This paper will appear in the 15th ACM ASIA Conference on Computer and Communication Security (ASIA CCS 2020), October 5–9, Taipei, Taiwan. © ACM, 2020.
https://doi.org/10.1145/3320269.3384769

Privacy-Preserving Incentive Systems with Highly Efficient Point-Collection*

Jan Bobolz[1], Fabian Eidens[1],
Stephan Krenn[2], Daniel Slamanig[2], Christoph Striecks[2]

[1] Paderborn University, Germany
firstname.lastname@uni-paderborn.de
[2] AIT Austrian Institute of Technology, Austria
firstname.lastname@ait.ac.at

Abstract. Incentive systems (such as customer loyalty systems) are omnipresent nowadays and deployed in several areas such as retail, travel, and financial services. Despite the benefits for customers and companies, this involves large amounts of sensitive data being transferred and analyzed. These concerns initiated research on privacy-preserving incentive systems, where users register with a provider and are then able to *privately* earn and spend incentive points.

In this paper we construct an incentive system that improves upon the state-of-the-art in several ways:

- We improve efficiency of the Earn protocol by replacing costly zero-knowledge proofs with a short structure-preserving signature on equivalence classes.
- We enable tracing of remainder tokens from double-spending transactions without losing backward unlinkability.
- We allow for secure recovery of failed Spend protocol runs (where usually, any retries would be counted as double-spending attempts).
- We guarantee that corrupt users cannot falsely blame other corrupt users for their double-spending.

We propose an extended formal model of incentive systems and a concrete instantiation using homomorphic Pedersen commitments, ElGamal encryption, structure-preserving signatures on equivalence classes (SPS-EQ), and zero-knowledge proofs of knowledge. We formally prove our construction secure and present benchmarks showing its practical efficiency.

Keywords: Incentive systems · Privacy · Provable security

1 Introduction

Customer loyalty and incentive systems represent an entire industry with massive promising benefits for many businesses. Those systems offer incentives to users to, for example, buy certain products or repeatedly visit the same shop (through customer loyalty incentives). They are widely implemented via dedicated programs ranging from simple point-based programs (e.g., IHG Rewards Club) over tiered programs (e.g., Shop.com, Cashback, and Best Buy) to paid and punch systems (e.g., Amazon Prime and Hotels.com) serving 3.3 billion customers alone in the US.[3] Incentive systems are mostly deployed in practice in form of physical loyalty cards, rewards cards, points cards, advantage cards or club cards, and can be found in traditional business areas such as retail, travel, or financial services that come with a variety of promotion and condition flavors.[4]

While in the past most of these program required dedicated plastic cards, nowadays either smartphone apps provided by the vendors of the incentive system or dedicated mobile applications (e.g., Apple's Wallet)

* This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (GZ: SFB 901/3) under the project number 160364472 and by the European Union under the H2020 Programme Grant Agreement No. 830929 (CyberSec4Europe).

[3] https://www.ecommerceceo.com/customer-loyalty-programs

[4] https://www.businessinsider.de/best-loyalty-programs-stores-2019-4

are becoming more and more popular. Through such programs, a significant amount of data is generated and provided to the organization behind those incentive systems. While clearly most customers have to consent in providing their data to organizations via accepting the company's terms and conditions, undoubtedly sensitive data is processed in various and sometimes even undesired ways. One famous example is demonstrated by the retailer Target that hit the headlines in 2012 when they exposed a girl's pregnancy to her father with coupons sent by Target.[5] Given these facts and an ever increasing number or large-scale data breaches[6] — including large loyalty-card providers such as British Airways, Gap, Starbucks, and Target —, it is evident that privacy concerns of customers need to be seriously considered. A large-scale consumer survey conducted by COLLOQUY in 2017[7] among Americans and Canadians who participate in at least one loyalty program, shows that 49% of these people do not trust loyalty programs with their personal information. Although they still consent to those programs, it needs to be expected that there is a significant number of people who do not participate in such programs due to privacy concerns. This is supported by another study[8] which shows that around 24% of consumers say that they would not enroll in a loyalty program because it was a risk to their privacy.

There are various lines of research which investigate the design of privacy-preserving incentive systems (see Sect. 1.2 for a detailed discussion). In these systems, users generally store their own point counts (in authenticated form) as a single continually updated token. Users can interact with the incentive system provider to earn points (e.g., at supermarket checkout) or spend points (to reap rewards). To ensure privacy for users, the provider should not be able to recognize which user is earning or spending points in any particular transaction. On the other hand, the provider wants to ensure that users can only spend as many points as they have earned. In particular, users must not be able to use the same token twice to spend points (i.e., double-spend). Double-spending is usually prevented by embedding an identifier *dsid* into each token, which needs to be revealed when spending it. If the user tries to spend a token that he has already spent, the provider can recognize the *dsid* and reject the attempt. However, this means that all locations where points can be spent must be continually online and connected to a database to check *dsid*. This poses many practical challenges and limits applicability of cryptographic incentive systems. For this reason, modern systems support *offline double-spending protection*, meaning that if the provider accepts two transactions with the same *dsid* (e.g., because he is currently unable to check whether *dsid* has been used before), there is a mechanism in place that allows the provider to repair this: the provider can reveal the double-spending user's identity and reclaim any benefits he illegally gained from double-spending.

There are several works on privacy-preserving incentive systems [MDPD15,JR16,HHNR17,BBDE19]. Unfortunately, the schemes are limited in their applicability in practice (as discussed in Sect. 1.2). With our new construction, we improve upon the state-of-the-art of privacy-preserving incentive systems in the following ways:

- We significantly improve the efficiency of the Earn protocol (which is the most frequent operation). We do this by replacing costly zero-knowledge proofs with a short structure-preserving signature on equivalence classes [HS14,FHS19].
- When spending k out of v points from a token, the spend operation yields a remainder token for the remaining point count $v - k$. In case of double-spending, these remainder tokens must be dealt with by the provider. Otherwise, a user can effectively double his point count by double spending, receiving *two* remainder tokens for $v - k$ points. Prior work either was not able to trace these remainder tokens at all [JR16,HHNR17], or did so at the cost of losing backward unlinkability [BBDE19] (i.e., if a user

[5] https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html
[6] https://en.wikipedia.org/wiki/List_of_data_breaches
[7] https://www.the-cma.org/Contents/Item/Display/327325
[8] https://www.businessnewsdaily.com/4615-loyalty-programs-privacy-concerns.html

double-spends once, *all* past transactions can be traced by the provider). We provide a mechanism for remainder token tracing that is compatible with backward unlinkability.
- In practical scenarios, the spend protocol may be interrupted (e.g., through a technical fault). In prior work, this meant that users would lose all of their points, as retrying the transaction would be detected as double-spending. We present a recovery mechanism that can handle arbitrary faults (as well as malicious attempts at provoking faults).
- Prior work ensures that malicious users (and a dishonest provider) cannot blame honest users for double-spending. In practice, one also should require that a malicious user $\mathcal{B}$ cannot simply blame another malicious user $\mathcal{A}$ for double-spending conducted by $\mathcal{B}$. Without this property, malicious users would be incentivized to double-spend while avoiding any blame by simply blaming another party. We give a formal definition to model framing resistance for (non-colluding) malicious users. Our construction provides this property.

1.1 Our Construction

For the sake of exposition, we will explain the ideas behind our construction in several steps, starting with a basic version of an incentive system and working towards our full scheme and omit details the sake of simplicity of exposition.

A basic construction. In the most basic version (reminiscent of [JR16]), the user holds a commitment C to the user's current point total v (initially, $v = 0$), the user's secret usk, a token identifier $dsid$, and a random value $dsrnd$. Additionally, he holds a signature σ on C. To earn points, the user sends a randomized version C' of C to the provider (and proves possession of a corresponding signature). The provider homomorphically increments the point total v within C' by k points and signs the resulting commitment C'', which is a commitment to $(v + k, usk, dsid, dsrnd)$.

Spending points is more complicated because we need to account for double-spending. To spend k out of $v \geq k$ points, the user reveals the token identifier $dsid$ to the provider. If the provider is aware that the token with id $dsid$ has been spent before, he can immediately reject the attempt as double-spending. However, the provider may not be aware of $dsid$ if it has been spent before at an offline location. To cover such cases, the provider sends a random value $\gamma \leftarrow \mathbb{Z}_p$ and the user reveals $c = usk \cdot \gamma + dsrnd \mod p$ (explained below). The user then chooses fresh random values $dsid'$ and $dsrnd'$ and prepares a commitment C' to $(v - k, usk, dsid', dsrnd')$ for his remainder token. The user proves possession of his old signed commitment on $(v, usk, dsid, dsrnd)$, that all the sent values have been computed correctly, and that $v \geq k$. Then the provider signs the new token commitment C', which is on $(v - k, usk, dsid', dsrnd')$.

The idea is that a double spending user would be forced to reveal $c = usk \cdot \gamma + dsrnd$ in the first transaction and $c' = usk \cdot \gamma' + dsrnd$ in the second (with, likely, $\gamma \neq \gamma'$). c, γ and c', γ' allow the provider to compute usk, both revealing the identity of the double-spender and providing proof of guilt (as usk cannot be computed if no double-spending occurs). If the user does not double-spend, then usk is perfectly hidden within c (because $dsrnd$ is only used once), and every revealed $dsid$ is just some random identifier. We will call this the "Schnorr trick".

Through this mechanism, a double-spending user can be identified and punished for any time he double-spends a token. However, after double-spending a token with v points, the user holds two remainder tokens with $v - k$ points each. These remainder tokens can be used without being traceable to the double-spender, almost doubling his point value.

Tracing remainder tokens. To fix the remainder token problem from the previous construction, we augment the basic construction as follows (using ideas from [BBDE19]): When spending points, we now

Table 1. Comparison of incentive systems. We list the features (e.g., double-spending protection) and the security guarantees (soundness, anonymity, framing-resistance, backward unlinkability, and framing-resistance for non-colluding malicious users). *Earn protocol* highlights the main technique used: GS (Groth-Sahai NIZK), ZKP (zero-knowledge proof of knowledge based on Σ-protocols), SPS-EQ. "?" is used where the guarantee is not clear.

	features							security		
	online ds protection	offline ds protection	partial spending	remainder token tracing	recoverability	earn protocol	setting	sound. & anon. & fr.	backward unlink.	malicious fr.
BBA [JR16]	✓	✗	✗	✗	✗	GS	CRS	✓	✗	?
BBA+ [HHNR17]	✓	✓	✗	✗	✗	GS	CRS	✓	✓	?
BBW [HKRR19]	✓	✓	✓	✗	✗	ZKP	ROM	✓	✓	?
UACS [BBDE19]	✓	✓	✓	✓	✗	ZKP	CRS/ROM	✓	✗	?
This work	✓	✓	✓	✓	✓	SPS-EQ	ROM	✓	✓	✓

force the user to additionally reveal an encryption of $dsid'$ (which he chose for his remainder token) under usk. This way, if the user double-spends, the provider can extract usk as in the construction above and use it to obtain the remainder token's id $dsid'$. Then if $dsid'$ has also already been spent, he can again use usk to decrypt that transaction's remainder token id, and so on. Basically, this allows to provider to find all tokens that resulted directly or indirectly from a double-spending transaction.

Unfortunately, this modification also means that as soon as a user double-spends, the provider can use usk not only to decrypt encryptions of remainder tokens of double-spending transactions, but also *all other* transactions of that user. This means that as soon as a user double-spends, all past transactions also become traceable; the scheme does not have backward unlinkability.

Backward unlinkability. To add backwards-unlinkability to the previous construction (while still allowing the provider to trace "forwards"), we effectively create a chain of encryption keys instead of encrypting everything with usk. More specifically, we now interpret $dsid$ as the public key of an encryption scheme with secret key esk. The user's commitment now contains $(v, usk, esk, dsrnd_0, dsrnd_1)$, where esk replaces $dsid$ and we now have two $dsrnd$ values. Earning points is essentially unchanged (as it only touches v). When spending points, the user now chooses a fresh encryption key esk^* as well as random $dsrnd_0^*, dsrnd_1^*$ for his remainder token. He encrypts esk^* under the old token's $dsid$. He then reveals $c_0 = usk \cdot \gamma + dsrnd_0$ and $c_1 = esk \cdot \gamma + dsrnd_1$ for the provider's random challenge γ. Now if the user double-spends, the provider can compute the secret key esk for $dsid$ through the Schnorr trick and hence decrypt the remainder token's esk^*. Then esk^* can be used to decrypt the next esk^{**} from the remainder token and so on.

With this mechanism, the provider can still trace all remainder tokens resulting (directly or indirectly) from double-spending, but cannot use esk to decrypt any older ciphertexts. This preserves backward unlinkability, i.e., if users are corrupted (and, for example, double-spend as a result), their past transactions stay private (provided they discard the esks for old transactions).

Recovering failed spends. The previous construction already improves upon the state-of-the-art by combining remainder token traceability with backward unlinkability. However, assume that some spend transaction is aborted after c_0, c_1 have been sent, but before the user receives his signature on his new commitment. This can either happen accidentally (e.g., equipment failure), or maliciously. In this case, the user has effectively lost all his points, as re-trying to run the spend protocol (with the same token) would allow the provider to extract the user's secret. This is unacceptable for an incentive system in practice.

For this, we introduce a recovery functionality. We need to ensure security for the user (repeat recovery attempts do not threaten anonymity) and for the provider (users must not be able to abuse recovery to double-spend). First, for every spend transaction, the provider chooses a (unique) transaction id *tid*. In contrast to before, the challenge γ for the Schnorr trick is now deterministically derived as $\gamma = \mathcal{H}^{(\gamma)}(k, dsid, tid, C')$ using a (collision-resistant) hash function $\mathcal{H}^{(\gamma)}$. This means that for the same transaction *tid*, if the user spends the same amount k, reveals the same *dsid* and uses the same commitment C' for the remainder token, he will receive the same challenge γ. This prevents the provider from kicking off the mechanism to punish double-spending. As before, for *different* transactions *tid*′, the user will (likely) get a different challenge γ, which enables double-spending punishment. Furthermore, trying to change C' will (likely) result in a different challenge. This prevents the user from receiving a second/different remainder token from the retry (any attempt will reveal the user's identity and void the transaction using the mechanisms explained above).

Removing zero-knowledge proofs from Earn. When earning points, currently, the user needs to send a randomized version C' of his commitment C and then prove knowledge of a signature (in zero-knowledge). The user cannot simply send his signature to the provider in plain, as this would de-anonymize the user. To solve this, we need a signature scheme that (1) allows randomizing signatures and (2) allows randomizing messages C to C' without making the signature invalid. A perfect fit for this scenario are structure-preserving signatures on equivalence classes (SPS-EQ [HS14,FHS19]), for which a signature on a vector (h, g) of group elements can be adapted to a signature on (h^s, g^s) (in a sense, the signer actually signs all exponentiations of (h, g)).

For this, the user holds an SPS-EQ signature σ on the vector (C, g_1) for his (Pedersen) commitment $C = h_1^v \cdot h_2^{usk} \cdot h_3^{esk} \cdot h_4^{dsrnd_0} \cdot h_5^{dsrnd_1} \cdot h_6^r$ and some fixed element g_1. To earn points, the user randomizes (C, g_1) to (C^s, g_1^s), which looks random to the provider (assuming decisional Diffie-Hellman). He also adapts the signature σ to a signature σ' on (C^s, g_1^s). The provider can then, using g_1^s and knowledge of x such that $g_1^x = h_1$, add k to the v component of the commitment: $C^s \cdot ((g_1^s)^x)^k$ is a Pedersen commitment on $(v+k, \dots)$ (just raised to the power of s). The provider then signs $(C^s \cdot ((g_1^s)^x)^k, g_1^s)$ with the SPS-EQ scheme and sends the signature to the user. The user can transform this into a signature on $(C \cdot h_1^k, g_1) = (C', g_1)$, where C' is a Pedersen commitment on $v + k$.

This way of running the earn protocol means that the user simply sends two group elements plus a signature (which he can even prepare in advance) and receives a single signature in return, which is a significant improvement over the zero-knowledge proof that had to be conducted previously. However, the previous version of the earn protocol was much more convenient to prove secure: Zero-knowledge ensured anonymity for the user. The proof of knowledge property allowed a reduction to trace *every single* operation and the commitments, signatures, and openings involved, allowing it to immediately catch signature forgers or commitment breakers. Replacing the zero-knowledge proof of knowledge with an SPS-EQ introduces many technical challenges into the concrete construction (cf. Sect. 4) and its security proofs.

1.2 Related Work

Ideas on mechanisms similar to incentive systems have been pursued in various fields, for instance, in public-transport payment with refunds [RBHP15], privacy-preserving cloud-resource payments [Sla12,PSW12] or privacy-preserving multi-coupon systems [CES+05,CEL+07,CGH06,Ngu06]. We stress that these approaches essentially use pre-paid mechanisms, where in contrast to incentive systems, the spending is a much more frequent operation than earning/debt clearance. As those solutions do not match the requirements of incentive systems, we now discuss schemes that are more closely related to the collection of points (as done in incentive systems).

In the basic incentive system by Milutinovic et al. [MDPD15] points can neither be accumulated nor updated, and so the system induces storage and communication cost linear in the number of tokens. Moreover, double-spending protection is limited to online providers. In [JR16], Jager and Rupp introduce black-box accumulation (BBA) as a building block for incentive systems. BBA has been improved in two follow-up works to a system called BBA+ [HHNR17] by Hartung et al. and Black-box Wallet (BBW) by Hoffmann et al. [HKRR19]. BBA+ adds offline double-spending protection and the collection of negative points. Further, they introduce in BBA+ [HHNR17] the important and strong notion of forward and backward unlinkable transactions. While BBA and BBA+ use Groth-Sahai NIZK proofs [GS08] in all protocols, in BBW [HKRR19] the system is further optimized for smart-card usage. In contrast to our construction, the concurrent work BBW [HKRR19] aims at applications where the spend operation is frequent, e.g. subway turnstiles, and optimizes it using Bulletproofs [BBB$^+$18]. BBA+ has been further extended for electronic toll collection [HFN$^+$18] and vehicle-to-grid systems [SNF$^+$19], where in contrast to incentive systems, tokens are used to accumulate *debt* instead of points. We stress that BBA, BBA+, and BBW do not allow to partially spend credits, i.e., they lose all guarantees after a spend where credits remain. Our approach does not suffer from this problem, since we support remainder token tracing.

Blömer et al. [BBDE19] introduce a model for updatable anonymous credentials (UACS) and show how incentive systems can be generically constructed from UACS. They also provide an instantiation based on UACS obtained from Pointcheval-Sanders signatures [PS16]. Their scheme introduces a remainder token tracing, but their tracing mechanism rules out backward unlinkability. We take their definitions as a starting point but extend and strengthen it in various ways. In particular, we add the features recoverability, backward unlinkability, and framing-resistance for non-colluding malicious users.

To the best of our knowledge, no prior schemes considered recovery mechanisms or framing resistance for non-colluding malicious users. Furthermore, all prior work relied on zero-knowledge proofs for (in particular) the Earn protocol, which comes with a significant performance impact.

An overview of the features of directly related incentive systems is presented in Tab. 1. Since the definition of framing-resistance for non-colluding malicious users is introduced in this paper it is not clear if existing systems satisfy it.

Apart from these specific solutions, there are some classical well-known concepts that are clearly related, e.g., e-cash systems. However, modern approaches to e-cash such as compact e-cash [CHL05] are not suitable as once a double-spending occurs, *all* coins of the respective user can be traced. In our approach, however, we want old tokens to stay anonymous, but tokens spent starting from the double-spending have to be traceable (see remainder token tracing in Tab. 1).

In transferable e-cash systems [BCFK15,BCF$^+$11] or divisible e-cash systems [CPST15a,CPST15b,Oka95], tracing is only possible for individual coins, i.e., yields only double-spending detection, and, hence, does not provide the desired security (e.g. handling of tokens resulting from a double-spending) that we want to achieve. Moreover, most of the existing e-cash solutions are far too inefficient for our use cases.

In [CHK$^+$06], Camenisch et al. present glitch protection which preserves anonymity of users even if they use their one-time authentication token not more than n times, but all authentications with the same token are linkable. Users in [CHK$^+$06] can then be identified after $n + 1$ uses of the same token. Applied to our scenario this would lead to unsatisfactory guarantees, because then we essentially allow n double-spendings before we can identify the user to claim the losses. Our recovery feature (recoverability) compared to the glitch protection in [CHK$^+$06] also preserves anonymity in case something goes wrong, e.g., hardware failure, but unlike the protection in [CHK$^+$06] our goal is to bring the user back into the system and we cannot consider double-spending as a glitch, rather we consider failure during the spend process as a glitch. The additional tracing feature presented in [CHK$^+$06] enables the provider to identify all possible authentication tokens of a user after the user reuses one token. This is similar to the all-or-nothing

unlinkability provided in [BBDE19] where all users' tokens can be linked after one double-spending. Our backward unlinkability still guarantees user anonymity for tokens issued before the double-spending. The backward unlinkability and recoverability that our scheme provides are new features that can be added to the elaborate list presented in [CHK+06] and improves upon the work in [BBDE19].

2 Building Blocks

Notation. For $n \in \mathbb{N}$, let $[n] := \{1, \ldots, n\}$, and let $\lambda \in \mathbb{N}$ be the security parameter. For a finite set $\mathcal{S}$, we denote by $s \leftarrow \mathcal{S}$ the process of sampling s uniformly from $\mathcal{S}$. For an algorithm A, let $y \leftarrow A(x)$ be the process of running A on input x with access to uniformly random coins and assigning the result to y.

PRF. We denote a pseudorandom function PRF as follows: $\beta \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$ generates a key. For a finite set S and a bit string $x \in \{0,1\}^*$, we denote the evaluation as $y = \mathsf{PRF.Eval}_\beta^S(x)$, which outputs some $y \in S$. As usual, no PPT adversary should be able to distinguish evaluations of the PRF from a truly random function.

SPS-EQ. Structure-preserving signatures on equivalence classes ($\mathsf{SPS\text{-}EQ}$) [HS14,FHS19] sign messages-vectors of length ℓ from one of the prime order p source groups $\mathbb{G}_i$ ($i \in \{1,2\}$) of a bilinear group and $\ell > 1$. An $\mathsf{SPS\text{-}EQ}$-scheme signs equivalence classes $[M]$ of vectors $M \in (\mathbb{G}_i^*)^\ell$ with equivalence relation: $M, N \in \mathbb{G}_i^\ell : M \sim_{\mathcal{R}} N \Leftrightarrow \exists\, s \in \mathbb{Z}_p^* : M = N^s$.

An $\mathsf{SPS\text{-}EQ}$ scheme now signs an equivalence class $[M]$ by signing a representative M of $[M]$ and given a signature for $[M]$ one can publicly derive a signature for each representative. More precisely, given the description of a bilinear group BG and a length parameter ℓ, KeyGen outputs a key pair (pk, sk) and Sign on input a message $M \in (\mathbb{G}_i^*)^\ell$, which is a representative of the class $[M]$, and a secret key sk outputs a signature. The algorithm ChgRep on input an representative M of $[M]$, a corresponding signature σ, an exponent $\mu \in \mathbb{Z}_p^*$ and a public key pk outputs a pair $M' = M^\mu$ and σ' where σ' is a valid signature for the representative M^μ. The verification algorithm on input a message M, a signature σ and a public key pk outputs a bit indicating whether the signature is valid. Additionally, there is a VKey algorithm that on input sk and pk outputs a bit indicating whether the key pair is valid. Unforgeability is defined analogous to digital signatures, but for equivalence classes instead of messages. One of the goals of $\mathsf{SPS\text{-}EQ}$ is to guarantee that message-signature pairs output by Sign and ChgRep are identically distributed and that it is hard to distinguish different equivalence classes, which yields a type of unlinkability. For more details, consult Appendix B.1.

Zero-Knowledge PoKs. Let $\mathcal{R}$ be a witness relation for some NP language $\mathcal{L}_\mathcal{R} = \{x \mid \exists w \text{ s.t. } (x,w) \in \mathcal{R}\}$. A zero-knowledge proof of knowledge is a a proof system that allows a prover P to convince a verifier V that it knows a witness w to some word $x \in \mathcal{L}_\mathcal{R}$, i.e., such that $(x,w) \in \mathcal{L}_\mathcal{R}$, without revealing anything beyond this fact and in particular no information about w to V. Consider, for example, a prime order p group $(\mathbb{G}, p, g)$ with fixed generator g and let us define $(x,w) \in \mathcal{R}$ if $g^w = x$. In our notation a zero-knowledge proof of knowledge for this relation is denoted as $\mathsf{ZKP}[(w) :\ g^w = x]$. For more details, consult Appendix B.2.

3 Definition of Incentive Systems

In this section, we formally define incentive systems, their syntax, and their security. We start with a high-level description of an incentive system for point collection. Our definitions are based on [BBDE19], which we extend with a recovery feature and completely new security definitions for *backward unlinkability* and *framing-resistance for non-colluding malicious users.*

3.1 Syntax Definition of Incentive Systems

In a nutshell, after having set up the system parameters, the *provider* and the *users* generate their private key material. In order to join the system, the user engages in an *issuance* protocol with the provider, where he receives an initial token initialized with 0 points. After this, the user can *earn* and *spend* points through interactions with the provider, after which she receives updated tokens for the new balance. To increase fault tolerance, the *spend* protocol can be run repeatedly with the same input without it being interpreted as double-spending or losing points; this is useful, e.g., if the spend-protocol aborts when removing the user device from the terminal too early, or in case that the user does not receive the announced benefit from the provider after having spent the points. Transaction ids *tid* are used to differentiate repeat attempts from new transactions. Finally, *double-spending protection*, in particular required when considering offline spending, is achieved through linking and verification algorithms; furthermore, *tracing* of remainder tokens that are a direct or indirect result of double-spending is supported. More details are also given below.

Definition 1. *An incentive system consists of the following PPT algorithms:*

$\mathsf{Setup}(1^\lambda) \to pp$ *generates public parameters* pp*. We assume that some maximum point score* $v_{max} \in \mathbb{N}$ *and a bounded maximum number* $k_{max} \in \mathbb{N}$ *of points a user can earn at once are (implicitly) encoded in* pp *(we assume both limits to be large enough never to be hit in practice).*

$\mathsf{ProviderKeyGen}(pp) \to (pk, sk)$ *generates a key pair* (pk, sk) *of an provider.*

$\mathsf{KeyGen}(pp, pk) \to (upk, USK)$ *generates a user's public key* upk *and secret key* USK *for the given provider.*

$\mathsf{Issue}(pp, pk, upk, sk) \leftrightarrow \mathsf{Join}(pp, pk, upk, USK) \to (token, dsid)$ *is an interactive protocol with common input* pp *and public keys* pk, upk*. The provider gets its secret key* sk *as private input and the receiver gets its user secret key* USK *as private input. After the protocol, the receiver outputs a token* $token$ *with* 0 *points and a double-spend id* $dsid$*, or the failure symbol* $\perp$*.*

$\mathsf{Credit}(pp, pk, k, sk) \leftrightarrow \mathsf{Earn}(pp, pk, k, USK, token) \to token^*$ *is an interactive protocol with common input* pp*, the provider's* pk*, and earn amount* $k \in \mathbb{N}$*. As private input, the provider gets* sk *and the receiver gets its key* USK *and token-to-be-updated* $token$*. The earner outputs an updated token* $token^*$ *or the failure symbol* $\perp$*.*

$(token^*, dsid^*) \leftarrow \mathsf{Spend}(pp, pk, k, dsid, tid, USK, token) \leftrightarrow \mathsf{Deduct}(pp, pk, k, dsid, tid, sk) \to (b, dstag, \gamma)$ *is an interactive protocol with common input* pp*, the provider's* pk*, spend amount* $k \in \mathbb{N}$*, a double-spend id* $dsid$*, and a (fresh, unique) transaction ID* tid*. The spender gets its key* USK *and token* $token$ *as private input. The provider outputs a bit* b*, a double-spend tag* $dstag$*, and a transaction tag* γ*. The spender outputs a token* $token^*$ *and its* $dsid^*$ *or an error symbol.*

$\mathsf{Link}(pp, pk, dstag, dstag') \to (upk, dsblame, dstrace)$ *given two double-spend tags* $dstag, dstag'$*, deterministically outputs the spending user's* upk*, double-spending blaming information* $dsblame$*, and tracing information* $dstrace$*, or* $\perp$*.*

$\mathsf{VerifyDs}(pp, dsblame, upk) \to b$ *given* $dsblame$ *and* upk*, outputs a bit* b *indicating whether* $dsblame$ *proves that* upk *has double-spent.*

$\mathsf{Trace}(pp, dstrace, dstag) \to (dsid^*, dstrace^*)$ *given* $dstrace$ *and* $dstag$*, deterministically outputs the remainder token's* $dsid^*$ *and* $dstrace^*$*.*

Double-spending protection is inspired by [BBDE19]. The provider sets up a central database $\mathcal{DB}$. This database is a directed bipartite graph, which contains (1) one token node *dsid* for each *dsid* the provider has seen, and (2) one transaction node (tid, γ) for a corresponding successful spend process with id *tid* and transaction tag γ. Two Spend protocol runs with the same *tid* and same γ are considered to be attempts of the same transaction and only one of them appears in the database. Edges establish known consume/produce relations for transactions: every transaction (tid, γ) effectively consumes exactly one

$\mathsf{DBsync}(tid, \gamma, dsid, dstag, k, \mathcal{DB})$:

- If (tid, γ) is not yet in the database, add new spend operation node (tid, γ) to $\mathcal{DB}$, associate it with k and $dstag$.
- If $dsid$ is not in $\mathcal{DB}$, add the node $dsid$ and an edge from $dsid$ to (tid, γ).
- Otherwise, add the edge from $dsid$ to (tid, γ), and:
  - If $dsid$ has no $(upk, dsblame, dstrace)$ associated with it, then there exist two outgoing edges from $dsid$ to transactions $(tid, \gamma), (tid', \gamma')$. In this case, compute $(upk, dsblame, dstrace) = \mathsf{Link}(pp, pk, dstag, dstag')$ using the two tags $dstag$, $dstag'$ associated with (tid, γ) and (tid', γ'), respectively. Associate $(upk, dsblame, dstrace)$ with $dsid$.
  - Mark (tid, γ) invalid (this triggers the steps below).
- Whenever some node (tid, γ) with incoming edge from some $dsid$ is marked invalid
  - Use record $(upk, dsblame, dstrace)$ associated with $dsid$ and $dstag$ associated to tuple (tid, γ) to compute $(dsid^*, dstrace^*) = \mathsf{Trace}(pp, dstrace, dstag)$. Add $dsid^*$ to the graph (if it does not already exist), associate $(upk, dsblame, dstrace^*)$ with $dsid^*$, and add an edge from (tid, γ) to $dsid^*$. If there is an edge from $dsid^*$ to some (tid', γ'), mark all such (tid', γ') invalid (if they were not already marked). This triggers this routine again.

Fig. 1. DBsync algorithm

$dsid$ (which the user reveals), inducing an edge $dsid \to (tid, \gamma)$. If double-spending occurs, Trace may uncover the remainder token's $dsid^*$ produced by (tid, γ), in which case the graph would contain an edge $(tid, \gamma) \to dsid^*$.

Providers periodically send their observed transactions (tid, γ) (together with their k, $dsid$, and $dstag$) to the central database. If $dsid$ is not yet in the database, the database simply adds $dsid \to (tid, \gamma)$ to the graph. If $dsid$ was already in the database, then it has already been spent in a transaction (tid', γ'). tid is marked invalid and $\mathsf{Link}, \mathsf{Trace}$ are used to find (tid, γ)'s successor $dsid^*$. If $dsid^*$ is already in the database, any transaction descendants of $dsid^*$ are marked invalid and the process repeats. The exact algorithm $\mathsf{DBsync}(tid, \gamma, dsid, dstag, k, \mathcal{DB})$ to add a transaction to the graph is given in Fig. 1.

Correctness. We omit a formal definition of correctness here. In a nutshell, the property should guarantee that no algorithm or protocol will terminate in an error state if all participants take honestly generated inputs and follow the protocol specification.

3.2 Security Definitions

To define security for incentive systems, we first define several stateful oracles, a selection of which will be available to an adversary $\mathcal{A}$ in each of the subsequent security games. For these definitions, we assume that pp has been generated honestly and is available to each of the oracles. Oracles modeling the behavior of honest users are presented in Fig. 2, and those for honest providers in Fig. 3. Note that upon failing to spend a token, users will enter a state in which they must retry the exact same spend operation again. This models recovery of failed spend protocols in practice (where a user would also retry). It is noteworthy that users will also consider any spend operation failed if the provider claims that it has failed, allowing a corrupt provider to force users to retry transactions.

We next define anonymity, backward unlinkability, framing resistance, as well as soundness for incentive systems.

Anonymity. Anonymity guarantees that an adversary controlling the provider cannot link a specific transaction to a specific user. More precisely, we let the adversary arbitrarily interact with honest users, mimicked by the oracles defined before. Then, in the challenge phase, the adversary may trigger one further

RegisterKeys(pk):

– This interface must be called before any other honest-user oracles, and may only be called a single time. It registers the provider's public key pk for further use.

Keygen():

– It chooses a fresh user handle u, runs $(upk, USK) \leftarrow \mathsf{KeyGen}(pp, pk)$ and then stores user state $(upk_u, USK_u, v_u, token_u, dsid_u, mustRetry_u) \leftarrow (upk, USK, 0, \bot, \bot, \bot)$. It outputs (u, upk).

Join(u):

– Given user handle u, the oracle runs $(token, dsid) \leftarrow \mathsf{Join}(pp, pk, upk_u, USK_u)$. If $token = \bot$, the oracle outputs $\bot$. Otherwise, it stores $token_u \leftarrow token$, and $dsid_u \leftarrow dsid$. This oracle can only be called once for each u. It must be called before any calls to **Earn**$(u, \cdot)$ and **Spend**$(u, \cdot, \cdot)$.

Earn(u, k) :

– This oracle ignores the call if $token_u = \bot$, $k > k_{max}$, $v_u + k > v_{max}$, or $mustRetry_u \neq \bot$. Otherwise, the oracle runs $token^* \leftarrow \mathsf{Earn}(pp, pk, k, USK_u, token_u)$. If $token^* = \bot$, the oracle outputs $\bot$. Otherwise, it updates $token_u \leftarrow token^*$ and $v_u \leftarrow v_u + k$.

Spend(u, k, tid) :

– This oracle ignores the call if $token_u = \bot$, $v_u < k$, or $mustRetry_u = (k', tid') \neq (k, tid)$ holds. The oracle sends $dsid_u$ to its communication partner. It then runs $(token^*, dsid^*) \leftarrow \mathsf{Spend}(pp, pk, k, dsid_u, tid, USK_u, token_u)$. If the output of Spend is $\bot$ or the provider sends $b = 0$, it sets $mustRetry_u = (k, tid)$ and outputs 0. Otherwise, it updates $(v_u, token_u, dsid_u) \leftarrow (v_u - k, token^*, dsid^*)$ and $mustRetry_u = \bot$.

Fig. 2. Honest user oracles

ProviderKeyGen() :

– This oracle runs $(pk, sk) \leftarrow \mathsf{ProviderKeyGen}(pp)$. It stores pk and sk for further use. It initially sets the set of users $\mathcal{U} \leftarrow \emptyset$ and sets the double-spend database $\mathcal{DB}$ to the empty graph. Furthermore, initially $v_{\text{earned}}, v_{\text{spent}} \leftarrow 0$. Further calls to this oracle are ignored. This oracle must be called before any of the other provider-related oracles. The oracle outputs pk.

Issue(upk):

– If $upk \in \mathcal{U}$, the request is ignored. Otherwise, the oracle runs $\mathsf{Issue}(pp, pk, upk, sk)$ and adds upk to $\mathcal{U}$.

Credit(k):

– On input $k \leq k_{max}$, this oracle runs $\mathsf{Credit}(pp, pk, k, sk)$ and increments $v_{\text{earned}} \leftarrow v_{\text{earned}} + k$.

Deduct(k, tid):

– The oracle waits to receive $dsid$. Afterwards, it runs $\mathsf{Deduct}(pp, pk, k, dsid, tid, sk) \rightarrow (b, dstag, \gamma)$. It sends b to the user. If $b = 0$, it outputs $\bot$. Otherwise, it chooses a fresh token spend handle s and stores $(tid_s, \gamma_s, dsid_s, dstag_s, k_s) \leftarrow (tid, \gamma, dsid, dstag, k)$, then it outputs s. If this is the first time that $\mathsf{Deduct}(\cdot, \cdot, \cdot, \cdot, tid, \cdot)$ has output $b = 1$, update $v_{\text{spent}} \leftarrow v_{\text{spent}} + k$.

DBsync(s):

– This interface runs $\mathcal{DB}' \leftarrow \mathsf{DBsync}((tid_s, \gamma_s, dsid_s, dstag_s, k_s), \mathcal{DB})$. It then updates $\mathcal{DB} \leftarrow \mathcal{DB}'$. Let v_{invalid} be the sum of values k associated with invalid transactions within $\mathcal{DB}'$.

Fig. 3. Honest provider oracles

interaction. The adversary wins if he correctly guesses which user it was interacting with. To rule out trivial distinguishers, both users must have valid tokens and sufficient points to spend/must not hit the v_{max} limit through Earn. Furthermore, the users must not be in the *mustRetry* error state (multiple tries of the same user for the same transaction are linkable).

Definition 2 (Anonymity). *An incentive system Π is* anonymous *iff for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that for all λ it holds that* $\Pr[\mathrm{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq 1/2 + negl(\lambda)$, *where* $\mathrm{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda)$ *is as defined in Fig. 4.*

Backward-unlinkability. This property guarantees that a malicious provider cannot link past transactions to a user even if all of the user's long-term local information (key material, token) are leaked to the provider. This is reminiscent of forward secrecy for key exchange: even if long-term secrets are leaked, secrecy is still protected for operations that occurred *before* the leakage.

To model this, we imagine two honest users u_0, u_1 who run transactions with a corrupted provider. Eventually, both users are corrupted, their states are leaked to the provider. Then the provider should not be able to tell which user state belongs to which of the two users.

Experiment $\mathrm{Exp}^{\mathsf{anon}}_{\mathcal{A},\Pi}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$, $(pk, st) \leftarrow \mathcal{A}(pp)$
$\mathbf{RegisterKeys}(pk)$, $(u_0, upk_0), (u_1, upk_1) \leftarrow \mathbf{Keygen}()$
$st \leftarrow \mathcal{A}^{\mathbf{Join}(\cdot),\mathbf{Earn}(\cdot,\cdot),\mathbf{Spend}(\cdot,\cdot,\cdot)}(st, u_0, upk_0, u_1, upk_1)$
$b' \leftarrow \mathcal{A}^{\mathbf{Earn}(u_b,\cdot),\mathbf{Spend}(u_b,\cdot,\cdot)}(st)$
where $\mathcal{A}$ is restricted to a single oracle query, and:
If $\mathcal{A}$ calls $\mathbf{Earn}(u_b, \cdot)$ on input k,
if $\max\{v_{u_0}, v_{u_1}\} > v_{max}$ or $\perp \in \{token_{u_0}, token_{u_1}\}$
or $(mustRetry_{u_0}, mustRetry_{u_1}) \neq (\perp, \perp)$, output 0.
If $\mathcal{A}$ calls $\mathbf{Spend}(u_b, \cdot, \cdot)$ on input k, tid,
if $\min\{v_{u_0}, v_{u_1}\} < k$ or $\perp \in \{token_{u_0}, token_{u_1}\}$
or $(mustRetry_{u_0}, mustRetry_{u_1}) \neq (\perp, \perp)$, output 0.
return 1 iff $b = b'$

Fig. 4. Anonymity

Experiment $\mathrm{Exp}^{\mathsf{bu}}_{\mathcal{A},\Pi}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$, $(pk, st) \leftarrow \mathcal{A}(pp)$
$\mathbf{RegisterKeys}(pk)$, $(u_0, upk_0), (u_1, upk_1) \leftarrow \mathbf{Keygen}()$
$st \leftarrow \mathcal{A}^{\mathsf{oracles}}(st, u_0, upk_0, u_1, upk_1)$
where $\mathsf{oracles} = (\mathbf{Join}(\cdot), \mathbf{Earn}(u_b, \cdot), \mathbf{Earn}(u_{1-b}, \cdot),$
$\mathbf{Spend}(u_b, \cdot, \cdot), \mathbf{Spend}(u_{1-b}, \cdot, \cdot)$
$b' \leftarrow \mathcal{A}(st, USK_{u_0}, token_{u_0}, USK_{u_1}, token_{u_1})$
return 1 iff $b = b'$ and $v_{u_0} = v_{u_1}$ and $mustRetry_{u_0} = mustRetry_{u_1} = \perp$.

Fig. 5. Backward unlinkability

For the formal definition, we choose an equivalent formulation (which lends itself better to the security proofs): the provider does not know which user he interacts with in the first phase (he can interact with u_b and u_{1-b}, but does not know b). Then he receives the secrets of u_0 and u_1 (knowing which secrets belong to u_0 and u_1, respectively) and needs to decide which of the oracles belongs to which secrets (i.e., guess b). To exclude trivial distinguishers, the balances of the two users must be the same when their token is leaked and they must not currently be in a *mustRetry* state. The latter restriction is because if the user is stuck in a *mustRetry* state, he holds on to a token with a *dsid* he has already used, allowing the provider to easily link the user to his previous spend attempt. In practice, this means that if the most recently spend transaction has failed, it may be linked to the user upon corruption (but all other transactions remain unlinkable).

Definition 3 (Backward-unlinkability). *An incentive system Π is* backwards unlinkable *iff for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that for all λ it holds that* $\Pr[\mathrm{Exp}^{\mathsf{bu}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq 1/2 + negl(\lambda)$, *where* $\mathrm{Exp}^{\mathsf{bu}}_{\mathcal{A},\Pi}(\lambda)$ *is as defined in Fig. 5.*

Framing-resistance. Informally, framing-resistance guarantees that an honest user cannot be blamed for double-spending. To model this, the adversary plays the role of a corrupt provider, interacting with an honest user. Note that by definition, honest users do not perform double-spending (cf. Fig. 2). The adversary outputs some double-spending blaming information *dsblame*. He wins if *dsblame* is accepted as valid proof of double-spending.

Experiment $\mathrm{Exp}^{\mathsf{fr}}_{\mathcal{A},\Pi}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $(pk, st) \leftarrow \mathcal{A}(pp)$, $\mathbf{RegisterKeys}(pk)$
$(dsblame, u) \leftarrow \mathcal{A}^{\mathbf{Keygen}(),\mathbf{Join}(\cdot),\mathbf{Earn}(\cdot,\cdot),\mathbf{Spend}(\cdot,\cdot,\cdot)}(st)$
return 1 iff $\mathsf{VerifyDs}(pp, dsblame, upk_u) = 1$.

Fig. 6. Framing resistance

Experiment $\mathrm{Exp}^{\mathrm{soundness}}_{\mathcal{A},\Pi}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $pk \leftarrow \mathbf{ProviderKeyGen}()$, $\mathbf{RegisterKeys}(pk)$
$\mathcal{A}^{\mathbf{Issue}(\cdot),\mathbf{Credit}(\cdot),\mathbf{Deduct}(\cdot,\cdot),\mathbf{DBsync}(\cdot)}(pp, pk)$
return 1 if $v_{\mathrm{spent}} - v_{\mathrm{invalid}} > v_{\mathrm{earned}}$
    and $\mathcal{A}$ has queried $\mathbf{DBsync}(\cdot)$ for all spending record handles s output by $\mathbf{Deduct}$
return 1 if $\exists (upk, dsblame) \in \mathcal{DB}$ associated with some $dsid$ such that $\mathsf{VerifyDs}(pp, dsblame, upk) \neq 1$ or $upk \notin \mathcal{U}$
return 0

Fig. 7. Soundness

Definition 4 (Framing-resistance). *An incentive system Π is* framing resistant *iff for every PPT adversary $\mathcal{A}$ there exists a negligible function $negl$ such that for all λ it holds that* $\Pr[\mathrm{Exp}^{\mathsf{fr}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq negl(\lambda)$, *where* $\mathrm{Exp}^{\mathsf{fr}}_{\mathcal{A},\Pi}(\lambda)$ *is as defined in Fig. 6.*

Soundness. Soundness protects an honest provider against malicious users. On the one hand, it guarantees that users can spend at most as many points as they have previously earned. On the other hand, the property guarantees that double-spending can be detected and traced to a valid user.

To model this property, we allow the adversary to interact with honest-provider oracles in the role of corrupted users. We also give the adversary control over the order that double-spending information is synchronized into the central database. In the end, the adversary wins if some double-spending transaction cannot be traced to one of the corrupted users, or if the adversary was able to spend more points than he was credited by the provider. For this count of spent points, we do not count spending operations that have been detected as double-spending – the idea here is that as soon as the provider detects double-spending operations, he will recoup any losses from the blamed party.

Definition 5 (Soundness). *An incentive system Π is* sound *iff for every PPT adversary $\mathcal{A}$ there exists a negligible function $negl$ such that for all λ it holds that* $\Pr[\mathrm{Exp}^{\mathrm{soundness}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq negl(\lambda)$, *where* $\mathrm{Exp}^{\mathrm{soundness}}_{\mathcal{A},\Pi}(\lambda)$ *is as defined in Fig. 7.*

Framing-resistance for non-colluding malicious users. Via Definition 5, we have ensured that a malicious user cannot frame an honest user for double-spending. One may also want to ensure that *malicious* users cannot frame another *malicious* user for double-spending. This is because if a user $\mathcal{B}$ is able to frame some user $\mathcal{A}$, this would effectively allow $\mathcal{B}$ to circumvent the tracing mechanism, regardless of whether $\mathcal{A}$ behaves honestly or not.

Note that for colluding users, this requirement can never be fulfilled. If $\mathcal{A}$ colludes with $\mathcal{B}$, it may share all his cryptographic secrets with $\mathcal{B}$. Then obviously, $\mathcal{B}$ will be able to make it look as if $\mathcal{B}$'s actions were conducted by $\mathcal{A}$. Since we cannot possibly prevent colluding users from using each others' secrets, we clarify our security requirement to "*non-colluding* malicious users should not be able to frame one another for

Experiment $\mathrm{Exp}^{\mathsf{adv\text{-}fr}}_{\mathcal{A},\mathcal{B},\Pi}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$, $pk \leftarrow$ **ProviderKeyGen**(), **RegisterKeys**(pk)
$(st, upk_{\mathcal{A}}) \leftarrow \mathcal{A}(pp, pk)$
Run $st' \leftarrow \mathcal{A}^{\mathbf{Issue}(upk_{\mathcal{A}})}(st)$ (require that $\mathcal{A}$ queries the oracle exactly once)
Run $\mathcal{A}^{\mathbf{Credit}(\cdot),\mathbf{Deduct}(\cdot,\cdot),\mathbf{DBsync}(\cdot)}(pp, pk, st')$ and
$\mathcal{B}^{\mathbf{Issue}(\cdot),\mathbf{Credit}(\cdot),\mathbf{Deduct}(\cdot,\cdot),\mathbf{DBsync}(\cdot)}(pp, pk, upk_{\mathcal{A}})$ in parallel
(note that $\mathcal{A}$ and $\mathcal{B}$ cannot communicate with each other, but their oracles work on the same state).
return 1 iff there is transaction tid with predecessor $dsid$ in $\mathcal{DB}$ such that
$dsid$ is associated with $upk_{\mathcal{A}}$ but
the transaction id tid does not belong to one of $\mathcal{A}$'s **Deduct** queries.

Fig. 8. Framing-resistance for non-colluding malicious users

their actions". This is still a nontrivial property to define (essentially, we are protecting some malicious user from another malicious user). In this scenario, we require the provider to behave honestly (in reality, this would be enforced through legal means).

Formally, we model this property as a game between two malicious users $\mathcal{A}$ and $\mathcal{B}$ and an honest provider. $\mathcal{A}$ and $\mathcal{B}$ may both interact with the provider (through the usual oracles), but cannot interact with one another. During the experiment, we associate each transaction id (tid, γ) to either $\mathcal{A}$ or $\mathcal{B}$, depending on who issued the corresponding **Deduct** query. $\mathcal{B}$ wins if the provider's database $\mathcal{DB}$ contains some transaction (tid, γ) that does not belong to $\mathcal{A}$ but that has been traced to $\mathcal{A}$.

As a technical restriction, $\mathcal{A}$ may only use the **Issue**(upk) oracle once, as it returns the error symbol if it has already been called for upk. Without this restriction, the oracle could be used as a covert channel for $\mathcal{A}$ and $\mathcal{B}$ to communicate (to communicate a bit b_ℓ to $\mathcal{B}$, $\mathcal{A}$ would query **Issue**(ℓ) iff $b_\ell = 1$, then $\mathcal{B}$ can query **Issue**(ℓ) afterwards to check for the error symbol).

We stress that the *framing-resistance for non-colluding malicious users* property neither implies (normal) framing resistance as in Definition 4 (since in Definition 4, the provider is not assumed to be honest), nor vice versa (since in Definition 4, the relevant users are assumed to be honest).

Definition 6 (Framing-resistance for non-colluding malicious users). *An incentive system Π has framing-resistance for non-colluding malicious users iff for every pair of PPT adversaries $(\mathcal{A}, \mathcal{B})$, there exists a negligible function $negl$ such that for all λ it holds that* $\Pr[\mathrm{Exp}^{\mathsf{adv\text{-}fr}}_{\mathcal{A},\mathcal{B},\Pi}(\lambda) = 1] \leq negl(\lambda)$, *where* $\mathrm{Exp}^{\mathsf{adv\text{-}fr}}_{\mathcal{A},\mathcal{B},\Pi}(\lambda)$ *is as defined in Fig. 8.*

4 Instantiation

In this section we present a concrete and practical construction. It is particularly interesting from an efficiency perspective, as the operation that is most frequently conducted in such a system, i.e., the Earn operation run by the user, does not require zero-knowledge proofs. This is achieved by the use of SPS-EQ and some tricks to integrate them with other building blocks.

Before presenting the formal construction (see Alg. 1 to 4) we present the intuition behind it. As a central building block we use an SPS-EQ scheme, which we use in a way that is inspired by their use to construct round-optimal blind signatures in [FHS15], but significantly extended to handle updates. More precisely, the key of the provider represents an SPS-EQ key for vectors of length $\ell = 2$ and the commitment parameters $(h_1, \ldots, h_6)$ of a generalized Pedersen commitment, where the trapdoors $(q_1, \ldots, q_6)$ of the generalized Pedersen commitment are contained in the secret key (latter is important for our update

functionality). Furthermore, we obtain another base h_7 for the generalized Pedersen commitment via a random oracle $\mathcal{H}$, which allows us to equivocate the commitment in our security proofs. Every user possesses a unique key $upk = w^{usk}$ and how the key is used in our construction is reminiscent of signatures with efficient protocols [CL04,PS16], where users include their secret key into a commitment which is signed in an interactive protocol in a blind way. Both the user and the provider set up respective keys for a PRF to derive randomness for the spend protocol.

A user token is a Pedersen commitment $C = h_1^{usk} \cdot h_2^{esk} \cdot h_3^{dsrnd_0} \cdot h_4^{dsrnd_1} \cdot h_5^{v} \cdot h_6^{z} \cdot h_7^{t}$ and an SPS-EQ signature σ on (C, g_1), where usk is the user's secret key, esk is a secret encryption key (and, at the same time, the corresponding public key $dsid = w^{esk}$ is a unique identifier for the token), $dsrnd_0, dsrnd_1$ are random values to use with the Schnorr trick when spending the token, v is the token's current point count, and z, t are randomization values for the commitment. To ensure that esk is actually unique, whenever a new token is created (during Issue or Deduct), both the user and the provider contribute randomness to it, i.e., the user chooses a random share esk_{usr} and the provider contributes random share esk_{prov}. Overall, $esk = esk_{\text{usr}} + esk_{\text{prov}}$.

Join. The very basic idea is that a user when joining the system forms a commitment $C = h_1^{usk} \cdot h_2^{esk_{\text{usr}}} \cdot h_3^{dsrnd_0} \cdot h_4^{dsrnd_1} \cdot h_6^{z} \cdot h_7^{t}$ (this expression implicitly contains h_5^0, i.e. the user starts with $v = 0$ points). Now the user assembles (C, g_1) forming an equivalence class for the $\mathsf{SPS\text{-}EQ}$ scheme and switches it to random representative $C^{(\text{pre})} = (C^u, g_1^u)$ for $u \leftarrow \mathbb{Z}_p^*$ and sends it to the provider. Simultaneously, it provides a zero-knowledge proof of well-formedness of the sent vector.

The provider can now use its knowledge of the trapdoor q_2 in the secret key to update the Pedersen commitment C^u in $C^{(\text{pre})}$ homomorphically without knowing the opening and produces $C'^u := C^u \cdot (g_1^u)^{q_2 esk_{\text{prov}}}$, contributing his share esk_{prov} to the encryption key $esk = esk_{\text{usr}} + esk_{\text{prov}}$. Then it signs (C'^u, g_1^u) using the $\mathsf{SPS\text{-}EQ}$ and sends the signature σ to the user, who checks well-formedness, verifies the $\mathsf{SPS\text{-}EQ}$ signature σ and changes the representative to an unlinkable version $(C^*, \sigma^*) = ((C', g_1), \sigma')$. In our intuitive description, we omit the opening information in the tokens for brevity.

Earn. Earn is a very simple protocol. A user who holds a token for v points and earns k points, takes his pair $((C', g_1), \sigma')$ from his token, changes the representative obtaining $((C'^s, g_1^s), \sigma'')$ for $s \leftarrow \mathbb{Z}_p^*$ and sends it to the provider, who verifies the $\mathsf{SPS\text{-}EQ}$ signature. It then updates the commitment to $C''^s := C'^s \cdot (g_1^s)^{q_5 k}$ which adds k points to the user's token. Then it issues a fresh signature σ'' on (C''^s, g_1^s), which the user verifies and then unblinds, obtaining the new token $(C^*, \sigma^*) = ((C'', g_1), \sigma''')$ for $v + k$ points.

Spend. The user holds a token with commitment $C = h_1^{usk} \cdot h_2^{esk} \cdot h_3^{dsrnd_0} \cdot h_4^{dsrnd_1} \cdot h_5^{v} \cdot h_6^{z} \cdot h_7^{t}$ for v points and wants to spend $k \leq v$ points. The user sends C and the signature on (C, g_1) to the provider. He also forms a commitment $h_1^{usk} \cdot h_2^{esk^*_{\text{usr}}} \cdot h_3^{dsrnd_0^*} \cdot h_4^{dsrnd_1^*} h_5^{v-k} \cdot h_6^{z^*} \cdot h_7^{t^*}$ for a fresh remainder token with $v-k$ points. Similarly to Join, he sends a random representative $C^{(\text{pre})} = ((h_1^{usk} \cdot h_2^{esk^*_{\text{usr}}} \cdot h_3^{dsrnd_0^*} \cdot h_4^{dsrnd_1^*} h_5^{v-k} \cdot h_6^{z^*} \cdot h_7^{t^*})^{u^*}, g_1^{u^*})$ to the provider. The user derives the values $esk^*_{\text{usr}}, dsrnd_0^*, dsrnd_1^*, z^*, t^*, u^*$ from a pseudorandom function $\mathsf{PRF.Eval}_{\beta_{usr}}^{\mathbb{Z}_p^6}(dsid)$, such that if he retries this transaction, he will send the same $C^{(\text{pre})}$ every time.

The user computes the challenge $\gamma = \mathcal{H}^{(\gamma)}(k, dsid, tid, C^{(\text{pre})})$ and sends $c_0 = usk \cdot \gamma + dsrnd_0$. If the user spends this token only once, then usk is perfectly hidden. However, if the user tries to spend the same token again for a different transaction id tid (or if he tries to get a different remainder token using $C^{(\text{pre})'} \neq C^{(\text{pre})}$), he will be forced to reveal $usk \cdot \gamma' + dsrnd_0$ with $\gamma \neq \gamma'$ with overwhelming probability. This allows the provider to compute a double-spending user's key usk. To enable the provider to trace the remainder token prepared above, the user additionally publishes $c_1 = esk \cdot \gamma + dsrnd_1$ and encrypts esk^*_{usr} (or, rather, its decomposition to base $base$) under esk using lifted ElGamal for each digit of esk^*_{usr}. If the

$\mathsf{Setup}(1^\lambda)$

- $\mathsf{BG} \leftarrow \mathsf{SPS\text{-}EQ.Gen}(1^\lambda)$, where $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, p, e, g_1, g_2)$
- Choose hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1$ (modeled as random oracle)
- $w = \mathcal{H}(\text{'w'}||\mathsf{BG})$, $h_7 = \mathcal{H}(\text{'h}_7\text{'}||\mathsf{BG})$
- Output $pp = (\mathsf{BG}, w, h_7)$

$\mathsf{P.KeyGen}(pp)$

- $(pk_{\mathsf{SPS\text{-}EQ}}, sk_{\mathsf{SPS\text{-}EQ}}) \leftarrow \mathsf{SPS\text{-}EQ.KeyGen}(\mathsf{BG}, 2)$
- $(pk_\Sigma, sk_\Sigma) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$
- $q_1, \ldots, q_6 \leftarrow \mathbb{Z}_p$, $h_i = g_1^{q_i}$ for $i = 1, \ldots, 6$
- $\beta_{prov} \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$
- Output $sk = (sk_{\mathsf{SPS\text{-}EQ}}, sk_\Sigma, q_1, \ldots, q_6, \beta_{prov})$ and $pk = (pk_{\mathsf{SPS\text{-}EQ}}, pk_\Sigma, h_1, \ldots, h_6)$

$\mathsf{U.KeyGen}(pp, pk)$

- $usk \leftarrow \mathbb{Z}_p$
- $\beta_{usr} \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$
- Output $USK = (usk, \beta_{usr})$ and $upk = w^{usk}$

Alg. 1. Setup and key generation for provider and user

$\mathsf{Link}(pp, dstag, dstag')$

- Parse $dstag = (c_0, c_1, \gamma, ctrace, esk_{\text{prov}})$ and
- $dstag' = (c_0', c_1', \gamma', ctrace', esk_{\text{prov}}')$
- $dsblame = (c_0 - c_0')/(\gamma - \gamma')$
- $upk = w^{dsblame}$
- $dstrace = (c_1 - c_1')/(\gamma - \gamma')$
- Output $(dsblame, upk, dstrace)$

$\mathsf{Trace}(pp, dstrace, dstag)$

- Parse $dstag = (c_0, c_1, \gamma, ctrace, esk_{\text{prov}}^*)$
- Parse $ctrace = (ctrace_{1,i}, ctrace_{2,i})_{i \in [\rho]}$
- $m_i = \begin{cases} \beta, & \text{if } w^\beta = (ctrace_{1,i})^{-dstrace} \cdot ctrace_{2,i} \text{ for some } 0 \le \beta < base \\ \bot & \text{otherwise} \end{cases}$ for $i = 0, \ldots, \rho - 1$
- If $m_i = \bot$ for some i output $\bot$, otherwise
- $dstrace^* = \sum_{i=0}^{\rho-1} m_i \cdot base^i + esk_{\text{prov}}^*$
- Output $(dsid^* = w^{dstrace^*}, dstrace^*)$

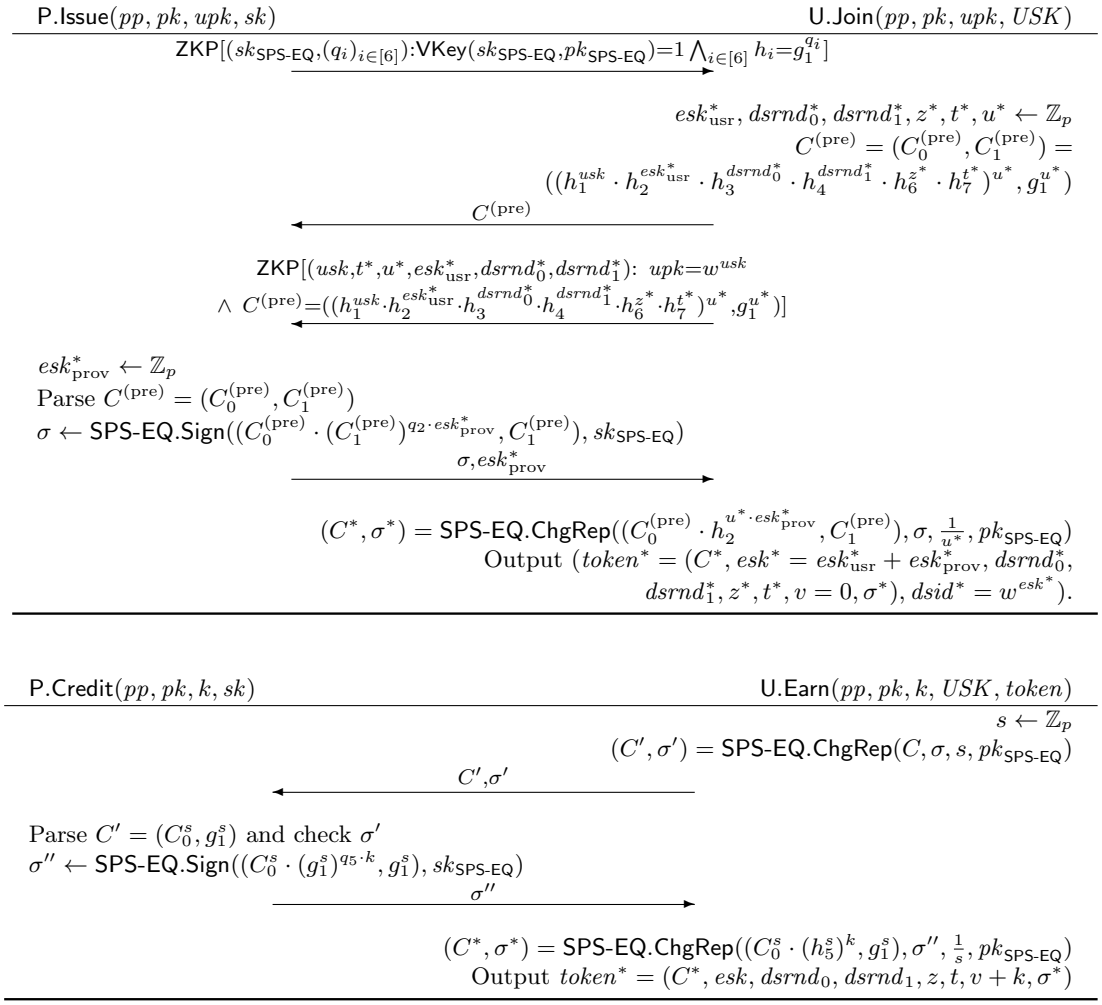
$\mathsf{VerifyDs}(pp, dsblame, upk)$

- Output 1 if $w^{dsblame} = upk$, and 0 otherwise

Alg. 2. Link and trace algorithms

user double-spends, the provider can compute esk and decrypt esk_{usr}^*, which ultimately enables tracing of remainder tokens.

The user then proves in zero-knowledge that all the sent values are well-formed. If the proof and the signature on the old token are valid, then the provider contributes his random share esk_{prov}^* to the remainder token commitment $C^{(\text{pre})}$, and produces an $\mathsf{SPS\text{-}EQ}$ signature σ on the changed $C^{(\text{pre})}$ and sends it back to the user. If everything is valid, then the user obtains the new token (C', g_1) by changing the representative of the $\mathsf{SPS\text{-}EQ}$. Note that the provider's share is deterministically derived from a PRF: $esk_{\text{prov}}^* = \mathsf{PRF.Eval}_{\beta_{prov}}^{\mathbb{Z}_p}(C^{(\text{pre})})$. This means that if the user retries this transaction, the resulting token will get the exact same $esk^* = esk_{\text{usr}}^* + esk_{\text{prov}}^*$ (if this were not the case, the user would be able receive a



Alg. 3. Joining and earning protocols

second token with a different $dsid = w^{esk^*}$ from retrying, which he could then legally spend without being detected).

Linking and tracing. Linking is based on the fact that a double-spending of the same token will with overwhelming probability involve two challenges γ, γ' from the provider such that $\gamma \neq \gamma'$ and given two pairs (c_0, c_0') with $c_0 = usk \cdot \gamma + dsrnd_0$ and $c_0' = usk \cdot \gamma' + dsrnd_0$, and (c_1, c_1') with $c_1 = esk \cdot \gamma + dsrnd_1$ and $c_1' = esk \cdot \gamma' + dsrnd_1$ allows to extract usk and esk. The key usk, then easily allows to identify the double-spender, and esk allows to track all the actions of the respective user after double-spending. For tracing, when given esk, remember that $ctrace$ is an ElGamal encryption of esk^*_{usr} under secret key esk. The provider also remembers his share esk^*_{prov} for every transaction. Basically, then trace can work through all transactions by always decrypting the next encryption key, which is made efficient by the use of DBsync.

Alg. 4. Spending protocol

4.1 Our Construction

Subsequently, in Alg. 1 to 4 we present our concrete construction. Our convention is that we do assume that an algorithm always aborts if verification of a (SPS-EQ) signature fails or a ZKP proof does not verify and we will not make these checks explicit to keep the algorithms as compact as possible. Moreover, k_{max} is some polynomial in λ and $v_{max} \le p - 1$. We model $\mathcal{H}$ as a random oracle and $\mathcal{H}^{(\gamma)}$ as collision-resistant (we omit the hash function key in the trusted setup for brevity).

We remark that although we are presenting our ZKPs in an interactive fashion it is easy to convert all ZKP to NIZK proofs in the random oracle model (ROM) (without a CRS) by applying the Fiat-Shamir (FS) heuristic. Care needs to be taken for the first ZKP in the issuing. We could include the proof in the pk of the provider, but then we can not apply the FS heuristic and would require a straight-line extractable NIZK in the ROM [Fis05] instead. We stress that the option of removing interaction from the ZKPs is interesting if communication rounds are a limiting factor.

4.2 Security

We state the following theorems to formalize that our construction from Sect. 4.1 satisfies all the desired security guarantees defined in Sect. 3. Formal definitions of assumptions can be found in Appendix A. The formal proofs are given in Appendix C.

Theorem 1. *If* PRF *is a pseudorandom function, the* ZKP *is zero-knowledge and a proof of knowledge,* SPS-EQ *provides perfect adaption, and the DDH assumption holds, then our incentive system provides backward unlinkability in the ROM.*

Theorem 2. *If* PRF *is a pseudorandom function, the* ZKP *is zero-knowledge and a proof of knowledge,* SPS-EQ *provides perfect adaption, and the DDH assumption holds, then our incentive system provides anonymity in the ROM.*

Theorem 3. *If* PRF *is a pseudorandom function, $\mathcal{H}^{(\gamma)}$ is collision-resistant,* ZKP *is zero-knowledge and a proof of knowledge, the discrete logarithm problem is hard in $\mathbb{G}_1$,* SPS-EQ *is EUF-CMA secure, and Σ is EUF-CMA secure, then our incentive system is sound.*

Theorem 4. *If* PRF *is a pseudorandom function,* ZKP *is zero-knowledge and the discrete logarithm problem is hard in $\mathbb{G}_1$, then our incentive system is framing resistant.*

Theorem 5. *If* PRF *is a pseudorandom function,* ZKP *is a zero-knowledge proof of knowledge, the discrete logarithm problem is hard in $\mathbb{G}_1$, and SPS-EQ is EUF-CMA secure, then our incentive system has framing-resistance for non-colluding malicious users.*

5 Performance Evaluation

Table 2. Average computation time over 50 runs in milliseconds. Emphasized: typical execution platform for each algorithm.

Device	Issue	Join	Credit	Earn	Deduct	Spend
Google Pixel (2016, Phone, Snapdragon 821)	88	**116**	28	**60**	1083	**1017**
Surface Book 2 (Laptop, i7-8650U)	**14**	19	**4**	9	**237**	220

We implemented our construction using the SPS-EQ scheme from [FHS19]. The zero-knowledge proofs were implemented as a non-interactive variant of Σ-protocols. For the range proofs, we follow the usual base-b-decomposition approach [CCs08]; PS signatures [PS16] are used to ensure the hidden digits are smaller than the base b. As a basis for the implementation, we used the open-source Java library `upb.crypto` with bilinear group bn256 by `mcl`[9]. Our implementation is not optimized and even for a rather outdated smartphone (Google Pixel, 2016) the performance figures in Tab. 2 are very promising. We envision the system to be used in an scenario where the smartphone communicates with the provider (terminal) over Near Field Communication (NFC).

The computation times highlighted in Tab. 2 give an estimated improvement of the most frequently used earn protocol of a factor of 4.5, 6.7 and 2 compared to BBA [JR16], BBA+ [HHNR17] and UACS [BBDE19] respectively. Our earn protocol consists of 5 pairings, 7 exponentiations in $\mathbb{G}_1$, and 2 exp. in

[9] upb.crypto: `https://github.com/upbcuk`. mcl: `https://github.com/herumi/mcl`

$\mathbb{G}_2$ for the user while the provider computes 5 pairings and 4 exp. in $\mathbb{G}_1$. We note that Spend is relatively costly, though still practical, compared to prior works. Its costs are mostly driven by the cost of range proofs to allow partially spending points (which, for example, are omitted in BBA and BBA+), and by the cost of encrypting the digits of the ElGamal secret key esk^* with ElGamal, which enables remainder token tracing with backward unlinkability (this combination is not offered by any other incentive system that we are aware of).

References

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBDE19. Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. Updatable anonymous credentials and applications to incentive systems. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1671–1685. ACM Press, November 2019.

BBM00. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.

BCF+11. Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 206–223. Springer, Heidelberg, July 2011.

BCFK15. Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable E-cash. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 101–124. Springer, Heidelberg, March / April 2015.

CCs08. Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008.

CEL+07. Liqun Chen, Alberto N. Escalante, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. A privacy-protecting multi-coupon scheme with stronger protection against splitting. In Sven Dietrich and Rachna Dhamija, editors, *FC 2007*, volume 4886 of *LNCS*, pages 29–44. Springer, Heidelberg, February 2007.

CES+05. Liqun Chen, Matthias Enzmann, Ahmad-Reza Sadeghi, Markus Schneider II, and Michael Steiner. A privacy-protecting coupon system. In Andrew Patrick and Moti Yung, editors, *FC 2005*, volume 3570 of *LNCS*, pages 93–108. Springer, Heidelberg, February / March 2005.

CGH06. Sébastien Canard, Aline Gouget, and Emeline Hufschmitt. A handy multi-coupon system. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 66–81. Springer, Heidelberg, June 2006.

CHK+06. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 201–210. ACM Press, October / November 2006.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, Heidelberg, May 2005.

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

CPST15a. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible E-cash made practical. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 77–100. Springer, Heidelberg, March / April 2015.

CPST15b. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Scalable divisible E-cash. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 287–306. Springer, Heidelberg, June 2015.

FHS15. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253. Springer, Heidelberg, August 2015.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

HFN+18. Max Hoffmann, Valerie Fetzer, Matthias Nagel, Andy Rupp, and Rebecca Schwerdt. P4TC - provably-secure yet practical privacy-preserving toll collection. *IACR Cryptology ePrint Archive*, 2018:1106, 2018.

HHNR17. Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. BBA+: Improving the security and applicability of privacy-preserving point collection. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1925–1942. ACM Press, October / November 2017.

HKRR19. Max Hoffmann, Michael Klooß, Markus Raiber, and Andy Rupp. Black-box wallets: Fast anonymous two-way payments for constrained devices. *IACR Cryptology ePrint Archive*, 2019:1199, 2019. To appear in PoPETS 2020 Issue 1.

HS14. Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 491–511. Springer, Heidelberg, December 2014.

JR16. Tibor Jager and Andy Rupp. Black-box accumulation: Collecting incentives in a privacy-preserving way. *PoPETs*, 2016(3):62–82, July 2016.

MDPD15. Milica Milutinovic, Italo Dacosta, Andreas Put, and Bart De Decker. uCentive: An efficient, anonymous and unlinkable incentives scheme. In *TrustCom/BigDataSE/ISPA (1)*, pages 588–595. IEEE, 2015.

Ngu06. Lan Nguyen. Privacy-protecting coupon system revisited. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 266–280. Springer, Heidelberg, February / March 2006.

NR04. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, 2004.

Oka95. Tatsuaki Okamoto. An efficient divisible electronic cash scheme. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 438–451. Springer, Heidelberg, August 1995.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

PSW12. Martin Pirker, Daniel Slamanig, and Johannes Winter. Practical privacy preserving cloud resource-payment for constrained clients. In *Privacy Enhancing Technologies - 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings*, pages 201–220, 2012.

RBHP15. Andy Rupp, Foteini Baldimtsi, Gesine Hinterwälder, and Christof Paar. Cryptographic theory meets practice: Efficient and privacy-preserving payments for public transport. *ACM Trans. Inf. Syst. Secur.*, 17(3):10:1–10:31, 2015.

Sla12. Daniel Slamanig. Efficient schemes for anonymous yet authorized and bounded use of cloud resources. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 73–91. Springer, Heidelberg, August 2012.

SNF+19. Rebecca Schwerdt, Matthias Nagel, Valerie Fetzer, Tobias Gräf, and Andy Rupp. P6v2g: a privacy-preserving v2g scheme for two-way payments and reputation. *Energy Informatics*, 2(1):1–21, 2019.

Sta96. Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

A Complexity Assumptions

We recall the DDH assumptions and we define the advantage of an adversary $\mathcal{A}$ relative to a group generator $(\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$ as $\mathbf{Adv}^{\mathsf{DDH}}_{\mathcal{A},\mathsf{GGen}}(\lambda) :=$

$$\left|\Pr\left[\begin{matrix} r,s,t \leftarrow \mathbb{Z}_p, b \leftarrow \{0,1\}, \\ b^* \leftarrow A((\mathbb{G},p,g), g^r, g^s, g^{rs+bt}) \end{matrix} : b^* = b\right] - \frac{1}{2}\right|.$$

We say that the DDH assumption holds if $\mathbf{Adv}^{\mathsf{DDH}}_{\mathcal{A},\mathsf{GGen}}$ is negligible functions in the security parameter λ for all PPT adversaries $\mathcal{A}$, respectively. We note that in our construction we will require the problem relative to the group $\mathbb{G}_1$ in a bilinear group generated by BGGen.

Additionally, we rely on the generalized version of DDH for the backward unlinkability and anonymity proofs. In generalized DDH the adversary gets a set of DDH instances. We define $\mathbf{Adv}^{\mathsf{x,y\text{-}DDH}}_{\mathcal{A},\mathsf{GGen}}(\lambda) :=$

$$\left|\Pr\left[\begin{matrix} r_1,\ldots,r_x, s_{1,1},\ldots,s_{x,y}, \\ t_{1,1},\ldots,t_{x,y} \leftarrow \mathbb{Z}_p, b \leftarrow \{0,1\}, \\ b^* \leftarrow A((\mathbb{G},p,g), [g^{r_i}, g^{s_{i,j}}, \\ g^{r_i s_{i,j} + b t_{i,j}}]_{i=1,\ldots,x; j=1,\ldots,y}) \end{matrix} : b^* = b\right] - \frac{1}{2}\right|.$$

We say that the x, y-DDH assumption holds if $\mathbf{Adv}^{\mathsf{x,y\text{-}DDH}}_{\mathcal{A},\mathsf{GGen}}$ is a negligible function in the security parameter λ for all PPT adversaries $\mathcal{A}$. The x, y-DDH assumption holds under the DDH assumption under a tight security proof [Sta96,NR04,BBM00].

B Formal Definitions of Building Blocks

In the following we introduce the formal definitions of building blocks that we use to build our incentive system.

B.1 SPS-EQ

Definition 7 (SPS-EQ). *A structure-preserving signature on equivalence classes (SPS-EQ) scheme* $\mathsf{SPS\text{-}EQ}$ *on message space* $(\mathbb{G}_i^*)$ *for* $i \in \{1,2\}$ *consists of the following PPT algorithms.*

$\mathsf{Gen}(1^\lambda)$ *on input a security param.* λ*, outputs a bilinear group* BG*.*
$\mathsf{KeyGen}(\mathsf{BG}, \ell)$ *on input* BG *and message vector length* $\ell > 1$*, outputs a key pair* (pk, sk)*.*
$\mathsf{Sign}(M, sk)$ *on input representative* $M \in (\mathbb{G}_i^*)^\ell$ *and a secret key* sk*, outputs a signature* σ *for the equivalence class* $[M]$*.*
$\mathsf{ChgRep}(M, \sigma, \mu, pk)$ *on input representative* $M \in (\mathbb{G}_i^*)^\ell$ *of equivalence class* $[M]$*, a signature* σ *on* M*, a value* μ *and a public key* pk*, returns an updated message-signature pair* (M', σ')*, where the new representative is* $M' = M^\mu$ *and* σ' *its corresponding (or, updated) signature.*
$\mathsf{Vrfy}(M, \sigma, pk)$ *is a deterministic algorithm and, on input a representative* $M \in (\mathbb{G}_i^*)^\ell$*, a signature* σ*, and public key* pk *and outputs a bit* $b \in \{0,1\}$*.*
$\mathsf{VKey}(sk, pk)$ *is a deterministic algorithm and, on input secret key* sk *and a public key* pk*, outputs a bit* $b \in \{0,1\}$*.*

Below, we define correctness, $\mathsf{EUF\text{-}CMA}$ security and perfect adaption of signatures for $\mathsf{SPS\text{-}EQ}$ schemes.

Definition 8 (Correctness). *An* $\mathsf{SPS\text{-}EQ}$ *scheme on* $(\mathbb{G}_i^*)^\ell$ *is called correct if for all security parameters* $\lambda \in \mathbb{N}$*,* $\ell > 1$*,* $\mathsf{BG} \leftarrow \mathsf{Gen}(1^\lambda)$*,* $(sk, pk) \leftarrow \mathsf{KeyGen}(\mathsf{BG}, \ell)$*,* $M \in (\mathbb{G}_i^*)^\ell$ *and* $\mu \in \mathbb{Z}_p^*$*:*

$$\mathsf{VKey}(sk, pk) = 1 \ \wedge \ \Pr\left[\mathsf{Vrfy}(M, \mathsf{Sign}(M, sk), pk) = 1\right] = 1$$
$$\wedge \Pr\left[\mathsf{Vrfy}(\mathsf{ChgRep}(M, \mathsf{Sign}(M, sk), \mu, pk), pk) = 1\right] = 1.$$

$\mathsf{EUF\text{-}CMA}$ security is similar to that of conventional signatures, but the forgery output by the adversary needs to be with respect to an unqueried equivalence class.

Definition 9 ($\mathsf{EUF\text{-}CMA}$). *An* $\mathsf{SPS\text{-}EQ}$ *over* $(\mathbb{G}_i^*)^\ell$ *is existentially unforgeable under adaptively chosen-message attacks, if for any PPT adversaries* $\mathcal{A}$ *there exists a negligible function negl such that for all* $\lambda \in \mathbb{N}$ *and* $\ell \in \mathsf{poly}(\lambda)$ *it holds that*

$$\Pr[\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(\lambda, \ell) = 1] \leq negl(\lambda),$$

where $\mathcal{Q}$ *is the set of queries that* $\mathcal{A}$ *has issued to* **Sign**.

Experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(\lambda, \ell)$

$\mathsf{BG} \leftarrow \mathsf{Gen}(1^\lambda)$
$(sk, pk) \leftarrow \mathsf{KeyGen}(\mathsf{BG}, \ell)$
$(M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathbf{Sign}(sk,\cdot)}(pk)$
return 1 if $[M^*] \neq [M] \ \ \forall M \in \mathcal{Q} \ \wedge \mathsf{Vrfy}(M^*, \sigma^*, pk) = 1.$
return 0

Fig. 9. Unforgeability of $\mathsf{SPS\text{-}EQ}$

Definition 10 (Perfect Adaption of Signatures [FHS15]). *An* SPS-EQ *scheme on* $(\mathbb{G}_i^*)^\ell$ *perfectly adapts signatures if for all tuples* (sk, pk, M, σ, μ) *where it holds that* $\mathsf{VKey}(sk, pk) = 1$, $\mathsf{Vrfy}(M, \sigma, pk) = 1$, $M \in (\mathbb{G}_i^*)^\ell$, *and* $\mu \in \mathbb{Z}_p^*$, *the distributions* $(M^\mu, \mathsf{Sign}(M^\mu, sk))$ *and* $\mathsf{ChgRep}(M, \sigma, \mu, pk)$ *are identical.*

B.2 Zero-Knowledge Proofs of Knowledge

We consider an (interactive) protocol (P, V) between a (potentially unbounded) prover P and a PPT verifier V and denote the outcome of the protocol as $(\cdot, b) \leftarrow \big(P(\cdot,\cdot), V(\cdot)\big)$ where $b = 0$ indicates that V rejects and $b = 1$ that it accepts the conversation with P. We require the usual notions of *zero-knowledge* and *knowledge soundness*.

Definition 11 (Zero knowledge). *We say that an interactive protocol* (P, V) *for a relation* $\mathcal{R}$ *is zero-knowledge if for all PPT algorithms* V^* *there exists a PPT simulator* $\mathcal{S}$ *such that:*

$$\big\{\mathcal{S}^{V^*}(x)\big\}_{x \in \mathcal{L}_\mathcal{R}} \approx \big\{\big\langle P(x,w), V^*(x)\big\rangle\big\}_{(x,w)\in\mathcal{R}},$$

where $\langle P(\cdot,\cdot), V^*(\cdot)\rangle$ *denotes the transcript of the interaction of* P *and* V, *and "*$\approx$*" denotes (perfect) indistinguishability.*

Definition 12 (Knowledge soundness). *We say that* (P, V) *is a proof of knowledge (PoK) relative to an* **NP** *relation* $\mathcal{R}$ *if for any (possibly unbounded) malicious prover* P^* *such that* $(\cdot, 1) \leftarrow \big(P^*(x), V(x)\big)$ *with probability greater than* ϵ *there exists a PPT knowledge extractor* $\mathcal{K}$ *(with rewinding black-box access to* P^**) such that* $\mathcal{K}^{P^*}(x)$ *returns a value* w *satisfying* $(x, w) \in \mathcal{R}$ *with probability polynomial in* ϵ.

If an interactive protocol is perfect zero knowledge and satisfies knowledge soundness, then we call it a zero-knowledge proof of knowledge ZKP.

C Proof sketches

We now give intuitions for the security proofs and due to space constraints, more extensive proofs will appear in the full version of this paper.

C.1 Proof of Theorem 1: Backwards-Unlinkability

Proof. We prove the theorem using a sequence of games, where we use $\mathbf{Adv}_i$ to denote the advantage of the adversary in Game i.

Game 0. This is the original security experiment $\mathrm{Exp}^{\mathsf{bu}}$ except that $\mathsf{PRF.Eval}^{\mathbb{Z}_p^6}_{\beta_{usr}}$ is replaced with a random oracle. Consequently, we have $|\mathbf{Adv}_0 - \mathbf{Adv}^{\mathsf{bu}}_{\mathcal{A},\Pi}(\lambda)| \leq \mathbf{Adv}^{\mathsf{PRF}}_{\mathcal{B}}(\lambda)$.

Game 1. In this game, on the first call of the adversary to the **Join** oracle, run the extractor of ZKP to extract $sk = (sk_{\mathsf{SPS\text{-}EQ}}, \cdot, q_1, \ldots, q_6)$. Note that this is the first call an adversary has to do as all oracles return $\perp$ unless a token for u_0 and u_1 exist in the system. Consequently, the view of the adversary is the same as in the previous game and we have $|\mathbf{Adv}_1 - \mathbf{Adv}_0| \leq \mathbf{Adv}^{\mathsf{pk}}_{\mathcal{B},\mathsf{ZKP}}(\lambda)$.

Game 2. In this game, we simulate all ZKP proofs within the oracles **Join** and **Spend**. We have $|\mathbf{Adv}_2 - \mathbf{Adv}_1| \leq \mathbf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{ZKP}}(\lambda)$.

Game 3. In this game we always use $sk_{\mathsf{SPS\text{-}EQ}}$ to produce new signatures via SPS-EQ.Sign instead of running SPS-EQ.ChgRep. Under the perfect adaption of SPS-EQ we have $\mathbf{Adv}_3 = \mathbf{Adv}_2$.

Game 4. In this game we guess the index i_0 of the last query (or the issue query if there is no spend

query) to the spend (or issue) oracle $\mathbf{Spend}(u_0,\cdot,\cdot)$ among the $q_{e,0}$ earn and $q_{s,0}$ spend queries (including the issue query) that the adversary makes to the $\mathbf{Earn}(u_0,\cdot)$ and $\mathbf{Spend}(u_0,\cdot,\cdot)$ oracles. If our guess is incorrect, we abort. We note that in the following we only speak of spend queries as handling the issue query (if there are no spend queries) is identical. We have $\mathbf{Adv}_4 = \frac{1}{(q_{e,0}+q_{s,0})} \cdot \mathbf{Adv}_3$.

Game 5. In this game for query i_0 we choose $R \leftarrow \mathbb{G}_1$, $r, r' \leftarrow \mathbb{Z}_p$, and program the random oracle s.t. $h_7 = H(\text{"h}_7\text{"}\|\mathsf{BG}) := R^{\frac{1}{r}} \cdot g_1^{-\frac{r'}{r}}$. On the i_0'th query (i.e., the last spend query), we set $C^{(\text{pre})} = (Z^{w^*}, g_1^{w^*})$ for $w \leftarrow \mathbb{Z}_p$ and $Z \leftarrow \mathbb{G}_1$. In the guess phase we provide the adversary with an opening $(usk_{u_0}, token_{u_0} = (usk_0, ((R', g_1), esk_{\text{usr}}, dsrnd_0, dsrnd_1, t, z, v, \sigma))$. Thereby, if i_0 is the last spend oracle call before the guessing phase, then $R' = R \cdot h_2^{esk^*_{\text{prov}}}$ with esk^*_{prov} being the value chosen by $\mathcal{A}$ in the last spend query. Otherwise let $v^* = \sum_j k_j$ be the sum of all earns following the last spend query and we have $R' = R \cdot h_2^{esk^*_{\text{prov}}} \cdot h_5^{v^*}$. Moreover, we have $esk_{\text{usr}}, dsrnd_0, dsrnd_1 \leftarrow \mathbb{Z}_p$, $esk = esk_{\text{usr}} + esk^*_{\text{prov}}$ and $v := v + v^*$ explains the current state of user u_0 and sets $t = r$ and $z = \frac{1}{q_6}(r' - (q_1 \cdot usk_{u_0} + q_2 \cdot esk + q_3 \cdot dsrnd_0 + q_4 \cdot dsrnd_1 + q_5 \cdot v)$.
Game 4 $\rightarrow$ Game 5. We claim that the above changes are not noticeable by $\mathcal{A}$ under the DDH assumption. First, note that choosing Z at random within the last $\mathbf{Spend}(u_0,\cdot,\cdot)$ call does not change the view as $C_0^{(\text{pre})}$ represents a freshly sampled perfectly hiding commitment. It remains to argue that instead the adversary does not notice that we replace Z with R. Let $(U, V, W) = (g_1^a, g_1^u, g_1^{au+bc})$ be our challenge DDH tuple, then we program the random oracle using $R := U$. Note that this programming as well as delivering the final output for u_0 in the guessing phase is identical to the original view of $\mathcal{A}$. For the i_0'th query made by $\mathcal{A}$ to **Spend** set $C^{(pre)} := (W, V) = (g_1^{au+bc}, g_1^u)$ and in the received token set $C = (U \cdot h_2^{esk^*_{\text{usr}}}, g_1) = (g_1^a \cdot h_2^{esk^*_{\text{usr}}}, g_1)$ if i_0 is the last query for u_0 or $C = (U \cdot h_2^{esk^*_{\text{usr}}} \cdot h_5^{v^*}, g_1)$ if it is followed by earn queries and $v^* = \sum k_j$ is the sum of all these earn queries. Now, it is easy to see that if $b = 0$ we are in Game 4, and if $b = 1$ we are in Game 5. Consequently, we have $|\mathbf{Adv}_5 - \mathbf{Adv}_4| \leq \mathbf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathsf{BGGen}}(\lambda)$.

Game 6. Exactly the same as in Game 4, but now for user u_1. We have, $\mathbf{Adv}_6 = \frac{1}{(q_{e,1}+q_{s,1})}\mathbf{Adv}_5$.

Game 7. Exactly the same as in Game 5, but now for user u_1. We have, $|\mathbf{Adv}_7 - \mathbf{Adv}_6| \leq \mathbf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathsf{BGGen}}(\lambda)$.

Game 8. In this game in **Spend** we always output as *ctrace* ρ, many random tuples $(R, R') \leftarrow \mathbb{G}_1^2$ instead of $((w^{r_j}, (w^{r_j})^{esk} \cdot w^{esk^*_j}))_{j\in[\rho]}$. Hence, we change it from ElGamal encryptions of esk^*_{usr} to an output that is independent of esk^*_{usr} and esk. Note that in the Game 8, the values $(usk_{u_0}, token_{u_0}, rtoken_{u_0}, usk_{u_1}, token_{u_1}, rtoken_{u_1})$ received in the guessing phase are independent of the view of all interactions with the oracles in the first phase. Consequently, we have $\mathbf{Adv}_8 = \frac{1}{2}$.

Game 7 $\rightarrow$ Game 8. We claim that the above changes are not noticeable by $\mathcal{A}$ under the x, y-DDH assumption (Appendix A), which is tightly equivalent to DDH.

First, notice that the x, y-DDH assumption gives us x many y extended DDH instances. Extended means that instead of challenge $(U, V, W) = (g_1^a, g_1^u, g_1^{au+bc})$ as in normal DDH, we get x many extended instances like $(U_i, V_{i,1}, \ldots, V_{i,y}, W_{i,1}, \ldots, W_{i,y}) = (g_1^{a_i}, g_1^{u_{i,1}}, \ldots, g_1^{u_{1,y}}, g_1^{a_i u_{i,1}+bc_{i,1}}, \ldots, g_1^{a_i u_{i,y}+bc_{i,y}})$ for $i = 1, \ldots, x$. Next, we describe how we embed the x, y-DDH challenge. First, let $\mathcal{A}$ make x many calls to phase 2 of **Spend**. In each call we reveal the current $dsid_i = w^{esk_i}$. We embed U_i as $dsid_i = U_i$. Additionally, in each call we send $ctrace_i$ to $\mathcal{A}$ which is a set of ρ many encryptions $((w^{r_j}, (w^{r_j})^{esk} \cdot w^{esk_{\text{usr}}{}^*_j}))_{j\in[\rho]}$. With $y = \rho$ we use the tuples $(V_{i,j}, W_{i,j} \cdot w^{esk_{\text{usr}}{}^*_j})_{j\in[y]}$ to embed the remaining parts of the x, y-DDH challenge. Now, it is easy to see that if $b = 0$ we are in Game 7, and if $b = 1$ we are in Game 8. Consequently, we have $|\mathbf{Adv}_7 - \mathbf{Adv}_8| \leq \mathbf{Adv}^{\mathsf{x,y\text{-}DDH}}_{\mathcal{B},\mathsf{BGGen}}(\lambda)$. This concludes the proof.

C.2 Proof of Theorem 2: Anonymity

Proof. The proof is done through a sequence of games and is done exactly as for the backward unlinkability property in the proof of Theorem 1. The difference is that in the anonymity experiment we do not need to reveal the token values to the adversary in the challenge phase. Instead we execute either one **Earn** or **Spend** call in the challenge phase and use the prepared token values to execute them. As in the proof of Theorem 1 we embed the changes in the last **Spend** (resp. **Issue**) call to prepare the token values for the challenge phase.

C.3 Proof of Theorem 3: Soundness

Our proof of soundness is similar *in structure* to the proof in [BBDE19], but in detail our construction calls for completely new arguments.

As a first step, we replace invocations of $\mathsf{PRF.Eval}^{\mathbb{Z}_p}_{\beta_{prov}}$ with proper random choices (the adversary cannot notice this change).

Then essentially, we run the zero-knowledge proof of knowledge extractor on all (accepting) proofs. This allows us for every **Issue** query to extract the token and all its hidden values $usk, esk, dsrnd_0, dsrnd_1, v, z, t$ from the user. Similarly, for every **Credit** query, we can extract the values of the token that the user spends, as well as the values of the remainder token the user receives from that transaction. For **Credit** queries, we cannot efficiently figure out what token points were added to, but it is still well-defined: the user sends a tuple $C' = (C'_0, g_1^s)$ of the token he wants to credit points to. Because of $\mathsf{SPS\text{-}EQ}$ unforgeability, with overwhelming probability, ${C'_0}^{1/s}$ is one of the commitments the provider has signed.

Using this, we can (as a thought experiment) define a bipartite graph in which on type of node are commitments and their openings and the other type is oracle queries. An edge from a commitment to an oracle query is present if the commitment was used as input to the oracle query, and an edge from the oracle query to the commitment if the oracle query resulted in the commitment.

Let $\mathsf{error}_{\mathsf{bind}}$ be the event that this graph contains a commitment with two different openings. Using a standard reduction, we show that $\Pr[\mathsf{error}_{\mathsf{bind}}]$ is negligible. Note that we cannot efficiently construct the graph (because the input and output of **Credit** queries are computationally hidden as explained above). However, the reduction can overapproximate the set of commitment nodes present in the graph by essentially homomorphically adding up to $poly \cdot k_{max}$ points to each commitment extracted during **Issue** and **Credit**.

It then remains to show that this graph fulfills certain properties (essentially that *esk*s are unique, every node input is explained by some other node's output, etc.). This then allows us to reason about the $\mathcal{DB}$ graph and, ultimately, that users cannot spend more than they have earned (after subtracting transactions detected as double-spending).

C.4 Proof of Theorem 4: Framing-Resistance

Proof (Framing-resistance). Let $\mathcal{A}$ be a PPT adversary against $\mathrm{Exp}^{\mathsf{fr}}$. We construct $\mathcal{B}$ against the discrete logarithm assumption. $\mathcal{B}$ gets as input a group description $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, p, e, w, g_2)$ and a some $w^x \in \mathbb{G}_1$. $\mathcal{B}$ programs the random oracle s.t. $\mathcal{H}(\text{"w"}||\mathsf{BG}) = w$ and sets $pp = (\mathsf{BG}, w, h_7)$ (for some $h_7 = \mathcal{H}(\text{"h}_7\text{"}||\mathsf{BG}) \leftarrow \mathbb{G}_1$).

Furthermore, $\mathcal{B}$ chooses a random index $j \leftarrow \{1, \ldots, \ell\}$, where ℓ is an upper bound on the number of users $\mathcal{A}$ creates. Then $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{B}$ plays the role of the framing resistance experiment honestly, but for the jth **Keygen**() query, $\mathcal{B}$ outputs (u_j, upk_{u_j}) with $upk_{u_j} := w^x$ instead of generating a key. $\mathcal{B}$ sets $usk_{u_j} := 0$. For the **Join**(u_j) query, $\mathcal{B}$ simulates the user's ZKP protocol. For all other queries, $\mathcal{B}$ follows

the protocol (with $usk_{u_j} = 0$). Eventually, $\mathcal{A}$ outputs some $(dsblame, u)$. If $u \neq u_j$ or $w^{dsblame} \neq upk_u$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs $x = dsblame$.

Simulation. $\mathcal{B}$ perfectly simulates $\mathrm{Exp}^{\mathsf{fr}}$. For this, note that usk_{u_j} only appears in the following contexts: (1) in Pedersen commitments $C_0^{(\mathrm{pre})}$ and C_0, in which usk_{u_j} is perfectly hidden (note that z^* and z are never revealed), (2) in zero-knowledge arguments about Pedersen commitments, and (3) in the expression $c_0 = usk_{u_j} \cdot \gamma + dsrnd_0$ when spending points. By design of the incentive system, for every $token$, there is at most one γ such that $c_0 = usk_{u_j} \cdot \gamma + dsrnd_0$ is revealed. Also, for every $token$, $dsrnd_0$ is chosen freshly random (and perfectly hidden from $\mathcal{A}$). Hence c_0 is uniformly random, independent of usk_{u_j}. We conclude that setting $usk_{u_j} = 0$ does not alter $\mathcal{A}$'s view.

Win probability. $\mathcal{B}$ outputs the correct discrete logarithm x if $u = u_j$ and $w^{dsblame} = upk_u$. It holds that $\Pr[\mathcal{B}(\mathsf{BG}, w, w^x) = x] = \Pr[u = u_j \wedge w^{dsblame} = upk_u] = \Pr[u = u_j] \cdot \Pr[w^{dsblame} = upk_u] = \frac{1}{\ell} \cdot \Pr[\mathrm{Exp}^{\mathsf{fr}}_{\mathcal{A},\Pi}(\lambda) = 1]$. By assumption, $\Pr[\mathcal{B}(\mathsf{BG}, w, w^x) = x]$ is negligible and the maximum number of users ℓ that $\mathcal{A}$ may create is polynomially bounded. Hence $\Pr[\mathrm{Exp}^{\mathsf{fr}}_{\mathcal{A},\Pi}(\lambda) = 1]$ is negligible.

C.5 Proof of Theorem 5: Framing-Resistance for Non-Colluding Malicious Users

For the intuition of the proof, we highlight two cases: in $\mathcal{DB}$, when $upk_{\mathcal{A}}$ is computed from two transactions tid, tid' to illegally blame $\mathcal{A}$, it is done such that (1) neither of the transactions was done by $\mathcal{A}$, or (2) one of the was. In the first case, we can effectively run $\mathcal{B}$ by itself (we run $\mathcal{A}$ until it outputs its $upk_{\mathcal{A}}$, then never run it again). Because $\mathcal{A}$ and $\mathcal{B}$ cannot communicate, $\mathcal{B}$ cannot tell that $\mathcal{A}$ is not actually run and will still blame $\mathcal{A}$. The proof of knowledge in Deduct forces $\mathcal{B}$ to have a signature on (a commitment on) $usk_{\mathcal{A}}$. We will never have given such a signature to $\mathcal{B}$ (note that any $\mathbf{Issue}(upk_{\mathcal{A}})$ queries by $\mathcal{B}$ will be ignored, as $upk_{\mathcal{A}}$ can only be registered once). Hence $\mathcal{B}$ must forge a signature (or break the commitment's binding property). In order for the second case to occur, $\mathcal{B}$ and $\mathcal{A}$ must both get a token with the same $dsid$. Because the provider effectively signs only (commitments on) randomized $dsid$s, this is unlikely, except if $\mathcal{A}$ or $\mathcal{B}$ forge a signature (or break the commitment's binding property). The third potential case (3) is that $upk_{\mathcal{A}}$ was computed from two of $\mathcal{A}$'s transactions, but later DBsync adds some by $\mathcal{B}$ for the same $dsid$. Similarly to the second case, in this case $\mathcal{A}$ and $\mathcal{B}$ must get a signature on (a commitment on) $dsid$.

Hence both cases effectively reduce to the security of the "commit-proof-sign" part of our construction. The proof is straightforward using techniques from the soundness proof.