CheckOut: User-Controlled Anonymization for Customer Loyalty Programs

Matthew Gregoire
University of North Carolina at Chapel Hill
mattyg@cs.unc.edu

Rachel Thomas
University of North Carolina at Chapel Hill
rthomase@cs.unc.edu

Saba Eskandarian
University of North Carolina at Chapel Hill
saba@cs.unc.edu

ABSTRACT

To resist the regimes of ubiquitous surveillance imposed upon us in every facet of modern life, we need technological tools that subvert surveillance systems. Unfortunately, while cryptographic tools frequently demonstrate how we can construct systems that safeguard user privacy, there is limited motivation for corporate entities engaged in surveillance to adopt these tools, as they often clash with profit incentives. This paper demonstrates how, in one particular aspect of everyday life – customer loyalty programs – users can subvert surveillance and attain anonymity, *without* necessitating any cooperation or modification in the behavior of their surveillors.

We present the CheckOut system, which allows users to coordinate large anonymity sets of shoppers to hide the identity and purchasing habits of each particular user in the crowd. CheckOut scales up and systematizes past efforts to subvert loyalty surveillance, which have been primarily ad-hoc and manual affairs where customers physically swap loyalty cards to mask their real identities. CheckOut allows increased scale while ensuring that the necessary computing infrastructure does not itself become a new centralized point of privacy failure.

Of particular importance to our scheme is a protocol for loyalty programs that offer reward points, where we demonstrate how CheckOut can assist users in paying each other back for loyalty points accrued while using each others' loyalty accounts. We present two different mechanisms to facilitate redistributing rewards points, offering trade-offs in functionality, performance, and security.

1 INTRODUCTION

A great deal of recent research and deployed technology have focused on addressing the challenge of building increasingly private mechanisms for financial transactions [2, 10, 26, 39, 40, 52]. However, merely making financial exchanges private does not suffice as a means to protect user privacy in important everyday transactions. Many businesses operate deeply invasive loyalty programs that give customers financial incentives in the form of discounts and rewards in exchange for tracking their purchasing behavior over time [41, 51]. Previous works have demonstrated the feasibility of various privacy-preserving loyalty systems [4, 6, 27, 33, 35, 37, 49], but these systems often clash with companies' interests in not only promoting customer loyalty, but also profiting from the data they collect from their customers.

Meanwhile, outside the security research community, various ad-hoc efforts have attempted to protect privacy in the presence of unavoidable surveillance. Of particular interest to us are efforts to subvert tracking of customer behavior via loyalty cards, which typically operate by having customers scan a barcode on a loyalty card when making a purchase. These efforts include Rob's Giant Bonus-Card Swap Meet [13], where participants would manually exchange loyalty barcodes to masquerade as each other while shopping, and The Ultimate Shopper [19], where an individual encouraged others to collectively use his loyalty barcode when shopping, thus concealing the true identity of the purchaser. These efforts have been described by Nissenbaum and Brunton as instances of obfuscation [8, 9] and constitute a valiant grassroots effort at undermining surveillance.

Figure 1: An outline of the solution architecture. From the retailer's perspective, a normal transaction occurs, and the customer provides a loyalty account at checkout time. But because the loyalty account is selected from a third-party anonymity set, the seller is unable to track customer purchasing habits. The customer then relays the number of loyalty points earned, so the third party can keep tally of who owes whom loyalty points.

Unfortunately, previous loyalty obfuscation efforts have been limited in scale and effectiveness, both as a means of protest and of protecting privacy, by their ad-hoc nature. Manual card swapping, for instance, remains constrained by its capacity to accommodate only a limited number of participants, and the frequent swapping needed to assume a different identity for each shopping trip requires a great deal of effort. Conversely, a collective adoption of a single barcode may offer scalability; however, it invariably grants a single individual the opportunity to reap all the benefits of the loyalty program, such as accruing exclusive rewards points that can be redeemed for discounts, thereby diminishing the program's inherent value for others.

Inspired by the manual loyalty obfuscation efforts of the past, this paper introduces the CheckOut system for obfuscating loyalty card transactions, which scales up and systematizes card-swapping operations while offering mechanisms for users to fairly benefit from their participation in loyalty programs.

CheckOut allows users to contribute to a database of loyalty cards and retrieve a new, random loyalty card identity for each purchasing transaction at the store. In this sense, it is a directly scaled-up version of manual card swaps. However, introducing the technological infrastructure needed to facilitate card swapping at scale opens the door to new privacy challenges. In particular, the

card swapping infrastructure could itself become a point of failure for user privacy. We investigate the sometimes subtle ways in which privacy could be compromised by such a system (or its operator), and we design CheckOut to avoid the privacy risks introduced by a potentially malicious card swap operator.

Since many loyalty programs offer rewards points whose values can easily be mapped to dollars, CheckOut includes features for users to keep track of how many loyalty points they have received while using others' identities, facilitating repayment of these rewards. This aspect of the system requires careful consideration. Given the lack of "ground-truth" data about what transactions actually happened at the store, it is not immediately clear what kinds of security and integrity properties can be achieved. For example, a user can always misreport the amount of a transaction, and a fraudulently claimed transaction cannot be falsified without access to information about real transactions at the merchant.

Nonetheless, we show how to define appropriate confidentiality and integrity notions to provide reliable and internally consistent private bookkeeping for scalable loyalty card swaps. We show that CheckOut enables strong confidentiality protections against the server while protecting against disruptive clients who may wish to compromise the system. We present two different approaches to redistributing reward points – one for a semihonest server and one for a fully malicious server – and explore the functionality and performance trade-offs between them.

CheckOut's core functionality relies only on standard cryptographic primitives – hash functions, digital signatures, and public key encryption – and the private reward point tracking feature additionally requires the use of non-interactive zero-knowledge proofs (NIZKs) of statements about users' hidden balances and transaction amounts. [5, 21, 28–30] To demonstrate the feasibility of deploying CheckOut at scale, we have implemented each component of the protocol and report on the computational costs of the cryptographic protocols involved.

In summary, this paper makes the following contributions.

- A scheme for scalable swapping of loyalty card identities that is secure against a malicious server operator.
- Two mechanisms for users to privately keep track of reward points accrued while using each other's loyalty accounts, offering security, functionality, and performance trade-offs.
- An implementation and evaluation of the CheckOut algorithms, which realizes the new protocols introduced here.

The CheckOut source code and evaluation data are open source and publicly available at https://github.com/MatthewGregoire42/LoyaltyPointsCrypto.

2 OVERVIEW AND SECURITY GOALS

This section gives an overview of the CheckOut system and describes our overall security goals. CheckOut is run on a server operated independently of the loyalty program, which it augments, and does not require any cooperation with a business offering a loyalty program in order to function. In practice, one server could run multiple instances of CheckOut, one for each of several different loyalty programs, each offered by a different merchant.

2.1 CheckOut for Users

Users interact with CheckOut by installing an app on their mobile devices. During an initial setup phase, users submit their loyalty account barcode to the system. Then, whenever a user wishes to scan their loyalty card when making a purchase, instead of taking out their loyalty card, they open the CheckOut app on their phones. The app will display a barcode which they will use as their loyalty card for that purchase. For loyalty schemes that allow users to earn rewards points in addition to receiving discounts at the time of purchase, the user can also take a photo of the receipt or enter the total amount of points gained in the purchase to record their reward credit on the app as well. The app also allows users to check the balance of rewards they have accrued through the CheckOut system.

We describe our implementation of CheckOut in Section 6. Apart from the fairly simple pattern of interaction with the app described here, all functions included in the description of the scheme are carried out between the app and the server without additional user input or interaction. Our aim is for CheckOut to provide a user experience comparable to a typical loyalty program after the initial setup, allowing users to earn discounts and rewards as they would without CheckOut, while also protecting their privacy.

2.2 System Components

The CheckOut system is composed of two core parts. We give a brief overview of the functionality of each part here before addressing them in greater detail in Sections 4 and 5. Apart from the overall design of the system, our core technical contributions are in the mechanism for point redistribution described in Section 5.

Barcode distribution. The barcode distribution scheme forms the heart of CheckOut. This protocol ensures that a malicious server cannot unduly influence the loyalty card swapping process, ensuring that the CheckOut card swap does indeed behave as a scaled-up version of manual card swapping. We prevent the server from tampering with the randomness of the chosen cards via a coin-flipping protocol between the client and server to choose the barcode for each transaction.

In order to use CheckOut, a client must first register with the CheckOut server. To register, the client will need a valid loyalty account number at the store in question. For convenience, we display this account number as a barcode, and we will refer to it as a barcode throughout in this paper. The client registration step is also where the user is assigned a user ID by the server, and when the client generates any cryptographic secret key material needed. If a deployment also implements point redistribution, users generate long-term cryptographic secrets, as described in Section 5.

As internal state, the server keeps a list of (user ID, barcode, public key) tuples that form the leaves of a Merkle tree [47]. The root of this Merkle tree serves as a binding commitment to the contents of the loyalty card database, and an up-to-date version of the root is regularly distributed to CheckOut clients. Similar to other protocols that use Merkle trees to commit to lists of information, e.g., Certificate Transparency [20, 42, 43], the clients can gossip among themselves to ensure they have up-to-date Merkle roots. The same proofs used in Certificate Transparency can also be used

to ensure that new versions of the Merkle root are append-only extensions of previous versions.

Point redistribution. We describe mechanisms for users to privately record and repay rewards points that they have earned while using others' loyalty card barcodes. These schemes may not be applicable to all loyalty programs, but they often make for a useful addition. While these points are not always easily mapped to dollar values, there exist some programs where it is easy to think of the points as being easily exchanged for money. For example, in the loyalty program at Giant Food, every 100 points earned can be used for $1 off a future purchase, so each point can directly be assigned a value of 1 cent [1]. When the point redistribution feature is used, we augment the core card swapping protocol with additional steps to record the number of points earned in a given transaction, and we add a new *settling* protocol that allows users to retrieve how much they owe to, or are owed by, the system.

In our first of two schemes, providing security against a semihonest server, the server will keep a list of user's point balances. Whenever user i makes use of the loyalty card of another user j to buy x points worth of products, the server increases user i's balance by x and reduces user j's balance by x. The idea is that, since user i spent x points, they should be entitled to x points worth of rewards, but these rewards were actually given to user j, whose card was used. Thus, the system records that user j owes the system x reward points, while user i is owed x points. Our scheme will allow the server to handle this process privately, maintaining user balances and ensuring their integrity in the face of disruptive users, all without ever seeing the transaction values (that is, the values x) themselves.

In order to prevent a malicious server from having clients decrypt arbitrary messages during balance settling, we also need to change the clients' bookkeeping process when recording transactions, so clients can verify that they only decrypt correctly-computed balances. To facilitate this, each transaction will produce a cryptographic "receipt" that the server sends to the barcode owner when it next comes online. A client now keeps a local record of all its receipts in parallel with the server-recorded balance, and clients only reveal balances that match the one recorded locally.

2.3 Security Goals

The primary security goal for our system is to ensure that CheckOut allows users to receive random loyalty card barcodes from the server while minimizing the impact the server can have on the choice of barcode given to the user. Additional security goals also become relevant if CheckOut is used to track and repay reward point balances. This section describes these two sets of security requirements.

Ensuring random card swapping. It is important to ensure that neither a user nor the server can tamper with random barcode selection, lest malicious behavior cause the system to subtly provide different privacy guarantees to different users. For example, a malicious server could act as an honest card swapper for some users, while giving other users the same card every time, thereby not anonymizing their transactions. A more subtle attack would be for the server to partition the users of the system so that some users benefit from a large anonymity set while others have a smaller one. In a financially motivated attack, a server operator could more often give users the barcode for their own loyalty card, thereby potentially gaining more reward points than other users. On the other hand, a user who can bias which barcodes they receive can reduce the frequency with which some other user's barcodes are selected.

Our scheme will use a combination of a coin-flipping protocol to randomly select a barcode from a database of registered users and a transparency mechanism to ensure that the server does not tamper with the database. This suffices to ensure that, as long as the protocol completes, the choice of barcode given to each user will be random.

Note that our approach only achieves "security with abort," in that a party to the protocol who does not like the outcome can always abort. In our actual protocol, the client will learn the index i of the barcode selected prior to the server, meaning that clients can feasibly detect a server that regularly avoids distributing the barcode of some index i^*. We will discuss how to achieve stronger notions of security in Section 4.2 after presenting our core card swapping protocol.

Point redistribution. When CheckOut is deployed for loyalty programs that support reward point balances, we require two additional properties.

- **Point Confidentiality**: the server will not know the number of loyalty points used in any transaction, even if it maliciously deviates from the protocol. We will separately consider definitions and corresponding constructions that achieve point confidentiality for a semihonest and malicious server.
- **Balance Integrity**: the sum of balances across users of the system will always sum to zero, even in the presence of disruptive users. This means that it is always possible for users of the system to collectively settle their balances such that every user could be repaid the loyalty points they should have received while using others' loyalty barcodes. Moreover, malicious users should not be able to affect the balances of honest users whose loyalty cards they have not used.

While simple to state informally, it turns out that a great deal of care must be taken to formalize these definitions in a way that achieves meaningful security without relying on access to information held by the operator of the loyalty program. We formalize our definitions in Section 3.

Confidentiality and settling balances. Because users do not necessarily have a mechanism for directly exchanging money with each other, CheckOut can be used as a medium of exchange for settling balances. But in order to do this, the CheckOut server needs to see users' balances. While this is not necessarily a privacy problem, as the user's balance is the sum of various transaction values from when others used the user's loyalty barcode, it does introduce a potential security concern. A malicious server could tamper with balances, exploiting the settling feature to uncover user transactions of its choosing. Without information on the actual transactions taking place within the loyalty program and their timing, this kind of attack becomes difficult to mitigate.

Our approach addresses this challenge by introducing the concept of a *receipt* generated for each transaction entered into the CheckOut system. For each transaction, the owner of the barcode must verify the *receipt* before permitting updates to their balance. This added layer of security reduces the server's ability to arbitrarily tamper with user balances, but adds some bookkeeping complexity to the scheme. Thus, we have semi-honest and malicious secure versions of our scheme, depending on whether or not receipts are included. These two schemes satisfy different security definitions, as presented in Section 3.

2.4 Limitations and Deployment Considerations

This section addresses deployment considerations for the CheckOut system, including a discussion of what kinds of loyalty programs are appropriate candidates for a CheckOut deployment, as well as general limitations of CheckOut and potential strategies for mitigating them in practice.

CheckOut balances and real-world balances. CheckOut's security properties enforce that balances in CheckOut always remain *internally* consistent with users' inputs. They do not, however, prevent a malicious user from simply lying about the value spent at a store and introducing incorrect, yet internally consistent, data into the system. We have no way of ensuring that balances input to CheckOut correspond to the amounts that users actually spent in real life. This is an inherent limitation of approaches that do not externally verify the value of users' purchases at a store to prevent incorrect inputs, which would require cooperation and integration with the store. We explicitly aim to avoid such kind of cooperation and must therefore deal with this limitation.

In light of this inherent limitation, CheckOut can instead provide usability features that nudge and encourage users to participate honestly, even if the protocol cannot fundamentally do anything to defend against liars. For example, receipt values can be read off of receipts via OCR, making it harder to simply type in a fake number. Regardless of how users input transaction information, CheckOut aims to provide users with a reliable bookkeeping aid, and real-world disagreements about what numbers are entered need to be handled out-of-band. We present one potential approach for handling this kind of abuse in Appendix E.

Balance settling and repayment in practice. To entirely avoid issues with point confidentiality, clients could simply record their balances using CheckOut and handle settling out of band, therefore not revealing aggregate balances to the server. But this may be difficult in settings where strangers who live very far from each other use the same instance of the CheckOut system, which is the ideal setting for building a large anonymity set of users. To address this, CheckOut includes settling functionality that requires users to reveal aggregate balances to the server.

Because CheckOut is deployed without cooperation from the retailer in question, we have no way of transferring loyalty points between users directly, and we handle repayments by, e.g., equating numbers of points to dollar amounts. Since this feature requires more knowledge of user balances, as well as integration with some kind of payment processing mechanism, some deployments of CheckOut may wish to turn this feature off and just rely on the randomness of card swaps to allow for a best-effort balancing of point balances over time. But in this case, over time heavy spenders will tend to be undercompensated by CheckOut, and light spenders will tend to be overcompensated.

An important parameter in the CheckOut system is how frequently balance settling occurs. Longer windows increase ease of use, and also allow time for more transactions to enter the system, increasing the degree to which any given transaction's impact on a user's balance is masked. However, because balances can increase in magnitude over time, longer windows are also associated with higher risk. Malicious users can simply walk away with positive balances, and a malicious server responsible for payment processing can fail to deliver payouts. As no technical solution to this problem exists, the balance settling interval must be set such that the benefits of deciding to defect are sufficiently low.

Appropriate loyalty program structures. What can be achieved with card swapping depends, to some extent, on the structure of the underlying loyalty program. For example, some loyalty programs allow users to both accrue and spend points by simply scanning their card. In such a program, allowing others to make purchases with a user's card also means permitting them to spend that user's loyalty points, as no additional credential is required for point redemption. On the other hand, certain programs allow users to accrue points using a card but have a separate online portal for spending points. The latter category of loyalty programs are amenable to schemes that handle redistribution of loyalty points, while the best we can hope for in the former category is a scheme that redistributes cards without addressing the points. Although CheckOut's more technical contributions are focused on facilitating point redistribution, the system as described here can be used for either kind of loyalty program.

Clients' trust in the database. The security of CheckOut relies on clients trusting that updates to the server's database represent valid user registrations. If, for example, a malicious server operator simply floods the database with many user records sharing one barcode, they can funnel a large portion of transactions into one account. In some cases, this can be solved easily by publishing the database of user records publicly. Clients can also mitigate this issue by observing the distribution of barcodes they receive, as our system ensures that each barcode is selected uniformly at random from all accounts in the system.

3 FORMALIZING SECURITY DEFINITIONS

This section formalizes the syntax of card swapping schemes and introduces the definitions of point confidentiality and balance integrity that our card swapping schemes will seek to meet.

Notation. Before we continue, we briefly summarize the notation used throughout this paper. By $x \leftarrow f(y)$ we denote assignment to x of the value of $f(y)$, and by $x \xleftarrow{R} S$ we denote assignment to x of a value chosen uniformly at random from a set S. Tables, denoted with capital letters and initialized as $T \leftarrow \{\}$, act as key-value stores, where values can be accessed by $T[\mathsf{key}]$. Vectors, denoted $\vec{v}$ and initialized as $\vec{v} \leftarrow []$, are ordered lists of values. Elements can be added to vectors via $\vec{v}.\mathsf{add}(\mathsf{val})$. A function $\mathsf{negl}(x)$ is *negligible* if, for all $c > 0$, there exists an x_0 such that, for all

$x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We use $\perp$ as a special character indicating protocol failure.

In addition, many functionalities we introduce will rely heavily on state relating to a *shopper* who is processing a transaction, and the *barcode owner*, who supplies the loyalty account barcode. By convention, we suffix variables related to these two parties with "s" and "b", respectively (e.g. pks and pkb). Finally, for interactive protocols between a client and server, we will denote each party's view of the protocol transcript as optional additional outputs, named View_C and View_S, respectively.

3.1 Syntax

A *point tracking scheme* is a collection of four interactive protocols, (RegUser, CardSwap, TxProcess, BalSettle), and five algorithms, (ClientInit, ServerInit, ProcessRct, UpdateBalClient, UpdateBalServer). It is defined with respect to a barcode space $\mathcal{B}$, public and private key spaces $\mathcal{K}_{\mathsf{Enc}}$ and $\mathcal{K}_{\mathsf{Dec}}$, a ciphertext space $\mathcal{C}$, a hash space $\mathcal{H}$, signing and verifying key spaces $\mathcal{K}_{\mathsf{Sign}}$ and $\mathcal{K}_{\mathsf{Ver}}$, a signature space $\mathcal{S}$, a random seed space $\mathcal{R}$, and a masked balance space $\mathcal{M}$.

- $\mathsf{ClientInit}(\mathsf{bc}, 1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{bal})$: takes a loyalty account barcode $\mathsf{bc} \in \mathcal{B}$ and a security parameter, and outputs an encryption key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}_{\mathsf{Enc}} \times \mathcal{K}_{\mathsf{Dec}}$, as well as bal, a balance representation initialized to zero.
- $\mathsf{ServerInit}(1^\lambda) \rightarrow (k_{\mathsf{sig}}, \mathsf{vk}, \mathsf{DB}, \mathsf{MTree})$: takes a security parameter and outputs a signature key pair $(k_{\mathsf{sig}}, \mathsf{vk}) \in \mathcal{K}_{\mathsf{Sign}} \times \mathcal{K}_{\mathsf{Ver}}$. It also outputs an empty database DB of user records $(\mathsf{uid}, \mathsf{bc}, \mathsf{pk}, \mathsf{mbal}) \in \mathbb{N} \times \mathcal{B} \times \mathcal{K}_{\mathsf{Enc}} \times \mathcal{M}$, where mbal is each user's masked balance, and an empty merkle tree MTree which stores hashes of (uid, bc, pk) entries.
- $\mathsf{RegUser}\langle C(\mathsf{bc}, \mathsf{pk}), S(\mathsf{DB}, \mathsf{MTree})\rangle \rightarrow \langle \mathsf{uid}, (\mathsf{DB}', \mathsf{MTree}')\rangle$: the client inputs a barcode *bc* and a public key $\mathsf{pk} \in \mathcal{K}_{\mathsf{Enc}}$, and receives a user ID $\mathsf{uid} \in \mathbb{N}$. The server inputs their database DB of users and a MTree computed over the user database, and receives an updated database DB′ and merkle tree MTree′ where the new user has been added.
- $\mathsf{CardSwap}\langle C, S(\mathsf{DB}, \mathsf{MTree})\rangle \rightarrow \langle(\mathsf{uidb}, \mathsf{bc}, \mathsf{pkb}), \mathsf{uidb}\rangle$: the client has no input, and receives a randomly chosen barcode owner's record: (uidb, bc, pkb). The server inputs its database DB and MTree, and receives uidb (and can recover bc and pkb via database lookups).
- $\mathsf{TxProcess}\langle C(x, \mathsf{pks}, \mathsf{pkb}), S(k_{\mathsf{sig}}, \mathsf{pks}, \mathsf{pkb})\rangle \rightarrow \langle \mathsf{ptrct}, \mathsf{rct}\rangle$: the shopper client inputs a number of spent points x, the agreed upon barcode owner's public key pkb, and its public key pks. It receives a plaintext *receipt* ptrct from the server. The structure of ptrct and rct is explained in detail in Section 5. The server takes as inputs its signing key k_{sig}, and receives a masked rct as output.
- $\mathsf{ProcessRct}(\mathsf{sk}, \mathsf{rct}, \mathsf{vk}) \rightarrow (x, \mathsf{ptrct})/\perp$: takes as input the client's secret key sk, a transaction receipt rct and the server's verification key vk. If the transaction is accepted, this algorithm outputs the number of points x used in the transaction. It also outputs a plaintext representation of the receipt, to be used at settling time. If the transaction is rejected, this algorithm outputs $\perp$.
- $\mathsf{UpdateBalClient}(\mathsf{bal}, x, \mathsf{rct}) \rightarrow (\mathsf{bal}')$: takes as input a client's balance bal, a number x of points used, and a corresponding transaction receipt rct. It outputs an updated bal′.
- $\mathsf{UpdateBalServer}(\mathsf{DB}, \mathsf{uids}, \mathsf{uidb}, \mathsf{rct}) \rightarrow \mathsf{DB}'$: takes as input the server's database of user records DB, shopper and barcode owner user IDs uids and uidb, and a transaction receipt rct. This algorithm outputs an updated database where both users' balances are updated to reflect the transaction.
- $\mathsf{BalSettle}\langle C(\mathsf{sk}, \mathsf{bal}, \vec{\mathsf{ptrct}}), S(\mathsf{DB}, \mathsf{vk}, \mathsf{uid})\rangle \rightarrow \langle \perp, x/\perp\rangle$: the client inputs its secret key sk, its balance bal, and a vector of plaintext receipts $\vec{ptrct}$. The server takes as input its database DB, signature verification key vk, and the uid of the client settling. The server outputs an accepted balance value x, or rejects and outputs $\perp$. The client receives no output.

Note that this syntax captures all three protocols: card-swapping only, semihonest-secure point tracking, and malicious-secure point tracking, although schemes targeting different security and functionality goals may not use certain aspects of it.

3.2 Security Definitions

We now describe definitions for the security notions introduced informally in Section 2. We defer the full definitions to Appendix A.

Point confidentiality. Both our semihonest and malicious secure point confidentiality games allow the adversary to control a set of malicious users of its choosing. After a setup phase, the adversary can compel users of its choosing to run the card swapping protocol with the server, using transaction values of the adversary's choosing. If both the shopper and barcode owner in a transaction are honest users, the adversary specifies two potential transaction values (x_0, x_1), and the challenger executes the transaction with the value x_b specified by its input b. If either party is malicious, the challenger always executes the transaction with the value x_0, as otherwise the adversary could simply learn the value of b by revealing which point value was processed.

After completing as many transactions as the adversary wants, the confidentiality games allow the adversary to perform a settling operation, where each user settles with the server. The games enforce the condition that $\sum x_0 = \sum x_1$ over all transactions where each honest user's loyalty account was used, as otherwise the adversary can simply infer b from the balances. After the settling phase, the adversary outputs a distinguishing bit b', its guess at the challengers input b.

The difference between the semihonest and malicious games depends largely on who controls the server during the transaction and settling phases. In the semihonest game, the challenger always controls the server and runs the card swapping protocol honestly, but it sends the adversary all server secrets and the view of the server in each protocol it runs with any client, as well as the view of adversary-controlled clients. In the malicious game, after an honest setup phase, the adversary is sent all server secrets and directly controls the server and malicious clients throughout the rest of the game. The malicious game also includes additional bookkeeping to deal with handling receipts, which are not relevant to the semihonest setting. We present the full semihonest and malicious definitions as Definitions A.1 and A.2, respectively, in Appendix A.

Balance integrity. In the balance integrity experiment, the adversary controls a set of malicious users, and the experiment controls the remaining users, as well as the server.

The adversary is allowed to make transactions of its choosing with the server (playing the role of a malicious user) or compel honest users to make transactions with values of the adversary's choosing. It can also compel all users to settle their balances with the server.

The goal of the adversary is to break the integrity of the scheme in one of two ways:

- Break the invariant that all balances in the system always sum to zero.
- Unduly modify the balance of an honest client.

There are thus two ways for the adversary to win the experiment, and both of these conditions are checked when settling. The first is if the sum of all users' balances does not come out to zero when users settle. Meeting this condition means that the adversary has violated the condition that the overall system must remain solvent, so it is not possible for all users' debts to be repaid.

The second way for the adversary to win is to illegally modify the balance of any individual honest user, regardless of the sum of all balances. Formalizing this notion is tricky, as the adversary *is* allowed to modify the balance of an honest user in the course of the honest protocol when a malicious user is given the barcode of an honest user and records the value of the transaction made with that barcode. Our definition handles this situation by considering an honest user's balance "tainted" when a malicious user is given the honest user's barcode. In these cases, the adversary is expected to be able to subtract an arbitrary amount from the user's balance, and the relevant security consideration is the first condition, that balances remain consistent overall. However, for those users whose balances are not directly touched by the adversary, the experiment separately keeps track of their expected balance according to honest transactions with other users. The adversary wins if it can cause the balance of one of these users to deviate from the expected amount when settling, or if it can cause any user to fail to settle their balance, i.e., make the settling protocol output $\perp$. We also define a weakened version of balance integrity that does not count settling failure as an adversary winning because it has not caused a client to output an incorrect balance. These definitions are presented in Definition A.3 in Appendix A.

4 SCALING CARD SWAPS

This section describes the core CheckOut system, including both the client registration and card swapping processes, but excluding point redistribution. We will describe the process of requesting and retrieving a loyalty barcode with respect to a fixed Merkle tree with root R that commits to a list of N barcodes bc_i for $i \in \{1, \ldots, N\}$.

4.1 Retrieving a Loyalty Barcode

A naïve way to implement CardSwap, given a database of loyalty cards, would be for the server to choose one at random each time a client requests a barcode. But this allows a malicious server to preferentially choose certain barcodes or provide different levels of anonymity to different users, as described in Section 2. Allowing the client to choose the barcode presents a similar problem. Thus, our CardSwap implementation ensures that each client receives a random barcode by having the index of the barcode chosen via randomness jointly selected by the client and server via a commit-and-reveal protocol. Once the index of the barcode is fixed, the server can provide a Merkle inclusion proof relative to Merkle root R that it has provided the client with the correct barcode.

More precisely, the CardSwap protocol begins with a client hello message where the client sends a commitment to a random value $r_c \xleftarrow{R} \{1, \ldots, N\}$. We instantiate our commitments using a hash function $H : \{1, \ldots, N\} \times \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ modeled as a random oracle, where λ is a security parameter. The client computes

$$r_c \xleftarrow{R} \{1, \ldots, N\}$$
$$r \xleftarrow{R} \{0,1\}^\lambda$$
$$\mathsf{com} \leftarrow H(r_c, r)$$

and sends com to the server. The server responds by choosing and sending server randomness $r_s \xleftarrow{R} \{1, \ldots, N\}$.

The client's second message is an opening of the commitment, which is achieved by sending the server r_c and r, allowing the server to verify that $\mathsf{com} = H(r_c, r)$, and for both parties to compute $i \leftarrow r_c + r_s \mod N$.

Finally, the server responds to the client with the user entry $C_i = (\mathsf{uid}, \mathsf{bc}, \mathsf{pk})$, and a Merkle inclusion proof that C_i is the ith element committed to by R. After verifying the proof, the client can present bc as the barcode to be used at the grocery store checkout.

All communication in the barcode selection protocol incurs constant ($O(1)$) communication size complexity, with the exception of the generated Merkle inclusion proofs. These scale with the depth of the Merkle tree, or logarithmically ($O(\log(N))$) with the number of users of CheckOut.

A formal diagram of this CardSwap functionality is deferred to Figure 10 in Appendix B. By implementing this, as well as ClientInit, ServerInit, and RegUser as described above, we can model our card swapping protocol as a point swapping scheme, where all other functionalities are null protocols.

Intuitively, this scheme ensures that neither the client nor the server can affect the randomness of barcode selection. A malicious client cannot affect the random choice of barcode because the client's randomness r_c is chosen before that of the server, and if the client could change the randomness it reveals when opening the commitment com, this would break the binding property of the commitment scheme. A malicious server cannot affect the choice of barcode because the server's randomness r_s is chosen before the server sees r_c, by the hiding property of the commitment. Moreover, once i is selected, the server cannot change the corresponding choice of barcode because of the binding of the Merkle tree commitment.

4.2 Extension: Security Against Aborts

Our scheme ensures that the distribution of loyalty barcodes output by the protocol is uniformly random, even if the client or server maliciously deviate from the protocol. However, an adversary can still abort the protocol. In the case of the client, this is the best we can hope for, because a user who does not like the barcode they have received can simply not use the barcode to make their store

purchase and try again later. Features external to the protocol, such as rate limiting of client requests, can mitigate this kind of behavior.

On the server side, we can achieve stronger security, albeit at a higher cost. In our current protocol, since the client learns the final index i before the server does, this means that clients can detect servers who repeatedly abort when presented with a particular index. However, we may wish to strengthen the scheme so that the server's decision to abort *cannot* depend on the randomly selected index i.

We can achieve this stronger security notion by introducing PIR into the scheme [18]. A PIR scheme allows a client to retrieve an element from a database held by the server without the server learning which element the client retrieved. If the client retrieves the user record C_i using PIR, then it does not need to reveal r_c or i to the server and the server cannot base its decision to abort on this information. If we wish to further ensure that the client retrieves C_i from the database and does not learn additional information, e.g., other barcodes other than the one held in C_i, we could instead use a symmetric PIR (SPIR) scheme [34], which extends PIR to place limits on what the client learns from the protocol as well.

Adding PIR (or SPIR) to the protocol requires two other changes. First, since the server will not know i, it will not know the index for which it must provide a Merkle inclusion proof. To get around this, the server precomputes an inclusion proof π_i for each $C_i \in \mathsf{DB}$, and the server's PIR database includes tuples of the form (bc_i, π_i). This increases server storage costs from $O(N)$ in the database size to $O(N \log N)$ because the server now needs to hold N inclusion proofs, each of which consist of $O(\log N)$ hashes.

The second addition to the protocol is a more elaborate proof from the client. Since the client no longer needs to tell the server r_c or i, there is nothing in the protocol as described to prevent the client from picking an i of its choice. While this may be an acceptable trade-off, we can also avoid this problem by having the client provide the server with a proof π_c that the PIR query it sends is to an index i that is computed honestly from r_c and r_s. This proof can be constructed from generic composition of sigma protocols or other zero knowledge proof systems for general circuits [21, 22, 31], but it would lead to a significant increase in the concrete costs of the protocol.

We do not explore this direction further because hiding which user's barcode is used from the server also prevents the redistribution of loyalty points, which is the focus of the next section. A deployment of CheckOut focusing solely on card swapping, however, may wish to adopt this PIR approach. Our implementation uses the simpler scheme that allows servers to abort based on the value of i, but ensures that the intended value of i is visible to clients before this happens, allowing for out-of-band auditing of this kind of server misbehavior.

5 PRIVATE REDISTRIBUTION OF LOYALTY POINTS

In this section, we show how to expand the CheckOut system to handle tracking and repayment of loyalty rewards points for programs that offer such a feature. As all information exchanged in the point tracking scheme described below is constant size, we incur O(1) communication overhead for this protocol extension.

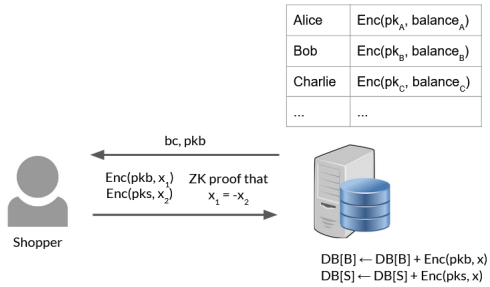


Figure 2: Alice requests a barcode from the server and allows the server to update its balances.

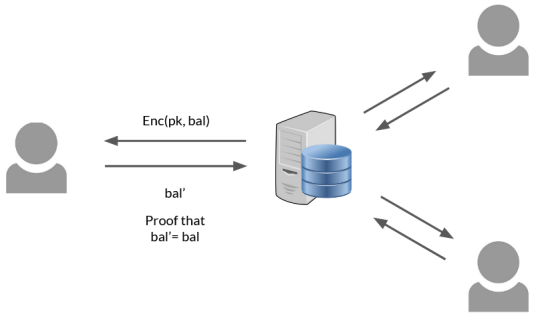


Figure 3: To settle balances, the server sends each user their encrypted balance. The users respond with their decrypted balances, and corresponding proofs that the decryptions are correct.

5.1 A Semihonest Solution

We begin by describing a solution that provides weak balance integrity against malicious clients but only provides point confidentiality against a semihonest server. This scheme trades weaker security properties for a simpler functionality, improved performance, and reduced client-side storage. We defer a formal description of our semihonest CheckOut scheme to Figure 11 in Appendix B. In Section 5.2, we describe a scheme that builds on the intuition of this scheme and achieves point confidentiality against a fully malicious server.

Additional client registration requirements. The only change needed to support reward points in the registration phase of CheckOut is that users additionally generate a key pair for an additively homomorphic public key encryption scheme and upload the public key to the server during the RegUser protocol.

Privately recording transactions. When recording point balances, the server keeps a database of N balances, where balance i is encrypted under the public key of user i using an additively homomorphic encryption scheme. The server additionally keeps users' public keys in the Merkle tree, and sends the public key of the barcode owner along with the barcode when a user requests one.

A view of processing a transaction while privately recording loyalty point balances is shown in Figure 2. To add support for

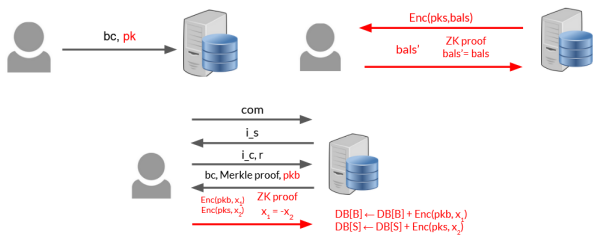


Figure 4: The complete semihonest CheckOut protocol summarized. Shown counter-clockwise from the top left: user registration, transaction processing, and balance settling. Protocol components only included when supporting loyalty points tracking are shown in red.

privately recording reward point values, a user additionally uploads the following to the server after scanning their barcode and checking out.

- The negated transaction loyalty point total $-x$, encrypted under their own public key.
- The transaction loyalty point total x, encrypted under the barcode owner's public key.
- A zero-knowledge proof asserting that one of these ciphertexts encrypts the additive inverse of the other.

The server then verifies this proof and adds each ciphertext to the respective users' encrypted balances. This reflects the fact that, after each transaction, the barcode owner will owe the shopper the number of loyalty points acquired. If a proof fails to verify, the server responds to the client with the message $\perp$.

We now describe the details of how to instantiate the encryption scheme and proofs needed to formalize the protocol sketched above.

Encryption scheme. We instantiate our additively homomorphic public key encryption scheme with multiplicative El-Gamal encryption [25]. To encrypt a point total, a client will generate a ciphertext of the form $(g^y, g^m h^y)$, where g is a public parameter, y is a random mask, $h = g^x$ is a public key with corresponding private key x, and m is the message. Decryption will reveal the value g^m, not m. But in our application, m is a relatively small integer representing the number of loyalty points used, so given g we can quickly compute the discrete logarithm of g^m via the baby-step giant-step algorithm [55]. In addition, this scheme is additively homomorphic: given ciphertexts $(g^{y_1}, g^{m_1} h^{y_1})$ and $(g^{y_2}, g^{m_2} h^{y_2})$, we can compute

$$(g^{y_1}, g^{m_1} h^{y_1}) \cdot (g^{y_2}, g^{m_2} h^{y_2}) = (g^{y_1+y_2}, g^{m_1+m_2} h^{y_1+y_2}), \quad (5.1)$$

which is an encryption of $m_1 + m_2$.

Proof of plaintext equality. At transaction time, the client sends two ciphertexts to the server: $(C_{11}, C_{12}) = (g^{y_s}, g^{m'} h_s^{y_s})$ encrypted under the shopper's public key h_s, and $(C_{21}, C_{22}) = (g^{y_b}, g^m h_b^{y_b})$, encrypted under the barcode owners public key h_b. The client must prove to the server it knows m, m', y_s, and y_b such that the following hold.

$$C_{11} = g^{y_s} \quad (5.2)$$

$$C_{12} = g^{m'} h_s^{y_s} \quad (5.3)$$

$$C_{21} = g^{y_b} \quad (5.4)$$

$$C_{22} = g^m h_b^{y_b} \quad (5.5)$$

$$C_{12} C_{22} = h_s^{y_s} h_b^{y_b} \quad (5.6)$$

The first four relations ensure that the ciphertext is well-formed. The fifth holds if and only if $m' = -m$.

In order to construct this proof, we use the "generic linear" generalization of Schnorr's protocol [53, 54] as described in [7], made non-interactive with the Fiat-Shamir transform [28].

Settling Balances. The implementation of BalSettle is diagrammed in Figure 3. To settle, the server sends each user their encrypted balance. That is, the server sends the client a ciphertext of the form $(C_1, C_2) = (g^y, g^m h^y)$, where $h = g^x$, and x is the client's private key. The users then decrypt their balance and send it back to the server, along with a zero-knowledge proof that they are providing the correct decryption of the ciphertext. Note that when the client reveals the decryption g^m, the server can compute

$$C_2 / g^m = h^y = (g^x)^y = (g^y)^x = C_1^x.$$

The client's supplied decryption is correct if and only if it can prove knowledge of an x such that its public key $\mathsf{pk} = g^x$ and $C_2/g^m = C_1^x$. Thus, the client's zero-knowledge proof is simply a proof that $(C_1, \mathsf{pk}, C_2/g^m)$ is a valid Diffie-Hellman triple, which can be proven via a Chaum-Pedersen proof [17]. The server verifies the proof, outputting $\perp$ if the proof fails to verify. As all messages sent are constant size, this operation incurs $O(1)$ communication overhead.

For ease of reference, we summarize the full semihonest protocol in Figure 4.

Security. The point confidentiality of this scheme follows from the semantic security of the El-Gamal public key encryption scheme and the zero knowledge property of the NIZK used, as these two properties ensure that everything sent to the server during the card swapping protocol reveals nothing about the underlying transaction amounts. Weak balance integrity relies on the soundness of the NIZKs used during transactions and at settling time to prevent users from lying about the well-formedness of their transactions or deceiving the server about their balances. The balance integrity is weak only because the client needs to compute a small discrete logarithm to decrypt their balance. If a malicious user makes a transaction value of astronomically large size, the client will not be able to complete this operation efficiently. The scheme can be strengthened to avoid this possibility by introducing range proofs into transactions to limit the size of a given transaction, but we do not include this to avoid the additional performance cost, as our second construction will meet the more demanding balance integrity definition. We prove the following theorems in Appendix C.

Theorem 5.1 (Semihonest Point Confidentiality). *Assuming the semantic security of El-Gamal encryption and the zero-knowledge*

property of the NIZKs used, our semihonest loyalty card swapping protocol with private redistribution of loyalty points satisfies semihonest point confidentiality (Definition A.1).

Theorem 5.2 (Weak Balance Integrity)**.** *Assuming the NIZKs used are proofs of knowledge, our semihonest loyalty card swapping protocol with private redistribution of loyalty points satisfies weak balance integrity (Definition A.3).*

Malicious attack on point confidentiality. Observe that this scheme does nothing to verify that a balance sent by the server to a client is the actual balance of points that client has accrued over time. A malicious server can take advantage of this by asking clients to decrypt arbitrary ciphertexts, claiming that the provided ciphertext is the client's balance. In particular, the server could take the ciphertexts produced in individual transactions and ask clients to decrypt them, thereby breaking point confidentiality.

5.2 Achieving Malicious Security

At a high level, our malicious security scheme proceeds as follows. Instead of keeping an encrypted balance for each client, the server keeps a group element mbal that represents a masked balance for each client. During the TxProcess protocol, the client and server agree upon a barcode owner ID in the same way as the semihonest protocol. However, instead of uploading a transaction encrypted under two different keys, the client notes the transaction amount locally, and sends a masked version of the transaction amount to the server. The receipt generated by this transaction is stored by the server until the barcode owner comes online, at which point the server relays the receipt and the barcode owner verifies its correctness in ProcessRct. The server then updates the masked balances for the accounts involved in the transaction. The server also informs the shopper client that its transaction was successful, and the shopper updates its own plaintext and masked balances. Finally, during BalSettle, each client reveals their balance to the server, and provides receipts to prove that it knows all transaction and mask values used in creating the masked balance held by the server. The full scheme is formalized in Figure 5.

Note that our version of CheckOut that provides malicious security differs from the one outlined in Section 5.1 in that each transaction is processed by both the shopper and loyalty account owner, and the transaction is not complete until the barcode owner successfully executes ProcessRct. By introducing this extra client-side overhead, we are able to ensure that a malicious server is unable to tamper with balances in a way that breaks point confidentiality. In particular, the server cannot generate false transactions as a function of previous transactions. We present the scheme in more detail below.

Client registration and server data. During ServerInit, in addition to initializing a DB of users and corresponding merkle tree, the server generates a long-term signature key. As in the semihonest scheme, the client generates an encryption key pair in ClientInit, and shares the public key in RegUser, but now the encryption scheme used must satisfy CCA security.

Transaction processing. Our implementation of TxProcess proceeds as follows. In each run of the protocol, the client provides the server with a masked transaction value g_i^{mx} where $g_i \in G$ is a random generator of the group G and $m \in \mathbb{F}_q$ is a random mask on the transaction value x. To pick g_i, the server selects a random bit string s to share with the client. Both parties hash this string to the group element, $H(s) = g_i$, where H is modeled as a random oracle. The client then selects a random mask exponent $m \xleftarrow{R} \mathbb{F}_q$ and encrypts the mask m, point value x, and bit string s under a CCA secure encryption scheme, using the barcode owner's public key pk_b, to get a ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}_b, (m, x, s))$. The shopper then computes the values $\mathsf{hm} \leftarrow h^m$ and the masked transaction value $\mathsf{mval} \leftarrow g_i^{mx}$ which, together with ct, serve as a receipt for the transaction. Here, $h \in \mathbb{G}$ is an additional public parameter such that the discrete log between g and h is unknown.

Since g_i is the output of a random oracle, g_i and h are group elements between which the discrete logarithm is unknown. Using the generic zero-knowledge protocol for non-linear relations presented in [7, 53, 54], made non-interactive by the Fiat-Shamir transform [28], the shopper can prove knowledge of m and x to show that the values it has sent the server are well-formed according to the protocol specification.

The server verifies this proof and generates a signature σ on (hm, s), which it sends to the shopper to be presented at balance settling time. The server stores the values ct, hm, mval, s as a receipt for the transaction, to be communicated to the account owner when they come online. Later, when the shopper client gets word from the server that the receipt was processed successfully, it adds x points to its local balance, and updates its local copy of the masked server-side balance.

Receipt bookkeeping. When a client comes online, it checks with the server if there are any unprocessed receipts associated with its account. If so, it performs the RctProcess protocol in Figure 5. In this protocol, the server forwards the receipt, along with the signature σ. The client then decrypts ct to recover the values (m, x, s), computes $g_i = H(s)$, and checks that $\mathsf{hm} = h^m$ and $\mathsf{mval} = g_i^{mx}$, rejecting any malformed transactions. The client then notifies the server that it accepts the receipt, subtracts x from its local balance bookkeeping, and updates its copy of the masked server-side balance. The client retains a plaintext receipt ptrct of its decrypted values m, x, s, as well as mval, hm, and σ for each transaction, until settling time.

Once notified that a client has accepted a transaction, the server updates the shopper's and barcode owner's balances as follows.

$$\text{Shopper bal} \leftarrow \text{Shopper bal} \cdot \text{mval}$$

$$\text{Barcode owner bal} \leftarrow \text{Barcode owner bal} \cdot (\text{mval})^{-1}$$

The server also informs the shopper that the transaction was successful, at which point the shopper adds x to its plaintext balance and updates its masked balance copy. UpdateBalClient and UpdateBalServer are detailed in Figure 5.

Balance settling. Our BalSettle protocol is as follows. The goal of the protocol is to reveal a user's aggregate balance to the server, without revealing any m_i or x_i values for each transaction $i \in \{1, \ldots, n\}$. Here, n represents the number of transactions that use that user's loyalty account. At settling time, each user has a balance x, consisting of transactions $\{x_i\}$, such that $x = \Sigma_{i=1}^{n} x_i$. To settle, each user sends the following information: their balance x, a tuple

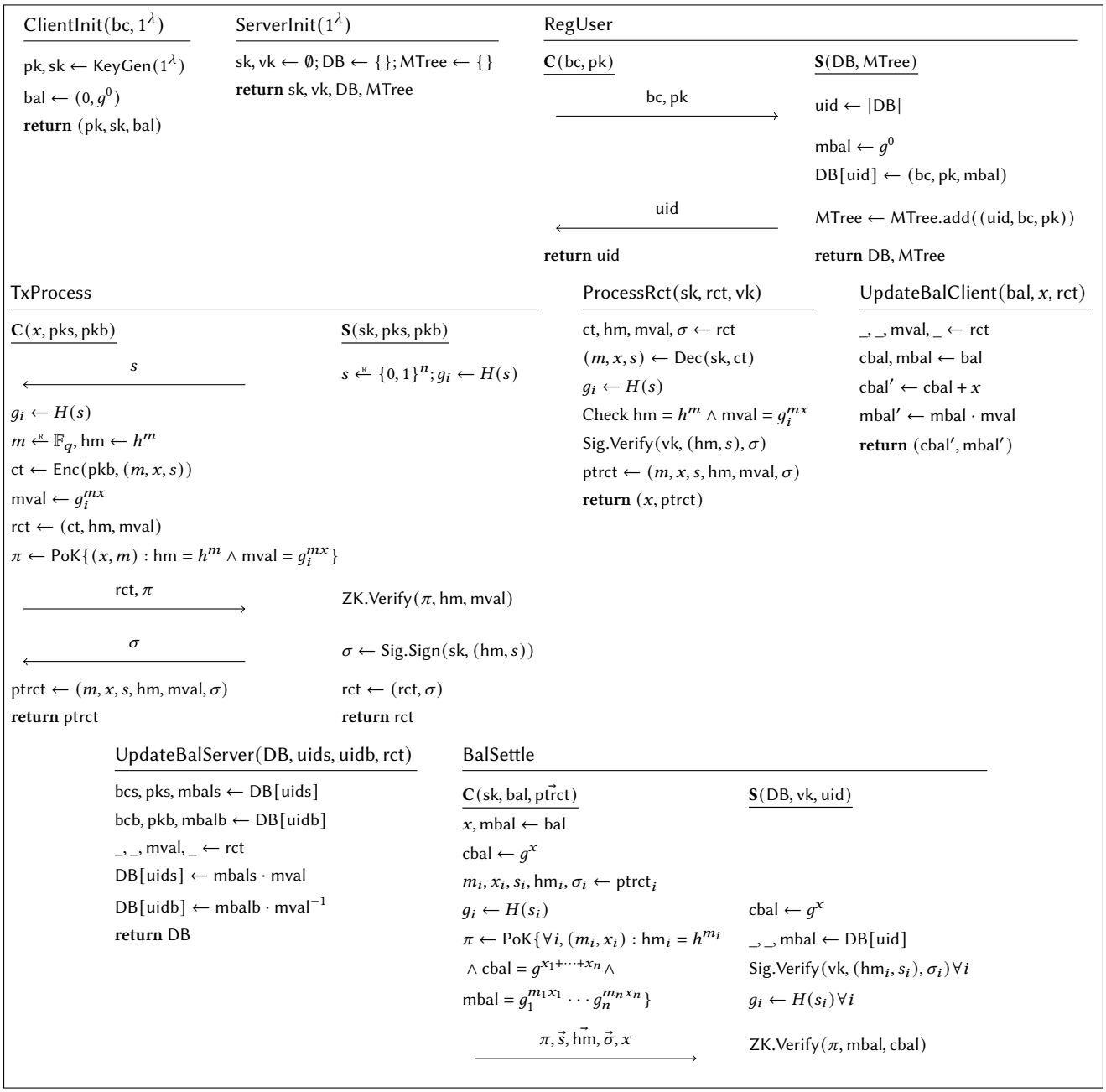


Figure 5: Malicious security scheme formalization.

(hm$_i$, s_i, σ_i) for $i \in \{1, \ldots, n\}$, and a proof π_{settle} of the correctness of the provided balance.

Since, for each user, the server can compute bal = $g^x = g^{x_1+\cdots+x_n}$ for a public generator g, and holds a masked balance mbal of the form

$$\text{mbal} = g_1^{m_1 x_1} \cdots g_n^{m_n x_n},$$

the goal of the proof π_{settle} is to show the server that the client-provided balance x is consistent with what the server holds. To this end, the client provides a zero knowledge proof of knowledge for the statement

$$\begin{aligned}\{(x_1, \ldots, x_n, m_1, \ldots, m_n), &\text{hm}_1, \ldots, \text{hm}_n, \text{mbal} : \\ &\text{bal} = g^{x_1+\cdots+x_n} \\ &\text{mbal} = g_1^{m_1 x_1} \cdots g_n^{m_n x_n} \\ &\text{hm}_i = h^{m_i}, \text{ for } i \in \{1, \ldots, n\}\}.\end{aligned}$$

The server verifies all the signatures σ_i, as well as the proof π, before accepting the client's claimed balance x as authentic.

Security. The malicious point confidentiality of this scheme follows from (1) CCA security of the underlying encryption scheme, which ensures confidentiality and integrity for (m, x, s), (2) the zero-knowledge property of the NIZKs used, which reveals nothing about m or x, and (3) the hardness of DDH in the group G, which ensures hardness of computing discrete logs between random oracle hash outputs in G. Balance integrity relies on the soundness of the NIZKs used, which ensures that receipts and balances are well-formed, and the existential unforgeability of the signature scheme and hardness of DDH in G, which ensures balances presented at settle time are composed of real transactions. We prove the following theorems in Appendix D.

Theorem 5.3 (Malicious Point Confidentiality)**.** *Assuming the CCA security of the encryption scheme, the zero-knowledge property of the NIZKs used, and the hardness of DDH in the group G, our malicious loyalty card swapping protocol with private redistribution of loyalty points satisfies malicious point confidentiality (Definition A.2) in the random oracle model.*

Theorem 5.4 (Malicous Scheme Balance Integrity)**.** *Assuming that the NIZKs used are proofs of knowledge, the signature scheme is existentially unforgeable, and that the discrete logarithm problem is hard in the group G, our malicious loyalty card swapping protocol with private redistribution of loyalty points satisfies balance integrity (Definition A.3) in the random oracle model.*

6 IMPLEMENTATION AND EVALUATION

We implemented both our semihonest and malicious point redistribution protocols from Section 5, as well as our base barcode swapping protocol, in Rust. We use the `curve25519-dalek` crate [23] for group operations in its implementation of the curve25519 Ristretto group [3, 32]. We instantiate our CCA secure encryption scheme using hashed El-Gamal encryption [7, 25] with AES-GCM.

In our evaluation, we grouped point tracking scheme operations into four components: user registration, transaction processing, receipt distribution, and balance settling. User registration includes ClientInit and RegUser, transaction processing includes TxProcess, UpdateBalClient, and UpdateBalServer, receipt processing refers to ProcessRct, and balance settling refers to BalSettle. We then analyzed the time taken to perform each component of the Check-Out protocol. We also benchmarked registration and transaction processing for our basic barcode swapping protocol described in Section 4. We measured performance of client-side operations on a Moto G Stylus 5G phone running Android 12, and measured performance of server-side operations on an Intel Core i7-11700K processor @ 3.60 GHz running Ubuntu 20.04.6 LTS.

Computation costs. Execution times for each protocol step are reported in Table 1. We recorded computation costs only, not including any network latency. The slowest constant-time step of our malicious protocol required 4 milliseconds of computation on the client device, and under 0.6 milliseconds on the server. Because transaction processing involves generating and verifying a Merkle proof for the server and client, respectively, execution time should scale logarithmically with the number of users. This effect is visible in the base swapping protocol, shown in Figure 6a. However, in the point tracking protocols this difference was small, so we report transaction times as constant.

In the semihonest setting, balance settling requires computing a discrete logarithm. We implement the baby-step giant-step algorithm [55], which has $O(\sqrt{N})$ time complexity in size of the exponent or, in our case, $O(\sqrt{N})$ in the number of points, as shown in Figure 6c.

In the malicious setting, computational overhead is greater, particularly for clients, as balance settling requires generation and verification of a NIZK that grows linearly with the number of transactions. This is demonstrated in Figure 6b. In practice, if clients settle regularly enough to accrue less than 20-30 transactions before settling, the combined server and client settling overhead of our malicious scheme is under 100 milliseconds. In addition, receipt processing adds approximately 1ms of processing for each transaction.

Communication costs. Using the primitives provided by our cryptographic libraries and the protocol specifications as described in Sections 4 and 5, overheads for each step of communication are provided in Table 2. The sizes of all primitives are listed in Table 3. As standard barcodes are as long as 13 decimal digits [50], we assume barcodes are encoded as 64-bit integers (up to 19 decimal digits).

Communication overheads are the same for client setup across both point swapping protocols. We also find similar communication overheads in transaction processing. This is because the two protocols only differ in the last few messages exchanged, and the NIZKs exchanged are of similar size in both protocols: in a system with 100,000 users, transaction processing incurs 556 bytes of overhead in the semihonest setting and 656 bytes in the malicious setting. In the balance settling phase, however, the malicious setting requires an extra 544 bytes to be sent by the client for every transaction using their account. This is because each transaction adds secret values m_i, x_i which the client needs to prove knowledge of in order to settle. In practice, if balance settling occurs often enough to expect 20 transactions in each user's account, settling requires 10,948 bytes of overhead in the malicious setting.

7 RELATED WORK

Private loyalty programs. A great deal of research explores how to build privacy-preserving systems that can, in principle, be applied to customer loyalty programs. Anonymous credentials can give customers unlinkable identities that they can present when checking out at the store [11, 12, 16]. Privacy-preserving points can be distributed and redeemed using ecash [2, 10] or the uCentive system [49], which is specifically designed for loyalty programs. A long line of works starting with the notion of black-box accumulation by Jager and Rupp [37] studies how to design efficient privacy-preserving incentives [4, 6, 27, 33, 35].

Obfuscation. Unfortunately, most cryptographic privacy-preserving systems are designed with the assumption that the server is making an effort to protect users' privacy, even if it may later maliciously deviate from its declared intentions. This approach often fails to align with the reality where users are constantly surveilled with

		Registration	Transaction Processing	Receipt Processing	Balance Settling
Swap only	Client	0.1	33	N/A	N/A
	Server	52	1.3	N/A	N/A
Semihonest	Client	72	1959	N/A	—
	Server	113	323	N/A	131
Malicious	Client	108	3948	977	—
	Server	99	579	16	—

Table 1: Average execution times for each component of both protocols, measured in microseconds (μs). Elements of the point tracking scheme syntax are divided into these components as described above. Receipt processing is not defined in the semihonest setting. Balance settling times are excluded because execution time varies (see Figures 6c, 6b). Transaction times in our swapping protocol are show in more detail in Figure 6a.

		Registration	Transaction Processing	Receipt Processing	Balance Settling
Swap only	Client	8	68	N/A	N/A
	Server	0	$16 + 32 \cdot \lceil \log_2(N_c) \rceil$	N/A	N/A
Semihonest	Client	40	708	N/A	228
	Server	0	$44 + 32 \cdot \lceil \log_2(N_c) \rceil$	N/A	64
Malicious	Client	40	768	0	$68 + 544 \cdot N_t$
	Server	0	$144 + 32 \cdot \lceil \log_2(N_c) \rceil$	272	0

Table 2: Communication costs for each CheckOut component, measured in bytes. Elements of the point tracking scheme syntax are divided into these components as described above. Each row specifies how many bytes of data are *sent* from either the client or server in each operation. Here N_c is the number of clients registered in the system, and N_t is the number of transactions in the client's balance.

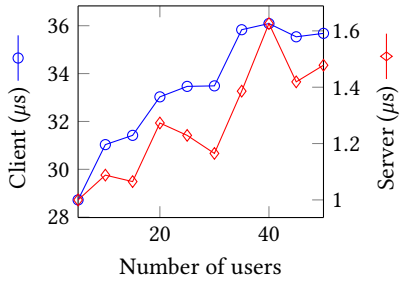
Primitive	Bytes
Barcode	8
Integer	4
Hash digest	32
Nonce	64
Group element	32
Group scalar	32
Nonce	12
Ciphertext	148

Table 3: Sizes of objects sent between client and server. A ciphertext always consists of an ElGamal encryption of a key, followed by an AES-GCM encryption of a 68-byte plaintext.

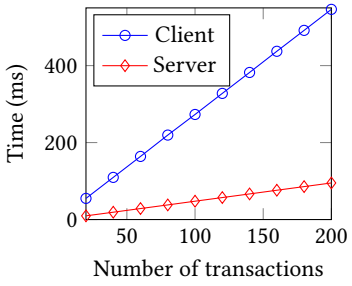
little apparent remorse on the part of surveillors. Nissenbaum and Brunton introduce the notion of obfuscation [8, 9], which they initially define as "producing misleading, false, or ambiguous data to make data gathering less reliable and therefore less valuable," to refer to a broad family of approaches to protecting privacy in the context of an adversarial and uncooperative surveillor. This includes diverse approaches ranging from physical world protections, like wartime pilots deploying chaff to deceive radar, to technical defenses like the celebrated Tor network [24].

The examples of obfuscation most relevant to this work in their motivation are the previous attempts at manual loyalty card swapping [13, 19] mentioned in Section 1. More similar in their approach, however, are those obfuscation schemes that create technological tools to combat privacy challenges. FaceCloak [45] prevents oversharing of personal data on social media. It shares fake data with social media platforms and makes the real data available only to those whom the user explicitly permits. This is achieved by encrypting and storing the real data on a third party server. CacheCloak [48] uses caching and carefully chosen cover traffic to offer its users a degree of location privacy. Vortex [44], conceived as an art project, allows users to explore the web through the eyes of others by swapping their locations and cookies, causing them to appear as someone else to the sites they visit. We see CheckOut as a new entry in this genre of obfuscation technologies that pry privacy back where possible in a world saturated with surveillance.

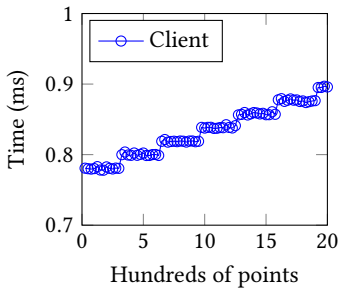
Private payment systems. Since loyalty points can be thought of as a proxy for dollars or other currency, another relevant set of tools are those used for privacy in digital payment systems. This includes the early works of Chaum [14–16] and subsequent work on e-cash mentioned above, as well as private transaction systems based on cryptocurrencies, e.g., Zcash, confidential transactions, etc. [36, 38, 46, 52]. While these systems achieve greater security in processing transactions, they do not achieve private assignment of aggregate repayment duties as CheckOut does. In principle, one could build a loyalty point redistribution system by using a privacy-preserving cryptocurrency to handle payment processing after settling balances, as CheckOut stops short of specifying how repayment of points due is implemented. Our CheckOut protocol could also be run entirely in a smart contract ecosystem such as Ethereum [56]. In either case, the privacy benefits of CheckOut would be independent of the benefits of the underlying payment system.



(a) Transaction processing time, in microseconds, by number of users in the barcode-swapping only scheme. Note the differing y-axes and y-intercepts.



(b) Server and client settling times by number of transactions in the malicious setting. The NIZK generated in the settling protocol is linear in the number of transactions performed.



(c) Client settling time by number of points in the semihonest setting. Note the y-intercept. The visible discrete jumps result from "giant steps" taken in the baby-step giant-step algorithm.

Figure 6

8 CONCLUSION

We have presented CheckOut, a system for scaling up loyalty card swapping as a mechanism to help anonymize loyalty card programs without requiring merchant involvement. CheckOut belongs to a family of obfuscation techniques [8, 9] that aim to subvert surveillance, both to reclaim privacy and to build technology as a form of protest. CheckOut offers a scalable and secure mechanism for loyalty card swapping while additionally allowing users to track and repay rewards points they should have accrued while using others' identities. We have developed a free and open-source prototype implementation of the algorithms underlying CheckOut, available at https://github.com/MatthewGregoire42/LoyaltyPointsCrypto.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and shepherd for their extremely helpful comments and feedback to improve this paper.

This material is based upon work supported by the National Science Foundation under Grant No. 2234408, as well as a gift from Cisco. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Rewards overview | giant, 2023.
[2] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable vrfs revisited. In *Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings*, pages 114–131, 2009.
[3] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*, pages 207–228. Springer, 2006.
[4] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. Updatable anonymous credentials and applications to incentive systems. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1671–1685, 2019.
[5] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112. ACM, 1988.
[6] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. Privacy-preserving incentive systems with highly efficient point-collection. In *Asia CCS*, 2020.
[7] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2023.
[8] Finn Brunton and Helen Nissenbaum. Vernacular resistance to data collection and analysis: A political theory of obfuscation. *First Monday*, 2011.
[9] Finn Brunton and Helen Nissenbaum. *Obfuscation: A User's Guide for Privacy and Protest.* MIT Press, 2015.
[10] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 302–321, 2005.
[11] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 93–118, 2001.
[12] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 56–72, 2004.
[13] Rob Carlson. Rob's giant bonuscard swap meet, 2006.
[14] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
[15] David Chaum. Blind signature system. In David Chaum, editor, *CRYPTO*, 1983.
[16] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
[17] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Advances in Cryptology—CRYPTO'92: 12th Annual International Cryptology Conference Santa Barbara, California, USA August 16–20, 1992 Proceedings 12*, pages 89–105. Springer, 1993.
[18] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.
[19] Rob Cockerham. Ultimate shopper: Modifying your club card, 2012.
[20] Individual Contributors. Certificate transparency. https://certificate.transparency.dev/, 2023.
[21] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols.* PhD thesis, January 1997.
[22] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18*, pages 424–441. Springer, 1998.
[23] Henry de Valence and Isis Agora Lovecruft. curve25519-dalek (version 3.2.1), 2023.
[24] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.
[25] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.
[26] Muhammed F Esgin, Ron Steinfeld, and Raymond K Zhao. Matrict+: More efficient post-quantum private blockchain payments. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1281–1298. IEEE, 2022.
[27] Saba Eskandarian. Fast privacy-preserving punch cards. *PoPETs*, 2021.
[28] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86: Proceedings 6*, pages 186–194. Springer, 1987.
[29] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May*

6-8, 1985, Providence, Rhode Island, USA, pages 291–304. ACM, 1985.
[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
[31] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965, pages 415–432. Springer, 2008.
[32] Mike Hamburg, Henry de Valence, Isis Lovecruft, and Tony Arcieri. https://ristretto.group/.
[33] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. BBA+: improving the security and applicability of privacy-preserving point collection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1925–1942, 2017.
[34] Ryan Henry, Femi G. Olumofin, and Ian Goldberg. Practical pir for electronic commerce. In *CCS*, pages 677–690, 2011.
[35] Max Hoffmann, Michael Klooß, Markus Raiber, and Andy Rupp. Black-box wallets: Fast anonymous two-way payments for constrained devices. *PoPETs*, 2020(1):165–194, 2020.
[36] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 4:220, 2016.
[37] Tibor Jager and Andy Rupp. Black-box accumulation: Collecting incentives in a privacy-preserving way. *PoPETs*, 2016(3):62–82, 2016.
[38] Tom Elvis Jedusor. Mimblewimble, 2016.
[39] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.
[40] Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–48, 2019.
[41] Clinton D Lanier and Amit Saini. Understanding consumer privacy: A review and future directions. *Academy of Marketing Science Review*, 12(2):1–45, 2008.
[42] B. Laurie, E. Messeri, and R. Stradling. Certificate transparency version 2.0. RFC 9162, 2021.
[43] Ben Laurie. Certificate transparency. *Communications of the ACM*, 57(10):40–46, 2014.
[44] Rachel Law. Vortex. http://mfadt.parsons.edu/2013/projects/vortex/, 2013. Accessed on 9/6/2022.
[45] Wanying Luo, Qi Xie, and Urs Hengartner. Facecloak: An architecture for user privacy on social networking sites. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada, August 29-31, 2009*, pages 26–33. IEEE Computer Society, 2009.
[46] Gregory Maxwell. Confidential transactions, 2015.
[47] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*, pages 369–378. Springer, 1988.
[48] Joseph T. Meyerowitz and Romit Roy Choudhury. Hiding stars with fireworks: location privacy through camouflage. In Kang G. Shin, Yongguang Zhang, Rajive L. Bagrodia, and Ramesh Govindan, editors, *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking, MOBICOM 2009, Beijing, China, September 20-25, 2009*, pages 345–356. ACM, 2009.
[49] Milica Milutinovic, Italo Dacosta, Andreas Put, and Bart De Decker. ucentive: An efficient, anonymous and unlinkable incentives scheme. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 588–595. IEEE, 2015.
[50] Global Trade Item Number. Barcode 101, 2015.
[51] Joachim Plesch and Irenaeus Wolff. Personal-data disclosure in a field experiment: Evidence on explicit prices, political attitudes, and privacy preferences. *Games*, 9(2):24, 2018.
[52] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.
[53] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.
[54] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4:161–174, 1991.
[55] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc., 1971*, volume 20, pages 415–440, 1971.
[56] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

A DEFERRED DEFINITIONS

We now formalize the semihonest point confidentiality, malicious point confidentiality, and balance integrity definitions introduced in Section 3.

Definition A.1 (Semihonest Point Confidentiality)**.** The semihonest point confidentiality experiment $\mathsf{CONF}[\mathcal{S}, \mathcal{A}, N, \lambda, b]$ is defined with respect to a point tracking scheme $\mathcal{S}$, an efficient adversary $\mathcal{A}$, a number N of system users, a security parameter λ, and a bit $b \in \{0, 1\}$. The experiment is described in Figure 7.

We say that a scheme $\mathcal{S}$ has semihonest point confidentiality if, for any choice of N and any security parameter λ, no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can win the point confidentiality game with greater than negligible advantage. In other words,

$$\Big|\Pr\big[\mathsf{CONF}[\mathcal{S}, \mathcal{A}, N, \lambda, 1] = 1\big] - \Pr\big[\mathsf{CONF}[\mathcal{S}, \mathcal{A}, N, \lambda, 0] = 1\big]\Big| \leq \mathsf{negl}(\lambda).$$

Definition A.2 (Malicious Point Confidentiality)**.** The malicious point confidentiality experiment, denoted $\mathsf{CONFMAL}[\mathcal{S}, \mathcal{A}, \lambda, b]$, is defined with respect to a point tracking scheme $\mathcal{S}$, an efficient adversary $\mathcal{A}$, a number N of user accounts, a security parameter λ, and a bit $b \in \{0, 1\}$. The experiment is described in Figure 8.

We say that a scheme $\mathcal{S}$ has malicious point confidentiality if, for any choice of DB and any security parameter λ, no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can win the point confidentiality game with greater than negligible advantage. In other words,

$$\Big|\Pr\big[\mathsf{CONFMAL}[\mathcal{S}, \mathsf{DB}, \mathcal{A}, \lambda, 1] = 1\big] - \Pr\big[\mathsf{CONFMAL}[\mathcal{S}, \mathsf{DB}, \mathcal{A}, \lambda, 0] = 1\big]\Big| \leq \mathsf{negl}(\lambda).$$

Definition A.3 (Balance Integrity)**.** The (weak) balance integrity experiment $\mathsf{BALINT}[\mathcal{S}, \mathcal{A}, N, \lambda, Q_O]$ is defined with respect to a point tracking scheme $\mathcal{S}$, an efficient adversary $\mathcal{A}$, a number N of system users, a security parameter λ, and a list of numbers , Q_O setting upper limits on the number of queries $\mathcal{A}$ makes to each of its oracles. The experiment is described in Figure 9.

We say that a scheme $\mathcal{S}$ has balance integrity if, for all efficient adversaries $\mathcal{A}$, any $N \in \mathbb{N}$, and any security parameter λ,

$$\Pr\left[\mathsf{BALINT}[\mathcal{S}, \mathcal{A}, N, \lambda, Q_O] = 1\right] \leq \mathsf{negl}(\lambda).$$

B DEFERRED CONSTRUCTION DETAILS

Figure 10 formalizes our approach to card swapping (without point redistribution). This approach is used for the card swapping component of all our schemes. Figure 11 formalizes the semihonest scheme described in Section 5.1.

C SECURITY PROOFS FOR SEMIHONEST SCHEME

We now repeat the security theorems for the semihonest construction that were stated in the body of the paper and then provide proof sketches for each.

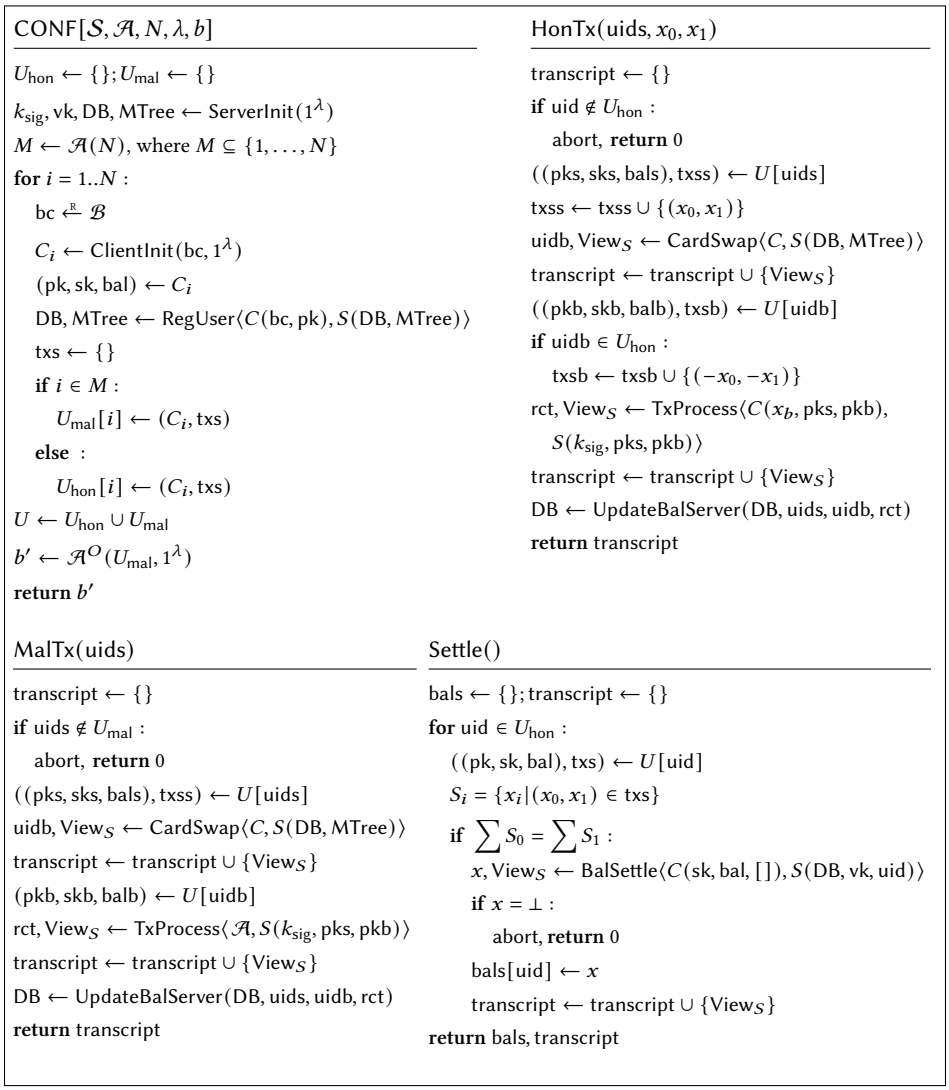


Figure 7: Semihonest point confidentiality experiment.

Theorem C.1 (Semihonest Point Confidentiality)**.** *Assuming the semantic security of El-Gamal encryption and the zero-knowledge property of the NIZKs used, our semihonest loyalty card swapping protocol with private redistribution of loyalty points satisfies semihonest point confidentiality (Definition A.1).*

Proof Sketch. The proof of point confidentiality follows a series of hybrids.

- **H0**: This hybrid is the security experiment CONF[$\mathcal{S}, \mathcal{A}, N, \lambda, 0$].
- **H1**: This hybrid is identical to the preceding one, except that for each transaction processed in the HonTx oracle, the experiment replaces the proof π of equal ciphertexts with a simulated proof. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each query to the HonTx oracle.
- **H2**: This hybrid is identical to the preceding one, except that each time an honest client settles in the Settle oracle, the experiment replaces the Chaum-Pedersen proof of knowledge of the plaintext with a simulated proof. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each settle operation. Observe that in this hybrid, the view of the adversary is independent of the challenger's input b because the proofs throughout the experiment reveal nothing about the transaction values x_b or the honest users' secret keys.
- **H3**: This hybrid is identical to the preceding one, except that, for each transaction where the barcode owner u_j is

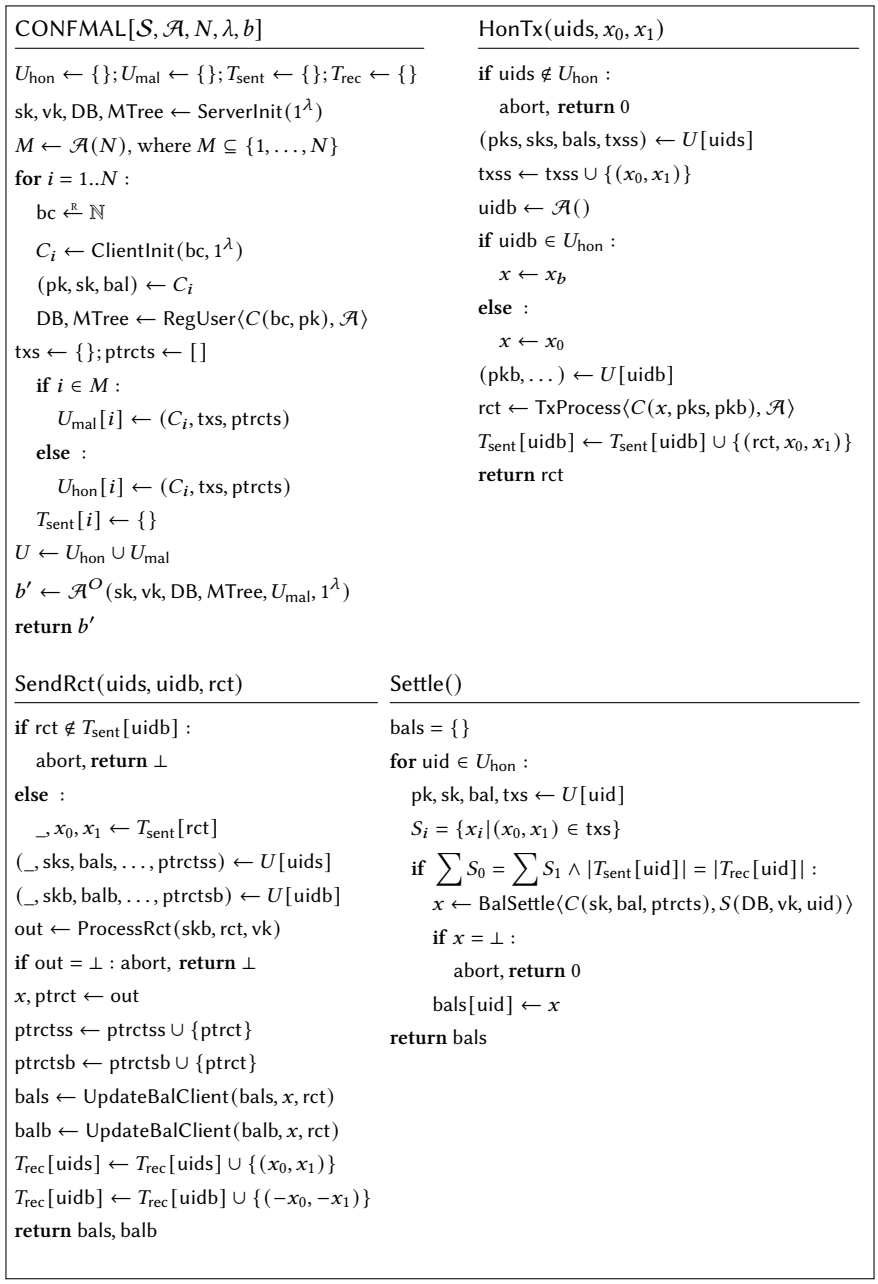


Figure 8: Malicious point confidentiality experiment.

honest, ciphertexts encrypting some x_0 are replaced with ciphertexts encrypting the corresponding x_1.
This hybrid is indistinguishable from the preceding one by the CPA security of El-Gamal encryption.

- **H4**: This hybrid undoes the change made in **H2**, replacing simulated Chaum-Pedersen proofs during the Settle oracle with real proofs. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each settle operation.
- **H5**: This hybrid undoes the change made in **H1**, replacing the simulated proofs π in the HonTx oracle with real proofs. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each transaction in calls to the HonTx oracle.

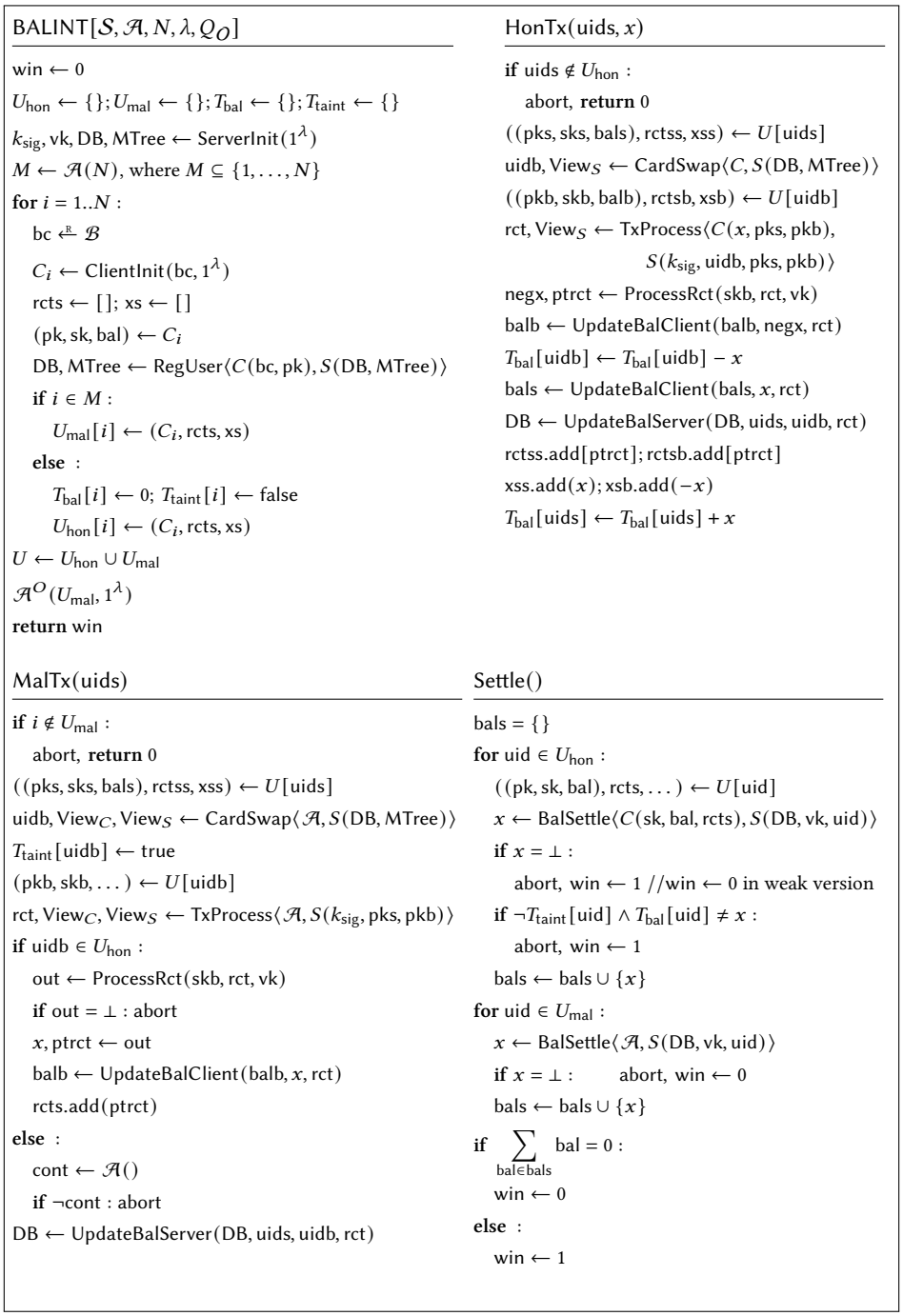


Figure 9: Balance integrity experiment.

Note that the final hybrid is precisely the adversary's view of the experiment CONF[$\mathcal{S}$, DB, $\mathcal{A}, \lambda$, 1]. Since each hybrid is indistinguishable from the last, the proof of the theorem follows from this series of hybrids and the triangle inequality. □

Theorem C.2 (Weak Balance Integrity). *Assuming the NIZKs used are proofs of knowledge, our semihonest loyalty card swapping protocol with private redistribution of loyalty points satisfies weak balance integrity (Definition A.3).*

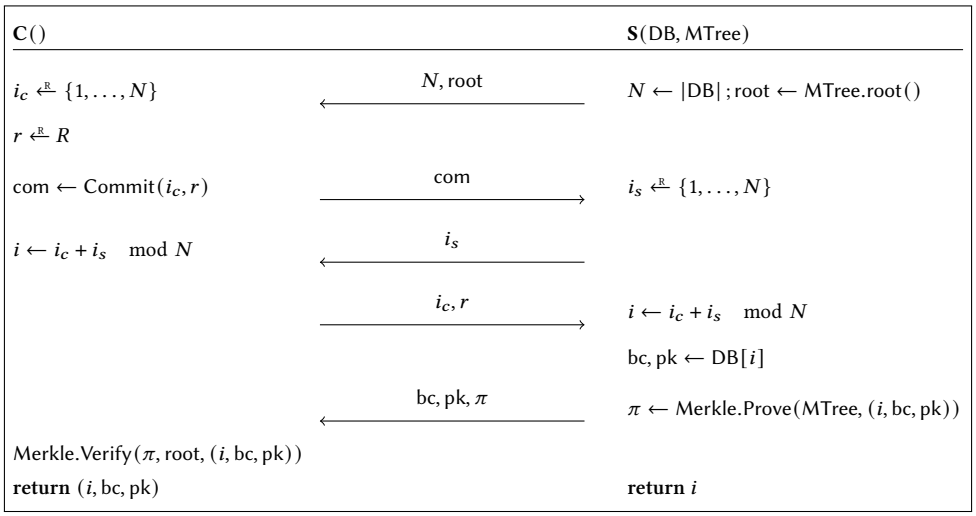


Figure 10: CardSwap **algorithm for all implementations.**

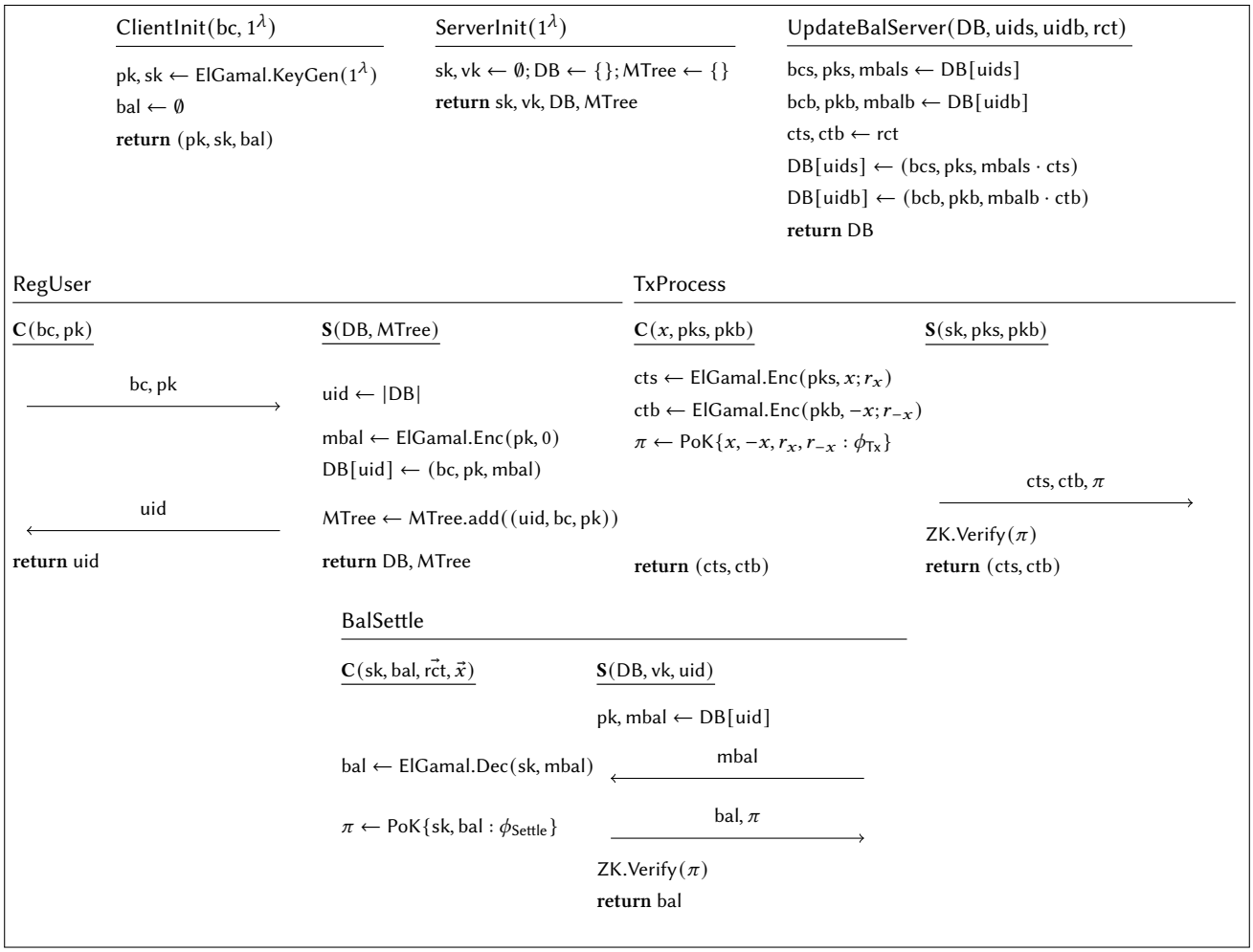


Figure 11: Semihonest scheme. We use ϕ_{Tx} and ϕ_{Settle} as shorthand for the proof statements described in Section 5.1.

Proof Sketch. The proof of this theorem follows a series of hybrids.

- **H0**: This is the original balance integrity experiment.
- **H1**: This hybrid is identical to the preceding one, except that when each adversary-controlled user settles its balance in a call to the Settle oracle, we run the extractor guaranteed to exist by the proof of knowledge property of the Chaum-Pedersen NIZK, aborting if any extractor fails.
  This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the Chaum-Pedersen NIZK. This hybrid consists of $Q_{\mathsf{Settle}} \cdot |M|$ subhybrids (one for each malicious user each time it settles), where the ith hybrid runs the extractor for the ith call to the Settle oracle.
- **H2**: This hybrid is identical to the preceding one, except that for in each call to the MalTx oracle, the experiment runs the extractor guaranteed to exist by the proof of knowledge property of the NIZK, aborting if any extractor fails. This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the NIZK. It consists of Q_{MalTx} subhybrids, where the ith hybrid runs the extractor for the ith call to the MalTX oracle.

We now show that the hybrid experiment **H2** never outputs 1. The proof of the theorem thus follows from the preceding hybrid argument and the triangle inequality.

Observe that one condition of balance integrity – that the balance of an honest user is not unduly modified – is satisfied quite easily. The only time a balance $T_{\mathsf{bal}}[\mathsf{uid}]$ is modified but $T_{\mathsf{taint}}[\mathsf{uid}]$ is not true is when a given uid only appears as an input or output to a protocol in the HonTx or Settle oracles, and not in the MalTx oracle. But in these cases, by the correctness of the additive homomorphism and decryption for the El-Gamal encryption scheme, the numbers held in $T_{\mathsf{bal}}[\mathsf{uid}]$ exactly match the ones that will be returned by BalSettle. This is the case because these balances are only affected by honest calls to the protocol functions, and no adversary-provided values are ever multiplied into them.

It remains to prove that the other condition of balance integrity – that all balances sum to zero when settling – is satisfied. We argue that each call to MalTx must result in a transaction consisting of encryptions of inverse values m and $-m$ under the public keys of the two users involved. The proof of the well-formedness of the ciphertexts follows immediately from statements 5.2-5.5. From statement 5.6, we have that $C_{12}C_{22} = h_s^{y_s} h_b^{y_b}$. But since we know that $C_{12} = g^{m'} h_s^{y_s}$ and $C_{22} = g^{m} h_b^{y_b}$, we have

$$C_{12}C_{22} = g^{m'} h_s^{y_s} g^{m} h_b^{y_b} = g^{m+m'} h_s^{y_s} h_b^{y_b}.$$

So statement 5.6 requires that

$$g^{m+m'} h_s^{y_s} h_b^{y_b} = h_s^{y_s} h_b^{y_b},$$

which implies that $m + m' = 0$.

Next, we argue that the decryptions in calls to Settle are always honest decryptions of the balances held by the server. The experiment has a group element $g^{m'}$ from the client and has extracted a secret x such that for the client's public key $h = g^x$ and encrypted balance (c_1, c_2), $c_2/g^{m'} = c_1^x$. Now, since all the ciphertexts used in calls to MalTx were well-formed, the ciphertext has the structure $(c_1 = g^y, c_2 = g^m h^y)$, which means that

$$c_2/g^{m'} = g^m h^y / g^{m'} = g^{m+xy-m'}$$

and $c_1^x = g^{yx}$. Thus, we have that

$$g^{m+xy-m'} = g^{yx},$$

which is only true when $m = m'$.

Finally, since we have shown that every transaction adds m to the balance of one user and subtracts m from the balance of another, this means that the change in the sum of balances between the two users in a given transaction is always zero. This satisfies the remaining condition of balance integrity, completing the proof. □

D SECURITY PROOFS FOR MALICIOUS SCHEME

We now repeat the security theorems for the malicious construction that were stated in the body of the paper and then provide proof sketches for each.

Theorem D.1 (Malicious Point Confidentiality)**.** *Assuming the CCA security of the encryption scheme, the zero-knowledge property of the NIZKs used, and the hardness of DDH in the group G, our malicious loyalty card swapping protocol with private redistribution of loyalty points satisfies malicious point confidentiality (Definition A.2) in the random oracle model.*

Proof Sketch. The proof of point confidentiality proceeds via a series of hybrids.

- **H0**: This hybrid is the security experiment $\mathsf{MALCONF}[\mathcal{S}, \mathsf{DB}, \mathcal{A}, \lambda, 0]$.
- **H1**: This hybrid replaces the ciphertexts produced in transactions where users i, j are both honest with encryptions of zero, using a lookup table to recover the plaintexts corresponding to these ciphertexts when needed. It consists of a series of subhybrids, one for each honest user, and it is indistinguishable from the preceding hybrid by the CCA security of the encryption scheme.
  The reduction for each subhybrid works by playing the role of the adversary in the CCA security game and the challenger in the confidentiality game. It sets the public key for one honest user to the public key in the CCA security game. It then plays the role of the challenger as described in **H0**, except whenever the value x_0 is to be encrypted in a transaction between two honest users, it sends the CCA challenger the two plaintexts $x_0, 0$. Decryption of known challenge ciphertexts is completed via a lookup table. Ciphertexts not found in the table are decrypted via the CCA decryption oracle. Note that if the CCA challenger's input bit is $b = 0$, the reduction behaves as in **H0**, and if $b = 1$, it behaves as in **H1**. Thus, the adversary wins the CCA security game with the same advantage that it distinguishes between **H0** and **H1**.
  Note that in this hybrid, intuitively, it becomes impossible for the adversary to create new transactions that are a function of transactions between honest users. It's always impossible to simply replay a transaction because the client checks for duplicate ciphertexts, and since the contents of

all ciphertexts between honest users are just zeros, modifying ciphertexts is no longer potentially useful.

- **H2**: This hybrid is identical to the preceding one, except that for each call to the HonTx oracle where both uids and uidb correspond to honest users, the experiment replaces the proof π in the transaction with a simulated proof. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each such transaction in calls to the HonTx oracle.
- **H3**: This hybrid is identical to the preceding one, except that each time an honest user participates in the settling protocol in a call to the Settle oracle, we replace the settling proof π with a simulated proof. This hybrid is indistinguishable from the preceding one by the zero knowledge property of the NIZK. The hybrid is made up of a series of subhybrids, one for each honest user.
- **H4**: In this hybrid, for receipts in transactions between honest users, the challenger computes the value mval by sampling a random group element $R \xleftarrow{R} G$ and setting $\mathsf{mval} \leftarrow R^{x_0}$. This hybrid consists of a series of subhybrids, one for each such transaction. It is indistinguishable from the preceding one under the DDH assumption in G in the random oracle model.
  The reduction for each subhybrid works by playing the role of the DDH adversary and the confidentiality challenger. It first receives a DDH challenge (g, X, Y, Z) where $X = g^{x_{\mathsf{DH}}}$, $Y = g^{y_{\mathsf{DH}}}$, and either $Z = g^{x_{\mathsf{DH}} y_{\mathsf{DH}}}$ or $Z = g^{z_{\mathsf{DH}}}$ where $x_{\mathsf{DH}}, y_{\mathsf{DH}}, z_{\mathsf{DH}} \xleftarrow{R} \mathbb{Z}_q$. The reduction samples $\alpha \xleftarrow{R} \mathbb{Z}_q$ and sets the public parameters h, g for the loyalty card swapping scheme as $h \leftarrow g^{\alpha}$.
  When responding to a random oracle query $H(s_i)$, the reduction samples $r_i \xleftarrow{R} \mathbb{Z}_q$ and sets $H(s_i) \leftarrow Y^{r_i}$. When handling a transaction between two honest users, instead of choosing $m_i \xleftarrow{R} \mathbb{Z}_q$, $\mathsf{hm}_i \leftarrow h^{m_i}$, the reduction chooses $r'_i \xleftarrow{R} \mathbb{Z}_q$ and sets $\mathsf{hm}_i \leftarrow X^{\alpha r'_i}$. It then retrieves the value of r_i from the RO programming for the corresponding string s_i and sets $\mathsf{mval} \leftarrow Z^{r'_i r_i x_b}$. Since the value of m_i is not explicitly known by the challenger anymore, the checks conducted after decrypting the ciphertext in these transactions simply check that the same group elements have been sent as were sent in the original transaction.
  Observe that the value of m_i is now implicitly set to be $x_{\mathsf{DH}} r'_i$, and the value of $H(s_i)$ is $g^{y_{\mathsf{DH}} r_i}$. Moreover, the value of mval is now either $g^{x_{\mathsf{DH}} y_{\mathsf{DH}} r'_i r_i x_b}$ (if $Z = g^{x_{\mathsf{DH}} y_{\mathsf{DH}}}$) or it is $g^{z_{\mathsf{DH}} r'_i r_i x_b}$ (if $Z = g^{z_{\mathsf{DH}}}$). But the former is exactly the value of mval as described in **H3**, whereas the latter is the value of mval in **H4**. Thus, reduction wins the DDH security game with the same advantage that the confidentiality adversary distinguishes between these two hybrids.
- **H5**: This hybrid is identical to the preceding one, except that, for each transaction where the barcode owner u_j is honest, we replace the transaction value x_0 with x_1. This hybrid is identical to the preceding one because the only time the value x_b still appears in the view of the adversary is in the exponent of the random group element mval.

From this point, additional hybrids **H6**–**H9** undo the changes made in hybrids **H1** – **H4** in reverse order. The final hybrid is precisely the adversary's view of the experiment $\mathsf{MALCONF}[\mathcal{S}, \mathsf{DB}, \mathcal{A}, \lambda, 1]$. Since each hybrid is indistinguishable from the last, the proof of the theorem follows from this series of hybrids and the triangle inequality. □

Theorem D.2 (Malicous Scheme Balance Integrity)**.** *Assuming that the NIZKs used are proofs of knowledge, the signature scheme is existentially unforgeable, and that the discrete logarithm problem is hard in the group G, our malicious loyalty card swapping protocol with private redistribution of loyalty points satisfies balance integrity (Definition A.3) in the random oracle model.*

Proof Sketch. Observe that one condition of balance integrity – that the balance of an honest user is not unduly modified – is satisfied quite easily. The only time a balance $T_{\mathsf{bal}}[\mathsf{uid}]$ is modified but $T_{\mathsf{taint}}[\mathsf{uid}]$ is not true is when a given uid only appears as an input or output to a protocol in the HonTx or Settle oracles, and not in the MalTx oracle. But in these cases, by the correctness of our scheme, the numbers held in $T_{\mathsf{bal}}[\mathsf{uid}]$ exactly match the ones that will be returned by BalSettle. This is the case because these balances are only affected by honest calls to the protocol functions, and no adversary-provided values are ever multiplied into them.

The adversary can also never cause an honest user to fail to settle because the only opportunities for the client to fail to settle are if a signature σ_i fails to verify or if the proof π fails to verify. But signature verification will always accept by the correctness of the signature scheme, and the proof that the server checks during the BalSettle protocol is simply a proof of the properties that the client verifies (in the clear) via the ProcessRct algorithm during calls to HonTx or MalTx. Thus any attempted malicious transaction that would put a client in an "un-settleable" state will be filtered out by the ProcessRct function.

Thus the rest of our proof focuses on the other condition, that users' balances sum to zero when settling. The proof of the second condition follows a series of hybrids.

- **H0**: This is the original balance integrity experiment.
- **H1**: This hybrid is identical to the preceding one, except the experiment aborts and outputs 0 if values hm_i, s_i that do not appear together in any run of the TxProcess protocol do appear during a successful run of the Settle protocol. An adversary who reaches this abort condition can be used to break the unforgeability of the signature scheme because the integrity challenger will not have issued a signature on such an hm_i, s_i. Thus, this abort condition is reached with negligible probability, due to the unforgeability of the signature scheme, and this hybrid is indistinguishable from the preceding one.
- **H2**: This hybrid is identical to the preceding one, except that in each call to MalTx, the experiment runs the extractor guaranteed to exist by the proof of knowledge property of the NIZK, aborting if any extractor fails. This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the NIZK. Since the extractor must be run for each transaction, this hybrid must include Q_{MalTx} subhybrids.

- **H3**: This hybrid is identical to the preceding one, except that for each Settle operation and for each uid $\in U_{\text{mal}}$, the experiment runs the extractor guaranteed to exist by the proof of knowledge property of the NIZK, aborting if any extractor fails. This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the NIZK. Since the extractor must be run for each settle operation, this hybrid must include $Q_{\text{Settle}} \cdot |M|$ subhybrids (one for each malicious user each time it settles).
- **H4**: This hybrid is identical to the preceding one, except it aborts if the hash function H is ever queried on two strings $s, s', s \neq s', H(s) = H(s')$, i.e., if a collision is found for the hash function. This event unconditionally occurs with negligible probability in the random oracle model, so this hybrid is statistically indistinguishable from the preceding one.

We now show that an adversary who wins the game by causing all settled balances to sum to something other than zero in **H4** can be used to break discrete logarithm in G. The proof of the theorem thus follows from this proof, the preceding hybrid argument, and the triangle inequality.

We build a reduction that plays the role of the adversary for discrete log in G, receiving a discrete log challenge (g, g_c), and who plays the role of the challenger in **H4**. Our reduction responds to random oracle queries $H(s_i)$ by sampling values $\alpha_i, \beta_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H(s_i) \leftarrow g^{\alpha_i} g_c^{\beta_i}$.

Observe that at the end of the experiment, the reduction has all the values $m_i, x_i, i \in \{1, ..., n_{\text{tx}}\}$ used during runs of the TxProcess protocol, as well as all the values $m'_j, x'_j, j \in \{1, ..., n_{\text{settle}}\}$ used during calls to Settle(). Here n_{tx} and n_{settle} represent the number of transactions or receipts included in a balances, respectively. As a notational shorthand, we will always use the subscript i to refer to values extracted from transactions and the subscript j to refer to values extracted during settling.

Since the reduction has kept a representation of each user's balance during the experiment, it can also take the product P of all users' balances, and find a representation of P using the m_i, x_i values it has extracted or saved from calls to malTx and HonTX, respectively. Because the server always updates balances by multiplying mval into one balance and mval^{-1} into another balance, we know that $P = g^0 = 1$. Moreover, because of the statements proven during the BalSettle protocol, we know that the extracted or saved values of x'_j, m'_j are also a representation of P because proofs provided while settling are made with respect to the masked balances produced during transactions.

Let $y_i = x_i m_i$ and $y'_j = x'_j m'_j$ for convenience. By the reasoning above, we have that

$$P = \Pi_i H(s_i)^{y_i} = \Pi_j H(s_j)^{y'_j},$$

where the first equality follows from the statements proved during transactions and the second follows from the statements proved during settling.

Substituting in $H(s_i) = g^{\alpha_i} g_c^{\beta_i}$, we can solve for g_c as follows.

$$\Pi_i (g^{\alpha_i} g_c^{\beta_i})^{y_i} = \Pi_j (g^{\alpha_j} g_c^{\beta_j})^{y'_j}$$
$$g^{\Sigma_i \alpha_i y_i} g_c^{\Sigma_i \beta_i y_i} = g^{\Sigma_j \alpha_j y'_j} g_c^{\Sigma_j \beta_j y'_j}$$
$$g_c^{\Sigma_i \beta_i y_i - \Sigma_j \beta_j y'_j} = g^{\Sigma_j \alpha_j y'_j - \Sigma_i \alpha_i y_i}$$
$$g_c = g^{(\Sigma_j \alpha_j y'_j - \Sigma_i \alpha_i y_i)/(\Sigma_i \beta_i y_i - \Sigma_j \beta_j y'_j)}$$

Now it only remains to show that

$$\Sigma_i \beta_i y_i - \Sigma_j \beta_j y'_j \neq 0,$$

which would mean that the quotient

$$\frac{\Sigma_j \alpha_j y'_j - \Sigma_i \alpha_i y_i}{\Sigma_i \beta_i y_i - \Sigma_j \beta_j y'_j}$$

is the discrete log between g_c and g.

First, observe that it is not possible for the adversary to win the balance integrity game if the sets of $\{x_i\}$ and $\{x'_j\}$ values used in these representations to be the same, or else the balances during settling (the x'_j values) would sum to zero, and the adversary would not win the balance integrity game.

Now, there are two cases we need to cover.

(1) The set of strings $\{s_i\}, \{s'_j\}$ used in the two representations of P are the same.
(2) The set of strings $\{s_i\}, \{s'_j\}$ used in the two representations of P are different.

Case 1. In the first case, the sets $\{\beta_i\}, \{\beta_j\}$ are also the same, as they are fully determined by the choice of s. This means we need to show that the values of $\{y_i\}, \{y'_i\}$ differ in order for the denominator of the discrete log value to be non-zero.

Suppose toward contradiction that these sets are the same. We know that the x_i and x'_j values cannot be the same, so it must be that the m_i and m'_j values must also not be the same if their products are to form sets of the same values. But then since all the signatures used during Settle() verify as being signatures used in calls to TxProcess, each hm used in settling must also have been used during a transaction (as the hms are tied to values of s by the signatures σ). But since hm uniquely determines m, this means that the extracted values of m are also the same values of m that appeared in a transaction alongside the corresponding values of s (as choices of s are never repeated except with negligible probability). From this we conclude that all the values m_i used in transactions match the values m'_j used in settling, except with negligible probability. This is a contradiction because we previously concluded that these values must not match.

Case 2. In this case, since the choices of $\{s_i\}, \{s_j\}$ differ, the values of $\{\beta_i\}, \{\beta_j\}$ will differ except with negligible probability, as they are sampled uniformly at random and independently of each other. Moreover, these values are perfectly hidden in $g^{\alpha_i} g_c^{\beta_i}$, so the view of the adversary is independent of $\{\beta_i\}, \{\beta_j\}$. This means that the probability that $\Sigma_i \beta_i y_i - \Sigma_j \beta_j y'_j = 0$ is simply $1/q$.

We have now shown how to calculate the discrete log of g_c and demonstrated that the calculated value is defined with all but negligible probability. This completes the proof of the theorem. □

E DETECTING INCORRECT INPUTS

There are many loyalty programs that allow users to check their balances online, or even look at the values of individual transactions. We can use this to allow users to detect when someone using their loyalty card has entered an incorrect input, and identify the offending transaction(s).

Our idea is simple. A user can check their outstanding loyalty balance online. This balance bal_{debt} represents the user's total debts incurred when others used their card. CheckOut can also record the total amount of loyalty points $\text{bal}_{\text{credit}}$ a user is owed from using others' cards. If all users correctly enter data into CheckOut, then we expect that for each user's overall balance bal,

$$\text{bal} = \text{bal}_{\text{credit}} - \text{bal}_{\text{debt}}.$$

Users can perform this verification out-of-band at settle time. If this expression does not hold, an honest user knows that someone who has used their card has entered an incorrect value, resulting in an inaccurate $\text{bal}_{\text{credit}}$. To facilitate identifying the offending transaction, the server can keep logs of each transaction sent, and—since the value of each transaction can be unmasked by the user whose account is used—the user can check which one does not align with the expected transactions it can see through the online interface to the loyalty program. From the user ID associated with this transaction in the server's logs, the malicious user can be identified and dealt with by the server.