Soundness)**.** *A blacklistable anonymous credential with whitelisting scheme is sound if for all p.p.t. algorithms $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c} (pp, bpk, L^{(\emptyset)}) \leftarrow \mathsf{Setup}(1^\lambda), (pk) \leftarrow \mathcal{O}^{\texttt{IssuerKeyGen()}} \\ \mathcal{O}^{\texttt{RegisterIssuerKeys}(pk)}, \mathcal{A}^{\mathcal{O}^{\texttt{Query()}}}(pp, pk) \\ (b_1, dsid, bid) \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{VerifyAuthChl}(j)}} \\ L_k^* \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{Blacklist}(\cdot, bid)}} \\ \mathcal{A}^{\mathcal{O}^{\texttt{Query()}}}(pp, pk) \\ (b_2, dsid, bid) \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{VerifyAuth}(\cdot, k)}} \end{array} : \begin{array}{c} b_1 = 1 \wedge L_k^* \neq \perp \wedge \\ (upk_u \notin \mathcal{U} \vee b_2 = 1) \end{array}\right] \leq \mu(\lambda)$$

where upk_u is the public key of the user u which is used by $\mathcal{A}$ in steps 3 and 6.

3.2 Construction and Security Analysis of Our BLACW Scheme

In this section, we first provide the construction of our blacklistable anonymous credential with whitelisting property scheme (BLACW) scheme using structure-preserving signatures on equivalence classes, time-lock puzzles, accumulators, zero-knowledge proofs of knowledge, and collision-resistant hash functions. Subsequently, we prove the anonymity and soundness of this construction according to Definition 16 and Definition 17, respectively.

$\mathsf{Setup}(1^\lambda)$	$\mathsf{IssuerKeyGen}()$
$\mathcal{BG} \leftarrow \mathsf{SPS\text{-}EQ.BGGen}(1^\lambda)$	$(sk_{\mathsf{SPS\text{-}EQ}}, pk_{\mathsf{SPS\text{-}EQ}}) \leftarrow \mathsf{SPS\text{-}EQ.KeyGen}(1^\lambda, \mathcal{BG}, 3)$
$h_4 \leftarrow \mathcal{H}(\text{"}h_4\text{"} \| \mathcal{BG})$	$(q_i)_{i\in[3]} \leftarrow (\mathbb{Z}_p)^3, (h_i)_{i\in[3]} = (g_i)_{i\in[3]}$
$(bpk, a_0, m_0) \leftarrow \mathsf{Acc.Gen}(1^\lambda, \mathcal{BG}, \emptyset)$	$sk = (sk_{\mathsf{SPS\text{-}EQ}}, (q_i)_{i\in[3]})$
$L^{(\emptyset)} = (a_0, m_0), w \leftarrow \mathcal{H}(\text{"}w\text{"} \| \mathcal{BG})$	$pk = (pk_{\mathsf{SPS\text{-}EQ}}, pk_\Sigma, (h_i)_{i\in[3]})$
$pp = (\mathcal{BG}, h_4, w)$	Initialize $BIDS = \emptyset, DSIDS = \emptyset$
return $(pp, bpk, L^{(\emptyset)})$	**return** (sk, pk)
	$\mathsf{UserKeyGen}(pk)$
	return $(usk, upk = w^{usk})$ for $usk \leftarrow \mathbb{Z}_p$

Figure 3.1 Setup, IssuerKeyGen, and UserKeyGen algorithms

3.2.1 Construction

Our BLACW construction benefits from the concept of updatable anonymous credentials. To realize the construction of our scheme, we need to enclose the set of *bid*'s, $BIDS_u$, as an attribute of user credentials. A credential in our scheme is mainly a signature of the issuer on two commitments. While the first one is a regular Pedersen commitment on an attribute vector, the second is a set commitment that commits to the set of *bid*'s a user has. Similarly to updatable anonymous credentials, we perform updates on old credential values using the commitments in old credentials during authentication and whitelisting operations. Nevertheless, the existing updatable anonymous credential schemes in the literature do not allow us to enclose both a set and a vector as a credential attribute at the same time. Thus, we can not use existing updatable anonymous credential definitions in black-box fashion, and we propose our method to represent a set as an attribute in our credential by relying on traditional accumulator schemes and set commitment ideas from (Fuchsbauer et al., 2019). In the rest of this part, we explain our construction in detail.

Setup algorithm in Fig. 3.1 creates initial parameters for pairing operations, commitment, zero-knowledge proofs of knowledge, and accumulator. Subsequently, it returns the public parameters. It is assumed that Setup is run honestly at the very beginning of the system. This is a reasonable assumption since there are several works that propose methods to perform the generation of parameters in these algorithms employing distributed computation in multi-party setting (Bowe, Gabizon & Miers, 2017; Chen, Cohen, Doerner, Kondi, Lee, Rosefield & Shelat, 2020).

$\mathsf{IssueCred}(sk, pk, upk) \leftrightarrow \mathsf{ReceiveCred}(pk, usk, upk)$

Issuer **User**

$$\xrightarrow{\mathsf{ZKP}_0}$$

$$dsid^*_{user}, z^*, t^*, u^*, y^* \leftarrow \mathbb{Z}_p$$
$$\overrightarrow{C} = (g, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*}), g^{y^*})$$
$$C^{(pre)} = \overrightarrow{C}^{u^*}$$

$$\xleftarrow{C^{(pre)}, \mathsf{ZKP}_1}$$
$$dsid^*_{issuer} \leftarrow \mathbb{Z}_p$$
$$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)} (C_0^{(pre)})^{q_2 dsid^*_{issuer}}, C_2^{(pre)})$$
$$\sigma \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}')$$

$$\xrightarrow{\sigma, dsid^*_{issuer}}$$
$$\overrightarrow{C}^{(tmp)} = (C_0^{(pre)}, C_1^{(pre)} h_2^{u^* dsid^*_{issuer}}, C_2^{(pre)})$$
if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma) = 0$
  then return $\perp$
$$(\overrightarrow{C}^*, \sigma^*) = \mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma, \frac{1}{u^*})$$
return $cred^* = (\overrightarrow{C}^*, dsid^*, z^*, t^*, y^*, \emptyset, \sigma^*)$

$\mathsf{ZKP}_0 = ZKP[(sk_{\mathsf{SPS\text{-}EQ}}, (q_i)_{i\in[3]}) : \mathsf{SPS\text{-}EQ.VerKey}(sk_{\mathsf{SPS\text{-}EQ}}, pk_{\mathsf{SPS\text{-}EQ}}) = 1 \wedge_{i\in[3]} h_i = g^{q_i}]$
$\mathsf{ZKP}_1 = ZKP[(usk, dsid^*_{user}, z^*, t^*, u^*, y^*) : upk = w^{usk} \wedge$
$C^{(pre)} = (C_0^{(pre)}, C_1^{(pre)}, C_2^{(pre)}) = (g^{u^*}, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*})^{u^*}, g^{y^* u^*})]$

Figure 3.2 IssueCred and ReceiveCred algorithms of BLACW

Fig. 3.1 also shows how an issuer creates her keys using the IssuerKeyGen algorithm. An issuer key includes secret and public parameters of SPS-EQ together with trapdoors for the commitment to be signed. Trapdoors will be used when it is necessary to update an attribute on an old credential in further operations. The issuer also initializes two sets as empty sets during key generation, the set of used *dsid* values $DSIDS$, and the set of used *bid* values, $BIDS$.

A new user in the system runs the UserKeyGen algorithm in Fig. 3.1 to initialize his public and secret parameters. After the issuer creates her keys, any user who created his keys may be registered by running IssueCred $\leftrightarrow$ ReceiveCred algorithms with the issuer, which is presented in Fig. 5.3. At the beginning of this procedure, the issuer proves that her public key is well-formed. Then, the user creates two commitments. While one of them contains the user part of $dsid$, $dsid^*_{user}$, and user's secret key, usk, the other one is the commitment to set of *bid*'s the user has. Since the set of *bid*' s for the user u, $BIDS_u$, is initially empty, it is actually a commitment to an empty set. The user then puts them in an instance of SPS-EQ's equivalence class and sends it to the issuer together with a proof of well-formedness of all these values. Issuer checks the proof to see if the values are indeed well-formed, and halts otherwise. The issuer chooses the issuer part of $dsid$, $dsid^*_{issuer}$, and adds it to the user part of $dsid$. As a result of this process, both the user and the issuer are sure that the final double-spent identifier value $dsid^*$ is a random choice, since $dsid^* = dsid^*_{user} + dsid^*_{issuer}$. Subsequently, issuer signs the credential for the user and sends both the signature and the issuer part of $dsid$ to the user. Since the issuer changed the first commitment that the user sent by adding the issuer part to $dsid$, the user needs to compute that member first. Finally, the user changes the representation of the SPS-EQ signature so that the first element of the signed message is g.

To perform authentication, a user with an issued credential must run the VerifyAuth $\leftrightarrow$ ProveAuth algorithms demonstrated in Fig. 3.3, with the issuer. The main flow of a authenticaion operation is similar to a credential update operation of an updatable anonymous credential. Particularly, old $dsid$ value from the old credential is updated with a new one, and the fresh blacklist identifier value (denoted by bid^* in the Fig. 3.3) is added to $BIDS_u$. While doing these operations, the user performs the similar operations to the credential issuance process for the updated credential. Together with these parameters, the user also sends $dsid$ value of old credential so that old credential becomes invalid. She also sends bid which can be used to blacklist the user for this session. The last element that the user sends to the issuer is ZKP_2, which contains proof of nonmembership to the input blacklist, and proof of well-formedness of other sent values. Proof of non-membership relies on accumulator non-membership witness. Proof of well-formedness of the sent values is performed as follows. User sends $\overrightarrow{C}$ and the signature on $\overrightarrow{C}$ of the old credential. While proving the well-formedness of the updated credential, she proves that the updated credential only has two updates, which we mentioned above on the old credential. The issuer checks this proof of well-formedness and the signature of the old credential. Issuer also checks that $dsid \notin DSIDS$ and $bid \notin BIDS$. If any of them do not hold, the issuer returns $(0, \perp, \perp)$. Otherwise, the issuer follows steps similar to the issuance phase while creating the updated credential and returns $(1, dsid, bid)$. The user also performs similar steps she followed while she gets her credential, and she returns the updated credential value.

$\mathsf{VerifyAuth}(sk, pk, L) \leftrightarrow \mathsf{ProveAuth}(pk, usk, cred, L)$

Issuer | **User**

Parse $L = (a, m)$

and $cred = (\overrightarrow{C}, dsid, z, t, y, BIDS_u, \sigma)$

$(dsid^*_{user}, z^*, t^*, y^*, u^*, bid^*) \leftarrow \mathbb{Z}_p$

$BIDS^*_u = BIDS_u + \{bid^*\}$

$\overrightarrow{C}' = (g, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*}), g^{f_0^*(\alpha) y^*})$

$\overrightarrow{C}^{(pre)} = (\overrightarrow{C}')^{u^*}$

$wit = \mathsf{Acc.CreateNonMem}(bpk, m, BIDS_u)$

$$\xleftarrow{dsid, \sigma, \overrightarrow{C}, \overrightarrow{C}^{(pre)}, bid^*, \mathsf{ZKP}_2}$$

Issuer:

$b = \mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, C, \sigma)$

if $b = 0 \vee dsid \in DSIDS \vee bid \in BIDS$

then return $(0, \perp, \perp)$

$DSIDS = DSIDS \cup \{dsid\}$

$BIDS = BIDS \cup \{bid\}$

$dsid^*_{issuer} \leftarrow \mathbb{Z}_p$

$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)} (C_0^{(pre)})^{q_2 dsid^*_{issuer}}, C_2^{(pre)})$

$\sigma' \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}')$

return $(1, dsid, btag)$

$$\xrightarrow{\sigma', dsid^*_{issuer}}$$

$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)} h_2^{u^* dsid^*_{issuer}}, C_2^{(pre)})$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma) = 0$

then return $\perp$

$(\overrightarrow{C}^*, \sigma^*) =$

$\mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}', \sigma, \frac{1}{u^*})$

$cred^* = (\overrightarrow{C}^*, dsid^*, z^*, t^*, y^*, BIDS^*_u, \sigma^*)$

return $cred^*$

$\mathsf{ZKP}_2 = ZKP[(usk, v, z, z^*, t, t^*, u^*, dsid, dsid^*_{user}, BIDS_u, wit):$
$\overrightarrow{C} = (g, h_1^{usk} h_2^{dsid} h_3^z h_4^t, g^{y f_0(\alpha)}) \wedge \overrightarrow{C}^{(pre)} = (g^{u^*}, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*})^{u^*}, g^{u^* y^* f_0^*(\alpha)}) \wedge$
$e(C_2, \hat{g}^{\alpha + bid^*})^{y^* \cdot u^*} = e(C_2^{(pre)}, \hat{g})^y \wedge e(C_2, \hat{w}_1) = e(w_2, a)^y e(g, \hat{g})^y]$

Figure 3.3 VerifyAuth and ProveAuth algorithms of BLACW

Whitelisting operations can be performed after each service if it turns out that the user does not get into any act to be blacklisted. This operation which is displayed explicitly in Fig. 3.4, is relatively easier task than the authentication operation, since they produce a credential with a higher credit for users at the end. To be clear, users do not have an incentive to use the old credential they have at the beginning of the whitelisting operation because their new credential is more advantageous than the old one by the nature of the whitelisting procedure. Therefore, there is no need to reveal the double-spent identifier *dsid* of the old credential, and they can be directly used for the freshly created credential with higher credibility. Hence, we only need to deal with changes according to the revealed *bid* value, which means that the only updated credential attribute is $BIDS_u$. This gives us the following optimization that the first commitment, which includes *usk* and *dsid* does not change. Hence, it can be used directly. However, the second commitment is updated, and user computes a new commitment, C_2^*, to the updated set of *bid*'s, $BIDS_u^*$, for that. Other procedures include the proof of well-formedness of the new commitment to $BIDS_u^*$. If the proof of well-formedness holds, σ' is a valid signature on $\overrightarrow{C}'$, the issuer signs the new credential and sends it to the user. User side to complete the process is similar to the credential issuance and authentication operations.

Lastly, the $\mathsf{Blacklist}$ algorithm in Fig. 3.5 performs blacklisting simply by adding the given *bid* to the accumulator. The Setup outputs anything which is necessary to manage a blacklist and anyone can manage arbitrary numbers of blacklists using the public parameters. However, by default, we assume that the input blacklist for authentication operation is managed by the issuer.

3.2.2 Security Proofs

The following theorems show that our scheme is *anonymous* and *sound* according to Definition 16 and Definition 17, respectively.

Theorem 2. *Given blacklistable anonymous credential scheme is anonymous in the random oracle model, if the underlying zero-knowledge proofs of knowledge are HVZK and sound,* DDH *assumption holds, and underlying SPS-EQ signature scheme* $\mathsf{SPS\text{-}EQ}$ *is a signature scheme with perfect adaptation property.*

Proof. For the rest of the proof, the event S_i is the case where the adversary $\mathcal{A}$ wins $\mathsf{Game_i}$. $\mathsf{Game_0}$ is identical to the original anonymity game.

$\mathsf{VerifyWhitelist}(sk, pk, bid) \leftrightarrow \mathsf{ProveWhitelist}(pk, usk, cred, bid)$

Issuer **User**

Parse $cred = (\overrightarrow{C}, dsid, z, t, y, BIDS_u, \sigma)$

$BIDS_u^* = BIDS_u \setminus \{bid\}$

$C_2^* \leftarrow g^{f_0^*(\alpha)y^*s}$ for $s, y^* \leftarrow \mathbb{Z}_p$

$(\overrightarrow{C'}, \sigma') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}, \sigma, s)$

$\xleftarrow{\overrightarrow{C'}, \sigma', bid, C_2^*, \mathsf{ZKP}_3}$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma) = 0$

then return $\perp$

Parse $\overrightarrow{C'} as (C_0', C_1', C_2')$

$\sigma'' \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}}, (C_0', C_1', C_2^*))$

$\xrightarrow{\sigma''}$

$\overrightarrow{C}^{(tmp)} = (g^s, C_1^s, C_2^*)$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma) = 0$

then return $\perp$

$(\overrightarrow{C}^*, \sigma^*) =$

$\mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma'', \frac{1}{s},)$

return $cred^* = (\overrightarrow{C}^*, dsid, z, t, y^*, BIDS_u^*, \sigma^*)$

$\mathsf{ZKP}_3 = ZKP[(y, y^*, s, BIDS_u^*) : e(C_2', \hat{g})^{y^*} = e(C_2^*, \hat{g}^{\alpha+bid})^{y \cdot s}]$

Figure 3.4 VerifyWhitelist and ProveWhitelist algorithms

$\mathsf{Blacklist}(bpk, bid, L)$

Parse L as (a_t, m_t)

$(a_{t+1}, m_{t+1}) \leftarrow \mathsf{Acc.Add}(bpk, m_t, \{bid\})$

return $L^* = (a_{t+1}, m_{t+1})$

Figure 3.5 Blacklist algorithm

Game_1: In Game_1, on the first call of the adversary to $\mathsf{ReceiveCred}$, we run the ZKP extractor and get $sk = (sk_{\mathsf{SPS\text{-}EQ}}, \cdot, q_1, ..., q_3)$. Let the event F be the case that Game_1 can not extract the zero-knowledge witness. In this case, Game_1 aborts. Since Game_0 and Game_1 behaves identically if F does not hold,

$$\Pr[S_1] - \Pr[S_0] \leq \Pr[F]$$

we can also imply that,

$$\Pr[F] \leq \mu_{snd}(\lambda)$$

Hence, Game_0 and Game_1 are computationally indistinguishable.

Game_2: In this game, we simulate all zero-knowledge proofs of knowledge in $\mathcal{O}^{\mathsf{ProveAuth}}$ and $\mathcal{O}^{\mathsf{ProveWhitelist}}$.

$$\Pr[S_2] - \Pr[S_1] \leq \mu_{zk}(\lambda)$$

Game_3: In this game, all calls to $\mathsf{SPS–EQ.ChgRep}$ are changed to $\mathsf{SPS–EQ.Sign}$. Due to perfect adaptation of $\mathsf{SPS–EQ}$,

$$\Pr[S_2] = \Pr[S_3]$$

Game_4: In Game_4, we guess the index of last call to $\mathcal{O}^{\mathsf{ProveAuth}}$ (or $\mathcal{O}^{\mathsf{ReceiveCred}}$) for u_0, i_0. If the guess is wrong, we choose a random bit b' and simulate it as the adversary $\mathcal{A}$'s output. Let the event E_4 be the case that Game_4's guess is wrong. Since the events E_4 and S_3 are independent events and the event $\neg E_4$ occurs with non-negligible probability,

$$|\Pr[S_4] - 1/2| = \Pr[\neg E_4] \cdot |\Pr[S_3] - 1/2|$$

Game_5: In this game, we choose $Z, R \leftarrow \mathbb{G}$, and $r, r', r_u, \leftarrow \mathbb{Z}_p$. Then, we program random oracle such that

$$h_4 = \mathcal{H}(\text{"}h_4\text{"} \| \mathcal{BG}) := R^{\frac{1}{r}} g^{-\frac{r'}{r}}$$

For the query i_0, we initialize $C^{(pre)}$ as

$$C^{(pre)} = \left(g^{r_u}, Z^{r_u}, g^{f_0^*(\alpha) y^* r_u}\right), y^* \leftarrow \mathbb{Z}_p, f_0^*(X) = \prod_{bid \in BIDS_u^*} (X + bid)$$

Again, for i_0'th query, we change the stored $cred$ value to

$$cred = \Big((g, R', g^{f'_0(\alpha)y'}), dsid_{user}, z, t, y', BIDS'_u, \sigma\Big)$$

where $dsid_{user} \leftarrow \mathbb{Z}_p$. $R' = Rh_2^{dsid^*_{issuer}}$ and $BIDS'_u = BIDS^*_u$ if i_0 is the last oracle call before the guessing phase. Otherwise, for the set of all whitelisted bid's after the i_0'th query, $\bigcup_j \{bid_j\}$, $BIDS'_u = BIDS^*_u \setminus (\bigcup_j \{bid_j\})$. For appropriately initialized $BIDS'_u$, let $f'_0(X) = \prod_{bid \in BIDS'_u}(X + bid)$. Finally, $t = r$ and $z = \frac{1}{q_3}(r' - (q_1 usk_{u_0} + q_2 dsid))$.

$\mathsf{Game}_4 \rightarrow \mathsf{Game}_5$: Indistinguishability of these two games can loosely be reducted to class-hiding property of SPS-EQ. However, we can provide a strict reduction to DDH still inspired by the security reduction of the class hiding property ((Fuchsbauer et al., 2019)), but also by considering our special case. We prove that by using any adversary $\mathcal{A}$ that can distinguish between Game_4 and Game_5 with non-negligible probability, we can create an adversary that can win DDH game with a non-negligible probability. Before starting this reduction, let us point out that using Z instead of a new commitment instance in the i_0'th query is indistinguishable, since the underlying commitment is perfectly hiding. Thus, we only need to prove that $\mathcal{A}$ cannot distinguish whether Z or R has been used in the very last query. To prove this, we take a DDH instance. For a DDH tuple $(U, V, W) = (g^r, g^s, g^{(1-b)t+brs})$, let us program the random oracle as $R := V$. Then, we initialize $C^{(pre)} = (U, W, Y^*)$ for the i_0'th query. Let $V' = Vh_2^{dsid^*_{issuer}}$, and $C = (g, V', Y)$ for the very last query, where $Y^*, Y \leftarrow \mathbb{G}^*$. Observe that using Y^* and Y instead of $g^{f^*_0(\alpha)y^*u^*}$ for some u^* and $g^{f'_0(\alpha)y'}$ is indistinguishable, since both $g^{f^*_0(\alpha)y^*u^*}$ and $g^{f'_0(\alpha)y'}$ are uniformly random in $\mathbb{G}^*$ for $y^*, y' \leftarrow \mathbb{Z}_p$. Hence, if $b = 1$ for DDH tuple, we are in Game_4. Otherwise, we are in Game_5.

$$|\Pr[S_5] - \Pr[S_4]| \leq \mu_{DDH}(\lambda)$$

Game_6: The repetition of Game_4 for u_1.

$$|\Pr[S_6] - 1/2| = \Pr[\neg E_6] \cdot |\Pr[S_5] - 1/2|$$

Game_7: Game_7 is the repetition of Game_5 for u_1.

$$|\Pr[S_7] - \Pr[S_6]| \leq \mu_{DDH}(\lambda)$$

In Game_7, both u_0 and u_1 answer the last $\mathcal{O}^{\mathsf{Borrow}}$ call with freshly created random credentials, and Game_7 is indistinguishable from Game_0, which is equal to original experiment, under our security assumptions. $\square$

Theorem 3. *If underlying hash functions are collision-resistant hash functions, underlying zero-knowledge proofs of knowledge is HVZK and sound, the discrete logarithm problem is hard in $\mathbb{G}_1$, underlying accumulator is a collision-resistant accumulator, and sunderlying SPS-EQ scheme is EUF-CMA, then the given blacklistable anonymous credential with whitelisting scheme is sound.*

Proof. Our construction includes a signature on two commitments which contain the attributes of a credential. Obviously, any adversary who is able to forge signatures would be able to create arbitrary credentials. However, we should also consider that any adversary who can provide different openings to signed commitments would also be able to win the game.

Case 1. In this case, we look at where $b_1 = 1 \wedge upk_u \notin \mathcal{U}$. We would like to point out that the user secret key $usk \in \mathbb{Z}_p$ is an attribute of credentials in our system, and there is a one-to-one mapping between user secret keys and user public keys. Hence, if any adversary wins the game in this case, it means that the adversary is able to create a valid credential on a user secret key usk' by following one of two possible ways. Obviously, the former performs a SPS-EQ signature forgery so that the adversary can create a signature on a message which is not signed by an honest issuer oracle $\mathcal{O}^{\texttt{IssueCred}}$. The latter provides a second opening to the Pedersen commitment in the credential. In this case, there are two cases that an adversary can follow. The adversary can either find a different opening directly or the adversary can perform the zero-knowledge proof of knowledge protocols for a user secret key usk', which is different from the one in opening that the user knows.

Case 2. Now we investigate the remaining case where $b_1 = 1 \wedge upk \in \mathcal{U}$. For this case, the only possible way that an adversary wins the game is the case that $b_2 = 1$. We can easily conclude that if $b_1 = 1$ and $b_2 = 1$, then the adversary is able to present a credential on the user secret key usk which corresponds to $upk \in \mathcal{U}$, and set of bid's $BIDS_u$ such that the blacklisted $bid \notin BIDS'_u$. Again, this is possible in three ways.

- Adversary performs a SPS-EQ signature forgery.
- Adversary performs a zero-knowledge proof of knowledge for an old credential that does not contain bid with a fresh $dsid'$ value. This case is similar to performing a zero-knowledge proof of knowledge for a usk'. $dsid$ is another member of the signed Pedersen commitment. Hence, if adversary can perform this zero-knowledge proof, the underlying commitment algorithm is

not binding, or underlying zero-knowledge proof of knowledge protocol is not sound.

- Adversary performs zero-knowledge proof of non-membership for a $BIDS'_u$ such that $bid \notin BIDS'_u$. This is again possible in two ways. The adversary must be able to find an opening to the commitment for a $BIDS'_u$ such that $bid \notin BIDS'_u$ or the adversary must perform the zero-knowledge proof of non-membership for $BIDS_u$ even if $bid \in BIDS_u$. For the first case we can easily have the following reduction. Assume that there is an adversary that can provide openings y_0 and y_1 for two different sets S_0 and S_1. Let $f_0(X) = \prod_{d \in S_0}(X+d)$ and $f_1(X) = \prod_{d \in S_1}(X+d)$. We know that they are openings for the same commitment value, so $g^{y_0 f_0(\alpha)} = g^{y_1 f_1(\alpha)}$. Subsequently, we know $y_0 f_0(\alpha)$ **mod** $p = y_1 f_1(\alpha)$ **mod** p. Since $S_0 \neq S_1$, we know $f_0(X)$ and $f_1(X)$ have different roots. Hence, $y_0 f_0(X) \neq y_1 f_1(X)$, which means $f'(X) = y_0 f_0(X) - y_1 f_1(X)$ is a non-zero polynomial, but $f'(\alpha) = 0$. By finding the roots of $f'(\alpha)$, one can learn the accumulator trapdoor α. Hence, we can construct an adversary against underlying accumulator's collision-resistence, if the set commitments are not binding. Now the remaining way is performing a successful zero-knowledge proof of non-membership even if blacklisted $bid \in BIDS_u$. One can straightforwardly design an adversary against the soundness of the underlying zero-knowledge protocol's soundness or accumulator's collision resistance by using any such adversary.

□

4. PERFORMANCE ANALYSIS OF OUR BLACW SCHEME

In this chapter, we analyze the performance of our BLACW scheme. We implemented our scheme in proof-of-concept level to measure its run-time. We also implemented two previously existing blacklistable anonymous credential schemes, (Aikou, Sadiah & Nakanishi, 2017) and (Tsang et al., 2007) to compare it with our BLACW scheme.

(Tsang et al., 2007) applies anonymous credentials and accumulators with a different approach than ours. They use an anonymous credential scheme which can enclose a vector of attributes with size K, and a user must prove that none of these attributes are blacklisted during an authentication procedure. To do that, the user instantiates K non-membership proofs to an accumulator. In each authentication, the blacklist id related to the oldest authentication session is unlinked with the credential. Thus, if a misbehaved user is not blacklisted before K authentication, he cannot be blacklisted anymore.

(Aikou et al., 2017) has a closer approach to our scheme. First, it provides constant-time algorithms on the service provider side, as our algorithms. Secondly, it also utilizes accumulators. While the accumulator construction they use allows for more efficient witness updates, it comes with the following drawback. The accumulator they use can be used to accumulate values in the range $[1, n]$ for some n which is chosen during the accumulator setup and cannot be changed. It means that this accumulator cannot be used after n authentications for a service provider. Our accumulator choice also has a limitation of n as a maximum size for the accumulator. However, it can contain values from $\mathbb{Z}_p$ for some prime number p. Thus, it can be used until n blacklisted value. Another disadvantage of (Aikou et al., 2017) is that it lacks a formal security analysis.

We developed the implementation of three schemes in C++ language using MIRACL library for pairing-based/RSA-based cryptography and NTL library for polynomial arithmetic. All algorithms were run on a Lenovo Thinkpad laptop with a 1.60GHz Intel Core i5-10210U CPU and 16 GB memory.

Figures 4.1 and 4.2 compare the user's computation cost for authentication of our BLACW scheme with (Aikou et al., 2017)'s scheme for blacklist sizes 2500 and 4000,

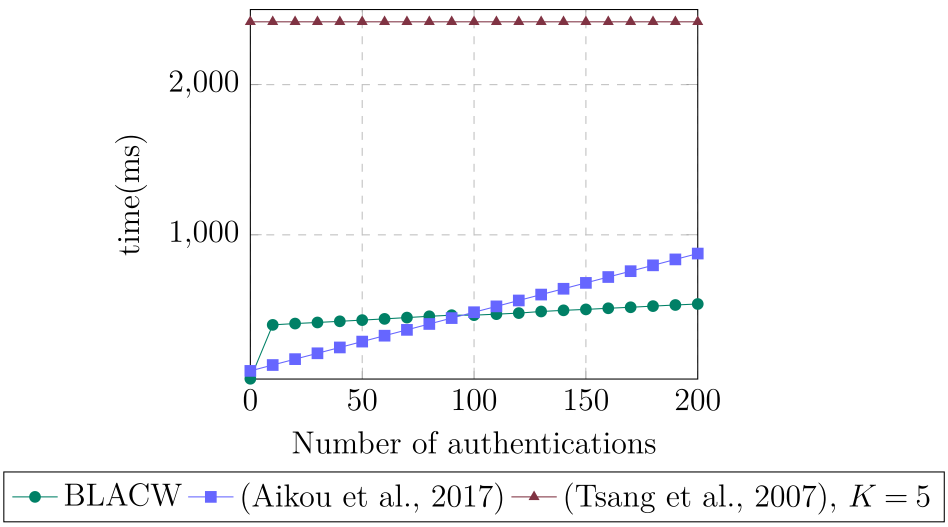

Figure 4.1 Authentication cost of users for blacklist size 2500

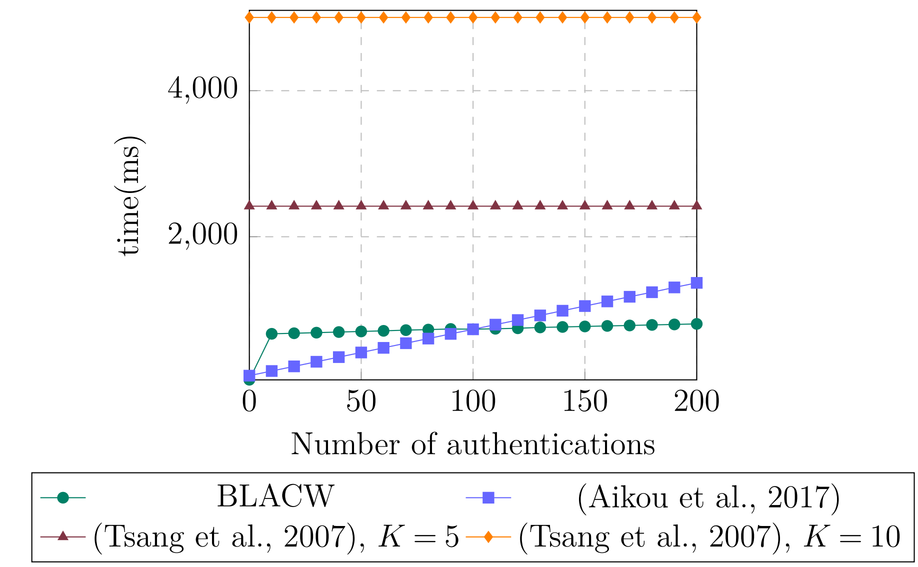

Figure 4.2 Authentication cost of users for blacklist size 4000

respectively. To make the constructions comparable, we measured our scheme without applying a whitelisting operation. Figures show that our scheme's cost gets more efficient after some number of authentications, while (Aikou et al., 2017)'s scheme is more efficient for the first few operations. (Tsang et al., 2007) assume that non-membership proofs are created by the service provider using accumulator trapdoor and users update the witnesses before each update. We kept the blacklist constant for (Tsang et al., 2007)'s favour, so the cost of non-membership proof updates are not included in the following costs. For $K = 5$, (Tsang et al., 2007) takes 2418.27 ms on the user side. For $K = 10$, (Tsang et al., 2007) takes 5003.04 ms on user side.

On the service provider side, both schemes have constant costs. While it takes 99.37 ms for a service provider to authenticate a user in (Aikou et al., 2017)'s scheme, it takes 46.94 ms for our BLACW scheme. (Tsang et al., 2007) takes 3726.15 ms for

Table 4.1 Credential Issuance Computation Cost Comparison

Protocol	**Time (ms)**	
	User	***Service Provider***
(Tsang et al., 2007)	51.17	45.47
(Aikou et al., 2017)	33.32	4.17
Our BLACW Construction	16.0	3.9

Table 4.2 Communication Cost Comparison

Operation	**Communication (kB)**		
	(Aikou et al., 2017)	**(Tsang et al., 2007)**	BLACW **(Section 3.2)**
Issuance	3.85	$2.63 + 0.13 \cdot K$	3.88
Authentication	10.45	$38.38 + 33.47 \cdot K$	9.06
Whitelisting	-	-	3.90

$K = 5$ and 1787.89 ms for $K = 10$ on the service provider side. Computational cost for credential issuance is displayed in Table 4.1 for all three schemes.

For the whitelisting operation, the service provider side has constant cost with average 18.20 ms, respectively. For the user side, whitelisting operations have computational costs that requires $O(d)$ exponentiations in pairing group for $d = |BIDS_u|$, which yields lightweight repayment operations such as 25.71 ms for $d = 20$.

Communication cost comparison of three schemes is presented in Table 4.2. The communication cost for the authentication operation in Table 4.2 covers only the constant cost and does not contain the communication cost related to the blacklists. For blacklist size n, (Aikou et al., 2017)'s blacklist is $n \cdot \log_2 n/8$ bytes, our BLACW scheme's blacklist is $n \cdot 64$ bytes, and (Tsang et al., 2007)'s blacklist is $n \cdot 127$ bytes.

5. PRIVACY PRESERVING BORROWING SCHEME

In this chapter, we first define the privacy preserving borrowing (PPB) scheme and its security properties. Then, we provide construction of our PPB scheme and the construction of time-lock puzzle scheme that is used in our PPB scheme. Subsequently, we provide the security analysis of the PPB construction. Finally, we review the applications of our PPB scheme.

5.1 Definition of Our Privacy Preserving Borrowing Scheme

In this section, we present the formal definitions of our *privacy-preserving borrowing* scheme and related security definitions. In this scheme, we provide a variant of our BLACW scheme by considering the sharing-economy environment. We define three entities for this system. The user/borrower is the entity who wishes to borrow assets in a sharing economy environment. Lender is the entity which correspond to the service providers of sharing-economy environment. Lenders share their asset with users/borrowers according to the agreement between them. Finally, issuer entity serves to register new users into the system by issuing credentials to them, checking that a user has a valid credential which not blacklisting during borrowing-lending procedure, and ending a asset sharing service by performing repayment-collecting procedure if there is no problem during the service. The main difference of this definition from blacklistable anonymous credentials is that the service provider entity, the lender, is separate from the credential issuer. Previous work in the literature defines these two authorities separately from each other and we follow this logic. This distinction mainly serves to supporting multiple service provider with the same credential. In our scheme, service providers, lenders, only have the role of informing the issuer in both borrowing-lending and repayment-collecting operations (for starting a service and for ending a service). Our PPB scheme is not vulnerable against the collision of lenders with the issuer as it will become obvious in our security definitions.

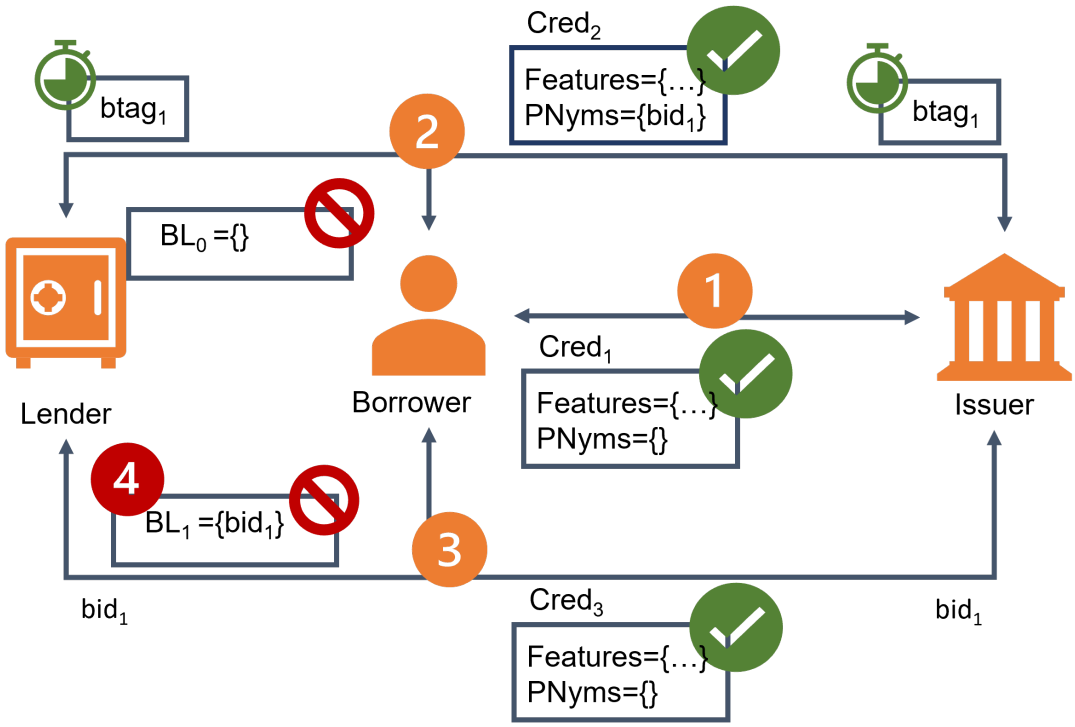

Figure 5.1 System Architecture. Step 1 corresponds to credential issuance. Step 2 corresponds to a borrowing-lending operation. Step 3 and Step 4 are alternatives of each other. If Step 3, repayment-collecting, is performed successfully, than blacklisting, Step4, the user is not possible. Otherwise, the user can be efficiently blacklisted as in Step 4.

Figure 5.1 displays the relation among the system entities. Similar to blacklistable anonymous credentials, there is a initial credential issuance procedure for each user to register them. When a user borrows and asset from a lender, the issuer and the user run borrowing-lending operation which is similar to the authentication operation in blacklistable anonymous credentials. However, instead of revealing the blacklist id directly, the user provides a tag, *btag* which can be used to learn blacklist id only after a time. When an asset sharing service is completed successfully between the user and the lender, the issuer and the user perform a repayment collection operation, similar to the whitelisting operation of our BLACW scheme. As a result of this operation, the users credential is updated in a way that it can not be linked to this sharing service anymore. Since the related blacklist id is not revealed directly by the user, if the user returns the shared asset honestly in time, then the user's credential is updated by running the repayment-collecting operation. It means that the user is whitelisted for the sharing service before the corresponding blacklist id is revealed to anyone else.

5.1.1 Formal Definition of Algorithms

A privacy-preserving borrowing scheme PPB is a tuple of algorithms (Setup, $\mathsf{IssueCred}$, $\mathsf{ReceiveCred}$, Lend, Borrow, $\mathsf{Collect}$, Repay, $\mathsf{ExtractBid}$, $\mathsf{Blacklist}$, $\mathsf{IssuerKeyGen}$, $\mathsf{UserKeyGen}$) such that:

- $\mathsf{Setup}(1^\lambda, T)$: Generates public parameters pp, blacklist public parameters, bpk, for the blacklist and an empty blacklist $L^{(\emptyset)}$. Finally, outputs $(pp, bpk, L^{(\emptyset)})$. For remaining algorithms, we assume that pp is known by all algorithms and we do not express it explicitly.
- $\mathsf{IssuerKeyGen}()$: Generates and outputs a key pair (sk, pk) for an issuer.
- $\mathsf{UserKeyGen}(pk)$: Generates and outputs a user key pair (usk, upk).
- $\mathsf{IssueCred}(sk, pk, upk) \leftrightarrow \mathsf{ReceiveCred}(pk, usk, upk)$: Generates a credential $cred$ for the user on the double-spent identifier $dsid$ (not known by the issuer) and outputs $cred$ to the user. Initially, the credential does not include any borrow identifier bid, so the set of bid's user has, $BIDS_u = \emptyset$.
- $\mathsf{Lend}(sk, pk, L) \leftrightarrow \mathsf{Borrow}(pk, usk, cred, L)$: Generates a borrow tag $btag$ on the borrow identifier bid that reveals bid only after T amount of time has passed, and outputs $btag$ to all parties. Reveals $dsid$ from $cred$ to the issuer, and the issuer checks if $dsid$ belongs to a previously used credential. Checks if

$cred$ includes any blacklisted borrow identifier by the lender using blacklist L. Subsequently, generates a credential $cred^*$ for the user on borrow identifier set $BIDS_u \cup \{bid\}$, and a fresh double-spent identifier $dsid^*$ (not known by the issuer). Afterwards, the issuer sends them to the user. If there is a failure during the process, it outputs $\perp$ to all parties.

- $\mathsf{Collect}(sk, pk, btag) \leftrightarrow \mathsf{Repay}(pk, usk, cred, btag, bid)$: Reveals user's bid related to $btag$ to all parties, and generates a credential $cred^*$ for the user on the set of borrow identifiers $BIDS_u \setminus \{bid\}$ and double-spent identifier $dsid$ from the credential $cred$. The issuer then sends all these values to the user. If there is a failure during the process, outputs $\perp$.
- $\mathsf{Blacklist}(bpk, bid, L)$: Adds bid to L and computes L^*. Outputs L^*.
- $\mathsf{ExtractBid}(btag)$: Computes and outputs bid for input $btag$, which takes at least T amount of time.

5.1.2 Security Definitions

We define the security experiments for the properties *anonymity*, *backward-unlinkability*, and *soundness* of our scheme. These experiments are built on top of the oracles below.

- $\mathcal{O}^{\texttt{RegisterIssuerKey}(pk)}$: It stores the issuer key for the user oracles. The oracle must be called before calling one of the oracles $\mathcal{O}^{\texttt{UserKeyGen}}$, $\mathcal{O}^{\texttt{Borrow}}$, or $\mathcal{O}^{\texttt{Repay}}$. It also initializes two sets, the set of successful borrowing calls $\mathcal{B} = \emptyset$, and the set of successful repayment calls $\mathcal{W} = \emptyset$. Lastly, it sets $btag_{buc} = \perp$ which will be used to keep the challenge value $btag$ for the backward-unlinkability game.
- $\mathcal{O}^{\texttt{UserKeyGen}()}$: Generates a user handle u and a key pair $(usk_u, upk_u) \leftarrow \mathsf{UserKeyGen}(pk)$ and stores $(upk_u, usk_u, cred_u) \leftarrow (upk_u, usk_u, \perp)$.
- $\mathcal{O}^{\texttt{ReceiveCred}(u)}$: Runs $cred^* \leftarrow \mathsf{ReceiveCred}(pk, usk_u, upk_u)$. If $cred^* = \perp$, it outputs $\perp$. Otherwise, it updates $cred_u$ as $cred^*$.
- $\mathcal{O}^{\texttt{Borrow}(u,L,k)}$: Outputs $\perp$ if $cred_u = \perp$. Otherwise, it runs $(cred^*, btag) \leftarrow \mathsf{Borrow}(pk, usk_u, cred_u, L)$ with the issuer. If $cred^* = \perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$. Lastly, the oracle updates $\mathcal{B}$ as $\mathcal{B}' = \mathcal{B} \cup \{(u, btag, k)\}$ outputs $btag$.
- $\mathcal{O}^{\texttt{Repay}(u,btag,k)}$: Outputs $\perp$ if $cred_u = \perp$ or there does not exist any $(u, btag, \cdot) \in \mathcal{B}$. Otherwise, it runs $cred^* \leftarrow \mathsf{Repay}(pk, usk_u, cred_u, btag, bid)$ with the issuer

for bid value which corresponds to $btag$. If $cred^* = \perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$, $\mathcal{W}$ as $\mathcal{W}' = \mathcal{W} \cup \{(u, btag, k)\}$, and outputs $btag$.

- $\mathcal{O}^{\texttt{BorrowChl}(u,L,k)}$: Outputs $\perp$ if $cred_u = \perp$. Otherwise, it runs $(cred^*, btag) \leftarrow \mathsf{Borrow}(pk, usk_u, cred_u, L)$ with the issuer. If $cred^* = \perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$. Lastly, the oracle updates $\mathcal{B}$ as $\mathcal{B}' = \mathcal{B} \cup \{(u, btag, k)\}$, updates $btag_{buc}$ as $btag_{buc} = btag$, and outputs $btag$.
- $\mathcal{O}^{\texttt{RepayChl}(u,k)}$: Outputs $\perp$ if $cred_u = \perp$ or $btag_{buc} = \perp$ or there does not exist any $(u, btag, \cdot) \in \mathcal{B}$. Otherwise, it runs $cred^* \leftarrow \mathsf{Repay}(pk, usk_u, cred_u, btag_{buc}, bid)$ with the issuer for bid value which corresponds to $btag_{buc}$. If $cred^* = \perp$, it outputs $\perp$. Otherwise, it updates $cred_u \leftarrow cred^*$, $\mathcal{W}$ as $\mathcal{W}' = \mathcal{W} \cup \{(u, btag, k)\}$, and outputs $btag$.
- $\mathcal{O}^{\texttt{IssuerKeyGen}()}$: Generates and stores $(sk, pk) \leftarrow \mathsf{IssuerKeyGen}()$, and outputs only pk. Sets the set of users $\mathcal{U} \leftarrow \emptyset$. Oracle also sets the set of different blacklists $\mathcal{L} = \emptyset$. Adversary is allowed to create arbitrary number of blacklists using $\mathcal{O}^{\texttt{RegisterBlacklist}}$ and $\mathcal{L}$ is used to keep track of these blacklists. Lastly, it sets $btag_c = \perp$ which is the challenge $btag$ value in soundness game.
- $\mathcal{O}^{\texttt{IssueCred}(upk)}$: If $upk \notin \mathcal{U}$, then runs $\mathsf{IssueCred}(sk, pk, upk)$, and updates $\mathcal{U} \leftarrow \mathcal{U} \cup \{upk\}$.
- $\mathcal{O}^{\texttt{Lend}(j)}$: Runs $(b, dsid, btag) \leftarrow \mathsf{Lend}(sk, pk, L_j)$. If $b = 0$, it outputs $(b, dsid, btag)$. Otherwise, it stores $(dsid, btag)$ and outputs $(b, dsid, btag)$.
- $\mathcal{O}^{\texttt{LendChl}(j)}$: Runs $(b, dsid, btag) \leftarrow \mathsf{Lend}(sk, pk, L_j)$. If $b = 0$, it outputs $(b, dsid, btag)$. Otherwise, it stores $(dsid, btag)$, sets $btag_c \leftarrow btag$ and outputs $(b, dsid, btag)$.
- $\mathcal{O}^{\texttt{Collect}(btag)}$: If $btag = btag_c$ outputs $\perp$. Otherwise, it runs $\mathsf{Collect}(sk, btag)$ with the user and outputs $btag$.
- $\mathcal{O}^{\texttt{RegisterBlacklist}()}$: This oracle just adds another empty blacklist to $\mathcal{L}$. For $j = |\mathcal{L}|$, it sets $L_j = L^{(\emptyset)}$, adds j to $\mathcal{L}$, and outputs j.
- $\mathcal{O}^{\texttt{Blacklist}(j,bid)}$: This oracle runs $L^* \leftarrow \mathsf{Blacklist}(bpk, L_j, bid)$, sets $L_j = L^*$, and outputs L^*.

5.1.2.1 Anonymity Experiment

Definition 18 ensures anonymity for honest users in the scheme. The intuition behind the anonymity experiment is as follows. While steps 1, 2 and 3 perform the setup steps for two users u_0 and u_1, the adversary $\mathcal{A}$ can make an arbitrary number of calls to oracles which are denoted by $\mathcal{O}^{\texttt{Query}}$ in steps 4 and 5. In step 6, $\mathcal{A}$ makes a single borrowing oracle call to u_b, which is the main challenge phase of the experiment. An important restriction on the oracle calls made by the adversary is that the sum of depths of all $\mathcal{A}_k$'s between a borrowing oracle query and the corresponding repayment oracle query cannot be greater than the define upper bound in the definition. This restriction models the behavior of an honest user, as an honest user must return the shared asset in time.

Definition 18 (Anonymity)**.** *A privacy preserving borrowing scheme is anonymous with gap $0 < \epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot)$ fulfilling that $T(\cdot) \geq \tilde{T}(\cdot)$, for all polynomials n in λ, for all polynomial size adversaries $\mathcal{A} = (\mathcal{A}_k)_{k\in[n]}$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$, it holds*

$$\left| \Pr\left[\begin{array}{c} (pp, bpk, L^{(\emptyset)}) \leftarrow \mathsf{Setup}(1^\lambda, T), b \leftarrow \{0,1\} \\ (pk, st_0') \leftarrow \mathcal{A}_0(pp), \mathcal{O}^{\texttt{RegisterIssuerKey}(pk)} \\ ((u_i, upk_i) \leftarrow \mathcal{O}^{\texttt{UserKeyGen}()})_{i\in\{0,1\}} \\ st_0 \leftarrow \mathcal{A}_0(st_0', u_0, upk_0, u_1, upk_1) \\ \textbf{\textit{for}}\ k\ \textbf{\textit{in}}\ [1, n-1], st_k \leftarrow \mathcal{A}_k^{\mathcal{O}^{\texttt{Query}}}(st_{k-1}) \\ b' \leftarrow \mathcal{A}_n^{\mathcal{O}^{\texttt{Borrow}(u_b,\cdot,n)}}(st_{n-1}) \end{array} : \begin{array}{c} \perp \notin \{cred_{u_0}, cred_{u_1}\} \\ \wedge b = b' \end{array} \right] - \frac{1}{2} \right| \leq \mu(\lambda)$$

where each $\mathcal{A}_k$ in step 5 can make at most a single query from $\mathcal{O}^{\texttt{Query}} := \{\mathcal{O}^{\texttt{ReceiveCred}(\cdot)}, \mathcal{O}^{\texttt{Borrow}(\cdot,\cdot,k)}, \mathcal{O}^{\texttt{UserKeyGen}()}, \mathcal{O}^{\texttt{Repay}(\cdot,\cdot,k)}\}$. We require that for all $(u_i, btag, k) \in \mathcal{B}$ with $i \in \{0,1\}$, if there exists a k' such that $(u_i, btag, k') \in \mathcal{W}$, then the sum of depths from $\mathcal{A}_k$ to $\mathcal{A}_{k'}$ is at most $T^\epsilon(\lambda)$. Otherwise, the sum of depths from $\mathcal{A}_k$ to $\mathcal{A}_n$ is at most $T^\epsilon(\lambda)$.

5.1.2.2 Backward-Unlinkability Experiment

Definition 19 ensures that completed honest events of a user can not be linked to the users future misbehaviour. The definition of backward-unlinkability is built on top of anonymity definition (Definition 18). Similar to the anonymity game, steps 1, 2, and 3 perform setup steps of the system and two honest users. Steps 6 and 8 perform the challenge event for the honest user u_b, and the adversary can make arbitrary number of calls to $\mathcal{O}^{\texttt{Query}'}$ in step 9.

Definition 19 (Backward-Unlinkability). *A privacy preserving borrowing scheme is backward-unlinkable with gap $0 < \epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot)$ fulfilling that $T(\cdot) \geq \tilde{T}(\cdot)$, for all polynomials n in λ, for all polynomial size adversaries $\mathcal{A} = (\mathcal{A}_k)_{k\in[n]}$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$, it holds*

$$\left| \Pr\left[\begin{array}{c} (pp, bpk, L^{(\emptyset)}) \leftarrow \mathsf{Setup}(1^\lambda, T), b \leftarrow \{0,1\} \\ (pk, st'_0) \leftarrow \mathcal{A}_0(pp), \mathcal{O}^{\texttt{RegisterIssuerKey}(pk)} \\ ((u_i, upk_i) \leftarrow \mathcal{O}^{\texttt{UserKeyGen}()})_{i\in\{0,1\}} \\ st_0 \leftarrow \mathcal{A}_0(st'_0, u_0, upk_0, u_1, upk_1) \\ \textbf{\textit{for}}\ k\ \textbf{\textit{in}}\ [1, m-1], st_k \leftarrow \mathcal{A}_k^{\mathcal{O}^{\texttt{Query}}}(st_{k-1}) \\ st_m \leftarrow \mathcal{A}_m^{\mathcal{O}^{\texttt{BorrowChl}(u_b,\cdot,m)}}(st_{m-1}) \\ \textbf{\textit{for}}\ k\ \textbf{\textit{in}}\ [m+1, n-2], st_k \leftarrow \mathcal{A}_k^{\mathcal{O}^{\texttt{Query}}}(st_{k-1}) \\ st_{n-1} \leftarrow \mathcal{A}_{n-1}^{\mathcal{O}^{\texttt{RepayChl}(u_b,n)}}(st_{n-2}) \\ b' \leftarrow \mathcal{A}_n^{\mathcal{O}^{\texttt{Query}'}}(st_{n-2}) \end{array} : \begin{array}{c} \perp \notin \{cred_{u_0}, cred_{u_1}\} \\ \wedge b = b' \end{array}\right] - \frac{1}{2} \right| \leq \mu(\lambda)$$

where each $\mathcal{A}_k$ in step 5 and step 7 can make at most a single query from $\mathcal{O}^{\texttt{Query}} := \{\mathcal{O}^{\texttt{ReceiveCred}(\cdot)}, \mathcal{O}^{\texttt{Borrow}(\cdot,\cdot,k)}, \mathcal{O}^{\texttt{UserKeyGen}()}, \mathcal{O}^{\texttt{Repay}(\cdot,\cdot,k)}\}$. In step 9, $\mathcal{A}_n$ can make arbitrary number of queries to $\mathcal{O}^{\texttt{Query}'} := \{\mathcal{O}^{\texttt{ReceiveCred}(\cdot)}, \mathcal{O}^{\texttt{Borrow}(\cdot,\cdot,n)}, \mathcal{O}^{\texttt{UserKeyGen}()}, \mathcal{O}^{\texttt{Repay}(\cdot,\cdot,n)}\}$. We require that for all $(u_i, btag, k) \in \mathcal{B}$ with $i \in \{0,1\}$ such that $k < n$, if there exists a $k' < n$ such that $(u_i, btag, k') \in \mathcal{W}$, then the sum of depths from $\mathcal{A}_k$ to $\mathcal{A}_{k'}$ is at most $T^\epsilon(\lambda)$. Otherwise, the sum of depths from $\mathcal{A}_k$ to $\mathcal{A}_{n-1}$ is at most $T^\epsilon(\lambda)$.

5.1.2.3 Experiment of Soundness

Definition 20 (Soundness). *A privacy-preserving borrowing scheme is sound if for all p.p.t. algorithms $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{c} (pp, bpk, L^{(\emptyset)}) \leftarrow \mathsf{Setup}(1^\lambda, T), (pk) \leftarrow \mathcal{O}^{\texttt{IssuerKeyGen}()} \\ \mathcal{O}^{\texttt{RegisterIssuerKeys}(pk)}, \mathcal{A}^{\mathcal{O}^{\texttt{Query}}}(pk) \\ (b_1, dsid, bid) \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{LendChl}(\cdot)}} \\ bid \leftarrow \mathsf{ExtractBid}(btag), L_k^* \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{Blacklist}(\cdot,bid)}} \\ \mathcal{A}^{\mathcal{O}^{\texttt{Query}}}(pk) \\ (b_2, dsid, bid) \leftarrow \mathcal{A}^{\mathcal{O}^{\texttt{Lend}(k)}} \end{array} : \begin{array}{c} b_1 = 1 \wedge L_k^* \neq \perp \wedge \\ (upk_u \notin \mathcal{U} \vee b_2 = 1) \end{array}\right] \leq \mu(\lambda)$$

where upk_u is the public key of the user u which is used by $\mathcal{A}$ in step 3 and step 6, and $\mathcal{A}$ can make arbitrary number of queries to $\mathcal{O}^{\texttt{Query}} =$

$\{\mathcal{O}^{\texttt{Lend}(\cdot)}, \mathcal{O}^{\texttt{Blacklist}(\cdot,\cdot)}, \mathcal{O}^{\texttt{Collect}(\cdot)}, \mathcal{O}^{\texttt{RegisterBlacklist}()}, \mathcal{O}^{\texttt{IssueCred}(\cdot)}\}$

Definition 20 provides our definition for soundness. This definition is similar to Definition 17, the soundness definition of BLACW except a small change. After getting $\mathcal{O}^{\texttt{LendChl}}$ query, the experiment must run ExtractBid as *btag* can not be blacklisted directly. Beyond this change, two definitions are similar.

5.2 Construction and Security Analysis of Our PPB Scheme

In this section, we first provide the construction of our privacy-preserving borrowing scheme (PPB) using structure-preserving signatures on equivalence classes, time-lock puzzles, accumulators, zero-knowledge proofs of knowledge, and collision-resistant hash functions. Subsequently, we prove the anonymity, backward-unlinkability, and soundness of this construction according to Definition 18, 19, and Definition 20, respectively.

5.2.1 Construction

Our PPB construction builds upon our BLACW construction from Chapter 3.2.

Similar to BLACW, Setup algorithm in Fig. 5.2 creates initial parameters for pairing operations, commitment, zero-knowledge proofs of knowledge, time-lock puzzle, and accumulator. Subsequently, it returns the public parameters. Since we assume that Setup is run honestly at the very beginning of the system, we would like to point out that these parameters can be computed by performing distributed computation in multi-party setting (Bowe et al., 2017; Chen et al., 2020). For notational simplicity, in the rest of the paper, we generate only a single time-lock puzzle in Setup algorithm. However, it can easily be generalized to multiple time-lock puzzles.

IssuerKeyGen, UserKeyGen, IssueCred, and ReceiveCred algorithms in Figure 5.2 and Figure 5.3 are identical to the related algorithms in our BLACW construction from Chapter 3.2.

Any user with a valid credential can perform a borrowing/lending operation if he can make a deal with a lender/service provider on the conditions of borrow. To perform a borrowing operation, a user needs to run Lend $\leftrightarrow$ Borrow algorithms demonstrated in Fig. 5.4, with the issuer. Of course, the issuer runs the protocol only if she is informed by the related lender of the borrowing operation. The main flow of a bor-

$\mathsf{Setup}(1^\lambda, T)$	$\mathsf{IssuerKeyGen}(pp)$
$\mathcal{BG} \leftarrow \mathsf{SPS\text{-}EQ.BGGen}(1^\lambda)$	$(sk_{\mathsf{SPS\text{-}EQ}}, pk_{\mathsf{SPS\text{-}EQ}}) \leftarrow \mathsf{SPS\text{-}EQ.KeyGen}(1^\lambda, \mathcal{BG}, 3)$
$pp_{\mathsf{TLP}} \leftarrow \mathsf{TLP.PSetup}(1^\lambda, T)$	$(q_i)_{i\in[3]} \leftarrow (\mathbb{Z}_p)^3, (h_i)_{i\in[3]} = (g_i)_{i\in[3]}$
$pp_{\mathsf{ZKP}} \leftarrow \mathsf{ZKP.Gen}(1^\lambda)$	$sk = (sk_{\mathsf{SPS\text{-}EQ}}, (q_i)_{i\in[3]})$
$h_4 \leftarrow \mathcal{H}(\text{"}h_4\text{"} \Vert \mathcal{BG})$	$pk = (pk_{\mathsf{SPS\text{-}EQ}}, pk_\Sigma, (h_i)_{i\in[3]})$
$(bpk, a_0, m_0) \leftarrow \mathsf{Acc.Gen}(1^\lambda, \mathcal{BG}, \emptyset)$	Initialize $DSIDS = \emptyset, BIDS = \emptyset$
$L^{(\emptyset)} = (a_0, m_0), w \leftarrow \mathcal{H}(\text{"}w\text{"} \Vert \mathcal{BG})$	**return** (sk, pk)
$pp = (\mathcal{BG}, h_4, w, pp_{\mathsf{TLP}}, pp_{\mathsf{ZKP}})$	
return $(pp, bpk, L^{(\emptyset)})$	$\mathsf{UserKeyGen}(pp, pk)$
	return $(usk, upk = w^{usk})$ for $usk \leftarrow \mathbb{Z}_p$

Figure 5.2 Setup, IssuerKeyGen, and UserKeyGen algorithms of PPB

rowing/lending operation is similar to the authentication operation of our BLACW construction. The difference from the BLACW is that, instead of sending the *bid* directly to the issuer, the user sends *btag* which includes a time-lock puzzle instance on bid^*. Additionally, since *bid* is not revealed during the operation, ZKP_2 also contains proof of knowledge of *bid*. Furthermore, the user also proves that the new element of $BIDS_u$, bid^*, is encapsulated in *btag*. This part of the proof contains cross group proofs between $\mathbb{Z}^*_{N^2}$ and $\mathbb{G}$ which are not straightforward to construct. However, thanks to the change we make in TLP.PSolve, we can perform this proof efficiently. The detailed explanation of this protocol and the function $recons()$ that is used in ZKP_2 are provided in Section 5.2.2. The issuer checks this proof of well-formedness and the signature of the old credential. If any of them do not hold, the issuer aborts. Otherwise, the issuer follows steps similar to the issuance phase while creating the updated credential and returns $(1, dsid, btag)$. The user also performs similar steps she followed while she gets her credential, and she returns the updated credential value.

Repayment/collection operations must be performed at the end of each service, and the issuer starts this only after the lender's approval. This operation which is displayed explicitly in Fig. 5.5, and it is similar to thw whitelisting operation of our BLACW construction. The difference is that since *bid* is not revealed to the issuer at the beginning, the user reveals it during the operation. If the proof of well-formedness holds, σ' is a valid signature on $\overrightarrow{C}'$, and *bid* corresponds to *btag* (it can be easily verified by checking that if $btag_2 = g^{bid}$ or not), the issuer signs the new credential and sends it to the user. User side to complete the process is similar to the credential issuance and borrowing/lending operations.

$\mathsf{IssueCred}(sk, pk, upk) \leftrightarrow \mathsf{ReceiveCred}(pk, usk, upk)$

Issuer **User**

$$\xrightarrow{\mathsf{ZKP}_0}$$

$dsid^*_{user}, z^*, t^*, u^*, y^* \leftarrow \mathbb{Z}_p$

$\overrightarrow{C} = (g, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*}), g^{y^*})$

$C^{(pre)} = \overrightarrow{C}^{u^*}$

$$\xleftarrow{C^{(pre)}, \mathsf{ZKP}_1}$$

$dsid^*_{issuer} \leftarrow \mathbb{Z}_p$

$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)} (C_0^{(pre)})^{q_2 dsid^*_{issuer}}$
$\quad , C_2^{(pre)})$

$\sigma \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}')$

$$\xrightarrow{\sigma, dsid^*_{issuer}}$$

$\overrightarrow{C}^{(tmp)} = (C_0^{(pre)}, C_1^{(pre)} h_2^{u^* dsid^*_{issuer}}, C_2^{(pre)})$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma)$

then return $\perp$

$(\overrightarrow{C}^*, \sigma^*) =$

$\quad \mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma, \frac{1}{u^*})$

return $cred^* = (\overrightarrow{C}^*, dsid^*, z^*, t^*, y^*, \emptyset, \sigma^*)$

$\mathsf{ZKP}_0 = ZKP[(sk_{\mathsf{SPS\text{-}EQ}}, (q_i)_{i\in[3]}) : \mathsf{SPS\text{-}EQ.VerKey}(sk_{\mathsf{SPS\text{-}EQ}}, pk_{\mathsf{SPS\text{-}EQ}}) = 1 \wedge_{i\in[3]} h_i = g^{q_i}]$

$\mathsf{ZKP}_1 = ZKP[(usk, dsid^*_{user}, z^*, t^*, u^*, y^*) : upk = w^{usk} \wedge$

$C^{(pre)} = (C_0^{(pre)}, C_1^{(pre)}, C_2^{(pre)}) = (g^{u^*}, (h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*})^{u^*}, g^{y^* u^*})]$

Figure 5.3 IssueCred and ReceiveCred algorithms of PPB

$\mathsf{Lend}(sk,pk,L) \leftrightarrow \mathsf{Borrow}(pk,usk,cred,L)$

User

Parse $L=(a,m)$

and $cred=(\overrightarrow{C},dsid,z,t,y,BIDS_u,\sigma)$

$(dsid^*_{user},z^*,t^*,y^*,u^*,bid^*) \leftarrow \mathbb{Z}_p$

$BIDS^*_u = BIDS_u + \{bid^*\}$

$\overrightarrow{C}' = (g,(h_1^{usk} h_2^{dsid^*_{user}} h_3^{z^*} h_4^{t^*}), g^{f_0^*(\alpha)y^*})$

$\overrightarrow{C}^{(pre)} = (\overrightarrow{C}')^{u^*}$

$wit = \mathsf{Acc.CreateNonMem}(bpk,m,BIDS_u)$

$btag \leftarrow (\mathsf{TLP.Create}(\lambda,bid^*), g^{bid^*})$

$$\xleftarrow{dsid,\sigma,\overrightarrow{C},\overrightarrow{C}^{(pre)},btag,\mathsf{ZKP}_2}$$

Issuer

$b = \mathsf{SPS\text{-}EQ.Verify}(C,\sigma,pk_{\mathsf{SPS\text{-}EQ}})$

if $b=0 \vee dsid \in DSIDS$

then return $(0,\perp,\perp)$

$DIDS = DSIDS \cup \{dsid\}$

$dsid^*_{issuer} \leftarrow \mathbb{Z}_p$

$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)}(C_0^{(pre)})^{q_2 dsid^*_{issuer}},$
$C_2^{(pre)})$

$\sigma' \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}},\overrightarrow{C}')$

return $(1,dsid,btag)$

$$\xrightarrow{\sigma',dsid^*_{issuer}}$$

User

$\overrightarrow{C}' = (C_0^{(pre)}, C_1^{(pre)} h_2^{u^* dsid^*_{issuer}}, C_2^{(pre)})$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}},\overrightarrow{C}^{(tmp)},\sigma)$

then return $\perp$

$(\overrightarrow{C}^*,\sigma^*) =$

$\mathsf{SPS\text{-}EQ.ChgRep}(\overrightarrow{C}',\sigma,\frac{1}{u^*},pk_{\mathsf{SPS\text{-}EQ}})$

$cred^* = (\overrightarrow{C}^*,dsid^*,z^*,t^*,y^*,BIDS^*_u,\sigma^*)$

return $cred^*$

$\mathsf{ZKP}_2 = ZKP[(usk,v,z,z^*,t,t^*,u^*,dsid,dsid^*_{user},BIDS_u,bid^*,wit,r):$

$\overrightarrow{C} = (g,h_1^{usk}h_2^{dsid}h_3^z h_4^t, g^{yf_0(\alpha)}) \wedge \overrightarrow{C}^{(pre)} = (g^{u^*},(h_1^{usk}h_2^{dsid^*_{user}}h_3^{z^*}h_4^{t^*})^{u^*}, g^{u^*y^*f_0^*(\alpha)}) \wedge$

$btag_1 = \mathsf{TLP.PGen}(pp,bid^*;r) \wedge btag_2 = g^{recons(bid^*)} \wedge$

$e(C_2,\hat{g}^{\alpha+bid^*})^{y^*\cdot u^*} = e(C_2^{(pre)},\hat{g})^y \wedge e(C_2,\hat{w}_1) = e(w_2,a)^y e(g,\hat{g})^y]$

Figure 5.4 Lend and Borrow algorithms

$\mathsf{Collect}(pp, sk, btag) \leftrightarrow \mathsf{Repay}(pp, usk, pk, cred, btag, bid)$

Issuer | **User**

Parse $cred = (\overrightarrow{C}, dsid, z, t, y, BIDS_u, \sigma)$

$BIDS_u^* = BIDS_u \setminus \{bid\}$

$C_2^* \leftarrow g^{f_0^*(\alpha) y^* s}$ for $s, y^* \leftarrow \mathbb{Z}_p$

$(\overrightarrow{C'}, \sigma') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(\overrightarrow{C}, \sigma, s, pk_{\mathsf{SPS\text{-}EQ}})$

$$\xleftarrow{\overrightarrow{C'}, \sigma', bid, C_2^*, \mathsf{ZKP}_3}$$

$b = \mathsf{SPS\text{-}EQ.Verify}(\overrightarrow{C'}, \sigma, pk_{\mathsf{SPS\text{-}EQ}})$

if $b = 0 \vee btag_2 \neq g^{bid}$

  then return $(0, \perp, \perp)$

Parse $\overrightarrow{C'} as (C_0', C_1', C_2')$

$\sigma'' \leftarrow \mathsf{SPS\text{-}EQ.Sign}(sk_{\mathsf{SPS\text{-}EQ}}, (C_0', C_1', C_2^*))$

$$\xrightarrow{\sigma''}$$

$\overrightarrow{C}^{(tmp)} = (g^s, C_1^s, C_2^*)$

if $\mathsf{SPS\text{-}EQ.Verify}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma) = 0$

  then return $\perp$

$(\overrightarrow{C}^*, \sigma^*) =$

  $\mathsf{SPS\text{-}EQ.ChgRep}(pk_{\mathsf{SPS\text{-}EQ}}, \overrightarrow{C}^{(tmp)}, \sigma'', \frac{1}{s},)$

return $cred^* = (\overrightarrow{C}^*, dsid, z, t, y^*, BIDS_u^*, \sigma^*)$

$$\mathsf{ZKP}_3 = ZKP[(y, y^*, s, BIDS_u^*) : e(C_2', \hat{g})^{y^*} = e(C_2^*, \hat{g}^{\alpha + bid})^{y \cdot s}]$$

Figure 5.5 Collect and Repay algorithms

$\mathsf{Blacklist}(bpk, bid, L)$

Parse L as (a_t, m_t)

$(a_{t+1}, m_{t+1}) \leftarrow \mathsf{Acc.Add}(bpk, m_t, \{bid\})$

return $L^* = (a_{t+1}, m_{t+1})$

$\mathsf{ExtractBid}(pp, btag)$

Parse $btag$ as $((u, v), btag_2)$

$bid \leftarrow \mathsf{TLP.PSolve}(u, v)$

return bid

Figure 5.6 ExtractBid and Blacklist algorithms

Lastly, the Blacklist algorithm in Fig. 5.6 is also identical to the Blacklist algorithm of BLACW construction in Chapter 3.2. The ExtractBid algorithm in Fig. 5.6 extracts *bid* from *btag* by force opening the time-lock puzzle instance. Our scheme allows an arbitrary number of blacklists in the system. However, by default, it is reasonable to assume that the lenders have their blacklists and run ExtractBid on each *btag* they get from the users of the service they provided.

5.2.2 Time-Lock Puzzle Construction

The following TLP scheme is used to instantiate our PPB scheme. It is the same construction as (Malavolta & Thyagarajan, 2019) except for the small change in the PSolve step. This change does not affect the security proof of the previous scheme, since there is no change in the algorithms PSetup and PGen. This change provides us to give efficient zero-knowledge proofs for time-lock puzzles we need in our construction which will be discussed later in this section.

- $\mathsf{PSetup}(1^\lambda, T)$: Generate RSA modulus $n = pq$. Choose $\dot{g} \leftarrow \mathbb{Z}_N^*$. Define $\tilde{g} = -\dot{g}^2 \textbf{ mod } N$ and $h = \tilde{g}^{2^T}$. Output $pp = (T, N, \tilde{g}, h)$.
- $\mathsf{PGen}(pp, s)$: Parse pp as (T, N, g, h). In the rest of the paper, let $\tilde{h} = (1+N) \textbf{ mod } N$ for notational simplicity. Choose $r \leftarrow \mathbb{Z}_{N^2}$ and compute $u = \tilde{g}^r \textbf{ mod } N$ and $v = h^{r \cdot N} \cdot \tilde{h}^s \textbf{ mod } N^2$. Output $Z = (u, v)$.
- $\mathsf{PSolve}(pp, Z)$: Parse pp and Z as $(T, N, \tilde{g}, h)$ and (u, v), respectively. Let $t = 2^{-1} \textbf{ mod } N$. Compute $w = u^{2^T} \textbf{ mod } N$ and output $s = \frac{\tilde{s}^{2t} \textbf{ mod } N^2 - 1}{N}$ for $\tilde{s} = v/(w)^N \textbf{ mod } N^2$.

Before showing the correctness of the scheme, it is noteworthy to recap the structure of group $\mathbb{Z}_{N^2}^*$ as we will also refer to it for our proof of knowledge protocol in the following of this section. $\mathbb{Z}_{N^2}^*$ can be decomposed to internal direct product $(\mathbf{G}_N \cdot \mathbf{G}_{N'} \cdot \mathbf{G}_2 \cdot \mathbf{T})$, for $N' = p'q'$, where $\mathbf{G}_i$ is a cyclic group of order i, and $\mathbf{T} = \langle (-1 \textbf{ mod } N^2) \rangle$ is a cyclic group of order 2. While $\mathbf{G}_N$, $\mathbf{G}_{N'}$, and $\mathbf{T}$ are unique, $\mathbf{G}_2$ has two possible groups choices:

- $\mathbf{G}_2 = \langle g_2 \rangle$ where $g_2 = 1 \textbf{ mod } p^2$ and $g_2 = -1 \textbf{ mod } q^2$
- $\mathbf{G}_2 = \langle g_2 \rangle$ where $g_2 = -1 \textbf{ mod } p^2$ and $g_2 = 1 \textbf{ mod } q^2$

The correctness of the scheme can be observed as follows.

$$\begin{aligned}
s' &= \frac{\tilde{s}^{2t} \textbf{ mod } N^2-1}{N} \\
&= \frac{\left(v/(w)^N \textbf{ mod } N^2\right)^{2t} \textbf{ mod } N^2-1}{N} \\
&= \frac{\left(h^{r\cdot N}\cdot(1+N)^s/(u^{2^T} \textbf{ mod } N)^N \textbf{ mod } N^2\right)^{2t} \textbf{ mod } N^2-1}{N} \\
&= \frac{\left(h^{r\cdot N}\cdot(1+N)^s/(h^r \textbf{ mod } N)^N \textbf{ mod } N^2\right)^{2t} \textbf{ mod } N^2-1}{N} \\
&= \frac{\left(h^{r\cdot N}\cdot(1+N)^s/h^{r\cdot N} \textbf{ mod } N^2\right)^{2t} \textbf{ mod } N^2-1}{N} \\
&= \frac{\left((1+N)^s\right)^{2t} \textbf{ mod } N^2-1}{N}
\end{aligned}$$

Now, we refer to the decomposition of $\mathbb{Z}^*_{N^2}$. Since $(1+N) \in \mathbf{G}_N$,

$$\frac{\left((1+N)^s\right)^{2t} \textbf{ mod } N^2-1}{N} = \frac{(1+N)^s \textbf{ mod } N^2-1}{N} = \frac{1+N\cdot s-1}{N} = s$$

5.2.2.1 Efficient Proof of Equality to Prime Order Discrete-Logarithm

We need an efficient zero-knowledge proof of knowledge of equality between a discrete logarithm in a prime order group and a secret value in a time-lock puzzle. We come up with the following protocol which satisfies our needs. This protocol closely follows a similar protocol from (Camenisch & Shoup, 2003) which also benefits from integer commitments (Damgård & Fujisaki, 2002). Informally, this protocol does not directly prove that solving time-lock puzzle reveals the discrete logarithm in a prime order group. However, it proves that a mapping $recons: \mathbb{Z}_N \to \mathbb{Z}_p$ maps the secret value which is revealed by solving time-lock puzzle to the discrete logarithm value in the prime order group. It is obvious that this is not a direct equivalence proof. Nevertheless, it is enough for our purpose as we only need to ensure verifiers of this protocol that by solving the related time-lock puzzle instance for an event they can learn the secret *bid* value for that event. It is also important that the mapping *recons* must be efficient and we use a function that only computes two modulo operations, which will be explained in detail.

First, we underline that we assume that the factorization of the RSA modulus N must be unknown to the prover for the soundness of this protocol. This is not a problem for our scheme since we already assume a trusted setup procedure in our scheme. Furthermore, this assumption is also used in (Camenisch & Shoup, 2003; Damgård & Fujisaki, 2002) and other protocols with similar aims. For $N = pq$,

$p = 2p' + 1$, and $q = 2q' + 1$, let $N' = p'q'$. Let $\mathfrak{g}$ and $\mathfrak{h}$ be generators of $\mathcal{QR}_N$ which is the group of quadratic residues modulo N. We would like to remind the reader that $\mathcal{QR}_N$ has order N'. Furthermore, let Γ be a prime order group of order ρ and let γ be a generator of Γ. Let $\delta = \gamma^m$. For the relation between group orders, we need to define two more parameters k and k' that determine the size of the challenge space and the quality of zero knowledge, respectively. We need $\rho < N2^{-k-k'-3}$ and $2^k < \min(p', q', \rho)$. We define a mapping **rem** $: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ to compute the balanced remainder st. a **rem** $b = a - \lceil a/b \rfloor b$. Similar to (Camenisch & Van Herreweghen, 2002), our protocol applies the mapping $recons(m) = m$ **rem** N **mod** ρ. As a result of the explanation above, we denote this protocol as:

$$PK\{(r,m) : Z = \mathsf{TLP.PGen}(pp, m; r) \wedge \delta = \gamma^{recons(m)}\}$$

Finally, we assume that the prover and the verifier both know two random generators $\mathfrak{g}$ and $\mathfrak{h}$ such that $\langle \mathfrak{g} \rangle = \langle \mathfrak{h} \rangle = \mathcal{QR}_N$. For our PPB scheme, these two parameters is generated in Setup with $\mathsf{ZKP.Gen}$ call.

Prover and verifier both know u, v, γ such that $u = \tilde{g}^r (mod\ N)$, $v = h^r \tilde{h}^m (mod\ N^2)$, and $\delta = \gamma^m$ for $r \leftarrow \{1, ..., N^2\}$. The prover further computes $\mathfrak{l} = \mathfrak{g}^m \mathfrak{h}^s$ for $s \leftarrow \{1, ..., n/4\}$ and sends it to the verifier.

Now, the prover computes the following values:

$$s' \leftarrow [-N2^{k+k'-2}, N2^{k+k'-2}]$$

$$r' \leftarrow [-N^2 2^{k+k'}, N^2 2^{k+k'}]$$

$$m' \leftarrow [-\rho 2^{k+k'}, \rho 2^{k+k'}]$$

$$t_u = \tilde{g}^{2r'} \ \mathbf{mod}\ N$$

$$t_v = h^{2r'N} \tilde{h}^{2m'} \ \mathbf{mod}\ N^2$$

$$t_\delta = \gamma^{m'}$$

$$t_{\mathfrak{l}} = \mathfrak{g}^{m'} \mathfrak{h}^{s'} \ \mathbf{mod}\ N$$

and sends them to the verifier. Subsequently, the verifier sends a challenge value $c \leftarrow \{0,1\}^k$ to the prover. The prover sends the following response values to the verifier.

$$z_r = r' - cr \quad z_m = m' - cm \quad z_s = s' - cs$$

$\mathcal{D}_0$	Same as the protocol
$\mathcal{D}_1$	$z_r \leftarrow [-cr - N^2 2^{k+k'}, -cr + N^2 2^{k+k'}]$ $z_s \leftarrow [-cs - N2^{k+k'-2}, -cs + N2^{k+k'-2}]$ $z_m \leftarrow [-cm - \rho 2^{k+k'}, -cm + \rho 2^{k+k'}]$ $t_v = h^{2N(z_r+cr)}\tilde{h}^{2(z_m+cm)} \textbf{ mod } N^2$ $t_\delta = \gamma^{z_m+cm}$ $t_{\mathfrak{l}} = \mathfrak{g}^{z_m+cm}\mathfrak{h}^{z_s+cs} \textbf{ mod } N$ $\tilde{g}^{2(z_r+cr)} \textbf{ mod } N$
$\mathcal{D}_2$	(z values are the same as $\mathcal{D}_1$) $t_{\mathfrak{l}} = \mathfrak{g}^{z_m}\mathfrak{h}^{z_s}\mathfrak{l}^c \textbf{ mod } N$ $t_u = \tilde{g}^{2z_r}u^{2c} \textbf{ mod } N$ $t_v = h^{2Nz_r}\tilde{h}^{2z_m}v^{2c} \textbf{ mod } N^2$ $t_\delta = \gamma^{z_m}\delta^c$
$\mathcal{D}_3$	t values are the same as $\mathcal{D}_2$ $z_s \leftarrow [-N2^{k+k'-2}, N2^{k+k'-2}]$ $z_m \leftarrow [-\rho 2^{k+k'}, \rho 2^{k+k'}]$ $z_r \leftarrow [-N^2 2^{k+k'}, N^2 2^{k+k'}]$

Figure 5.7 Distributions for Simulator Responses

Lastly, the verifier checks the relationships below.

$$t_v = h^{2z_r}\tilde{h}^{2z_m}v^{2c} \textbf{ mod } N^2$$

$$t_u = \tilde{g}^{2z_r}u^{2c} \textbf{ mod } N$$

$$t_\delta = \gamma^{z_m}\delta^c$$

$$t_{\mathfrak{l}} = \mathfrak{g}^{z_m}\mathfrak{h}^{z_s}\mathfrak{l}^c \textbf{ mod } N$$

$$-N/4 < z_m < N/4$$

If any of them does not hold, the verifier outputs 0. Otherwise, it outputs 1.

Proof. We provide a sketch for the statistical honest-verifier zero-knowledge property of the protocol above as follows. Fig. 5.7 contains four distributions such that the first one is identical to the output of a protocol run and the last one is the output of an efficient simulator which we use for statistical HVZK property. It is straightforward to observe that $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$ are identical. Therefore, we need to show that these distributions are statistically indistinguishable from $\mathcal{D}_3$. For distribution of z_r in $\mathcal{D}_i$, $\mathcal{D}^{(i)}_{z_r}$, let Δ_{z_r} denote the distance between $\mathcal{D}^{(2)}_{z_r}$ and $\mathcal{D}^{(3)}_{z_r}$. Also, for notational simplicity, let $\eta = N^2 2^{k+k'}$. Then

$$
\begin{aligned}
\Delta_{z_r} &= \sum_{Z=-cr-\eta}^{\eta} \left| \Pr\left[\ z \leftarrow \mathcal{D}^{(2)}_{z_r} \ :\ z = Z\ \right] - \Pr\left[\ z \leftarrow \mathcal{D}^{(3)}_{z_r} \ :\ z = Z\ \right] \right| \\
&= \sum_{Z=-cr-\eta}^{-1-\eta} \frac{1}{2\eta} + \sum_{Z=-cr+\eta+1}^{\eta} \frac{1}{2\eta} = cr\frac{1}{\eta} \leq 2^k N^2 \frac{1}{\eta} = 2^{-k'}
\end{aligned}
$$

Similarly, we can show that $\Delta_{z_m} \leq 2^{-k'}$ and $\Delta_{z_s} \leq 2^{-k'}$. Hence, the distance between $\mathcal{D}_2$ and $\mathcal{D}_3$ is bounded form above by $3 \cdot 2^{-k}$.

Now, we left with the soundness proof of the protocol. To prove the soundness of the protocol, we show that there is an efficient knowledge extractor that takes $t_u, t_v, t_\delta, t_\mathfrak{l}$, two valid answers $z_r^{(1)}, z_s^{(1)}, z_m^{(1)}$ and $z_r^{(2)}, z_s^{(2)}, z_m^{(2)}$ for challenges $c^{(1)}$ and $c^{(2)}$, respectively. Without loss of generality, suppose that $c^{(2)} > c^{(1)}$. Subsequently, let $\Delta_c = c^{(2)} - c^{(1)}$, $\Delta_r = z_r^{(1)} - z_r^{(2)}$, $\Delta_s = z_s^{(1)} - z_s^{(2)}$, and $\Delta_m = z_m^{(1)} - z_m^{(2)}$. From the verification equations, we know that

$$u^{2\Delta_c} = \tilde{g}^{2\Delta_r} \ \mathbf{mod}\ N$$

$$v^{2\Delta_c} = \tilde{g}^{2N\Delta_r} \tilde{h}^{2\Delta_m} \ \mathbf{mod}\ N^2$$

$$\delta^{\Delta_c} = \gamma^{\Delta_m}$$

$$\mathfrak{l}^{\Delta_c} = \mathfrak{g}^{\Delta_m} \mathfrak{h}^{\Delta_s} \ \mathbf{mod}\ N$$

Since we are able to compute $\mathfrak{l}$, Δ_c, Δ_m, and Δ_s for $\mathfrak{l}^{\Delta_c} = \mathfrak{g}^{\Delta_m}\mathfrak{h}^{\Delta_s}$ $\mathbf{mod}\ N$, we may assume $\Delta_c|\Delta_m$ and $\Delta_c|\Delta_s$, by Theorem 1. Moreover, Δ_c is invertible modulo all primes p, q, p', q', and ρ since $\Delta_c < min(p,q,p',q',\rho)$. Thus we are sure that $\hat{c} = \Delta_c^{-1}$ $\mathbf{mod}\ NN'$ exists. We also know that u^2 has an order which divides N', so $u^2 = \tilde{g}^{2\Delta_r\hat{c}}$ $\mathbf{mod}\ N$. Following that,

$$u = w_1 \tilde{g}^{\Delta_r \hat{c}} \ \mathbf{mod}\ N$$

$$\delta = \gamma^{\Delta_m/\Delta_c}$$

$$v = w_2 \tilde{g}^{N\Delta_r\hat{c}} \tilde{h}^{\Delta_m/\Delta_c} \ \mathbf{mod}\ N^2$$

for some w_1 and w_2 with order 2. Thanks to the exponentiation with $2t$ for $t = 2^{-1}$ $\mathbf{mod}\ N$ in our force opening algorithm, force opening (u,v) results with $\check{m} = \Delta_m/\Delta_c$ $\mathbf{mod}\ N$. Now we need to show that $recons(\check{m}) = \Delta_m/\Delta_c$ $\mathbf{mod}\ \rho$. Since $|z_m^{(1)}|, |z_m^{(2)}| < N/4$ and $\Delta_c|\Delta_m$, we have $|\Delta_m/\Delta_c| < N/2$. Due to the provided range,

$$\Delta_m/\Delta_c = ((\Delta_m/\Delta_c \ \mathbf{mod}\ N)\ \mathbf{rem}\ N) = (\check{m}\ \mathbf{rem}\ N)$$

Thus,

$$recons(\check{m}) = \Delta_m/\Delta_c \ \mathbf{mod} \ \rho$$

□

5.2.3 Security Proofs

The following theorems show that our scheme is *anonymous*, *backward-unlinkable*, and *sound* according to the related definitions in Section 5.1.

Theorem 4. *Given privacy-preserving borrowing scheme is anonymous in the random oracle model, if the underlying zero-knowledge proofs of knowledge are HVZK and sound,* DDH *assumption holds, underlying time-lock puzzle scheme* TLP *is secure, and underlying SPS-EQ signature scheme* $\mathsf{SPS\text{-}EQ}$ *is a signature scheme with perfect adaptation property.*

Proof. For the rest of the proof, the event S_i is the case that the adversary $\mathcal{A}$ wins $\mathsf{Game_i}$. $\mathsf{Game_0}$ is identical to the original anonymity game.

$\mathsf{Game_1}$: In $\mathsf{Game_1}$, on the first call of the adversary to $\mathsf{ReceiveCred}$, we run the ZKP extractor and get $sk = (sk_{\mathsf{SPS\text{-}EQ}}, \cdot, q_1, ..., q_3)$. Let the event F be the case that $\mathsf{Game_1}$ can not extract the witness of ZKP. In this case $\mathsf{Game_1}$ aborts. Since $\mathsf{Game_0}$ and $\mathsf{Game_1}$ behaves identically if F does not occur,

$$|\Pr[S_1] - \Pr[S_0]| \leq \Pr[F]$$

We can also imply that,

$$\Pr[F] \leq \mu_{snd}(\lambda)$$

Hence, $\mathsf{Game_0}$ and $\mathsf{Game_1}$ are computationally indistinguishable.

$\mathsf{Game_2}$: In this game, we simulate all zero-knowledge proofs of knowledge in $\mathcal{O}^{\mathsf{Borrow}}$ and $\mathcal{O}^{\mathsf{Repay}}$.

$$|\Pr[S_2] - \Pr[S_1]| \leq \mu_{zk}(\lambda)$$

$\mathsf{Game_3}$: In this game, all calls to $\mathsf{SPS–EQ.ChgRep}$ are changed to $\mathsf{SPS–EQ.Sign}$. Due to perfect adaptation of $\mathsf{SPS–EQ}$,

$$\Pr[S_3] = \Pr[S_2]$$

Game_4: Game_4 is the same with Game_3 except that for all calls to borrowing oracle for u_0 we change $btag_1$ responses of u_0 with $btag_1'$ for $btag_1' \leftarrow \mathsf{TLP.PGen}(bid')$ where $bid' \leftarrow \mathbb{Z}_p$ is freshly sampled for each borrowing oracle query for u_0.

$\mathsf{Game}_3 \rightarrow \mathsf{Game}_4$: In this transition, we instantiate an adversary against the security of time-lock puzzles by using each borrowing oracle call of our adversary. Let $q_{b,0}$ be the number of successful borrowing oracle calls for the user u_0. For this transition, we define the games $\mathsf{Game}_{\mathsf{u_0,0}}, ..., \mathsf{Game}_{\mathsf{u_0,q_{b,0}}}$. For notational simplicity, let $bid_{u_0,i}$ correspond to bid value which is used in i'th borrowing query for u_0. $\mathsf{Game}_{\mathsf{u_0,0}}$ is identical to Game_3. Each $\mathsf{Game}_{\mathsf{u_0,i}}$ for $i > 0$ is identical to $\mathsf{Game}_{\mathsf{u_0,i-1}}$ except that we change $btag_{u_0,i,1}$ which corresponds to $bid_{u_0,i}$ with $btag'_{u_0,i,1}$ for $btag'_{u_0,i,1} \leftarrow \mathsf{TLP.PGen}(bid'_{u_0,i})$ where $bid'_{u_0,i} \leftarrow \mathbb{Z}_p$. By the iterative definition, we have $\mathsf{Game}_{\mathsf{u_0,q_{b,0}}} = \mathsf{Game}_4$. Now, we need to prove that all these games are indistinguishable. We provide a general transition between $\mathsf{Game}_{\mathsf{u_0,i}}$ and $\mathsf{Game}_{\mathsf{u_0,i-1}}$. For any adversary that distinguishes these two games with non-negligible probability, we can generate an adversary wins TLP security game with non-negligible probability as follows. First, we use the time-lock puzzle public parameters which are provided by the TLP security game challenger as pp_{TLP} in our Setup algorithm. This does not change our adversaries view anyhow. Then, we run our anonymity game until the i'th borrowing query. At this point, we output the $s_0 = bid$ and $s_1 = bid'_{u_0,i}$ to the TLP security game challenger. We get Z as a challenged TLP instance and use this against our adversary. Let $(u_0, \cdot, k)$ be the member of $\mathcal{B}$ corresponding the i'th query and $(u_0, \cdot, k')$ be the corresponding member of $\mathcal{W}$ for the related repayment borrowing oracle query. If there is no such repayment oracle query, then we set $k' \leftarrow n$. Now, we can run A_k to $A_{k'}$ against the TLP security game. If the TLP security game challenger choose bid as the challenge, then we are in $\mathsf{Game}_{u_0,i-1}$. Otherwise, we are in $\mathsf{Game}_{u_0,i}$. Thus we can conclude that

$$|\Pr[S_{u_0,i}] - \Pr[S_{u_0,i-1}]| \leq \mu_{tlp}(\lambda)$$

By the iterative definition,

$$|\Pr[S_3] - \Pr[S_4]| \leq q_{b,0} \cdot \mu_{tlp}(\lambda)$$

Game_5: In Game_5, we guess the index of last call to $\mathcal{O}^{\mathsf{Borrow}}$ (or $\mathcal{O}^{\mathsf{ReceiveCred}}$) for u_0, i_0. If the guess is wrong, Game_5 chooses a random bit b' and simulates it as $\mathcal{A}$' output. Let the event E_5 be the case that Game_5's guess is wrong. Since the events E_5 and S_4 are independent events and the event $\neg E_5$ occurs with non-negligible probability,

$$|\Pr[S_5] - 1/2| = \Pr[\neg E_5] \cdot |\Pr[S_4] - 1/2|$$

Game_6: In this game, we choose $Z, R \leftarrow \mathbb{G}$, and $r, r', r_u, \leftarrow \mathbb{Z}_p$. Then, we program random oracle such that

$$h_4 = \mathcal{H}(\text{"}h_4\text{"} \| \mathcal{BG}) := R^{\frac{1}{r}} g^{-\frac{r'}{r}}$$

For query i_0, we initialize $C^{(pre)}$ as

$$C^{(pre)} = \left(g^{r_u}, Z^{r_u}, g^{f_0^*(\alpha) y^* r_u}\right), y^* \leftarrow \mathbb{Z}_p, f_0^*(X) = \prod_{bid \in BIDS_u^*} (X + bid)$$

Again, for i_0'th query, we change the stored $cred$ value to

$$cred = \left((g, R', g^{f_0'(\alpha) y'}), dsid_{user}, z, t, y', BIDS_u', \sigma\right)$$

where $dsid_{user} \leftarrow \mathbb{Z}_p$. $R' = R h_2^{dsid^*_{issuer}}$ and $BIDS_u' = BIDS_u^*$ if i_0 is the last oracle call before the guessing phase. Otherwise, for the set of all whitelisted bid's after the i_0'th query, $\bigcup_j \{bid_j\}$, $BIDS_u' = BIDS_u^* \setminus (\bigcup_j \{bid_j\})$. For appropriately initialized $BIDS_u'$, let $f_0'(X) = \prod_{bid \in BIDS_u'} (X + bid)$. Finally, $t = r$ and $z = \frac{1}{q_3}(r' - (q_1 usk_{u_0} + q_2 dsid))$.

$\mathsf{Game}_5 \to \mathsf{Game}_6$: Indistinguishability of these two games can loosely be reducted to class-hiding property of SPS-EQ. However, we can provide a strict reduction to DDH still inspired by the security reduction of the class hiding property ((Fuchsbauer et al., 2019)), but also by considering our special case. We prove that by using any adversary $\mathcal{A}$ that can distinguish between Game_5 and Game_6 with non-negligible probability, we can create an adversary that can win DDH game with a non-negligible probability. Before starting this reduction, let us point out that using Z instead of a new commitment instance in the i_0'th query is indistinguishable, since the underlying commitment is perfectly hiding. Thus, we only need to prove that $\mathcal{A}$ cannot distinguish whether Z or R has been used in the very last query. To prove this, we take a DDH instance. For a DDH tuple $(U, V, W) = (g^r, g^s, g^{(1-b)t + brs})$, let us program the random oracle as $R := V$. Then, we initialize $C^{(pre)} = (U, W, Y^*)$ for the i_0'th query. Let $V' = V h_2^{dsid^*_{issuer}}$, and $C = (g, V', Y)$ for the very last query, where $Y^*, Y \leftarrow \mathbb{G}^*$. Observe that using Y^* and Y instead of $g^{f_0^*(\alpha) y^* u^*}$ for some u^* and $g^{f_0'(\alpha) y'}$ is indistinguishable, since both $g^{f_0^*(\alpha) y^* u^*}$ and $g^{f_0'(\alpha) y'}$ are uniformly random in $\mathbb{G}^*$ for $y^*, y' \leftarrow \mathbb{Z}_p$. Hence, if $b = 1$ for DDH tuple, we are in Game_5. Otherwise, we are in Game_6.

$$|\Pr[S_6] - \Pr[S_5]| \leq \mu_{DDH}(\lambda)$$

Game_7: The repetition of Game_4 for u_1.

$$|\Pr[S_7] - \Pr[S_6]| \leq q_{b,1} \cdot \mu_{tlp}(\lambda)$$

Game_8: The repetition of Game_5 for u_1.

$$|\Pr[S_8] - 1/2| = \Pr[\neg E_8] \cdot |\Pr[S_7] - 1/2|$$

Game_9: Game_9 is the repetition of Game_6 for u_1.

$$|\Pr[S_9] - \Pr[S_8]| \leq \mu_{DDH}(\lambda)$$

In Game_9, both u_0 and u_1 answer the last $\mathcal{O}^{\mathsf{Borrow}}$ call with freshly created random credentials, and Game_9 is indistinguishable from Game_0 under our security assumptions. □

Theorem 5. *Given privacy-preserving borrowing scheme is backward-unlinkable in the random oracle model, if underlying zero-knowledge proofs of knowledge are HVZK and sound,* DDH *assumption holds, the underlying time-lock puzzle scheme* TLP *is secure, and the underlying SPS-EQ signature scheme* SPS-EQ *is a signature scheme with perfect adaptation property.*

The proof of backward unlinkability applies the same steps as the proof of anonymity, except the following changes. First, in games Game_4 and Game_7, we change the behavior only for borrowing oracle queries before $\mathcal{O}^{\mathtt{RepayChl}}$ call. Secondly, in games Game_5 and Game_8, we guess the last borrowing query index before $\mathcal{O}^{\mathtt{BorrowChl}}$ for u_0 (and u_1), i_0 (i_1) in Game_5 (in Game_7).

Theorem 6. *If underlying hash functions are collision-resistant hash functions, underlying zero-knowledge proofs of knowledge is HVZK and sound, the discrete logarithm problem is hard in $\mathbb{G}_1$, underlying accumulator is a collision-resistant accumulator, and sunderlying SPS-EQ scheme is EUF-CMA, then the given borrowing scheme is sound.*

Proof. Our construction includes a signature on two commitments which contain the attributes of a credential. Obviously, any adversary who is able to forge signatures would be able to create arbitrary credentials. However, we should also consider that any adversary who can provide different openings to signed commitments would also be able to win the game.

Case 1. In this case, we look at where $b_1 = 1 \wedge upk_u \notin \mathcal{U}$. We would like to point out that the user secret key $usk \in \mathbb{Z}_p$ is an attribute of credentials in our system, and there is a one-to-one mapping between user secret keys and user public keys. Hence, if any adversary wins the game in this case, it means that the adversary is able to create a valid credential on a user secret key usk' by following one of two possible ways. Obviously, the former performs a $\mathsf{SPS\text{-}EQ}$ signature forgery so that the adversary can create a signature on a message which is not signed by an honest issuer oracle $\mathcal{O}^{\texttt{IssueCred}}$. The latter provides a second opening to the Pedersen commitment in the credential. In this case, there are two cases that an adversary can follow. The adversary can either find a different opening directly or the adversary can perform the zero-knowledge proof of knowledge protocols for a user secret key usk', which is different from the one in opening that the user knows.

Case 2. Now we investigate the remaining case where $b_1 = 1 \wedge upk \in \mathcal{U}$. For this case, the only possible way that an adversary wins the game is the case that $b_2 = 1$. We can easily conclude that if $b_1 = 1$ and $b_2 = 1$, then the adversary is able to present a credential on the user secret key usk which corresponds to $upk \in \mathcal{U}$, and set of bid's $BIDS_u$ such that the blacklisted $bid \notin BIDS'_u$. Again, this is possible in three ways.

- Adversary performs a $\mathsf{SPS\text{-}EQ}$ signature forgery.
- Adversary performs a zero-knowledge proof of knowledge for an old credential that does not contain bid with a fresh $dsid'$ value. This case is similar to performing a zero-knowledge proof of knowledge for a usk'. $dsid$ is another member of the signed Pedersen commitment. Hence, if adversary can perform this zero-knowledge proof, the underlying commitment algorithm is not binding, or underlying zero-knowledge proof of knowledge protocol is not sound.
- Adversary performs zero-knowledge proof of non-membership for a $BIDS'_u$ such that $bid \notin BIDS'_u$. This is again possible in two ways. The adversary must be able to find an opening to the commitment for a $BIDS'_u$ such that $bid \notin BIDS'_u$ or the adversary must perform the zero-knowledge proof of non-membership for $BIDS_u$ even if $bid \in BIDS_u$. For the first case we can easily have the following reduction. Assume that there is an adversary that can provide openings y_0 and y_1 for two different sets S_0 and S_1. Let $f_0(X) = \prod_{d \in S_0}(X + d)$ and $f_1(X) = \prod_{d \in S_1}(X + d)$. We know that they are

openings for the same commitment value, so $g^{y_0 f_0(\alpha)} = g^{y_1 f_1(\alpha)}$. Subsequently, we know $y_0 f_0(\alpha) \textbf{ mod } p = y_1 f_1(\alpha) \textbf{ mod } p$. Since $S_0 \neq S_1$, we know $f_0(X)$ and $f_1(X)$ have different roots. Hence, $y_0 f_0(X) \neq y_1 f_1(X)$, which means $f'(X) = y_0 f_0(X) - y_1 f_1(X)$ is a non-zero polynomial, but $f'(\alpha) = 0$. By finding the roots of $f'(\alpha)$, one can learn the accumulator trapdoor α. Hence, we can construct an adversary against underlying accumulator's collision-resistence, if the set commitments are not binding. Now the remaining way is performing a successful zero-knowledge proof of non-membership even if blacklisted $bid \in BIDS_u$. One can straightforwardly design an adversary against the soundness of the underlying zero-knowledge protocol's soundness or accumulator's collision resistance by using any such adversary.

□

5.3 Practical Applications

In this section, we analyze the suitability of our PPB scheme to the practical applications.

Peer-to-Peer Lending. There are three main entities in a peer-to-peer lending system.

- Borrower: Party wishes to borrow some amount of money.
- Lender: Party wishes to lend some amount of money.
- P2PLP: Peer-to-peer lending platform. It gives services to both borrowers and lenders.

Transfer phase of the money is out of this paper's interest, and we only care about the privacy of borrowers in all phases. In this regard, (Xie et al., 2020) only cares about privacy during the transfer phase (on Bitcoin network), and our work has a complementary role. While entities borrower and lender have an exact match with the entities using the same names in our scheme, P2PLP takes the role of credential issuer to apply our conditional anonymity scheme to peer-to-peer lending schemes.

Car Sharing Services. We focus on car sharing services with service providers, which is displayed in Fig. 5.8. Recent applications of vehicle sharing utilizes wide

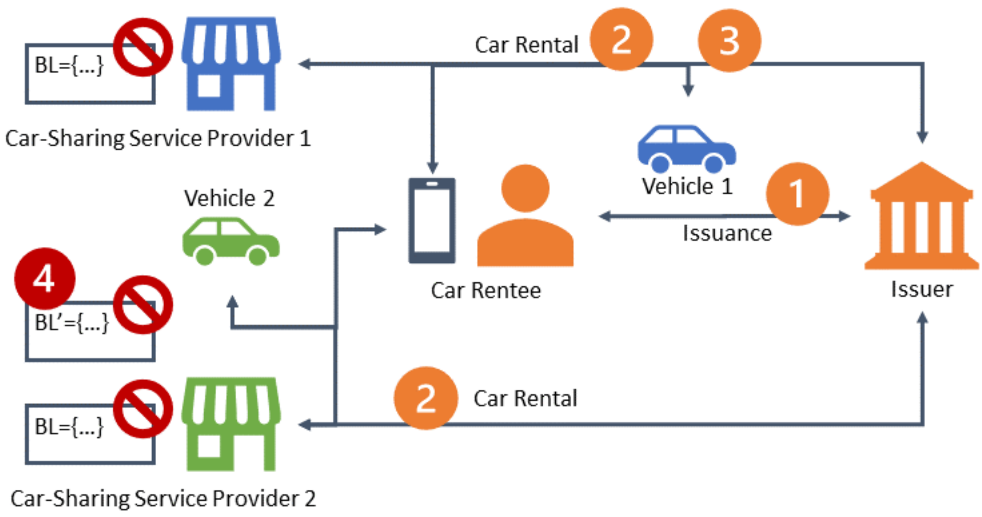

Figure 5.8 Car Sharing Architecture

range of vehicles from scooters and bicyles[1] to cars[2]. These applications provide access to the vehicles through the mobile phone of the user, so the user does not interact with a representative of the company during a rental. The system entities for a car-sharing environment we define are as follows:

- Car Rentee: Party wishes to rent a car from CSSP.
- Car-Sharing Service Provider (CSSP): Party provides a car rental service.
- Issuer: The party that registers users upon their real-world identity by issuing them credentials.

Our system matches with the system above. While car rentees correspond to borrowers, CSSP's take lender/service provider roles. We emphasize that our system allows multiple CSSP's to serve upon a single issuer. For the case that there is a single CSSP, it is possible that CSSP and Issuer are the same parties. Our PPB scheme is not vulnerable against the collision of the issuer and CSSP parties. Lastly, we underline that the proposed system does not have a TA role.

Steps 1, 2, 3, and 4 are identical to the steps in Fig. 5.1. A car renter registers to the system by running $\mathsf{ReceiveCred} \leftrightarrow \mathsf{IssueCred}$ with the issuer, which is displayed with Step 1 in Fig. 5.8. When a CSSP and a renter agree on the conditions, including the duration of car rental to perform a car sharing, they start the car rental service by performing $\mathsf{Borrow} \leftrightarrow \mathsf{Lend}$ together with the issuer as shown in Step 2 in Fig. 5.8 for both CSSP 1 and CSSP 2. A car-rental service is successfully completed,

[1]Bird. https://www.bird.co/

[2]ZipCar. https://www.zipcar.com/

if Repay $\leftrightarrow$ Collect operations performed without an error. In Fig. 5.8, CSSP 1, renter, and issuer perform Repay $\leftrightarrow$ Collect operations as pointed out with Step 3, since vehicle 1 was returned back without any problem. Finally, if a renter does not successfully return the rented vehicle in time, CSSP can add the dishonest renter to his blacklist using the Blacklist algorithm. In Fig. 5.8, CSSP blacklists the renter in Step 4, since the renter did not successfully return vehicle 2 in time.

6. PERFORMANCE EVALUATION OF OUR PPB SCHEME

We evaluate the performance of our scheme for different aspects. First, we present the benchmark of our PPB construction from Chapter 5. Then, we compare these benchmarks with another existing construction which has a similar application area (Huang et al., 2020). Finally, we discuss additional optimizations that could be possible for specific use cases. To measure the performance of our PPB protocol, we developed a proof-of-concept implementation. Our implementation is developed using the MIRACL[1] library in C++. We choose this library since we need pairing-based cryptography, RSA-based cryptography, and polynomial arithmetic algorithms at the same time. For bilinear pairings, we use BN-256 curve, and for time-lock puzzles, we use 2048-bits RSA modulus.

6.1 Computational Cost of Our PPB Scheme

Running time of our PPB implementation for borrowing/lending operations without non-membership witness update mechanisms is displayed in Figure 6.1 and 6.2. In the figures, we use *debt number* to refer to the size of $BIDS_u$ in the user credential at the beginning of the borrowing operation. On the credential issuer side, both the borrowing/lending and the repayment/collecting operations have constant costs with averages 122.25 ms and 18.20 ms, respectively. As additional information to the figure, repayment operations have light computational costs on the user side such that they require $O(d)$ group exponentiations for the debt number d, which yields lightweight repayment operations such as 25.71 ms for $d = 20$. Lastly, the issuance phase takes 21.43 ms on average including both the user and the issuer side.

The case in which a user always uses her credential for a single debt at a time in Fig. 6.1 is noteworthy. In this case, borrowing operations take a constant time for the user. Moreover, the user does not even have to learn the current accumulator

[1] https://github.com/miracl/MIRACL

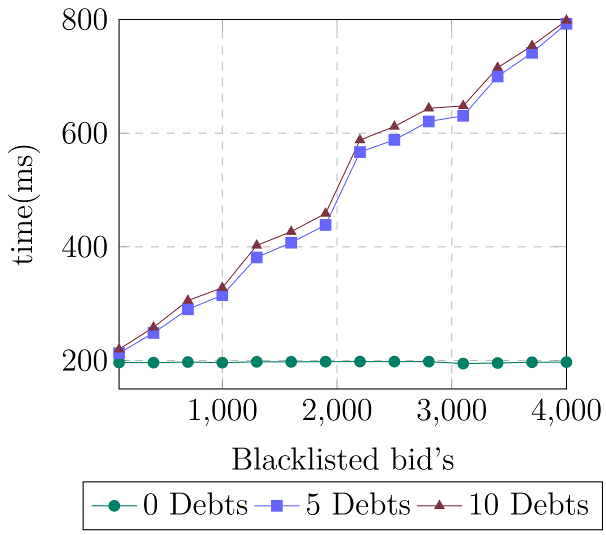

Figure 6.1 Borrow Cost at User for Blacklisted BID's

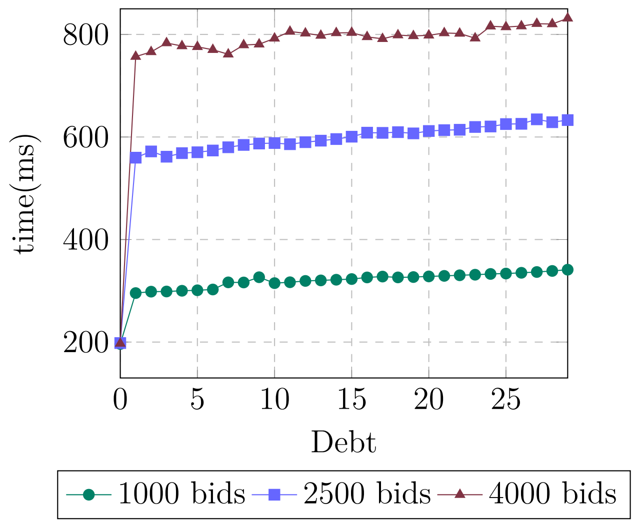

Figure 6.2 Borrow Cost at User Side for Debt Number

value for any blacklist. Since his credential does not include any *bid* at that time, it is enough for the borrower to store a constant witness for $BIDS_u = \emptyset$.

6.2 Comparative Evaluation with DAPA

In this part, we analyze the performance of our PPB scheme and DAPA, which is another scheme with privacy concerns on car-sharing, in detail.

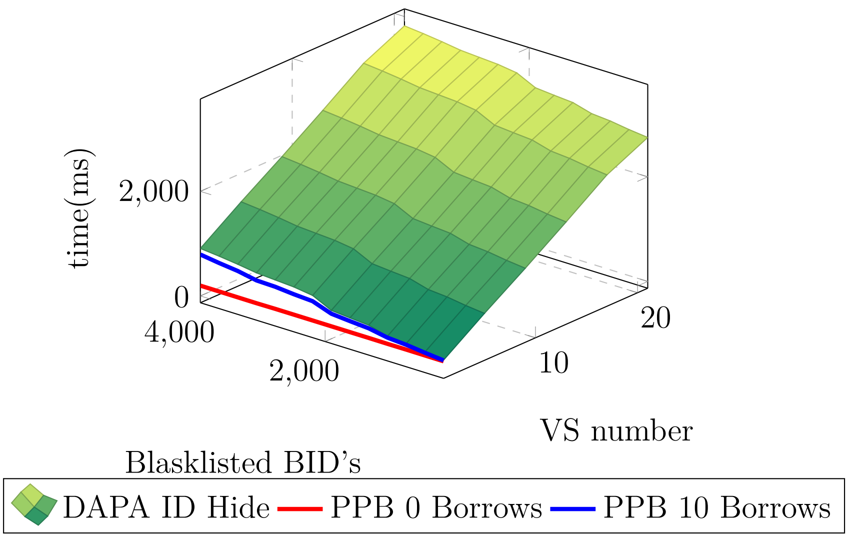

Figure 6.3 Comparison of borrowing-lending cost on the user side between PPB and DAPA

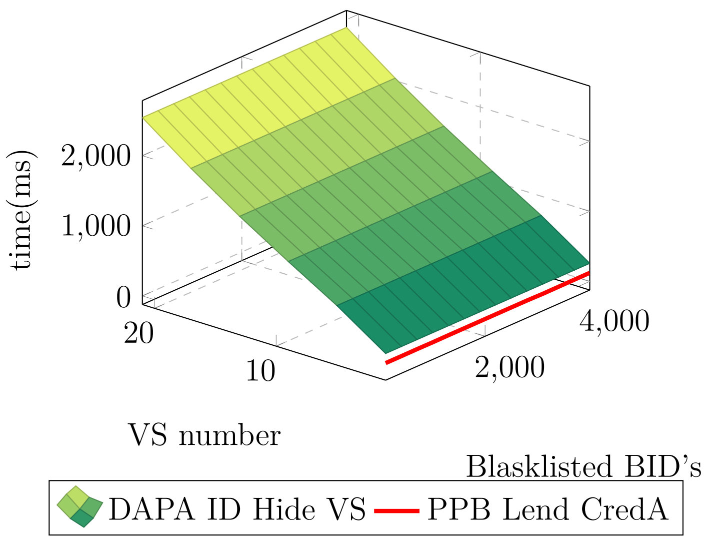

Figure 6.4 Comparison of borrowing-lending cost on the verifiable server and issuer sides between PPB and DAPA

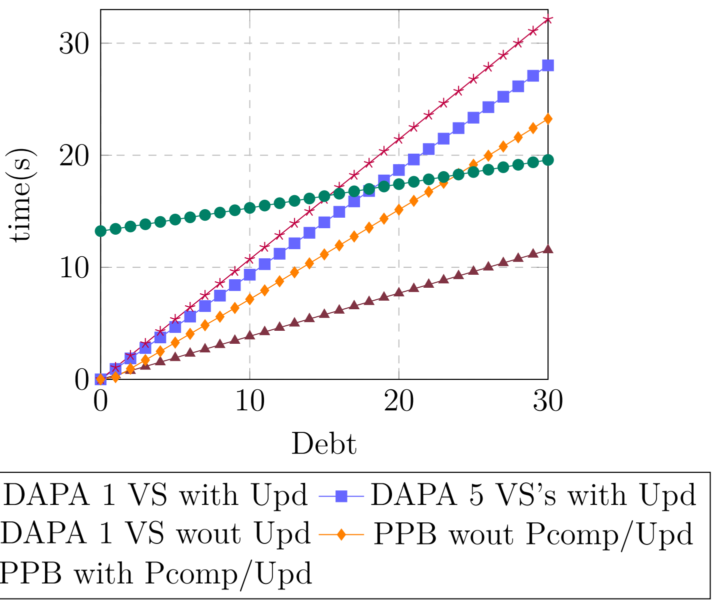

Figure 6.5 Comparison of cumulative cost for debt number at user side between PPB and DAPA

6.2.1 Comparison of Computational Cost

We compare our PPB scheme with DAPA (Huang et al., 2020), which has a similar application area. For this purpose, we implement DAPA scheme. DAPA applies an improved version of the traditional identity escrow concept to the car sharing use case. In summary, DAPA's purpose is mitigating TA's vulnerability by dividing its responsibility. Instead of employing a single TA, DAPA uses a committee of TA's, where each member of the committee is called *verifiable server*. Additionally, this committee transfers its authority to a committee of fresh members over periods. DAPA uses (Au, Tsang, Susilo & Mu, 2009)'s pairing-based accumulator for non-membership proofs. However, it is not explicitly declared whether they use the reference-string based construction or auxiliary information based construction. At this point, we implemented the reference string-based construction, since the auxiliary information-based construction has been broken by (Biryukov, Udovenko & Vitto, 2021). Reference-string based construction suffers from non-membership witness creation with non-constant time. Although the paper itself does not propose to use any kind of witness update operation, we would like to declare that witness update operation could be used for this scheme if a syncronization mechanism for witness update information can be met by user devices. Fig. 6.3 compares DAPA and PPB for user side and 6.4 compares verifiable server and issuer sides without witness update mechanisms. This figure clearly shows that, without employing any witness update mechanism, PPB is much more efficient than DAPA. DAPA's main motivation is to distribute the TA's responsibility, but PPB is more efficient than DAPA even when we configure DAPA for a single centralized TA.

6.2.2 Precomputation for PPB

PPB's accumulator does not have a previously known efficient update mechanism for our case. We prove the exclusion of two sets (set of user's *bid*'s, $BIDS_u$, and set of blacklisted *bid*'s) and both sets are dynamic. Thus, we need to have witness update mechanisms for both sets. Previously known mechanisms have $\mathcal{O}(n)$ computation complexity for $BIDS_u$ update case and $\mathcal{O}(d)$ computation complexity for blacklist update case where d is the size of $BIDS_u$, and n is the size of blacklist. We implement a restricted witness update mechanism for PPB to improve the performance for $BIDS_u$ updates. Namely, this mechanism makes precomputation for a set of *bid*' s, and the user can perform efficient witness updates until he uses all *bid*'s from this set. Since *bid*'s are uniformly random values, there is no difference between sampling them one by one in borrowing/lending operations and sampling k of them all together. In detail, the precomputation for k *bid*'s can be performed as follows. Let bid_i for $i = 1, ..., k$ denote *bid*'s to perform the precomputation on. Then, we compute polynomials $h(X)$ and $h_0(X)$ such that $f(X)h(X) - f'(X)h_0(X) = 1$ for $f'(X) = \prod_{i=1}^{k}(X + bid_i)$ and $f(X)$ is the polynomial represents the accumulator value. On top of these polynomials, we define $h_i(X) = h_0(X) \cdot X^i$ for $i = 1, ..., k-1$. The user computes and stores all $\hat{g}^{h_i(\alpha)}$ values. Using these group members, the user can compute $\hat{w}_1$ which forms a valid witness together with $w_2 = g^{h(\alpha)}$ for any $f_0(X)$ value which consists of arbitrary subset of $\{bid_1, ..., bid_k\}$ with at most k exponentiations.

Figure 6.5 compares several configurations of PPB and DAPA for the cumulative time on the user side when the user borrows an asset with the blacklist size, 4000. However, we would like to point out that the measurements for PPB with precomputation and witness updates and DAPA with witness updates does not perform any witness updates and just assume that witness for the current accumulator value was ready before starting the operation. We disregarded witness update costs, since the number of necessary updates is highly dependent to the use case. Figure 6.5 shows that PPB without witness update mechanisms may provide performance comparable to DAPA for 5 verifiable servers with witness update mechanisms. Furthermore, it shows that our precomputation method significantly reduces total computation cost.

6.2.3 Comparison of Communication Costs

The last metric that we compare DAPA and our PPB scheme is communication cost. For issuance operation, both DAPA and PPB have constant communication costs

Table 6.1 Communication Cost Comparison

Protocol	Communication Cost (kB)	
	IDHide/Borrowing	***Repayment***
DAPA (Huang et al., 2020) with 1 VS	16.6	-
DAPA (Huang et al., 2020) with 5 VS's	33.1	-
Our PPB Construction	16.0	3.9

0.06 kB's and 3.8 kB's, respectively. While obviously DAPA is more efficient for issuance operation, it is not a significant disadvantage to use PPB, since issuance is a one-time process for each user for her lifetime. Table 6.1 shows the communication cost comparison between DAPA and PPB. These cost measurements disregard the communication cost needed to update the accumulator witness. In this case, when we disregard the communication cost of the accumulator / blacklist update, PPB has a constant communication cost for issuance, borrowing, and repayment operations. DAPA's communication cost increases according to the number of verifiable servers and creates a bottleneck on the system for a high number of verifiable servers. Hence, PPB has better communication complexity and better communication cost than DAPA.

6.3 Discussion on Further Optimizations

Our goal was to come up with a scheme that covers as many use cases of sharing economy as possible. For that reason, further optimizations could be possible if the use cases to be supported are explicitly determined. To be clear, our scheme could be optimized considering the borrowing frequencies of users and the duration of borrowings. One point that is open to optimization is the utilization of time-lock puzzles. Since service providers are required to perform computation on a time-lock puzzle for each lended resource, they may need high computation power for some use cases. The methods proposed in (Abadi & Kiayias, 2021; Malavolta & Thyagarajan, 2019) can be used to perform the evaluation of time-lock puzzles for multiple *bid*' s with the cost of a single time-lock puzzle for some scenarios. Another possible optimization is related to using updatable anonymous credential concept. Independent of our usage, in case of frequent updates on credentials, updatable anonymous credentials may lead to used *dsid* lists with high storage costs. Together with the *dsid* checks, applying limited lifetimes for determined epochs to credentials may reduce the size of the used *dsid* list, as only storing used *dsid*'s for the current

Table 6.2 Specifications for experiment environments

User	Issuer	Lender
Sony Xperia XZ Premium	Cloud Run Container	Cloud Run Container
Android 9.0	Ubuntu 18.04 in a Container	Dart Lang. in a Container
4GB Memory	512 MiB Memory	512 MiB Memory
Qualcomm Kryo 1900 MHz (4-Core)	1 vCPU for each Container Instance	1 vCPU for each Container Instance

epoch would be sufficient. We note that a similar approach is also recommended in (Camenisch et al., 2010).

6.4 Performance Evaluation for Real-World Suitability

In this chapter, we further evaluate the performance of our protocol to see its suitability for real-world applications. We deploy a proof-of-concept application for each party in the system to a device that is suitable for their real-world matches. While we deploy the issuer and lender parties to a cloud system, we deploy the user application to a mobile phone. Figure 6.6 displays the architecture of this system. Detailed specifications for deployment environments are available in Table 6.2. We used Google Cloud Platform's Cloud Run[2] service to run issuer and lender sides. Cloud Run is a serverless, managed compute platform which allows users to run containerized applications.

On the user side, we used the same implementation code we have in Chapter 6 which uses C++ language. We wrapped our previously existing code to a library which takes inputs as C strings and returns all outputs as C strings. Finally, we used this library with Dart Language's foreign function interface (Dart:FFI library) to make our native C++ code easily accessible from Dart code. To run this code in an Android application, we used Flutter Framework which is a multi-platform application development framework for Dart language.

On the issuer side, we followed the same steps as on the user side to make our C++ library accessible from Dart using Dart:FFI library. Subsequently, we implemented

[2]https://cloud.google.com/run

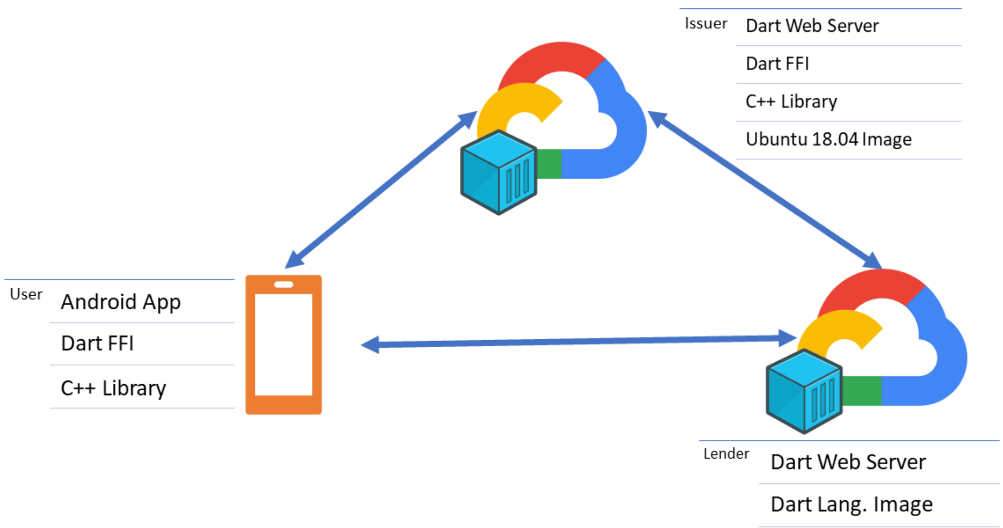

Figure 6.6 Architecture of the experiment environment for mobile benchmarks

a basic web server on Dart. Lastly, we created a Docker image based on the official Docker image for Ubuntu 18.04 [3].

On the lender side, we implemented a basic Dart web server, and created our image on the official Docker image for Dart [4].

We run each operation 50 times in the environment we explained above. The average and standard deviation of these operations' time costs are available in Tables 6.3, 6.4, and 6.5. "AgreeWithLender" procedure corresponds to the time that the user contacts the lender for the service, the lender contacts the issuer for the service and the lender returns to the user. Procedures starting with "Init" correspond to the time that the user performs the initial computation for each operation. Similarly, the procedures starts with "Finish" correspond to the time that the user performs the final computation for that operation after receiving necessary values from the issuer. The procedure "User2IssuerComm" is the total communication duration between the user and the issuer. The procedures "IssueCred", "GetBlacklist", "Lend", and "Collect" are self-explanatory.

The main result of these experiments is that our algorithms for user-side can run in reasonable time even on mobile devices. We did not test the performance of the

[3]https://hub.docker.com/_/ubuntu

[4]https://hub.docker.com/_/dart

Table 6.3 Issuance Time-Cost

IssueCred-ReceiveCred	Cost (ms)	
	Mean	***Std.***
InitReceiveCred	56.36	0.89
IssueCred	84.22	4.08
FinishReceiveCred	156.30	7.16
User2IssuerComm	606.68	226.76
Overall	903.56	227.78

Table 6.4 Borrowing Time-Cost for blacklist size, 3900.

Borrow-Lend	$\lvert BIDS_u \rvert = 0$		$\lvert BIDS_u \rvert = 3$	
	Mean (ms)	***Std. (ms)***	***Mean (ms)***	***Std. (ms)***
AgreeWithLender	1643.64	457.83	1455.04	362.34
GetBlacklist	1313.58	562.86	1106.26	487.71
InitBorrow	877.76	15.25	2218.94	16.01
Lend	876.90	7.20	867.72	7.06
FinishBorrow	105.34	13.42	101.92	8.09
User2IssuerComm	384.94	200.05	252.88	119.60
Overall	5202.16	725.36	6002.76	669.67

issuer against the number of users. However, considering the resource of the issuer-side implementation, we can say that our issuer-side algorithms can be deployed as a cloud service.

Table 6.5 Repayment Time-Cost for blacklist size 3900 and $|BIDS_u| = 4$.

Repay-Collect	**Cost (ms)**	
	Mean	***Std.***
AgreeWithLender	1511.92	305.13
InitRepay	138.74	7.94
Collect	129.06	2.88
FinishRepay	148.96	10.77
User2IssuerComm	591.18	116.80
Overall	2519.86	340.72

7. CONCLUSION

With the rapid adaptation of digital channels to the real world, various business models are adapted to computer systems through these channels. These applications bring the private user data they access through the provided service with them to the digital systems. Hence, the value of user privacy in these systems is increasing. In this thesis, we first proposed a blacklistable anonymous credential scheme, BLACW, that has an additional property, whitelisting property, to unlink the honest user credentials from the corresponding authentication sessions. Subsequently, we provide our construction for BLACW scheme and analyzed its performance by comparing with a rival scheme. Our experiments showed that BLACW has a comparable performance to previous schemes together with its stronger security requirements. While our BLACW scheme has a general application area, we extended this scheme further for a more specific business model, sharing economy. We defined a novel conditional anonymity scheme, namely, the privacy-preserving borrowing (PPB) scheme. Our PPB scheme provides anonymity to a user during an asset sharing, and the user becomes blacklistable if the shared asset is not returned in time. Our PPB scheme also assures backward unlinkability, so previous services that the user returned the shared asset honestly in time can not be linked to the further services even if the user misbehaved in further services. Our scheme is suitable to serve as a privacy-preserving mechanism for multiple sharing economy applications such as car-sharing or peer-to-peer lending. We provided the basic definition and construction of our PPB scheme using structure-preserving signatures on equivalence classes, accumulators, and time-lock puzzles. Finally, we analyzed the performance of our PPB scheme and compared it with a rival one. Our analyzes show that our PPB scheme outperforms it in both computational and communication cost metrics. The main drawback of this scheme is that the service provider must solve a time-lock puzzle for each sharing service. We can show a PPB scheme which is based on more efficient time-delayed cryptographic primitives as a further work. The possibility of conditionally anonymous schemes with more complex policies also stands as an interesting open problem.

BIBLIOGRAPHY

Abadi, A. & Kiayias, A. (2021). Multi-instance Publicly Verifiable Time-Lock Puzzle and Its Applications. In Borisov, N. & Diaz, C. (Eds.), *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, (pp. 541–559)., Berlin, Heidelberg. Springer.

Aikou, Y., Sadiah, S., & Nakanishi, T. (2017). An Efficient Blacklistable Anonymous Credentials without TTPs Using Pairing-Based Accumulator. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, (pp. 780–786). ISSN: 1550-445X.

Au, M. H., Tsang, P. P., Susilo, W., & Mu, Y. (2009). Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems. In M. Fischlin (Ed.), *Topics in Cryptology – CT-RSA 2009*, volume 5473 (pp. 295–308). Berlin, Heidelberg: Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

Badimtsi, F., Canetti, R., & Yakoubov, S. (2020). Universally Composable Accumulators. In Jarecki, S. (Ed.), *Topics in Cryptology – CT-RSA 2020*, (pp. 638–666)., Cham. Springer International Publishing.

Biryukov, A., Udovenko, A., & Vitto, G. (2021). Cryptanalysis of a dynamic universal accumulator over bilinear groups. In Paterson, K. G. (Ed.), *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, (pp. 276–298). Springer.

Blömer, J., Bobolz, J., Diemert, D., & Eidens, F. (2019). Updatable Anonymous Credentials and Applications to Incentive Systems. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, (pp. 1671–1685)., London United Kingdom. ACM.

Bobolz, J., Eidens, F., Krenn, S., Slamanig, D., & Striecks, C. (2020). Privacy-Preserving Incentive Systems with Highly Efficient Point-Collection. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, (pp. 319–333)., New York, NY, USA. Association for Computing Machinery.

Boneh, D. & Naor, M. (2000). Timed Commitments. In G. Goos, J. Hartmanis, J. van Leeuwen, & M. Bellare (Eds.), *Advances in Cryptology — CRYPTO 2000*, volume 1880 (pp. 236–254). Berlin, Heidelberg: Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

Boneh, D. & Shoup, V. (2020). *A Graduate Course in Applied Cryptography.*

Bowe, S., Gabizon, A., & Miers, I. (2017). Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050. `https://ia.cr/2017/1050`.

Bringer, J. & Patey, A. (2012). VLR group signatures - how to achieve both backward unlinkability and efficient revocation checks. In Samarati, P., Lou, W., & Zhou, J. (Eds.), *SECRYPT 2012 - Proceedings of the International Conference on Security and Cryptography, Rome, Italy, 24-27 July, 2012, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, (pp. 215–220). SciTePress.

Camenisch, J., Kohlweiss, M., & Soriente, C. (2010). Solving Revocation with Efficient Update of Anonymous Credentials. In Garay, J. A. & De Prisco, R. (Eds.), *Security and Cryptography for Networks*, Lecture Notes in Computer Science, (pp. 454–471)., Berlin, Heidelberg. Springer.

Camenisch, J. & Shoup, V. (2003). Practical Verifiable Encryption and Decryption of Discrete Logarithms. In Boneh, D. (Ed.), *Advances in Cryptology - CRYPTO 2003*, (pp. 126–144)., Berlin, Heidelberg. Springer Berlin Heidelberg.

Camenisch, J. & Van Herreweghen, E. (2002). Design and implementation of the *idemix* anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, (pp. 21–30)., New York, NY, USA. Association for Computing Machinery.

Chen, M., Cohen, R., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., & Shelat, A. (2020). Multiparty Generation of an RSA Modulus. In Micciancio, D. & Ristenpart, T. (Eds.), *Advances in Cryptology – CRYPTO 2020*, (pp. 64–93)., Cham. Springer International Publishing.

Chvojka, P., Jager, T., Slamanig, D., & Striecks, C. (2021). Versatile and Sustainable Timed-Release Encryption and Sequential Time-Lock Puzzles (Extended Abstract). In Bertino, E., Shulman, H., & Waidner, M. (Eds.), *Computer Security – ESORICS 2021*, (pp. 64–85)., Cham. Springer International Publishing.

Damgård, I. & Fujisaki, E. (2002). A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In G. Goos, J. Hartmanis, J. van Leeuwen, & Y. Zheng (Eds.), *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 (pp. 125–142). Berlin, Heidelberg: Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

Fiat, A. & Shamir, A. (1987). How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Odlyzko, A. M. (Ed.), *Advances in Cryptology — CRYPTO' 86*, Lecture Notes in Computer Science, (pp. 186–194)., Berlin, Heidelberg. Springer.

Freitag, C., Komargodski, I., Pass, R., & Sirkin, N. (2021). Non-malleable Time-Lock Puzzles and Applications. In K. Nissim & B. Waters (Eds.), *Theory of Cryptography*, volume 13044 (pp. 447–479). Cham: Springer International Publishing. Series Title: Lecture Notes in Computer Science.

Fuchsbauer, G., Hanser, C., & Slamanig, D. (2019). Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials. *Journal of Cryptology*, *32*(2), 498–546.

Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., & Triandopoulos, N. (2016). Zero-Knowledge Accumulators and Set Algebra. In *Advances in Cryptology – ASIACRYPT 2016*, (pp. 67–100). Springer, Berlin, Heidelberg.

Hu, X., Zhao, H., Zheng, S., & Wang, L. (2020). CBOL: Cross-Bank Over-Loan Prevention, Revisited. *Entropy*, *22*(6), 619. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

Huang, C., Lu, R., Ni, J., & Shen, X. (2020). DAPA: A Decentralized, Accountable, and Privacy-Preserving Architecture for Car Sharing Services. *IEEE Transactions on Vehicular Technology*, *69*(5), 4869–4882. Conference Name: IEEE Transactions on Vehicular Technology.

Katz, J., Loss, J., & Xu, J. (2020). On the Security of Time-Lock Puzzles and Timed Commitments. Technical Report 730.

Kilian, J. & Petrank, E. (1998). Identity escrow. In Krawczyk, H. (Ed.), *Advances in Cryptology — CRYPTO '98*, Lecture Notes in Computer Science, (pp. 169–185)., Berlin, Heidelberg. Springer.

Liu, Y., Xue, K., He, P., Wei, D. S. L., & Guizani, M. (2020). An Efficient, Accountable, and Privacy-Preserving Access Control Scheme for Internet of Things in a Sharing Economy Environment. *IEEE Internet of Things Journal*, *7*(7), 6634–6646. Conference Name: IEEE Internet of Things Journal.

Malavolta, G. & Thyagarajan, S. A. K. (2019). Homomorphic Time-Lock Puzzles and Applications. In Boldyreva, A. & Micciancio, D. (Eds.), *Advances in Cryptology – CRYPTO 2019*, (pp. 620–649)., Cham. Springer International Publishing.

Manevich, Y. & Akavia, A. (2022). Cross Chain Atomic Swaps in the Absence of Time via Attribute Verifiable Timed Commitments. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, (pp. 606–625).

Nakanishi, T. & Funabiki, N. (2005). Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps. In Roy, B. (Ed.), *Advances in Cryptology - ASIACRYPT 2005*, Lecture Notes in Computer Science, (pp. 533–548)., Berlin, Heidelberg. Springer.

Nakanishi, T. & Kanatani, T. (2018). An efficient blacklistable anonymous credential system with reputation using pairing-based accumulator. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, (pp. 1140–1148).

Symeonidis, I., Aly, A., Mustafa, M. A., Mennink, B., Dhooghe, S., & Preneel, B. (2017). SePCAR: A Secure and Privacy-Enhancing Protocol for Car Access Provision. In S. N. Foley, D. Gollmann, & E. Snekkenes (Eds.), *Computer Security – ESORICS 2017*, volume 10493 (pp. 475–493). Cham: Springer International Publishing. Series Title: Lecture Notes in Computer Science.

Symeonidis, I., Rotaru, D., Mustafa, M. A., Mennink, B., Preneel, B., & Papadimitratos, P. (2022). Hermes: Scalable, secure, and privacy-enhancing vehicular sharing-access system. *IEEE Internet of Things Journal*, *9*(1), 129–151.

Thakur, S. (2019). Batching non-membership proofs with bilinear accumulators. Cryptology ePrint Archive, Report 2019/1147. `https://ia.cr/2019/1147`.

Tsang, P. P., Au, M. H., & Kapadia, A. (2008). Perea: Towards practical ttp-free revocation in anonymous authentication. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, (pp. 333–344)., New York, NY, USA. Association for Computing Machinery.

Tsang, P. P., Au, M. H., Kapadia, A., & Smith, S. W. (2007). Blacklistable anonymous credentials: Blocking misbehaving users without ttps. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, (pp. 72–81)., New York, NY, USA. Association for Computing Machinery.

Verheul, E. R. (2016). Practical backward unlinkable revocation in fido, german e-id, idemix and u-prove.

Xie, Y., Holmes, J., & Dagher, G. G. (2020). ZeroLender: Trustless Peer-to-Peer Bitcoin Lending Platform. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, (pp. 247–258)., New Orleans LA USA. ACM.

Yang, R., Au, M. H., Xu, Q., & Yu, Z. (2019). Decentralized blacklistable anonymous credentials with reputation. *Computers & Security*, *85*, 353–371. Publisher: Elsevier BV.