Verito: A Practical System for Transparency and Accountability in Virtual Economies

Raghav Bhaskar, Saikat Guha, Srivatsan Laxman, Prasad Naldurg
Microsoft Research India, Bangalore, India
{rbhaskar,saikat,slaxman,prasadn}@microsoft.com

Abstract

Purchase of virtual goods and services is now a major source of revenue for developers on platforms like Facebook, Xbox, and iOS. These virtual economies are typically based on users maintaining a stored-value account of virtual-currency (purchased with real-currency) with the platform. While the model is similar to that of a bank, these economies lack transparency and regulatory oversight that protect a consumer's financial interests. We propose Verito, a practical solution that provides transparency and accountability in this context. We combine state-of-the-art cryptographic constructs in novel ways to design a system that provides four desirable properties, viz., transparency (money-in equals money-out), fairness (users treated equally), non-repudiation (users' virtual money is safe), and scalability (low processing and storage costs). Our design also accommodates nuances such as support for multiple-currencies, and defense against arbitrage, while addressing scalability bottlenecks. We present an experimental evaluation based on our implementation of Verito and study its performance characteristics. Overall, we show that it is possible to protect consumer interests in virtual economies in a practical manner, without relying only on regulation.

1 Introduction

Virtual economies, where users buy virtual goods and services, have seen phenomenal growth in recent years. Facebook transacts over 1.5 billion dollars in in-game purchases for social games like Farmville and Mafia Wars, accounting for 15% of its net 2011 revenue [5]; the market has averaged around 300% year-on-year growth over the last three years as reported by Facebook [5]. In-app purchases netted Apple, Google and Xbox over 2.1 billion dollars in 2011 [6]. Massively-multiplayer online games such as Second Life, World of Warcraft, and the Sims Online have thriving economies, including black-markets for users to convert between real and virtual currency for games that prohibit this practice [2, 14].

These economies typically use a virtual currency rather than a real currency for individual transactions. This is because there are a large number of small-valued transactions (sometimes of just a few cents). Using real currency for individual transactions is uneconomical due to regulatory and compliance obligations for financial transactions. As a result, most economies allow the user to purchase virtual currency in bulk (using real currency); merchants are similarly paid in bulk. Examples of these virtual currencies include Facebook credits, Linden dollars (Second Life), and Xbox points.

Virtual economies are not regulated. Facebook, for instance, declared "in July 2010, we seeded users with free seeded credits [...] in order to drive user awareness and education around credits. These credits are zero-value, and developers do not realize revenue from them when spent by users." [4]. In essence, Facebook unilaterally devalued its currency overnight. Unfortunately for developers, an unintentional consequence was that the effective conversion rate from Facebook credits to dollars became unpredictable. Since credits are opaque to the developer, for a batch of 100 credits, the developer could be paid anywhere between the fair value of 0 and 100 credits; the developer would have no way to verify if he was paid correctly. Facebook reserves for itself the right to seed these zero-valued credits at any time and for any reason [4].

The global reach of these virtual economies introduces additional complications. Xbox points were, for instance, priced similarly in all currencies when they were first introduced (around $1 for 80 points). However, due to fluctuating currency exchange rates, the current value of these points varies considerably between regions. This can create an arbitrage scenario where a merchant can turn a profit by buying points in one currency and encashing it in another. Currently, points purchased in one currency cannot be spent in a different geography. Further, trading points between user accounts is also not supported.

Our main contribution with Verito is a technical solution that provides transparency in accounting in such economies without relying on external audits or trusted third parties. Applied to Facebook, Verito would allow game developers to verify that they were paid correctly, while preserving user privacy. Applied to Xbox, Verito would allow promotional points, with flexible support for multiple currencies and user relocation, while preventing arbitrage.

We identify and define the *accountable virtual economy* problem as having the following properties. First, is transparency, i.e., money entering the economy (e.g., through user purchase of credits) should equal money exiting the economy (either as payments to merchants, or as commissions and fees charged by the platform). Second, is fairness, i.e., the merchants should not be able to distinguish between credits acquired at different rates and treat users differently. However, the platform should be able to distinguish between credits of different values (e.g., zero-value vs. non-zero value for Facebook, different currencies for Xbox points, etc.). Third, is non-repudiation of transactions to protect users, merchants and the platform. Fourth, is scalability of processing, storage, and network costs. We formalize these properties in Section 2.

Verito combines existing cryptographic constructs in a novel way. At a high-level, we use homomorphic commitments [32] to create an audit trail for virtual transactions. A naïve application of commitments, however, may compromise fairness when the commitments are eventually opened. A second problem is the storage requirements for non-repudiation (e.g., to prevent a user from spending a credit multiple times), which would normally require a large database. We address the first issue through aggregate commitments, and the second by combining commitments with dynamic accumulators [17]. While accumulators solve the storage issue, a naïve application of accumulators would incur high network and processing overheads. Our construction avoids both these issues. We present a detailed design of our approach in Section 3, and formally define and prove that the design achieves our properties in Section 4. Sections 5 and 6 report on our implementation of Verito and experimental evaluation that demonstrates its practicality. We also outline how Verito can be deployed incrementally, as a proxy service that is available to interested users and developers of any platform.

There is considerable interest in virtual currencies, though much of it is focused on legal and regulatory aspects (See Section 8). One technical solution, which is designed with similar goals of transparency and accountability in mind, is Bitcoin [1]. However, Bitcoin assumes a fully-decentralized world, where every Bitcoin user tracks the history of every credit and ratifies every transaction. This is impractical in our setting, with millions of credits and transactions. There is also a substantial body of work concerning e-cash [18] and related schemes, which seek to provide anonymity and unlinkability, and are not directly applicable.

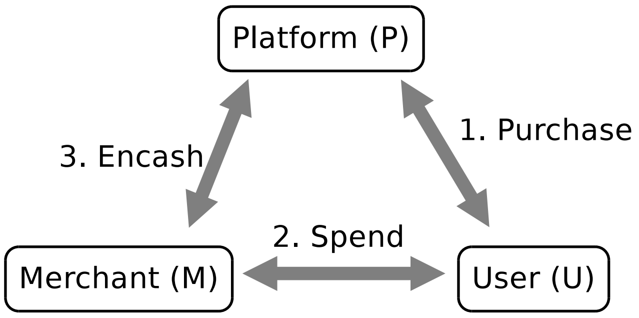


Figure 1. Architecture of an accountable virtual economy.

Overall this paper makes three contributions: First, we define the accountable virtual economy problem, and identify the properties which make it an interesting research problem that is highly-relevant to industry. Second, we present a rigorous solution that addresses this problem through a novel combination of commitments and accumulators. Finally, we build and evaluate our proposed approach, and find it to be practical and scalable on current hardware. In Section 7 we also discuss how it can be deployed incrementally, and in response to stronger regulatory laws.

2 Accountable Virtual Economy

In this section, we formulate the accountable virtual economy problem. We first introduce the players, define the transactions between them, and call out the properties that must be satisfied in this context.

2.1 Players

There are three types of players in a virtual economy as shown in Fig. 1. First, we have the platform ($\mathcal{P}$) which owns and maintains the infrastructure, e.g., an online-gaming ecosystem. Examples of such platforms include Facebook and Xbox Live. Next there are the merchants ($\mathcal{M}$) who are registered with a $\mathcal{P}$ and use $\mathcal{P}$'s infrastructure to host their services. Examples of $\mathcal{M}$s include companies like Zynga, which make games like Farmville, Cityville and MafiaWars hosted on Facebook. Finally, there are the users ($\mathcal{U}$) who are also registered with $\mathcal{P}$ and interface with the services provided by $\mathcal{M}$s.

As a part of its business infrastructure, $\mathcal{P}$ issues and regulates *credits*, which can be thought of as a form of virtual currency. Users playing a game can either purchase or earn these credits, and subsequently redeem them with $\mathcal{M}$ to buy

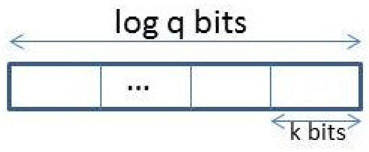


Figure 2. A credit message

virtual goods (such as tractors in Farmville) and services. In other words, $\mathcal{M}$s can offer products and services to users in-exchange for credits, and $\mathcal{M}$s can later encash spent credits at $\mathcal{P}$.

Each credit is associated with a *nominal value*, which is equal to the price that the $\mathcal{P}$ charged a user for its purchase. Credits can be associated with different nominal values. For example, in Xbox Live, 80 credits (Microsoft Points) can be purchased for 1 USD, but cost 54.4 INR in India, and are not tied to the current exchange rate. There are many reasons for this, including different market conditions, different regulations, and tax laws across geographical regions. In addition, $\mathcal{P}$s (or $\mathcal{M}$s) can hand out *free credits* to users, which do not have any monetary value when they are redeemed. These can be thought of as loyalty points or incentives to increase user participation as exercised by Facebook in 2010. Verito is architected to support multiple types of credits (including e.g., zero value credits, credits in different currencies, and discounted credits in the same currency), with a predefined maximum limit on the number of distinct nominal values a credit can be associated with.

One of the requirements in these virtual economies is that no matter what the nominal value, an $\mathcal{M}$ must provide the same experience for each credit spent by $\mathcal{U}$s. That is, at the time of spend by users, $\mathcal{M}$s cannot distinguish the nominal values of individual credits directly. This ensures a fair and consistent experience for users.

A credit in Verito is a commitment on a message which describes the nominal value of the credit. The message (hidden in the credit) comprises of exactly $\log_2 q$ bits, which is divided into $\lfloor \frac{\log_2 q}{k} \rfloor$ buckets each of size k, as shown in Figure 2. q is a security parameter in the system (i.e., related to a large prime), and k is a fairness parameter in the system (i.e., aggregation level for revealed commitments). Different buckets represent different distinct nominal values. A valid credit message has the least significant bit set to one for exactly one bucket. The rest of the message is filled with zeros. The bucket with the non-zero bit implicitly indicates the nominal value of the credit applied for its purchase. Thus, the number of distinct nominal values we can support is determined by system parameters q and k (and is equal to $\lfloor \frac{\log_2 q}{k} \rfloor$).

2.2 Transactions

We now describe the transactions (or interfaces) in a virtual economy. Let CREDITS denote a set of credits and let MONEY denote its corresponding aggregate nominal value. Since the set may contain credits with different nominal values, MONEY must be thought of as an array of aggregate values, where the j^{th} entry represents the aggregate corresponding to j^{th} distinct nominal value. Thus, $\texttt{MONEY}[1]$ may, for instance, correspond to credits purchased at 80 credits per USD in the US, $\texttt{MONEY}[2]$ to credits purchased at 1.47 credits per INR in India, etc. There are three transactions in a virtual economy, namely, PURCHASE, SPEND and ENCASH (See Fig. 1):

1. PURCHASE ($\mathcal{U}$, MONEY):
   - Description: $\mathcal{U}$ buys credits worth MONEY from $\mathcal{P}$. There will only be at most one non-zero entry in the MONEY array (which corresponds to the particular nominal value applied for this purchase). A zero-value credit, issued as incentive, may also be "purchased" from the $\mathcal{P}$ using the same interface.
   - Initiated by: $\mathcal{U}$
   - Output: CREDITS
2. SPEND ($\mathcal{U}$, $\mathcal{M}$, CREDITS):
   - Description: $\mathcal{U}$ gives CREDITS to $\mathcal{M}$ in-exchange for virtual goods and/or services. Different CREDITS may contain distinct nominal values.
   - Initiated by: $\mathcal{U}$
   - Output: None
3. ENCASH ($\mathcal{M}$, CREDITS):
   - Description: $\mathcal{M}$ deposits CREDITS to $\mathcal{P}$ and gets back MONEY corresponding to (1-PROFIT) times the aggregate value of CREDITS (where PROFIT represents the fixed profit margin of $\mathcal{P}$ that is published ahead of time). Again, for each j, $\mathcal{P}$ pays $\mathcal{M}$ a sum of $\texttt{MONEY}[j]$ for credits associated with the j^{th} distinct nominal value.
   - Initiated by: $\mathcal{M}$
   - Output: MONEY

2.3 Properties

Currently, virtual economies strongly rely on fair play by the platform for their accountability. Our goal is develop a system that minimizes trust assumptions on the platform.

To this end, we identify four system-wide properties that an accountable virtual economy should support, namely transparency, fairness, non-repudiation, and scalability. We do not require merchants or users to trust the platform as long as neither colludes with the platform to compromise the other.

- **Transparency**: This property ensures that all the money in the system can be properly accounted for.

  Let MONEYIN denote the total value of all the real currencies that users have used to purchase credits inside the virtual economy. Let MONEYOUT denote the total amount of real currencies that $\mathcal{P}$ has paid-out to merchants in ENCASH transactions. As earlier, MONEYIN and MONEYOUT are represented as arrays with the j^{th} entry corresponding to the j^{th} distinct nominal value.

  The set of currently valid (unspent) credits is denoted VALIDCR and the set of credits already encashed by $\mathcal{M}$ is denoted ENCASHEDCR. The subset of VALIDCR constituted by credits associated with the j^{th} nominal value is denoted $\mathtt{VALIDCR}_j$ and similarly $\mathtt{ENCASHEDCR}_j$ is the corresponding subset of ENCASHEDCR. Given any set of credits associated with same nominal value, its aggregate value is given by the function $\mathrm{Value}(\cdot)$.

  For transparency we will require that the money within each nominal value bucket is properly accounted for. To this end an accountable virtual economy should enforce the following system-wide property at all times (invariant) and for all j:

$$\begin{aligned}\mathtt{MONEYIN}[j] &= \mathtt{MONEYOUT}[j] + \mathrm{Value}(\mathtt{VALIDCR}_j) \\ &+ \mathtt{PROFIT} * \mathrm{Value}(\mathtt{ENCASHEDCR}_j)\end{aligned} \quad (1)$$

  where $\mathtt{MONEYOUT}[j]$ is given by:

$$\mathtt{MONEYOUT}[j] = (1 - \mathtt{PROFIT}) * \mathrm{Value}(\mathtt{ENCASHEDCR}_j) \quad (2)$$

  This security property can be satisfied if CREDITS ←PURCHASE ($\mathcal{U}$, MONEY) satisfies

$$\mathrm{Value}(\mathtt{CREDITS}) = \mathtt{MONEY}$$

  and MONEY ←ENCASH ($\mathcal{M}$, CREDITS) satisfies

$$\mathtt{MONEY} = (1 - \mathtt{PROFIT}) * \mathrm{Value}(\mathtt{CREDITS}) \quad (3)$$

  The first condition above guarantees transparency for the user, while the second one guarantees it for the merchant.

- **Fairness**: This property ensures that an $\mathcal{M}$ cannot preferentially treat users based on the values of credits used in the game. For this, we need to hide from $\mathcal{M}$, the nominal value of credits spent in any individual SPEND ($\mathcal{U}$,$\mathcal{M}$,CREDITS) transaction. Note that the system must allow $\mathcal{M}$s to check the correctness of money received from $\mathcal{P}$ in ENCASH ($\mathcal{M}$,CREDITS) transactions, without allowing an $\mathcal{M}$ to eventually infer the value of credits in any individual spend transaction.

- **Non-repudiation**: The third security property disallows any party from repudiating any transaction after-the-fact. For example, a user should not be able to deny having spent a credit (in an attempt to re-spend that credit), and a merchant should not be able to deny having encashed a credit.

- **Scalability**: For scalability, $\mathcal{P}$ must not have to store each credit issued. While users and merchants naturally have to store each credit in their possession, network costs for any transaction they engage in must be a function of the number of credits exchanged in that transaction (and not a function of the total number of credits in their possession).

Note that both $\mathcal{M}$s and $\mathcal{P}$s have a natural incentive to detect any fraud by $\mathcal{U}$s involving use of invalid credits or re-use of already spent credits. Further, $\mathcal{P}$ has a natural incentive to ensure that $\mathcal{M}$s do not encash credits that were already paid out. No other trust assumptions are needed except that there are no collusions between the players. Specifically, it is trivially possible for the user and platform to collude and and erode a merchant's revenue. Similarly, a merchant and the platform can collude to deny fair service to the user. Finally, since an accountable virtual economy by design ensures that the money associated with every distinct nominal value is separately accounted for, it automatically prevents arbitrage that can arise out of differentials in currency exchange rates. We explain how Verito provides support for these properties using cryptographic schemes in the next section.

3 Verito

In this section, we describe our approach to achieve the properties proposed in Section 2. To recall, the key properties are transparency, fairness, non-repudiation and scalability. We explain the underlying cryptographic primitives, and describe our Verito scheme that uses these primitives in detail. In Section 4, we formally define our security properties and explain how our design meets these goals.

3.1 Preliminaries

To ensure transparency while maintaining fairness, we use *homomorphic commitments* on the credit values. A commitment to the credit value hides the actual value but allows the credit values to be aggregated (see next subsection for details.) To get non-repudiation and scalability, we use an efficient cryptographic primitive called *dynamic accumulators*, without requiring zero-knowledge proofs that make them inefficient in traditional applications, to track all the credits "spent" and "encashed". These accumulators require much less storage at the platform than the traditional approach of maintaining lists or hashed trees.

3.1.1 Commitment scheme

A commitment scheme is a protocol executed by a sender and a receiver and has two stages, Commit and Reveal. During the Commit stage, the sender commits to a (secret) value to the receiver, by sending some function of the value. Knowledge of the committed value does not allow the receiver to learn anything about the secret value. In the second phase, the sender reveals the hidden value along with some useful auxiliary information, which the receiver can use to check the validity of the revealed value. Homomorphic commitments have the additional property that multiplying two commitments results in a commitment on the sum of their committed messages, without revealing individual values. More formally, a commitment scheme consists of three algorithms: Setup, Commit and Open. The Setup algorithm generates a public commitment key which is used by either party to commit to a message m to the other party. The committing party runs the Commit algorithm to commit to a message m and sends the output c of the algorithm to the other party. At some future time, the committing party can open the commitment by sending the message m with auxiliary information aux to the other party. The second party checks the correctness of the committed value by running $\mathsf{Open}(c, aux)$. A secure commitment scheme has two security properties: *hiding* and *binding*. The *hiding* property ensures that the other party cannot learn anything about the committed message m even after receiving the output of the Commit algorithm. The *binding* property ensures that the committing party cannot "open" the commitment to a different message m, once the Commit algorithm has been executed. An additively homomorphic commitment scheme has the additional property that a commitment on $m_1 + m_2$ can be computed directly from the individual commitments on m_1 and m_2, *i.e.* $\mathsf{Commit}(m_1 + m_2) = \mathsf{Commit}(m_1) \odot \mathsf{Commit}(m_2)$. Several homomorphic commitment schemes are known [10, 19, 32]. In our protocol, we use Pedersen's commitment scheme [32], which is one of the most efficient additively homomorphic commitment schemes.

$\mathsf{Setup}(1^k)$: Takes a security parameter k as input and generates a $(k+1)$ bit prime prime p such that $p = 2q+1$ where q is prime. Pick a random generator g of $\mathbb{Z}_q^*$ and a random element h of $\mathbb{Z}_q^*$.

$\mathsf{Commit}(m \in \mathbb{Z}_q)$: In order to commit m, choose random element $r \in \mathbb{Z}_q^*$ and output $c = g^r h^m \ mod \ p$.

Open(c, r, m): Outputs 1 if $g^r h^m \ mod \ p$ equals c else 0. Note r is revealed in this step allowing the validation of the hidden value m.

3.1.2 Dynamic Accumulators

Another cryptographic primitive that we use in Verito is a dynamic accumulator. Cryptographic accumulator schemes allow a publisher to "hash" a large set of input values to a single short value (typically, of constant-size) called the *accumulator* [13, 15, 17], and provide for checking if a value is contained (or alternately not contained) in the set represented by the accumulator, by generating a *witness*. It is infeasible to find a witness for a value that was not accumulated. Accumulators have been proposed for revocation lists in anonymous credential systems, for identity escrow, and group signatures etc. Dynamic accumulators [17] further extend this functionality by allowing dynamic addition or deletion of input values to this accumulator efficiently, where the cost of the add or delete is independent of the number of accumulated values. We use accumulators in the context of checking double-spending/double-encashing of credits (for non-repudiation by users or merchants), though we do not use the accompanying (often expensive) Zero-knowledge protocols that are often required in settings requiring anonymity. Different dynamic accumulator schemes have been proposed, including those based on the strong RSA assumption [17], and from bilinear pairings [16, 28]. Accumulators can represent either the list of valid values (white-list) or the list of invalid values (black-list).

Any secure accumulator consists of four algorithms AcGen, AccAdd, AccWitUpdate, and AccVerify. There are typically four actors: an accumulator authority, an (untrusted) update entity, users and verifiers. The accumulator authority runs the AccGen algorithm after creating a new accumulator key pair (sk_A, pk_A). New values can be added to the accumulator acc_V using the AccAdd to obtain a new value acc'_V. AccAdd produces a witness wit_i. Both acc_V and wit_i are of fixed length. Each time an accumulator changes, wit_i becomes invalid. All witnesses need to be updated, and this can be offloaded to a witness-update entity, which does not need any secret security parameters. Users who obtain the current witness for a value i can prove to any verifier that this value is in the accumulator using AccVerify. In our architecture, the role of the authority and the verifier is played by $\mathcal{P}$. The role of the users is played by $\mathcal{U}$ in the SPEND transaction and by $\mathcal{M}$ in the ENCASH transaction. There is no third party update entity. $\mathcal{U}$ and $\mathcal{M}$ can update their witnesses by receiving a witness update from $\mathcal{P}$.

Now we present technical details of the dynamic universal accummulator (DUA) scheme used in our implementation. This scheme is due to Au et al., [11], which is based on an original proposal by Nguyen [28]. The scheme uses Bilinear pairings and augments Nguyen's construction to work on the ring of polynomials over a finite field (i.e., a Euclidean domain). The security of this scheme is based on the strong or decisional Diffie-Hellman assumption.

A bilinear pairing is a mapping from a pair of group elements to another group element. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be cyclic groups of prime order p. Let g be a generator of $\mathcal{G}_1$. Function $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \longrightarrow \mathcal{G}_2$ is a bilinear map if:

- Each element in $\mathcal{G}_1$, $\mathcal{G}_2$ has a unique binary representation.
- $\hat{e}(A^x, g^y) = \hat{e}(A, B)^{xy}$ for all $A, B \in \mathcal{G}_1$ and $x, y \in \mathbb{Z}_p$ (bilinearity).
- $\hat{e}(g, g) \neq 1$, 1 is identity in $\mathcal{G}_2$.
- Computation of $\hat{e}(A, B)$ is efficient.

We now describe the various algorithms:
AccGen: Let $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \longrightarrow \mathcal{G}_2$ be a bilinear pairing such that $|\mathcal{G}_1| = |\mathcal{G}_2| = p$ for some λ bit prime p (λ is the security parameter). Let g_0 be a generator of $\mathcal{G}_1$ and $\mathcal{G}_q =< h >$ be a cyclic group of prime order q such that $\mathcal{G}_q \subset \mathbb{Z}_p^*$. The generation algorithm picks α randomly from $\mathbb{Z}_p^*$. Define function $f : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \longrightarrow \mathbb{Z}_p^*$ such that $f : u, y \rightarrow u(y + \alpha)$. Define function $g : \mathbb{Z}_p^* \times \mathcal{G}_1$ s. t., $g : y \rightarrow g_0^y$. The domain of accumulatable elements is $\mathcal{G}_q$ and the auxiliary information is α. To compute the accummulator $v = g \otimes f(1, Y)$ if α is available is efficient. If α is not available, one can publish $g_0^{\alpha^i}$ for $i = 0, k$ where k is the maximum number that can be accumulated. If we denote the polynomial $\prod_{y \in Y}(y + \alpha) = \sum_{i=0}^{i=k}(u_i \alpha^i)$ of maximum degree k as $v(\alpha)$, then $v = g \otimes f(1, Y)$ can be efficiently computed as $\prod_{i=0}^{i=k} g_i^{u_i} \in \mathcal{G}_1$ without knowledge of α.
AccAdd: To add element y to the accumulator v, compute $\hat{v} = v^{y+\alpha}$. Deleting y from v gives $\hat{v} = v^{\frac{1}{y+\alpha}}$. Both cases require knowledge of α.
AccWitUpdate: Let w is the original witness for y and v the original accumulator. If y' has been added, the new membership witness for y can be computed as $vw^{y'-y}$.
AccVerify: The verification relation $\Omega(w, y, v) = 1$ iff $\hat{e}(w, g_0^y, g_0^\alpha) = \hat{e}(v, g_0)$. For elements $Y = y_1, \cdots, y_k \in \mathcal{G}_q$, a membership witness for $y \in Y$ can be computed as:

- $w = (\prod_{i=1}^{i=k}(y_i + \alpha) . \frac{1}{\alpha+y}$ if α is known
- $\prod_{i=1}^{i=k} g_i^{u_i} \in \mathcal{G}_1$ otherwise

Please refer to [11] for a detailed description of its construction and properties.

3.2 Verito Protocol

At a high-level, Verito works as follows: When a user initiates a PURCHASE transaction, $\mathcal{P}$ creates the required number of credits by committing to the nominal value requested (using Commit), and sends them to $\mathcal{U}$. $\mathcal{P}$ also provides $\mathcal{U}$ with a commitment key so that $\mathcal{U}$ may open the credits and verify the nominal value of each credit. This is implemented as CheckCredit, implemented as Open(c, r, m) as described in the Pedersen's commitment scheme earlier. $\mathcal{P}$ also white-lists the credits generated by adding them to the accumulator (using AccAdd) it maintains per $\mathcal{U}$; this results in a witness for each credit added to the accumulator, which $\mathcal{P}$ sends to $\mathcal{U}$ along with the credits. Since modifying the accumulator requires that existing witnesses (of past credits still in the user's possession) are updated, $\mathcal{P}$ computes the witness update for the transaction and sends it to $\mathcal{U}$. The witness update is a single quantity (independent of the number of credits in the user's possession). $\mathcal{U}$ combines the witness update with the witness for each credit in his possession (using AccWitUpdate) to derive the new witness for that credit.

The SPEND transaction proceeds as follows: $\mathcal{U}$ first acquires a transaction nonce from a merchant $\mathcal{M}$; this is used to ensure freshness and protect against replay attacks. Next, $\mathcal{U}$ contacts $\mathcal{P}$ with the credit(s) it would like to spend along with their witnesses, the merchant $\mathcal{M}$ it would like to spend the credits with, and the nonce. $\mathcal{P}$ verifies that the credits are present in the accumulator (white-list using AccVerify) it is maintaining for $\mathcal{U}$. If the verification succeeds, it updates the accumulator by removing the credits spent (using AccAdd) and sends $\mathcal{U}$ a witness update, so $\mathcal{U}$ can update the witnesses of other credits in his possession. Note that if $\mathcal{U}$ attempts to re-spend credits he previously spent, the membership check in the accumulator (white-list) will fail and the transaction will be aborted.

When the SPEND transaction succeeds, $\mathcal{P}$ also adds the spent credits to the accumulator (using AccAdd) he is maintaining for the merchant (i.e., merchant $\mathcal{M}$ sent by the user when he initiated the transaction). This accumulator for $\mathcal{M}$ is used as a white-list to protect against $\mathcal{M}$ attempting to encash a credit multiple times. The witness update (for the merchant) that this generates, and the transaction nonce, are together signed by $\mathcal{P}$ as a receipt for the transaction. $\mathcal{P}$ sends this receipt to $\mathcal{U}$, who forwards it to $\mathcal{M}$ as proof of the transaction along with the actual credits. $\mathcal{M}$ stores these credits and applies the witness update (using AccWitUpdate).

In the ENCASH transaction, $\mathcal{M}$ sends a set of credits and their respective witnesses to $\mathcal{P}$. As before, $\mathcal{P}$ checks if the credits are indeed present in the merchant's accumulator (using AccVerify). If the verification succeeds, it updates the accumulator by removing the credits and returning the witness update. It also reveals to the merchant the aggregate commitment opening key (the sum of individual credit opening keys) for the set of credits encashed. The merchant opens the set of credits to learn the aggregate nominal value of the credits he encashed. When $\mathcal{M}$ is later paid, he can ensure his payment matches the aggregate he computed (while accounting for $\mathcal{P}$'s profit).

3.2.1 Concrete Instantiation

Our concrete construction uses the additively homomorphic commitment scheme of Pedersen described above to generate credits. The dynamic accumulator scheme of [11, 28] is used to check the validity of the credits in an efficient manner. The details are provided below:

1) SETUP (1^k): $\mathcal{P}$ runs this algorithm to set up the system parameters. It takes the security parameter as input and generates a symmetric key K and a public-private key pair (pk, sk). While K is used by $\mathcal{P}$ to encrypt parts of the credit that it wants to hide from $\mathcal{M}$, sk is used to sign messages. $\mathcal{P}$ shares this pk with all users and game developers. For each user $\mathcal{U}$ and game developer $\mathcal{M}$, it runs AccGen($1^k, n$) to initialize their respective dynamic accumulators.

2) PURCHASE ($\mathcal{U}$, MONEY): The Purchase transaction is initiated by user $\mathcal{U}$ by sending its id and the amount #credits to $\mathcal{P}$. $\mathcal{P}$ creates #credits number of credits, where each credit is constructed as follows: Depending on the currency type and other transaction details, $\mathcal{P}$ chooses the bucket i to be populated in the credit. It defines $m = 2^j$ where $j = (i-1) \times k$. It chooses r uniformly at random from $\mathbb{Z}_q$ and computes $g^r \cdot h^m \mod p$. Then, it adds the credit to the $\mathcal{U}$'s accumulator by calling AccAdd($\mathcal{U}$, $g^r \cdot h^m \mod p$). The returned witness update wit also forms part of the message returned by $\mathcal{P}$ to $\mathcal{U}$. $\mathcal{P}$ sends back {Credit=($g^r \cdot h^m \mod p$, $E_K(r, m)$), $\sigma_{sk}(Credit)$, r, m, wit}, where $\sigma_{sk}(Credit)$ is a signature on the credit under the signing key sk. Note that the encryption of (r, m) is also included in the credit as this is can be decrypted later by $\mathcal{P}$ to know the value of the credit. $\mathcal{U}$ runs the CheckCredit (c,m) = Open(c, r, m), which verifies if $g^r h^m \mod p$ equals the first part of the Credit, and verifies the signature on the credit using $\mathcal{P}$'s public key pk. $\mathcal{U}$ updates the witnesses for all existing credits by running AccWitUpdate and stores the credit along with its witness for future transactions.

3) SPEND ($\mathcal{U}$, $\mathcal{M}$, CREDITS): When$\mathcal{U}$ wants to spend in-app, it initiates the SPEND transaction by sending its id, $\mathcal{M}$'s id and the transaction number/nonce (received from $\mathcal{M}$) along with a set of credits with respective witnesses to $\mathcal{P}$. $\mathcal{P}$ retrieves r and m for each credit by decrypting $E_K(r, m)$ and checks if $g^r h^m \mod p$ indeed equals the first part of the credit. It also verifies its own signature on the credit. It calls AccVerify ($g^r h^m \mod p, wit$) to check if the credit is in the accumulator of that user. If all these checks are passed, $\mathcal{P}$ removes the credits from $\mathcal{U}$'s accumulator and adds it to $\mathcal{M}$'s accumulator using the AccAdd function. It sends back witness updates for both $\mathcal{U}$ and $\mathcal{M}$ along with a signed acknowledgment of the success of the transaction. $\mathcal{U}$ updates witnesses for all of its credits using AccWitUpdate and also forwards $\mathcal{M}$'s witnesses to the latter.

4) ENCASH ($\mathcal{M}$, CREDITS): Whenever $\mathcal{M}$ wants to encash the credits she has earned through various users, she can send these CREDITS in the ENCASH transaction along with their respective witnesses. Note, $\mathcal{M}$ can send at most $2^k - 1$ CREDITS at a time, to prevent overflow when these CREDITS are multiplied later. $\mathcal{P}$ retrieves r and m for each credit by decrypting $E_K(r, m)$ and checks if $g^r h^m \mod p$ indeed equals the first part of the credit. It also verifies its own signature on the credit. It calls AccVerify ($g^r h^m \mod p, wit$) to check if the credit is in $\mathcal{M}$'s accumulator. If all the checks are passed, it removes the CREDITS from the $\mathcal{M}$'s accumulator using AccAdd function, computes $R = \Sigma r$ and $M = \Sigma m$ of the presented credits and returns the witness update along with money worth $(1 - \texttt{PROFIT}) * M$ to the $\mathcal{M}$. $\mathcal{M}$ checks that the product of the submitted credits *i.e.* $\Pi g^r h^m \mod p$ equals $g^R h^M \mod p$.

4 Properties

In this section, we formally define our security properties as a game between a benign challenger and a malicious adversary (see [33]). We model the adversary and challenger as probabilistic processes that are allowed to communicate with each other, creating a probability space for the attack-game. Associated with each game is a security parameter which affects this probability space. For every "efficient" adversary (a probabilistic polynomial time algorithm), the probability that the adversary succeeds in winning the game should be negligible. Note that the protocol also uses signatures for non-repudiation, and since this is standard, we do not include it in the games. As noted in Sect 2, we assume that no parties collude in the system. Thus, each of our security games assumes one party is the challenger, and the other the adversary, while the third party is neutral. The properties are defined in Section 4.1. We present proofs and how can use these properties to achieve our global invariants in Sect 4.2.

4.1 Property Definitions

1) User transparency: For this game, the challenger is the user $\mathcal{U}$ and the adversary is the platform $\mathcal{P}$. $\mathcal{P}$ should not be able to claim different values for the credits once they are committed at the time of purchase.

- The challenger $\mathcal{U}$ initiates PURCHASE ($\mathcal{U}$, 1)

- The adversary creates a single credit c_i which includes the commitment, the nominal value m_i and its commitment key, and sends c_i to the challenger. The challenger runs CheckCredit (c_i, m_i) and accepts only if its output is true.

The adversary wins the attack-game if $\mathcal{P}$ can produce another $m_i' \neq m_i$ and CheckCredit (c_i, m_i') is true.

2) Merchant transparency: For this game, the challenger is $\mathcal{M}$ and the adversary is $\mathcal{P}$. The idea is that the platform cannot cheat merchants when credits are encashed.

- The challenger initiates an ENCASH ($\mathcal{M}$, CREDITS) transaction by presenting credits (c_k) along with their corresponding accummulator witnesses w_k.
- The adversary sends (M, R) in response to the challenger.
- The challenger computes Aggregate (c_k) (product of credits in our protocol) and Commit (M, R) and checks if Aggregate (c_k) = Commit (M, R)

The adversary wins the attack game if Commit (M, R) = Aggregate (c_k) and $M \neq \Sigma m_k$.

3) User fairness: For this game the challenger is $\mathcal{U}$ and the attacker is $\mathcal{M}$. We assume that $\mathcal{P}$ is not malicious. The idea is that the merchant cannot distinguish between different valued (say paid and free) credits.

- The challenger picks r from $\{0, 1\}$ uniformly at random and sends credit c_r to the adversary. Previously, the challenger has initiated PURCHASE ($\mathcal{U}$,1), to obtain c_0 and c_1 with nominal values m_0 and m_1 respectively from $\mathcal{P}$.
- The adversary returns $r' \in 0, 1$

The adversary wins the attack-game if $\mathcal{M}$ can correctly return $r' = r$ with probability non-negligibly higher than $\frac{1}{2}$.

4) Double spending prevention: For this game the challenger is $\mathcal{P}$ and the attacker is $\mathcal{U}$. $\mathcal{M}$ is only an observer, and has a natural incentive to be honest, as double spending by $\mathcal{U}$ will impact $\mathcal{M}$'s profit directly. The idea is that the user cannot spend a used credit at any merchant more than once. The adversary purchases n credits $\mathcal{C} = \{(c_1, m_1), \cdots, (c_n, m_n)\}$ and obtains the appropriate witnesses w_i.

- The adversary initiates a SPEND ($\mathcal{U}$, $\mathcal{M}$, $\mathcal{C}_1$) of $\mathcal{C}_1 \subset \mathcal{C}$ consisting of $l < n$ credits by sending the corresponding witnesses
- The challenger retrieves the m_is corresponding to the credits in $\mathcal{C}_1$, and calls AccVerify(m_i, w_i) on the corresponding witnesses and updates these witness values and communicates it to both $\mathcal{M}$ and $\mathcal{U}$
- The adversary initiates another SPEND ($\mathcal{U}$, $\mathcal{M}$, $\mathcal{C}_2$) of $\mathcal{C}_2 \subset \mathcal{C}$ consisting of $m < n$ credits and $(C)_1 \cap (C)_2 \neq \phi$, by sending the corresponding (updated) witnesses

The adversary wins the attack-game if upon retrieving the corresponding m_js, $j \in \mathcal{C}_2$, AccVerify(m_j, w_j) succeeds.

5) Double encashment prevention: For this game the adversary is $\mathcal{M}$ and the challenger is $\mathcal{P}$. Users are not involved in encash. The idea is that the merchant cannot encash the same credit twice. We assume that the adversary has n credits $\mathcal{C} = \{(c_1), \cdots, (c_n)\}$ and obtains the appropriate witnesses w_i.

- The adversary initiates an ENCASH ($\mathcal{M}$, $\mathcal{C}_1$) of $\mathcal{C}_1 \subset \mathcal{C}$ consisting of $l < n$ credits by sending the corresponding witnesses
- The challenger retrieves the m_is corresponding to the credits in $\mathcal{C}_1$, calls AccVerify(m_i, w_i) for $1 \leq i \leq l$ on the corresponding witnesses, and updates these witness values and communicates it to $\mathcal{M}$
- The adversary initiates another ENCASH ($\mathcal{U}$, $\mathcal{M}$, $\mathcal{C}_2$) of $\mathcal{C}_2 \subset \mathcal{C}$ consisting of $m < n$ credits and $(C)_1 \cap (C)_2 \neq \phi$, by sending the corresponding (updated) witnesses

The adversary wins the attack-game if upon retrieving the corresponding m_js, $j \in \mathcal{C}_2$, AccVerify(m_j, w_j) succeeds.

4.2 Analysis

We show how these properties are achieved in our instantiation, and describe what cryptographic hard-problems are needed for their security.

1) User transparency: We will show that the probability of the adversary $\mathcal{U}$ winning this game is negligible in the security parameter. The security parameter here is defined in terms of the number of bits needed to represent the commitments c_i, i.e., the number of elements n of a group G of cyclic order with generators g, h. We will use the discrete logarithm (DL) problem in G for a given $h \in G$, compute $r \in \mathbb{Z}_n$ such that $h = g^r$. The DL assumption is a computational hardness assumption: Given G, for all PPT algorithms A, $Pr[A(h) = \log_g h]$ is negligible.

From our definition of the User transparency game, $Pr[Adv\ wins] = Pr[\text{CheckCredit}(c_i, m_i) = 1 \wedge \text{CheckCredit}(c_i, m_i^{'}) = 1 \wedge m_i \neq m_i^{'}]$.

Thus, in our protocol,

$$\begin{aligned} Pr[Adv\ wins] &= Pr[c_i = g^{r_i} h^{m_i} \wedge c_i = g^{r_i^{'}} h^{m_i^{'}} \\ &\quad \wedge m_i \neq m_i^{'}] \\ &= Pr[g^{r_i} h^{m_i} = g^{r_i^{'}} h^{m_i^{'}} \wedge m_i \neq m_i^{'}] \end{aligned}$$

For given $m_i, m_i^{'}(\neq m_i)$ and r_i, coming up with $r_i^{'}$ that satisfies the above relation is equivalent to finding the discrete logarithm (DL) of $g^{r_i}h^{m_i - m_i^{'}}$ with respect to generator g, which can happen with only negligible probability (DL assumption).

2) From our definition of the merchant transparency attack-game, $Pr[Adv\ wins] = Pr[\mathsf{Commit}(M,R)] = \mathsf{Aggregate}(c_k) \wedge M \neq \Sigma m_k]$.

$$\begin{aligned} Pr[Adv\ wins] &= Pr[\mathsf{Commit}(M,R) = \\ &\qquad \mathsf{Aggregate}(c_k) \wedge M \neq \Sigma m_k] \\ &= Pr[g^R h^M\ mod\ p = \\ &\qquad \Pi g^r h^{m_k}\ mod\ p \wedge M \neq \Sigma m_k] \end{aligned}$$

From the homomorphic property of Pedersens commitments, we know that $\mathsf{Commit}(\Sigma m_k) = \Pi\mathsf{Commit}(m_k)$, so the adversary has to find a R equal to the discrete log of $g^{\Sigma r}h^{\Sigma m_k - M}$. Finding such a R is equivalent to solving the DL problem. The DL Assumption asserts that the probability of this is negligible.

3) From our definition of the user fairness attack-game $Pr[Adv\ wins] = Pr[r' = r]$. Now r is picked by the challenger uniformly at random, and causes the adversary to obtain either $c_0 = g_1^r h^{m_0}\ mod\ p$ or $c_1 = g_2^r h^{m_1}\ mod\ p$. Since, the values c_0 and c_1 are information-theoretically indistinguishable from random numbers (hiding property of Pedersen's commitment), the adversary cannot guess r with probability better than random chance.

4) Since in the double spending attack-game, $(C)_1 \cap (C)_2 \neq \phi$, let us focus on one credit c_k in $(C)_1 \cap (C)_2$. When this credit is presented in the SPEND transaction (as part of $(C)_1$), the challenger removes c_k from the accumulator and sends an update for the adversary's witnesses. Now, since credits get added only once to the accumulator using the PURCHASE transaction, there is only a negligible probability that the adversary can produce a witness for c_k when presenting it again in the SPEND transaction (as part of $(C)_2$). This follows from the fact that security of our accumulator scheme, which is equivalent to solving the q-strong Diffie-Hellman problem.

5) From our definition of the double encashment attack-game, the probability that the adversary wins depends on the same property as the double spending, and the analysis follows from 4) with the ENCASH instead of SPEND calling $\mathsf{AccVerify}$ under the same conditions.

Note that all transactions have a built in authentication and non-repudiation properties (using encryption and digital signatures). We summarize our properties and the cryptographic assumptions required for ready reference below

1. Transparency: User transparency follows from the binding property of Pedersen's commitment (based on the Discrete-Log assumption). Merchant transparency follows from the binding property and the homomorphic property of Pedersen commitments.
2. User Fairness follows from the hiding property (information-theoretic, no assumption) of Pedersen commitments
3. Double spending and Double encashment protection ensue from the Accumulator scheme (Decisional Diffie Hellman and q-Strong Diffie Hellman assumptions)

Global Properties: It is easy to see that Verito system guarantees global transparency, as the value of the credits in PURCHASE, SPEND and ENCASH transactions can be verified by the interested parties as soon the transaction completes by the concerned parties. Note that ENCASH is accompanied by a privacy policy that may not allow certain kinds of transactions, but if a transaction completes that adheres to this policy, the guarantee is still valid. Since the property is true for every transaction, it is true globally.

Let all users verify all their PURCHASE transactions by checking if the product of their credits equals the opened value of their sum revealed by $\mathcal{P}$, and no credits are duplicated when a user spends them with $\mathcal{M}$s. Further all $\mathcal{M}$s verify each ENCASH, assuming the privacy policy allows them. Arbitrage is not an issue as different currencies are not allowed to mix (by credit design).

Now let us examine the Fairness property. As long as the $\mathcal{M}$ can only see the aggregate value of the tokens presented, and cannot game the system (because of the privacy policy) to infer values of individual tokens, $\mathcal{M}$ cannot discriminate between credits with different nominal values.

Non-repudiation, i.e., both double-spending and double-encashment are prevented by the white-list accumulator scheme.

Scalability is achieved through the use of fixed-size accumulators (sz_a), fixed-size credits (sz_c), fixed-size witnesses (sz_w), and fixed-size witness updates (sz_{wu}). The storage at $\mathcal{P}$ is $\mathcal{O}\ (1)$ for each user and merchant, and independent of the number of credits they possess. The response to the PURCHASE transaction for n credits is of size $n(sz_c + sz_w) + sz_{wu})$ which is $\mathcal{O}\ (n)$ in the credits purchased, and independent of the total number of credits the user already has. The messages in the SPEND and ENCASH transaction for n credits are similarly $\mathcal{O}\ (n)$ and independent of the total number of credits the user or the merchant have.

5 Implementation

Our implementation of Verito comprises of the following components: Interfaces for $\mathcal{P}$, $\mathcal{M}$ and $\mathcal{U}$, a credit generation module using Pedersen's commitment scheme, a double-spending prevention module using dynamic accumulators from [28] and an example game which allows for in-game purchases. $\mathcal{P}$ exposes its functionality over HTTP, and can therefore interact with programs written in any language, including browser-based applications.

The commitment part of our $\mathcal{P}$ implementation consists of 760 lines of C code. The accumulator part consists of an additional 130 lines of C++ code. Our example $\mathcal{M}$ component consists of 470 lines of C, and $\mathcal{U}$ component consists of 130 lines of HTML and Javascript. We use the libevent library [25] for all HTTP communication. We use the NTL library [29] for performing modular arithmetic with $k = 1024$ unless otherwise specified, and an in-house library for pairing-based operations. The example game including both the $\mathcal{M}$ and $\mathcal{U}$ components was created in under six hours by a single person demonstrating the ease of programming to the $\mathcal{P}$ webservice API.

5.1 Optimizations

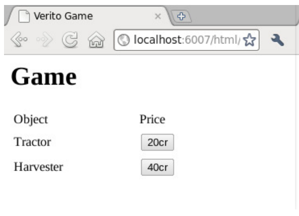
While our commitment module, and most of the accumulator module could perform at high-speed, one accumulator operation (AccVerify) was the performance bottleneck. We design an optimization that avoids this bottleneck in most cases.

We cache the last l credits added to an accumulator. Credits are removed from the cache either when they are deleted from the accumulator (AccUpdate), or evicted (e.g., in FIFO order) when cache size exceeds k. As a result, if a credit exists in the cache, it is guaranteed to exist in the accumulator (i.e., AccVerify is guaranteed to succeed). If, however, a credit does not exist in the cache, AccVerify must be executed to check if the credit exists in the accumulator. The optimization, thus, skips AccVerify when the credit exists in the cache. Note that generating the witness update (after deleting a credit from the accumulator) is not impacted by skipping AccVerify when the credit is guaranteed to exist in the accumulator.

Picking l and the eviction policy to maximize the cache-hit rate depends on the application, and is best arrived at during performance tests, and we do not prescribe a fixed value in this paper.

5.2 Putting it Together

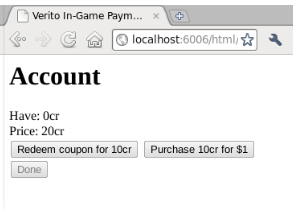
Figure 3 demonstrates through screenshots our proof-of-concept in-game purchase scenario, and how Verito can detect an accounting anomaly we deliberately injected into $\mathcal{P}$. Figures 3a and 3b show an in-game shopping menu, and the user's in-game credits account where the user can purchase credits by paying actual money, or redeem a promotional coupon (modeled along Facebook's credit system as


Verito Game
localhost:6007/html/

Game

Object	Price
Tractor	20cr
Harvester	40cr

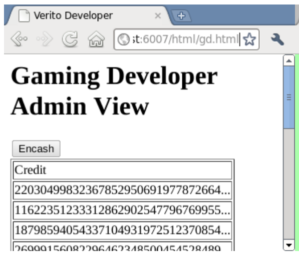
(a)


Verito In-Game Paym...
localhost:6006/html/

Account

Have: 0cr
Price: 20cr

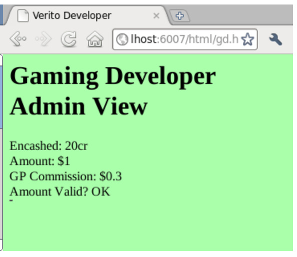
(b)


Verito Developer
st:6007/html/gd.html

Gaming Developer Admin View

Credit
22030499832367852950691977872664...
11622351233312862902547796769955...
18798594054337104931972512370854...
26999156082296462348500454528489

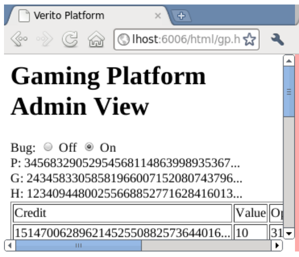
(c)


Verito Developer
lhost:6007/html/gd.h

Gaming Developer Admin View

Encashed: 20cr
Amount: $1
GP Commission: $0.3
Amount Valid? OK

(d)

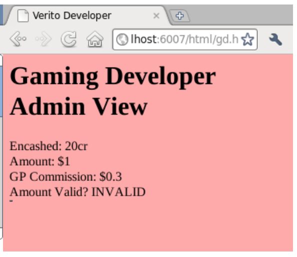

Verito Platform
lhost:6006/html/gp.h

Gaming Platform Admin View

Bug: Off [✓ On]
P: 34568329052954568114863998935367...
G: 24345833058581966007152080743796...
H: 12340944800255668852771628416013...

Credit	Value	Op
15147006289621452550882573644016...	10	31

(e)


Verito Developer
lhost:6007/html/gd.h

Gaming Developer Admin View

Encashed: 20cr
Amount: $1
GP Commission: $0.3
Amount Valid? INVALID

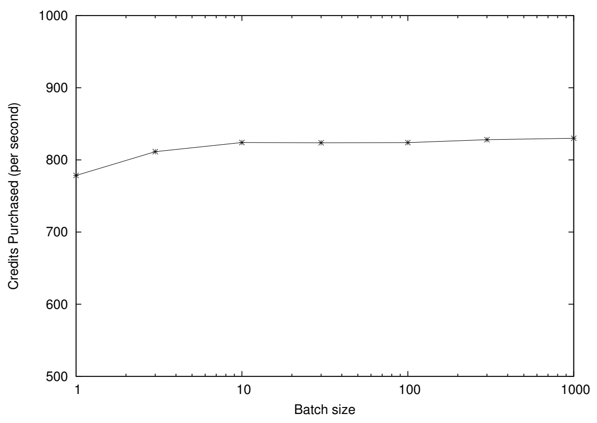
(f)

Figure 3. a,b: $\mathcal{U}$ purchases in-game items. c,d:$\mathcal{M}$ encashes credits. e,f: Bug in $\mathcal{P}$ results in incorrect pay-out. $\mathcal{M}$ catches the problem.

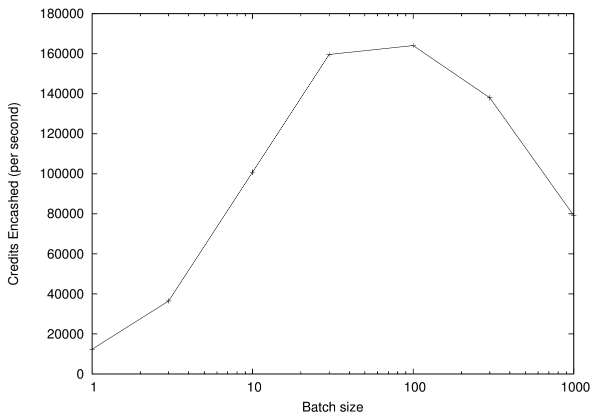
discussed earlier). In our test, the user redeems a coupon for 10cr, and purchases 10cr for $1. Once the user buys the Tractor, $\mathcal{M}$'s console (Figure 3c) shows the credits received from the user. When $\mathcal{M}$ chooses to encash his credits, the verification succeeds (Figure 3d) showing the value of the encashed credits as $1. Next, we enable an accounting anomaly in $\mathcal{P}$ (Figure 3e) where $\mathcal{P}$ siphons off $1 from the payout. We re-run the test, but let the user buy all 20cr by paying $2. The (anomalous) payout is therefore $1 instead of $2; this duplicates the previous scenario where $\mathcal{P}$ offers to payout $1 for the 20cr encashed. $\mathcal{M}$ can, however, detect the accounting anomaly since the verification fails as shown in Figure 3f.

6 Performance Evaluation

We evaluate Verito performance using macro-benchmarks, and perform a feasibility analysis based on transaction volumes we acquired from a large gam-



(a) PURCHASE



(b) ENCASH

Figure 4. Performance macro-benchmark

Operation (single-thread)	**Time**
PURCHASE	
$\mathcal{P}$: Accumulator Add	0.14 ms
$\mathcal{U}$: Witness Update	3.12 ms
SPEND	
$\mathcal{P}$: Accumulator Verify worst-case	156 ms
$\mathcal{P}$: Accumulator Verify best-case*	<0.1 ms
$\mathcal{P}$: Accumulator Update $\times$ 2	0.28 ms
$\mathcal{U}$: Witness Update	3.12 ms
$\mathcal{M}$: Witness Update	3.12 ms
ENCASH	
$\mathcal{P}$: Accumulator Verify worst-case	156 ms
$\mathcal{P}$: Accumulator Verify best-case*	<0.1 ms
$\mathcal{P}$: Accumulator Update	0.14 ms
$\mathcal{M}$: Witness Update	3.12 ms

*Cache-based optimization

Table 1. Accumulators micro-benchmark

ing company. Macro-benchmarks test the end-to-end aspects of our implementation, including not only the underlying cryptography performance, but also overheads associated with processing HTTP requests, and serializing and deserializing credits. All experiments are run on a typical workstation-class machine with an Intel Core2 Duo processor running at 2 GHz with 4 GB memory.

6.1 Macro-Benchmarks

We benchmark the performance of our $\mathcal{P}$ implementation (with the cache-based optimization) by measuring end-to-end latency of each of the operations it participates in. The measured latency includes HTTP overheads, however, we exclude network bandwidth and latency related overheads by using the loopback network interface. We restrict the $\mathcal{P}$ to only one core of the dual-core machine, and use the second core for the benchmarking process.

Purchase. Figure 4a plots the performance of PURCHASE transactions as a function of the size of the batches in which credits are purchased. Since the end-to-end latency includes a fixed communication overhead and variable cryptography cost, the larger the batch-size, the more credits the network overhead is amortized over, and therefore the higher the performance. That said, cryptographic cost clearly dominates network overhead as performance holds steady at around 830 credits per second with minimal improvement from batching. Each datapoint is the average per-second throughput computed for a benchmark-run lasting 15 seconds.

Spend/Encash. Figure 4b plots the performance of ENCASH transactions as a function of batch-size. The impact of batching is far more dramatic since the fixed overheads dominate. Encashing credits in batches of around 100 yields the best performance for our implementation. Network overheads dominate below this threshold. Above this threshold memory-related overheads (i.e., holding larger requests and responses in memory). While we believe we can optimize our implementation to maintain peak performance for larger batches, there is little reason to, considering ENCASH performance already outstrips PURCHASE performance by two orders of magnitude.

The performance of the validation step in the SPEND transaction (to guard against invalid credits and double-spent credits) is identical to ENCASH performance, and indeed uses identical code. The only difference is that ENCASH in addition generates the correctness proof (for opening commitments). Generating the proof adds negligible overheads since it involves only product operations on large numbers (and no modular exponentiations).

6.2 Micro-Benchmarks

We benchmark our accumulator performance in Table 1. PURCHASE transactions require very little processing at $\mathcal{P}$ (0.14 ms). $\mathcal{U}$ must apply the witness update to all witnesses he has, however, this operation does not need to be synchronous. The witnesses can be updated lazily at any time before the credit is spent. Each witness update operation takes 3.12 ms on a typical laptop (single-threaded), which can be trivially parallelized. Thus thousands of credits can be updated in just a few seconds.

As mentioned, the dominant performance cost comes from the unoptimized verify operation in the SPEND and ENCASH transactions. By using our cache-based optimization, and tuning the cache size for best performance, we can reduce the verification cost to practically nothing ($<$ 0.1 ms) for almost all requests. As before, witness updates can be performed lazily in parallel.

6.3 Feasibility Study

To estimate feasibility, we approached a large online-gaming platform for transaction volume data; the online-platform in question transacts several hundreds of millions of dollars each year. Given the exchange rate between \$1 and credits in the platform, and the platform's growth target over the next few years, a rough target for Verito would be to generate on the order of 100 billion credits in a year.

Our $\mathcal{P}$ implementation can generate 71 million credits per day (26 billion per year) on a single-core. Thus a single quad-core workstation we estimate can serve the demands of the real-world gaming platform we approached.

Overall we believe Verito is both practical (in terms of performance) and significantly raises the functionality bar (in terms of adding transparency and accountability) to Internet-scale gaming platforms.

7 Discussion

The biggest concern with Verito is how to incentivise adoption. It is true that in the current climate, Facebook or Xbox (perhaps) have little reason to give up transparency of their economies. We envision two complementary approaches that can drive Verito adoption.

Verito overlay. Verito can run as an overlay on top of an existing economy (e.g., Facebook credits). The user installs a browser extension that modifies the existing Facebook UI, say, to add the option of purchasing "verified credits", which are processed by a Verito platform that runs independently from Facebook. Game developers allow users to pay using regular Facebook credits, or using "verified credits". The independent Verito platform, which may be run by a startup, makes a commission per-transaction like Facebook does. Game developers don't get any guarantees for regular credits, but for the fraction of their income coming from "verified credits", they have strong guarantees.

Regulation. While virtual economies have, so far, escaped regulation, the same reasons for regulating banks applies to virtual economies — i.e., protecting customers. If today a user were to buy \$100 in Facebook or Xbox credits, he would not have any legal recourse if due to a bug or a breach Facebook or Xbox were to lose these credits. Regulators could encourage (or force) virtual economies to be transparent to protect consumers. Verito presents a proof-of-concept of how it is technologically feasible to add transparency while balancing competing interests and constraints that naturally arise in these virtual economies.

8 Related Work

The past decade has seen a growing interest in research revolving around virtual economies, what drives them [24], their legal implications [26], etc. (See [9] for an extensive bibliography). However, there has been very little work on the design and implementation of these virtual economies. Nevertheless, token (or credit) based virtual economy systems exist around the world: Xbox Live Marketplace uses Microsoft Points as the currency to purchase games and other online services without repetitive use of credit cards or banking accounts [7]; Facebook Credits is a system that enables users to buy digital and virtual goods in games and apps across Facebook [3]; Octopus Card (a contactless stored-value card) was originally introduced to collect fares in the Hong Kong Mass Transit Railway (MTR) system, and it is now a widely used payment system in supermarkets, fast-food restaurants, car parks, etc. [8]; and in Second Life, the virtual economy of Linden Dollars allows users to make in-game purchases to acquire virtual goods and services [2].

Microsoft Points, Octopus Cards and Linden Dollars are transparent systems, in that the nominal value associated with a spent credit is clearly known to the player encashing it for real money. However, these systems offer very little flexibility with respect to accommodating multiple nominal values (whether for pricing differently in different geographies or for offering promotions/discounts to encourage active user-participation in the virtual economy, etc.). On the other hand, the Facebook Credits system admits a flexible pricing of credits, but does not provide transparency – game developers cannot independently verify the correctness of the amount of money received from Facebook in-return for the spent credits. Facebook may be able to trivially offer a transparent payout summary, but that would compromise the privacy of users (exposing users to the risk of preferential/unequal treatment by game developers). Moreover, the differentials in credit nominal values may be exploited by game developers masquerading as users, raising the possibility of arbitrage.

Recently, a token-based scheme for privacy-preserving toll collection was proposed in [12, 27]. Their focus was on preserving privacy of user-locations during automatic toll collection. While their scheme uses similar primitives as ours, such as homomorphic commitment schemes, their architecture does not quite adapt to the virtual economy settings we consider, with multiple nominal values for credits.

Electronic cash (e-cash) systems [18] also have an architecture similar to our system in terms of the kinds of players and transactions. The three players in an e-cash system are: the Bank, customers and merchants, while the three main transactions are Withdraw, Payment and Deposit. Even some of the desired security properties are similar: Unforgeability of digital cash and over-spending (sometimes called double spending) prevention. The key differences are the requirements of anonymity of users (no one should be able to infer the identity of the user by viewing the e-cash tokens) and unlinkabilty of user transactions (no one should be able to infer that two payment transactions are with the same user). Because of these privacy requirements, e-cash systems tend to rely on stronger and less efficient cryptographic techniques. There are also two desirable properties in our system which are often absent in e-cash systems: flexibility (multiple-valued credits) and non-repudiation. While we support multiple nominal values for virtual credits in Verito, we also provide non-repudiation as a security property.

Finally, we also present references for homomorphic commitment schemes and dynamic accumulators: Only a few direct constructions of homomorphic commitments are known [20, 32]. Many other constructions can be derived from homomorphic encryption schemes [22, 30, 31], but [32] continues to remain one of the most efficient and widely used commitment schemes. Several other commitment schemes [21, 23] have been derived from it to achieve some additional properties. In terms related work for of dynamic accumulators, they were first introduced in Camenisch et al. [17], and many schemes have been proposed [13, 15]. The dynamic accumulator scheme used is this work is a modification [11] of the scheme presented by Nguyen [28] and we picked the scheme for its efficiency.

9 Summary and Future Work

In this paper we argue that the accountable virtual economy problem is very relevant to the industry today. Credits, which constitute the currency in virtual economies, may be associated with multiple nominal values, depending on the geographic region where they are sold, or depending on whether the credits are paid-for or offered free as part of a promotion, etc. Currently it is not possible for users and content-creators to audit the accounting practices of platform-providers, i.e., they cannot validate the value of credits spent on the platform without resorting to legal procedures (as in the case of a dispute). In addition to scalability as a performance requirement, we identify three desirable security properties viz., transparency, fairness and non-repudiation, which are currently enforced by simply trusting the platform provider. Using well-established cryptographic primitives such as commitment schemes and dynamic accumulators, we develop a framework that satisfies these security properties, and is efficient and scalable to a large number of users, credits and transactions.

From the adoption angle, a shortcoming of our approach is that it requires explicit cooperation from existing platforms. While one can imagine regulatory pressure to extract this cooperation, a better solution would be one that leverages existing platforms without major modifications. To address this, we propose a web-proxy based overlay solution that can be incrementally deployed. We hope this paper will spark further research within the community to explore all aspects of this important problem.

References

[1] Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/.

[2] Economy of second life.

[3] Facebook credits. http://developers.facebook.com/docs/credits/.

[4] Facebook credits for developers. https://www.facebook.com/help/?page=199374706772290.

[5] Facebook Inc. Form S1. http://www.sec.gov/Archives/edgar/data/1326801/0001193 12512175673/d287954ds1a.htm.

[6] Inside virtual goods the us virtual goods market 2011 - 2012. http://www.insidevirtualgoods.com/us-virtual-goods/.

[7] Microsoft points. http://www.xbox.com/en-US/Live/MicrosoftPoints.

[8] Octpus card. http://en.wikipedia.org/wiki/Octopus_ card.

[9] Virtual economy research network bibliography. http://virtual-economy.org/biblio.

[10] M. Abe, R. Cramer, and S. Fehr. Non-interactive distributed-verifier proofs and proving relations among commitments. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '02, pages 206–223, London, UK, UK, 2002. Springer-Verlag.

[11] M. Au, P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for ddh groups and their application to attribute-based anonymous credential systems. In M. Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 295–308. Springer Berlin / Heidelberg, 2009.

[12] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens. PrETP: Privacy-preserving electronic toll pricing. In *USENIX Security '10*, 2010.

[13] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *Advances in Cryptology (EUROCRYPT '97)*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer Berlin / Heidelberg, 1997.

[14] M. W. Bell. Virtual Worlds Research: Past, Present & Future. *Journal of Virtual Worls Research*, 1(1), July 2008.

[15] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In T. Helleseth, editor, *Advances in Cryptology (EUROCRYPT '93)*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer Berlin / Heidelberg, 1994.

[16] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Advances in Cryptology (CRYPTO 2002)*, volume 2442 of *Lecture Notes in Computer Science*, pages 101–120. Springer Berlin / Heidelberg, 2002.

[17] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and applications to efficient revocation of anonymous credentials. *Advances in Cryptology (Crypto'02)*, 2442:61–76, 2002.

[18] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[19] D. Chaum, J.-H. Evertse, and J. Van De Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Proceedings of the 6th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'87, pages 127–141, Berlin, Heidelberg, 1988. Springer-Verlag.

[20] R. Cramer and I. B. Damgard. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *IN PROC. CRYPTO*, pages 424–441. Springer-Verlag, 1997.

[21] I. Damgard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '02, pages 125–142, London, UK, UK, 2002. Springer-Verlag.

[22] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[23] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 16–30, London, UK, 1997. Springer-Verlag.

[24] Y. Guo and S. Barnes. Virtual item purchase behavior in virtual worlds: an exploratory investigation. *Electronic Commerce Research*, 9(1–2):97–113, 2009.

[25] LibEvent. http://monkey.org/ provos/libevent/.

[26] H. Lin and S. C-T. Cash trade within the magic circle: Free-to-play game challenges and massively multiplayer online game player responses. In *Proceedings of DiGRA 2007: Situated Play*, 2007.

[27] S. Meiklejohn, K. Mowery, S. Checkoway, and H. Shacham. The phantom tollbooth: privacy-preserving electronic toll collection in the presence of driver collusion. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 32–32, Berkeley, CA, USA, 2011. USENIX Association.

[28] L. Nguyen. Accumulators from bilinear pairings and applications. In *Proceedings of CT-RSA '05*, 2005.

[29] NTL: A Library for doing Number Theory. http://www.shoup.net/ntl/.

[30] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In K. Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 308–318. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0054135.

[31] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[32] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, 1992. Springer-Verlag.

[33] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. *Cryptography ePrint Archive 2004/32*, 2004.