zkLedger: Privacy-Preserving Auditing for Distributed Ledgers

Neha Narula, *MIT Media Lab;* **Willy Vasquez,** *University of Texas at Austin;*
Madars Virza, *MIT Media Lab*

https://www.usenix.org/conference/nsdi18/presentation/narula

This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-939133-01-4

zkLedger: Privacy-Preserving Auditing for Distributed Ledgers

Neha Narula
MIT Media Lab

Willy Vasquez
University of Texas at Austin*

Madars Virza
MIT Media Lab

Abstract

Distributed ledgers (e.g. blockchains) enable financial institutions to efficiently reconcile cross-organization transactions. For example, banks might use a distributed ledger as a settlement log for digital assets. Unfortunately, these ledgers are either entirely public to all participants, revealing sensitive strategy and trading information, or are private but do not support third-party auditing without revealing the contents of transactions to the auditor. Auditing and financial oversight are critical to proving institutions are complying with regulation.

This paper presents zkLedger, the first system to protect ledger participants' privacy and provide fast, provably correct auditing. Banks create digital asset transactions that are visible only to the organizations party to the transaction, but are publicly verifiable. An auditor sends queries to banks, for example "What is the outstanding amount of a certain digital asset on your balance sheet?" and gets a response and cryptographic assurance that the response is correct. zkLedger has two important benefits over previous work. First, zkLedger provides fast, rich auditing with a new proof scheme using Schnorr-type non-interactive zero-knowledge proofs. Unlike zk-SNARKs, our techniques do not require trusted setup and only rely on widely-used cryptographic assumptions. Second, zkLedger provides *completeness*; it uses a columnar ledger construction so that banks cannot hide transactions from the auditor, and participants can use rolling caches to produce and verify answers quickly. We implement a distributed version of zkLedger that can produce provably-correct answers to auditor queries on a ledger with a hundred thousand transactions in less than 10 milliseconds.

1 Introduction

Institutions engage trusted third-party auditors to prove that they are complying with laws and regulation. Traditionally this is done by auditing companies like Deloitte, Pricewaterhouse Coopers, Ernst and Young, and KPMG (known as the "Big Four"), which together audit 99% of the companies in the S&P 500 [19]. This type of auditing is laborious and time-consuming, so regulators and investors do not get real-time access to information about the financial status of institutions. In addition, trusted third parties can make mistakes. The most well-known example of this is the collapse of Arthur Anderson in 2002, after it failed to catch Enron's $100 billion accounting fraud.

Recently, financial institutions are exploring distributed ledgers (or blockchains) to reduce verification and reconciliation costs in an environment with multiple distrusting parties. Distributed ledgers enable real-time validation by all participants (known as *public verifiability*), but at the cost of privacy—every participant must download all transactions in order to verify their integrity. This is untenable for institutions that rely on secrecy to protect strategy and intellectual property (e.g. trading strategies), and for organizations that have to comply with laws and regulation around data privacy (for example, the General Data Protection Regulation in Europe [24]).

Distributed ledgers that support privacy generally operate in one of two ways: either by only committing to hashes of transactions on the ledger, using trusted third parties to independently verify transactions [22, 23], or by using cryptographic commitment schemes to hide the content of transactions [17, 42, 47, 51]. The former class of ledgers suffers from the fact that participants can no longer verify the integrity of private transactions, eliminating the distributed ledger benefit. The latter class still has public verifiability, but either reveals the transaction graph [17, 42] or requires trusted setup, which, if compromised, would let an adversary undetectably create new assets [47, 51]. None of the existing privacy-preserving distributed ledgers offer an important property for real-world systems—efficient auditing.

This paper presents zkLedger, the first distributed ledger system to support strong transaction privacy, public verifiability, and practical, useful auditing. zkLedger provides strong transaction privacy: an adversary cannot tell who is participating in a transaction or how much is being transacted, and crucially, zkLedger does not reveal the *transaction graph*, or linkages between transactions. The time of transactions and the type of asset being transferred are public. All participants in zkLedger can still

* Work completed at the MIT Media Lab.

† Source code and full version of the paper: `zkledger.org`.

verify transactions are maintaining important financial invariants, like conservation of assets, and an auditor can issue a rich set of auditing queries to the participants and receive answers that are provably consistent with the ledger. zkLedger supports a useful set of auditing primitives including sums, moving averages, variance, standard deviation, and ratios. An auditor can use these primitives to measure financial leverage, asset illiquidity, counterparty risk exposures, and market concentration, for the system as a whole or for individual participants.

A set of banks might use zkLedger to construct a settlement log for an over-the-counter market trading digital assets. In these markets, buyers and sellers are matched via electronic exchanges, trades are frequent and fast settlement helps lower counterparty risk. Once a trade is confirmed, a bank can initiate the transfer of the asset as a transaction in zkLedger, which, when accepted in the ledger, settles the transaction. Each bank stores plain-text transaction data in its own private datastores. In zkLedger, instead of storing plain-text transactions, participants store value *commitments* on the distributed ledger. Importantly, these commitments can be homomorphically combined. A bank can prove to an auditor how much of an asset it has on its balance sheet by opening up the product of all transaction commitments it has referencing that asset. The auditor can confirm that the opened product is consistent with the product of the commitments on the ledger.

Designing zkLedger required overcoming three key challenges:

Providing privacy and auditing. The first challenge is to preserve privacy while still allowing an auditor to compute provably correct measurements over the data in the ledger. zkLedger is the first system to simultaneously achieve this, by combining several cryptographic primitives. To hide values, zkLedger uses Pedersen commitments [41]. Pedersen commitments can be homomorphically combined, so a verifier can, for example, confirm that the sum of the outputs is less than or equal to the sum of the inputs, conserving assets. More than that, an auditor can combine commitments to compute linear combinations of values in different rows in the ledger. Previous confidential blockchain systems also use Pedersen commitments to hide values but end up revealing linkages between transactions, and do not support private auditing [17, 34, 42].

zkLedger uses an interactive *map*/*reduce* paradigm over the ledger with non-interactive zero-knowledge proofs (NIZKs) to compute measurements that go beyond sums. These are Generalized Schnorr Proofs [48], which are fast and rely only on widely accepted cryptographic assumptions. Banks can provably recommit to functions over values in the ledger, such as $f: v \rightarrow v^2$, which lets the auditor compute measurements like variance, skew, and outliers without revealing individual transaction details.

Auditing completeness. Since an auditor cannot determine who was involved in which transactions, zkLedger must ensure that during auditing, a participant cannot leave out transactions to hide assets from the auditor. We call this property *completeness*. At the same time, we do not want to reveal to the auditor who was involved in which transactions. zkLedger uses a novel table-construction in the ledger. A transaction is a row which includes an entry for every participant, and an empty entry is indistinguishable from an entry involving a transfer of assets. All of a participant's transfers are in its column in the ledger. An auditor audits *every* transaction when auditing a participant, meaning a participant cannot hide transactions. This presents efficiency challenges, which zkLedger addresses by using *commitment caches* and *audit tokens*, described below.

Efficiency. The third challenge is supporting all of this efficiently. zkLedger implements a number of optimizations: every participant and the auditor keeps *commitment caches*, which are rolling products of every participants' column in the ledger; this makes it fast to generate asset proofs and to answer audits. To reduce communication costs, zkLedger is designed so that participants do not have to interact to construct the proofs for the transaction; the spender can create the transaction alone (this is similar to how other blockchain systems work). But a malicious spender could try to encode incorrect values in the commitments for other banks—we must ensure all of the commitments and proofs are correct and that every participant has what they need to later respond to an audit. To do this, we designed a set of proofs that everyone can publicly verify—transactions with incorrect proofs will be ignored. These proofs ensure that every participant has an *audit token*, which they can use to later open up commitments for that row, and that all proofs and commitments are *consistent*. The audit token and the consistency proofs are publicly verifiable, but do not leak any transaction information. They are also non-interactive, so zkLedger makes progress even if banks cannot communicate, and they are encoded for a specific bank, so a token for one bank cannot be used by another bank to lie to the auditor.

The slowest part of transaction creation and validation are range proofs, which ensure that an asset's value is in a pre-specified range, and prevent a malicious attacker from undetectably creating new assets. Range proofs are $10\times$ the size of the other proofs and take $5\times$ as much time to prove and verify. A naive implementation of

zkLedger might require multiple range proofs, but by using disjunctive proofs, we can multiplex different values into one range proof per entry.

In summary, the contributions of this paper are:

- zkLedger, the first distributed ledger system to achieve strong privacy and complete auditing;
- a design combining fast, well-understood cryptographic primitives using audit tokens and map/reduce to compute provably correct answers to queries;
- an evaluation of zkLedger showing efficient transaction creation and auditing; and
- an analysis of the types of queries zkLedger can support, suggesting that zkLedger can efficiently handle a useful set of auditing measurements.

2 Related Work

zkLedger is related to work in auditing or computing on private data and privacy-preserving blockchains. zkLedger achieves fast, provably correct auditing by creating a new distributed ledger table model and applying a new scheme using zero-knowledge proofs.

2.1 Computing on Private Data

Previous work proposed a multi-party computation scheme in which participants use a secure protocol to compute the results of functions which answer questions about systemic financial risk, the same problem which zkLedger aims to address [3, 10], and network security [14]. This work provides privacy benefits over existing analytics systems by allowing participants to keep their data secret. However, it only supports overall system auditing, it is not a solution to audit individual participants. There is also nothing preventing participants from lying in the inputs to the multi-party computation; they do not achieve completeness.

Provisions [21] is a way for Bitcoin exchanges to prove they are solvent without revealing their total holdings. Provisions uses Proof of Assets and Proof of Liabilities, which are very similar to the zero-knowledge proofs we use in zkLedger. However, in Provisions, an exchange could "borrow" private keys from another Bitcoin holder and thus prove assets they do not actually hold; in fact multiple exchanges could share the same assets. Moreover, Provisions does not provide completeness. By using a columnar construction with a distributed ledger, zkLedger achieves completeness.

In Prio [18], untrusted servers can compute privately on mobile client data. Prio does not operate on distributed ledgers, and thus does not guarantee public verifiability. Prio requires all servers to cooperate in order for client proofs to validate; zkLedger can tolerate non-cooperating participants.

Several systems provide private and correct computing using trusted hardware [4–6, 49, 52]. In our setting, we cannot guarantee that all participants will trust the same hardware provider. In addition, it would be a conflict of interest to use such a system to audit the company providing the trusted hardware.

There are many systems which compute on encrypted data to protect user confidentiality in the event of a server compromise [25, 31, 32, 40, 43, 44, 50]. These systems address a different problem than what zkLedger is trying to solve. Instead, we provide interactive, provably correct auditing over private data generated by many parties.

2.2 Privacy-preserving blockchains

Bitcoin, a decentralized cryptocurrency released in 2009, was the first blockchain [37]. Many companies have explored using a blockchain to record the transfer of assets. These systems are marked by the following characteristics: (1) Multiple, possibly distrusting participants, all with write permissions and no single point of failure or control; (2) A consensus protocol to construct an append-only, globally ordered log with a chain of hashes to prevent tampering with the past; and (3) Digitally signed transactions to indicate intent to transfer ownership.

In Bitcoin and most other blockchains, all transactions are public: every participant receives each transaction, and can verify all the details. Users create pseudonyms by generating one-time use public keys for payment addresses, but transaction amounts and the links between transactions are still globally visible. Confidential Transactions [34] and Confidential Assets [17, 42] are extensions to Bitcoin which blind the assets and amounts in transactions while still ensuring that all participants can validate transactions. Though these systems hide assets and amounts, they leak the transaction graph and do not support private auditing—an auditor would require access to *all* plain-text transactions in order to ensure completeness. The transaction graph alone leaks substantial information [36, 38, 45, 46]; for example, the FBI followed linked transactions to trace bitcoins and used this as evidence in court [28]. zkLedger provides stronger transaction privacy and private auditing, but at the cost of scalability. Transactions in zkLedger are sized order the number of participants in the whole system, requiring more time to produce and verify as the number of participants grows. This makes zkLedger more suitable to ledgers with fewer participants who require more privacy.

Solidus [16] is a distributed ledger system that uses Oblivious RAM to hide the transaction graph and trans-

action amount between bank customers. While this construction also provides private transactions, Solidus can only support auditing by revealing all of the keys used in the system to an auditor, and opening transactions. zkLedger achieves performance similar to Solidus while providing private auditing.

R3's Corda [22], and Digital Asset Holding's Global Synchronization Log (GSL) [23] are distributed ledgers geared towards financial institutions that rely on trusted third parties to pass through information. In Corda, notaries verify transactions and maintain privacy of participants, while GSL segments its ledger, only storing a hash of the values globally and limiting access to fine-grained transaction data. Neither support private auditing.

Another approach is that of Zerocash [47], and its related implementation Zcash [51], an anonymous cryptocurrency based on Bitcoin. Zerocash uses zk-SNARKs [7] to hide transaction amounts, participants, and the transaction graph. The zk-SNARKs as used in Zcash can be extended to handle policies to enforce regulations, KYC/AML laws, and taxes [27]. These policies do not support arbitrary queries, but instead put limits on the new types of transactions that can take place. These ideas have not yet been implemented in a practical system.

zk-SNARKs are quite efficient for some statements but unfortunately, the price of this efficiency is paid in setup assumptions: as of now, all concretely efficient zk-SNARKs require a trusted third party for setup. The consequences of incorrect or compromised setup are potentially disastrous: an adversary who can learn the secret randomness used during setup can make fraudulent proofs of false statements that are indistinguishable from proofs of true statements. In our setting (international banking), such proofs would permit unrestricted creation or destruction of financial assets or liabilities. There may not even be a viable party to perform the one-time trusted setup. For example, Russia might not trust the Federal Reserve or the European Central Bank, or it might not be politically expedient to be seen as doing so. While it is possible to mitigate this concern, e.g., by distributing the setup between multiple parties [8, 11, 12], this process is onerous and expensive. Ideally, the financial integrity of the system would not rely on trusted setup at all. We choose to base consensus-critical portions of zkLedger's design on standard NIZKs.

3 zkLedger Overview

3.1 Architecture

System participants. There are n participants which we call *banks* that issue transactions to transfer digital assets, $\mathsf{Bank}_1, \ldots, \mathsf{Bank}_n$ and an auditor Auditor, that verifies certain operational aspects of transactions performed by the banks (e.g. "is a particular bank Bank_i solvent?"). These roles are not distinct; a bank could also audit. A Depositor or set of Depositors can issue and withdraw assets from the system; for example, the European Central Bank might issue 1M € to Bank_i in the system. Issuance and withdrawal of all assets are controlled by the Depositors and are global, public events.

Transactions. Banks exchange assets by creating *transfer* transactions, whose details are hidden. A transfer transaction captures an event where Bank_i is transferring v shares of asset t to Bank_j. Our scheme supports a bank transferring to multiple other banks, but for simplicity we assume there is one spending and one receiving bank in each transaction. Banks determine the details of a transfer transaction outside of zkLedger, perhaps through an exchange. We assume they use encrypted channels.

Append-only ledger. Banks submit transactions to an append-only ledger, which globally orders all valid transactions. If a digital asset only exists on the ledger, then transfer on the ledger *is* change in legal custody of the digital assets, not merely a record of ownership change, and an Auditor is guaranteed a Bank is not hiding assets. This ledger could be maintained by a trusted third party, by the banks themselves, or via a blockchain like Ethereum or Bitcoin. Maintaining a fault-tolerant, globally ordered log is outside the scope of this paper, but can be done using standard techniques [15, 30, 39].

3.2 Cryptographic building blocks

Commitment schemes. To protect their privacy participant banks do not broadcast payment details, such as the transaction amount, in plain. Instead the banks post hiding *commitments* to the append-only ledger; in particular, zkLedger uses Pedersen commitments [41]. Let $\mathbb{G}$ be a cyclic group with $s = |\mathbb{G}|$ elements, and let g and h be two random generators of $\mathbb{G}$. Then a Pedersen commitment to an integer $v \in \{0, 1, \ldots, s-1\}$ is formed as follows: pick commitment randomness r, and return the commitment $\mathsf{cm} := \mathsf{COMM}(v, r) = g^v h^r$.

Pedersen commitments are perfectly hiding: the commitment cm reveals nothing about the committed value v. In a similar way, the commitments are also computationally binding: if an adversary can open a commitment cm in two different ways (for the same r, two different values v and v'), then the same adversary can be used to compute $\log_h(g)$ and thus break the discrete logarithm problem in $\mathbb{G}$. In zkLedger we choose $\mathbb{G}$ to be the group of points on the elliptic curve secp256k1.

A very useful property of Pedersen commitments is that they are *additively homomorphic*. If cm_1 and cm_2 are

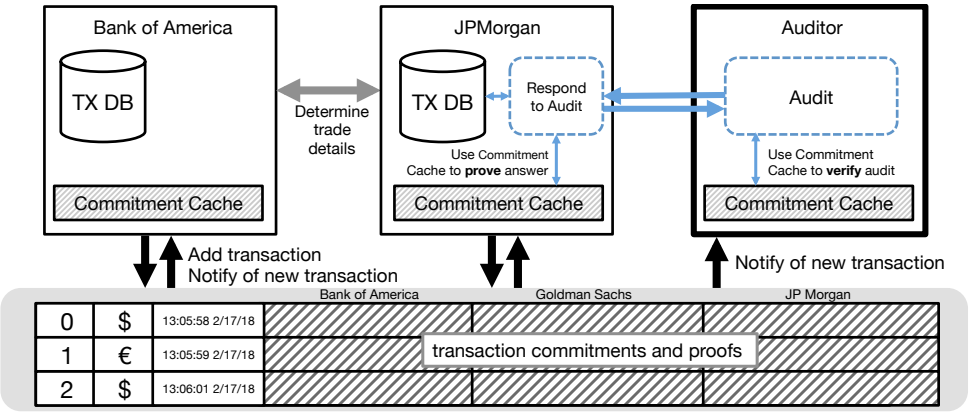

Figure 1: Overall zkLedger system design showing the interactions between the three main entities (banks, auditor, and the shared ledger) in our system. Each bank maintains private state, consisting of the transaction database for transactions the bank originated, and the bank's secret key.

two commitments to values v_1 and v_2, using commitment randomness r_1 and r_2, respectively, then $\mathsf{cm} := \mathsf{cm}_1 \cdot \mathsf{cm}_2$ is a commitment to $v_1 + v_2$ using randomness $r_1 + r_2$, as $\mathsf{cm} = (g^{v_1} h^{r_1}) \cdot (g^{v_2} h^{r_2}) = g^{v_1+v_2} h^{r_1+r_2}$. To speed up transaction generation and auditing zkLedger makes extensive use of the ability to additively combine commitments.

Public-key encryption. Every bank i also generates a Schnorr signature keypair $\mathsf{keypair}$ consisting of a secret key sk_i and public key $\mathsf{pk}_i := h^{\mathsf{sk}_i}$, and distributes the public key pk_i to all other system participants.

Non-interactive zero-knowledge proofs. To make privacy-preserving assertions about payment details zkLedger relies on non-interactive zero-knowledge proofs (NIZKs) [9]. In brief, zero-knowledge proofs concern two parties: the prover, who holds some private data, and the verifier, who wishes to be convinced of some property about this private data. For example, the prover might know the opening of a commitment cm, and wish to convince the verifier that the committed value v is in some range, e.g., $0 \leq v < 10^6$. Using NIZKs, the prover can produce a binary string π, the proof, that simultaneously persuades the verifier, yet does not reveal anything else about v. Verifying π does not require any interaction between the prover and the verifier, and the prover can append π to the ledger, where it can be verified by any party of the system.

In theory, NIZK proof systems exist for all properties in NP whereas the practical feasibility of NIZKs is highly-dependent on the complexity of the property at hand. In particular, algebraic properties in cyclic groups, such as, knowledge of discrete logarithm, equality of values committed in Pedersen commitments, or similar have very efficient NIZK proof systems. The design of zkLedger is carefully structured so that all NIZK proofs have particularly efficient constructions.

3.3 Security Goals

The goals of zkLedger are to hide the amounts, participants, and links between transactions while maintaining a verifiable transaction ledger, and for the Auditor to receive *reliable* answers to its queries. Specifically, zkLedger lets banks issue hidden transfer transactions which are still publicly verifiable by all other participants; every participant can confirm a transaction conserves assets and assets are only transferred with the spending bank's authority. For example, if Bank_i transfers 10,000 € to Bank_j, both the banks and amount are hidden. The asset (€) and time of the transaction are not hidden. zkLedger also hides the transaction graph, meaning which previous transaction(s) supplied the 10,000 € to Bank_i in the first place.

An Auditor can query a Bank about its contents on the ledger, for example "How many euros does Bank_j hold?" A bank should be able to produce commitments which will convince the auditor that the bank's answer to the auditing query is correct, meaning consistent with the transactions on the ledger. zkLedger ensures that if a bank gives the Auditor an answer that is inconsistent with the ledger, the Auditor will catch such attempt of cheating with high probability (and of course, a trustworthy answer must always be accepted).

3.4 Threat model

zkLedger does not assume that banks will behave honestly—they can attempt to steal assets, hide assets, manipulate their account balances, or lie to auditors. We assume banks can arbitrarily collude. zkLedger keeps the amounts and participants of transactions private as long as neither the spender nor receiver in a transaction collude with an observer, like the Auditor. We assume that the ledger does not omit transactions and is available. zkLedger does not protect against an adversary who observes traffic on the network; for example, if only two banks are exchanging messages, it's reasonable to assume the transactions in the ledger involve those banks. Nothing

beyond what is necessarily leaked by an audit is revealed. However, frequent auditing might reveal transaction contents; e.g. if an auditor asks for banks' assets after every transaction.

4 Design

The challenge in creating zkLedger is to practically support complete, confidential auditing—an Auditor should not be able to see individual bank transactions, but a Bank should not be able to hide assets from the Auditor during an audit, and the auditor should be able to detect an incorrect answer.

Figure 1 shows a general overview of zkLedger. There are banks which determine transactions out of band and then settle them by appending transactions to the ledger. The ledger makes sure all banks and any auditors see new transactions. Each bank and auditor maintains a *commitment cache*, which are commitments to summed values used to make creating transactions and responding to audits faster. Each bank also has private stores of plaintext transaction data.

The rest of this section describes the zkLedger transaction format, how banks create transactions, and how a bank can answer a simple query from the auditor.

4.1 Transactions

The ledger in zkLedger is a table where transactions correspond to *rows*, and Banks correspond to *columns*. Each transaction has an entry for each Bank. Figure 2 shows a ledger with n banks. Each entry in a transaction includes a commitment to a value which is the amount of the asset that is being debited or credited to the bank. For example, if Bank_i wants to transfer 100 shares of an asset to Bank_j, i's entry in the transaction would contain a commitment to -100 and j's would contain a commitment to 100. All other entries in the transaction would contain commitments to 0, since none of the other bank balances were changed. This scheme has the nice property that an outside observer can look at a bank's entire column and know that this represents the entirety of the bank's holdings.

Hiding amounts. As described in §3.2, zkLedger does not include the value in plain-text in the transaction. Instead, zkLedger uses Pedersen commitments to commit to the value in transfer transactions. This makes value commitments completely indistinguishable—an outside observer cannot tell the difference between a commitment to a positive value, a negative value, or 0. Recall that a commitment to a value v is $\mathsf{cm} := \mathsf{COMM}(v,r) = g^v h^r$. If desired, a prover can reveal v and r to a verifier who knows cm and the verifier can confirm this is consistent.

Since a transaction in zkLedger contains an entry for every Bank, there is a size-n vector $\vec{\mathsf{cm}}$ committing to values in $\vec{v}$. Each commitment cm_k uses a fresh commitment randomness r_k. Most of the entries will contain commitments to 0, for banks that are not involved with the transaction, but this is not apparent to an outside observer.

zkLedger maintains the following financial invariants:

- A transfer transaction cannot create or destroy assets
- The spending bank must give consent to the transfer and must actually own enough of the particular asset to execute this transaction

In a public blockchain, the validators could simply confirm that these things are true by looking at the history of transactions and the current transaction, and making sure the spending bank has the funds to spend. However, in zkLedger these values are not public. Instead, we create a set of proofs that the spender can create to prove the invariants are maintained. The spender can create a transaction without interacting with any of the other banks.

First, zkLedger introduces a *Proof of Balance* (π^{B}). This is a proof that the transaction conserves assets; no assets are created or destroyed (of course, public issuance and withdrawal transactions do not have such proofs). More formally, the committed values should satisfy $\sum_{k=1}^{n} v_k = 0$. To prove this, the prover chooses the r_k carefully: it should also be the case that $\sum_{k=1}^{n} r_k = 0$. If this is true and the values also sum to 0, then the verifier can check to make sure that $\prod_{k=1}^{n} \mathsf{cm}_k = 1$ for the commitments in the row.

Next, zkLedger must ensure that the spending bank actually has the assets to transfer. To do this, zkLedger introduces a *Proof of Assets* (π^{A}). Other privacy-preserving blockchain systems use *Unspent Transaction Outputs*, or UTXOs, to show proof of assets and prevent double spending. For example, if Alice wants to send a coin to Bob, she chooses one of her coins, creates a new transaction addressing the coin to Bob, and includes a pointer to the previous transaction where she received the coin. This previous transaction is an *output*. All of the validators in the system maintain the invariant that outputs can only be spent once. Unfortunately, in systems without zk-SNARKs, this leaks the transaction graph. In zkLedger, a bank proves it has assets by creating a commitment to the *sum* of the values for the asset in its column, including this transaction. If the sum is greater than or equal to 0, then the bank has assets to transfer. Note that this is true since the bank's column represents all the assets it has received or spent, and the Pedersen commitments can be homomorphically added in columns as well as in rows. In

order to produce a proof with the correct sum, the bank must have seen every previous transaction. This implies that banks must create transactions *serially*. In its own entry where the value is negative, the bank includes proof of knowledge of secret key to show that it authorized the transaction. This requires creating a disjunctive proof—either the committed value for entry i has $v_i \geq 0$, or the creator of the transaction knows the secret key for Bank_i.

Range proofs. Because commitment values are in an elliptic curve group and rely on modulus, we need to make sure that the commited values are within an acceptable range. To see why, note that if N is the order of the group, then $\mathsf{COMM}(v,r) = \mathsf{COMM}(v+N,r)$; there is no way to distinguish between the two. Without a check to make sure the commited value is within the range $[0, N-1]$, a malicious bank could undetectably create assets. To address this, we use range proofs as described in Confidential Assets [42], which uses Borromean ring signatures [35]. zkLedger supports asset value amounts up to a trillion. Range proofs are the most expensive part of the transaction; as described, our scheme requires two range proofs—one for the commitment value, and another for the sum of assets in the column. We can squash the two range proofs down to one range proof by introducing an auxiliary commitment, cm'_i. cm'_i is *either* a re-commitment to the value in cm_i or the sum of the values in the column up to row m, $\sum_{k=0}^{m} v_k$, which can be achieved by computing the product of the commitments in the column, $\prod_{k=0}^{m} \mathsf{cm}_k$. Then, we can do one range proof on the value in cm'_i. Either this is the spending bank, in which case cm'_i must be a commitment to the sum, or it is another bank which is receiving funds or not involved, in which case cm'_i could be either (and it does not matter which it is).

This satisfies the financial invariants described above. However, a particular design choice we made in zkLedger is that a spending bank can create a transaction spending its own assets without interacting with other banks. This means that a malicious bank could create transactions which maintain financial invariants but are ill-formed. We will address this problem after describing how auditing works.

Once created, a bank broadcasts the transaction, and it will be appended to the ledger. If the banks are maintaining the ledger, each bank is responsible for validating the transaction before accepting it to the ledger. If a third party is maintaining the ledger, then the third party should verify the proofs in a transaction before accepting it.

Example transaction. Figure 2 shows a transaction where Bank of America is transferring one million euros to Goldman Sachs. Bank of America creates the transfer transaction, publishing the transaction id, timestamp, and asset type (euros) publicly. Bank of America commits to the amount deducted from its own assets, $-1,000,000$, in its own entry and $1,000,000$ in Goldman Sachs's entry. For every other bank, Bank of America commits to 0. This serves to hide the banks involved in the transfer; no one except Bank of America can distinguish between the commitments to determine which are commiting to nonzero values. Bank of America then broadcasts the transaction to the ledger. The ledger maintainer validates the transaction and appends it to the ledger. Once accepted to the ledger, this serves as a complete transfer of $1,000,000$ euros from Bank of America to Goldman Sachs.

4.2 Auditing Protocol

The Auditor has a copy of the ledger and interacts with the banks to calculate functions on their private data, in order to get a view of the financial system represented by the ledger.

The Auditor audits Bank_i by issuing a query to Bank_i, for example, "How many euros do you hold at time t?". Bank_i responds to the auditor with an answer and a proof that the answer is consistent with the transactions on the ledger. The Auditor multiplies the commitments on the ledger in Bank_i's column for euros, and verifies the proof and answer with the total. This is a commitment to the the total amount of euros Bank_i holds.

The key insight here is that given this table construction, the Auditor can read bank i's column and know that it is seeing every asset transfer involving i. There is no way for i to "hide" assets on the ledger without *actually transferring assets* and giving control to another bank. In contrast, during a traditional audit, a bank could simply not show the auditor some of its balance sheet.

Banks could collude to hold assets for each other temporarily; for example Bank_j might transfer assets to Bank_i and take them back later. For that time period, the assets would be part of Bank_i's holdings. But banks cannot collude after an Auditor poses a query because the Auditor has already specified the time t at which the query applies. Any transfer would necessarily have to be after t. So at this point, it is too late for a malicious bank to create a new transaction transferring assets to another bank.

As described above, a bank can create a transaction transferring its own assets without any interaction. This is common with most blockchain systems, where only the signature of the sender is required to create a valid transaction. There is no in-protocol way for a receiver to object to a transfer. Given our table construction, every bank is affected by every transaction, because a bank must

Metadata			Bank_1	Bank_2	Bank_3	...	Bank_n
ID	Asset	Time	(Bank of America)	(Goldman Sachs)	(JPMorgan)	...	(Citigroup)
1	€	13:06:01 2/17/18	$\mathsf{COMM}(-10^6, r_1)$ Token_1 $\pi_1^{\mathsf{A}}, \pi_1^{\mathsf{B}}, \pi_1^{\mathsf{C}}$	$\mathsf{COMM}(10^6, r_2)$ Token_2 $\pi_2^{\mathsf{A}}, \pi_2^{\mathsf{B}}, \pi_2^{\mathsf{C}}$	$\mathsf{COMM}(0, r_3)$ Token_2 $\pi_3^{\mathsf{A}}, \pi_3^{\mathsf{B}}, \pi_3^{\mathsf{C}}$	⋯ ⋯ ⋯	$\mathsf{COMM}(0, r_n)$ Token_n $\pi_n^{\mathsf{A}}, \pi_n^{\mathsf{B}}, \pi_n^{\mathsf{C}}$

Figure 2: Contents of the ledger pertaining to a transaction that sends 10^6 euros from Bank_1 to Bank_2. Note that while asset type (euro) is visible as part of the metadata, the transaction amount (10^6 €) and participating institutions (Bank of America and Goldman Sachs) remain private. We use Pedersen commitments, so the commitment part of the row has values $g^{-10^6}h^{r_1}$, $g^{10^6}h^{r_2}$,and h^{r_3}, ..., h^{r_n}. Similarly, audit tokens pictured have values $(\mathsf{pk}_1)^{r_1}$, $(\mathsf{pk}_2)^{r_2}$, ... For each bank Bank_i, the corresponding column also includes proof-of-assets π_i^{A}, proof-of-balance π_i^{B}, and proof-of-consistency π_i^{C}.

total all of the commitments in its column to respond to the Auditor—even commitments for transactions in which it was not involved. A malicious Bank_i could create a transaction and not inform the another Bank_j of the r_j used in its entry, even if it is not transferring assets to Bank_j. Bank_j would be unable to respond to the Auditor because it would not be able to open up the product of the commitments in its column.

In order to prove the integrity of a transfer transaction, zkLedger must ensure an additional invariant:

- All banks have enough information in the transaction to open up commitments for the Auditor

zkLedger does this by requiring the spending bank to include a publicly verifiable Token in every entry. This is defined as $\mathsf{Token}_k := (\mathsf{pk}_k)^{r_k}$. Bank_k uses this token to open up the product of its commitments for the Auditor, without needing to know r_k.

Using audit tokens. Consider a query for a sum of values in a bank's column. One way of answering this query would be to reveal $\sum v_k$ and $\sum r_k$. Then the auditor would simply check that these plain values are consistent with the homomorphically computed value $\prod \mathsf{cm}_k = g^{\sum v_k} h^{\sum r_k}$.

However, a bank does not necessarily know all the commitment randomnesses r_k (in particular, these values are unknown for any transaction that the bank was not party to), so the naive approach does not work.

One approach could be to ask the preparer of the transaction (i.e. the sender) to encrypt r_k so that the non-participating bank Bank_k can decrypt it. To prevent the sending bank from placing a "garbage" ciphertext on the ledger (and thus making Bank_k fail the auditor's queries), one would need a zero-knowledge proof of consistency between the encrypted value and the commitment. Constructing a concretely efficient proof for this statement is non-trivial: in a nutshell, standard encryption schemes (e.g. ElGamal) embed plain-text in a group element, while Pedersen commitments would have this value in the exponent.

Our insight is that Bank_k does not need to open $\sum r_k$ to prove that $\sum v_k$ is correct. Instead, suppose that Bank_k wants to claim that $s = g^{\sum v_k} h^{\sum r_k}$ opens up to a value $\sum v_k$. To do so, the bank computes $s' = s/g^{\sum v_k} = h^{\sum r_k}$ and $t = \prod_k \mathsf{Token}_k = h^{\mathsf{sk}\sum r_k}$. Note that the auditor can also compute the values of s' and t from the ledger and the claimed answer $\sum v_k$.

It suffices for the bank to prove that $\log_{s'} t = \log_h \mathsf{pk}$. Observe that both logarithms evaluate to sk so a bank can produce this proof without knowing $\sum r_i$. Moreover, if this equation holds then $t^{1/\mathsf{sk}} = s' = s/g^{\sum v}$, but if the $\sum v$ was incorrect then knowledge of sk would reveal a linear relationship between g and h, which is ruled out by our security assumption.

To show that the r in the Token_k is the same as the r in r_k, we require an additional *Proof of Consistency* (π^{C}). This is a zero-knowledge proof asserting that for each k the value r_k used to form cm_k and Token_k is the same. (See Appendix B for details of how such proof is constructed.)

Note that audit tokens are only useful to the bank opening its commitment; though public, a malicious bank cannot use another bank's Token to successfully open an incorrect result or learn information about other bank's transactions.

4.3 Final transaction construction

For a transfer transaction in row m, each entry i contains the following items:

- **Commitment** (cm_i): $(g^{v_i} h^{r_i})$ a Pedersen commitment to the value we are transferring.
- **Audit Token** (Token_i): $(\mathsf{pk}_i)^{r_i}$. This is used to answer audits without knowing the randomness used in the commitment.
- **Proof of Balance** (π^{B}): a zero-knowledge proof asserting that the committed values satisfy $\sum_{k=1}^{n} v_k = 0$.
- **Proof of Assets** (π^{A}): a new commitment cm'_i, corresponding token Token'_i, and a zero-knowledge proof asserting that either cm'_i is a re-commitment of the value in cm_i or a recommitment to the sum of the values in $\prod_{j=0}^{m} \mathsf{cm}_j$, and cm'_i is in range $[0, 2^{40})$. If the committed value in cm_i is negative, the proof asserts bank i consented to the transfer.

- **Proof of Consistency** (π^{C}): two zero-knowledge proofs asserting the randomness used in cm_i and Token_i are the same, and the randomness used in cm'_i and Token'_i are the same. This is to prevent a malicious bank from adding data to the ledger that would stop another bank from being able to open its commitments for the Auditor.

Transactions may contain additional metadata in plaintext or not. For example, banks might want to include encrypted account numbers, addresses, or identifying information on behalf of a customer to satisfy the Travel Rule specified in the Bank Secrecy Act of 1970 [1]. zkLedger supports auditing over metadata in the transaction as well, but it does not have a way to publicly verify additional metadata.

4.4 Adding or removing banks

zkLedger can support dynamically adding or removing banks if done so publicly. The participants (or another authority) append a signed transaction to the ledger indicating which banks, and thus columns, should be added or removed. For example, to add a new bank to the ledger shown in Figure 2, the involved banks would append a transaction to the ledger indicating an intent to add Bank_{n+1}. From that point forward, all transactions should contain $n+1$ entries. The Proof of Assets for Bank_{n+1}'s entry in each transaction will start at the row where Bank_{n+1} was added. Similarly, if a bank is removed, later transactions should not include entries for that bank. Since all participants can see which banks were added or removed, they can adjust their proofs and verifications accordingly.

4.5 Optimizations

zkLedger employs several optimizations to make producing and verifying these proofs faster, and to support faster auditing. First, caching the product of the commitments in a bank's column improves auditing and proof creation speed. Each bank stores a rolling product of commitments by row and by asset so that it can quickly produce proofs of assets and answer queries from auditors. Using these caches, a bank can quickly answer an auditor's query on a subset of rows in the ledger.

Most transactions in zkLedger do not include every bank. Every bank can pre-generate many range proofs for the value 0. We speedup transaction throughput by parallelizing range proof generation and validation.

5 Auditing

Auditing is a critical component of the financial system, and regulators use various techniques to measure systemic financial risk. Through the use of sums, means, ratios, variance, co-variance, and standard deviation, an auditor in zkLedger can determine the following, among other measurements:

- **Leverage ratios**. zkLedger can show how much of an asset a bank has on its books compared to its other holdings. This is helpful to estimate counterparty risk.
- **Concentration**. Regulators use a measure called the Herfindahl-Hirschman Index (HHI) to measure how competitive an industry is [29].
- **Real-time price indexes**. Auditors can get a sense of the price of assets that are traded over-the-counter and thus not tracked through exchanges.

Natively, zkLedger supports sums, which means linear combinations of values stored in the ledger. This comes from the additive structure of Pedersen commitments. But zkLedger also supports a more general query model, which can be considered in two parts: A *map* step and a *reduce* step.

Basic auditing. Consider the basic example where an auditor wants to determine how much of an asset a bank has on its books. As described in §4.2, the auditor will filter the rows by asset, multiply the entries in the bank's column, and then ask the bank to open the commitment product. This only requires one round of communication between the auditor and the bank and the messages are a constant size, independent of the number of rows. Because of zkLedger's commitment caches, this is very fast.

Map/reduce. An auditor can issue more complex queries that might require the exchange of more data or might require the participants to look at most of the rows in the ledger. Let's consider an auditor which wants to know the mean transaction size for a given bank and asset. An auditor cannot verify a bank's answer by simply totaling the bank's column of commitments and dividing the opened value by the number of rows, because such a computation would have an incorrect denominator. Namely, when the bank is not involved in a transaction, its column in the row will be commitment to 0, and should be discounted. In order to determine the correct denominator, the auditor and the bank run the following protocol:

1. **Filter.** The bank will filter the rows by asset.
2. **Produce new commitments.** For each row, the bank will commit to a single bit b, 1 or 0, depending on if the bank was involved in the transaction or not, and create a proof that the bank has done this recommitment correctly. Crucially, the auditor cannot distinguish between these commitments and so the bank's transactions are not revealed. We call this act of producing new commitments the *map* step. The map step also requires producing proofs that the new values were

correctly computed; in our example, for each transaction, the bank would produce a NIZK proof that $b = 1$ if and only if the transaction value was not equal to 0.

3. **Compute number of non-zero transactions.** The bank computes the homomorphic sum of the new commitments to bits $\vec{b}$ and opens it to reveal how many transactions were non-zero. This is the *reduce* step. This is the correct denominator to compute the mean transaction size. The auditor cannot tell anything about the values in $\vec{b}$ beyond what is revealed by the sum.
4. **Respond to auditor.** The bank then sends the auditor the sum of the values in its column, the vector of bit commitments and corresponding NIZK proofs, the number of its non-zero transactions n, and the sum of the r values in the commitments.
5. **Verification.** The auditor verifies the map step by verifying the commitments were done correctly, and verifies the reduce step and the number of non-zero transactions by confirming that the product of the vector of bit commitments is $g^n h^{\sum_{k=1}^{N} r_k}$.
6. **Compute answer.** The auditor computes the mean from the sum of the bank's column and the number of non-zero transactions.

An auditor could ask a bank for outlier transactions using a similar technique. For each row, the bank will commit to a bit b where $b = 1$ if a transaction's value for that bank is outside a specified range. As when computing the mean, the auditor can verify these commitments were produced correctly and obtain the sum. The bank can then open only the transactions where $b = 1$, and the auditor knows exactly how many transactions should be opened.

More complex auditing queries require multiple *map* and *reduce* computations. For example, here is how an auditor can learn the variance of transaction values $v_1, \ldots, v_N$:

1. **Compute the average transaction value.** Execute the protocol described above to compute the number of non-zero transactions n, and their average value $\bar{v}$.
2. **Apply the squaring map.** For each entry v_i in its row, the bank produces a fresh commitment cm'_i to v_i^2 and sends these commitments to the auditor. The bank also supplies NIZK proofs that the value hidden in each cm'_i is exactly the square of the value v_i committed to on the ledger.
3. **Apply the reduce step.** The auditor computes the product of the commitments cm'_i, and the bank opens up this commitment as $V = v_1^2 + \cdots + v_N^2$ by revealing $R = \sum_{i=1}^{N} r_i$. The auditor confirms that the product of the commitments is equal to $g^V h^R$.

The auditor now computes the variance σ as follows: $\sigma^2 = \frac{1}{n}\sum_{v_i \neq 0}(v_i - \bar{v})^2 = \frac{1}{n}V - \bar{v}^2$.

We note that whereas the square mapping used above corresponds to the second moment (variance), zkLedger can also compute higher statistical moments (e.g. skewness and kurtosis) using similar techniques and using cubing and fourth power mappings, respectively. See Appendix A for a list of measurements zkLedger supports.

zkLedger can support limited information release by using more complex reduce mappings. For example, instead of releasing the sum of values, the bank can produce a commitment to the *rounded* sum of values (e.g. to the first two decimal places), and use range proofs, also implemented in zkLedger, to show that the rounding was done correctly. Revealing just the order of magnitude of the quantity at hand lets the parties balance the granularity of information disclosure.

6 Implementation

To evaluate zkLedger's design, we implemented a prototype of zkLedger in Go. Our prototype uses a modified version of the *btcec* library [2] that contains the parameters and methods to compute with the elliptic curve secp256k1. We use Go's built-in SHA-256 implementation for our cryptographic hash function, and deterministically pick g and h by applying point decompression to the "nothing-up-my-sleeve" strings $\texttt{SHA256}(0)$ and $\texttt{SHA256}(1)$. Our prototype consists of approximately 3,200 lines of code, of which 40% implement cryptographic tools used by zkLedger (zero-knowledge proofs, range proofs, etc).

The implementation of the curve in zkLedger uses Go's `big.Int` type, which we make no effort to compress or serialize in an efficient way. A more optimized implementation could compress curve points. Our range proofs implement the protocol used in Confidential Assets [42]. Our NIZKs are based on Generalized Schnorr Proofs, which are three move interactive protocols; to make them non-interactive we apply the Fiat-Shamir heuristic [26], where we instantiate the random oracle using the SHA-256 hash function. Our prototype implementation does not implement the complex queries described in §5, and thus we do not evaluate them in §7.

7 Evaluation

Our evaluation answers the following questions:

- How expensive is it to store, prove and verify the different proofs in zkLedger? (§7.2)
- How does auditing scale with the size of the ledger? (§7.3)
- How does zkLedger scale with the number of banks? (§7.4)

#	Component	Create	Verify	Size
$2k$	Commitment	0.5 ms	0.5 ms	64 B
$2k$	Consistency	0.7 ms	0.8 ms	224 B
k	Disjunctive	0.9 ms	0.9 ms	288 B
k	Range	4.7 ms	3.5 ms	3936 B

Table 1: Number of each proof component in a transaction for k banks. Size of and time to create and verify the components with 12 cores. The range proof create and verify benefit from the additional cores.

7.1 Experimental setup

Microbenchmarks. We run microbenchmarks on a 12 core Intel machine with i7-X980 3.33 GHz CPUs and 24GB of RAM, running 64-bit Linux 4.4.0 on Ubuntu 16.04.3. Each microbenchmark runs the same code a Bank runs to create and validate transactions.

Distributed experiment. We run the distributed experiments on a set of 12 virtual machines each with 4 cores of Intel Xeon E5-2640 2.5 GHz processors, 24GB of RAM, and the same software setup as above. There is one auditor, one server providing the service of the ledger, and a varying number of banks, one per server. Servers communicate using the `net/rpc` Go package over TCP. All experiments use Go version 1.9.

7.2 Proof overhead in zkLedger

Table 1 shows the time to prove and verify the proofs in a transaction in zkLedger. There are two commitments, two consistency proofs, and one each of the disjunctive and range proofs in a transaction entry. There is a transaction entry per Bank. Table 1 also shows the sizes of the various proofs, in bytes. These sizes are estimated based on the size of the underlying fields in the struct in memory; these proofs could be further compressed. Range proofs dominate the size of the transactions.

The left graph in Figure 3 shows the time it takes to create and verify a transaction varying the number of overall banks, which increases the number of entries per transaction. This indicates that as we increase the number of banks, both transaction creation and verification times per bank increase linearly, but parallelization helps. Proving and validating range proofs dominates transaction creation and verification, but this cost is also highly parallelizable. 12 cores gives a 2.8× speedup when creating a transaction with 20 banks; a bank can create or validate a transaction for up to 20 banks in less than 200ms.

As described in §4.1, zkLedger uses Borromean ring signatures to prove that a value is in a certain range, and supports values up to 2^{40}. Reducing the supported range of values would reduce range proof cost since that cost is linear in the number of bits in the size of the range. There are also newer proof systems, such as Bulletproofs, which might create much smaller range proofs [13]. We plan to evaluate zkLedger with Bulletproofs in future work.

7.3 Cost of auditing ledgers

The left graph in Figure 4 shows that for certain functions, the time to audit is independent of the number of transactions in the ledger. This is because the Auditor and Banks maintain commitment caches, which already have the commitment product necessary to prove to the auditor the sum of the values in its column. The audit function is measuring the Herfindhal-Hirschman Index, so the auditor communicates with each bank.

When the auditor cannot use a commitment cache, perhaps because it was offline, it must process the whole ledger to compute the commitment product. This also applies to more complex auditing like the types described in §5, when the auditor has to verify recommitments for every row in the ledger. These costs are shown in the middle graph in Figure 4. This graph shows how long it takes the auditor to compute the Herfindahl-Hirschman Index on a ledger of varying sizes without using the commitment caches, so the auditor must process every row of the ledger. In these measurements, the auditor does not verify each row. As expected, this time increases linearly with the number of rows. This indicates that maintaining commitment caches is important for real-time auditing. However, even without commitment caches, auditing time is reasonable: 3.5 seconds for 100K transactions. This suggests the complex auditing queries, in which the auditor computes a similar set of operations per row, will also be on the order of many seconds. zkLedger currently only maintains commitment product caches per asset per bank, but could maintain more.

For a fixed size ledger, this audit function costs order the number of banks. The right graph in Figure 4 demonstrates the auditing costs of computing the Herfindahl-Hirschman Index on a ledger of 2000 transactions as we vary the number of banks, both with and without commitment caches. The auditor audits the banks in parallel. Auditing cost for this function grows slightly with the number of banks, since more banks increase the variability in parallel auditing and the auditor must wait for the last bank to respond before computing the final answer.

In these figures, each point is the mean of running the auditing query 20 times, with error bars representing one standard deviation from the mean.

7.4 Scaling with more banks

There are two significant costs that grow with the number of banks in zkLedger: a serial step to create transactions that increases linearly, and verifying transactions which increases quadratically with the number of banks. As

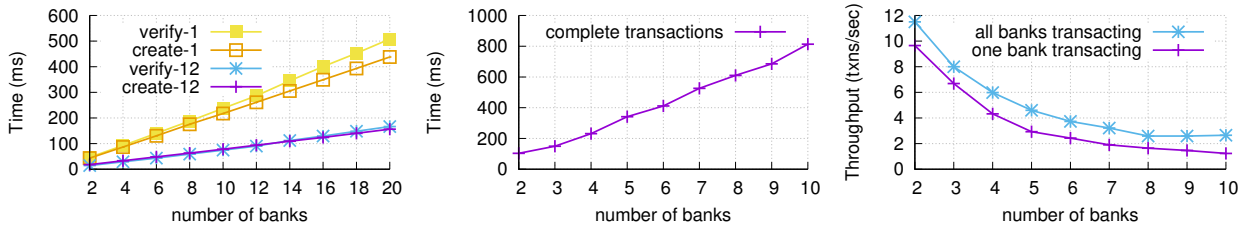

Figure 3: Transaction creation and verification time for one bank (left), varying the number of entries in the transaction. Single-threaded and multi-threaded performance, with 12 threads. Time to fully process a transaction including creation, broadcast to ledger, banks and auditor, and verification by all parties (middle). Throughput (right) varying the number of banks.

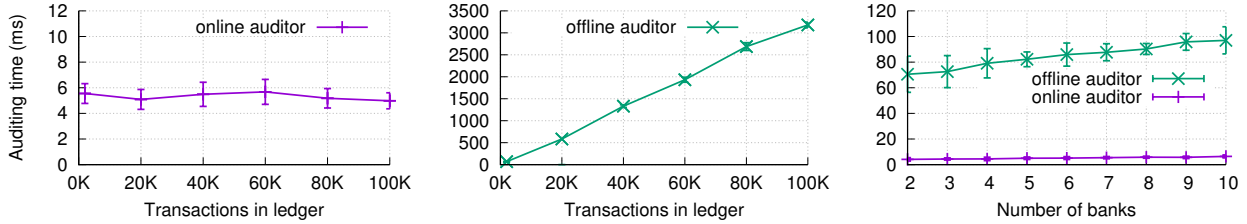

Figure 4: Time to audit ledgers of different sizes (4 banks), and with a varying number of banks (2000 row ledger). Audit time is independent of the size of the ledger (left) thanks to commitment caches maintained by the online auditor. When commitment cache optimization is turned off (middle) the audit time is linear in the size of the ledger. Audit time grows with the number of banks (right) and is much higher without commitment caches.

described in §4.1, a bank needs to use its entry from transaction $n-1$ to create transaction n. So though a bank can use many cores to produce the proofs for a single transaction in parallel, multiple banks cannot produce different transactions in parallel. In zkLedger, banks start creating transaction n before seeing $n-1$ but the bank cannot complete the transaction until $n-1$ is accepted to the ledger and verified, causing an inherent bottleneck.

The second major cost is around verification. Every bank must verify every transaction, so the more banks, the larger each transaction and thus the more work that needs to be done by each bank. The middle graph in Figure 3 measures the time it takes one bank to create and all participants in zkLedger to completely process a transaction. One bank creates a transaction and sends it to the ledger, which then broadcasts the transaction to all banks and an online auditor. The auditor and every bank verify the transaction. As we increase the number of banks, work increases quadratically; however, banks can verify transactions in parallel so the time to process transactions only increases linearly. The right graph in Figure 3 shows that as we surmised, zkLedger's throughput worsens with more banks. The one bank transacting line in this graph is the same data as the middle graph.

Since range proofs dominate the costs of transaction creation and verification, we are optimistic that a faster range proof implementation will directly improve performance. zkLedger's current performance is comparable to Solidus, a privacy-preserving distributed ledger which achieves 3-4 transactions per second with online validation but, unlike zkLedger, does not support auditing.

8 Future work

zkLedger focuses on providing provably correct auditing over private transaction data, but zkLedger does not have a way to recover if the distributed ledger is corrupted. In this case, the parties maintaining the ledger would have to come together to recreate historical transactions. zkLedger also does not provide recourse if a bank commits an unintended transaction to the ledger. A future version of zkLedger might provide rectifying transactions or participant agreed-upon rollback.

9 Conclusion

zkLedger is the first distributed ledger system to provide strong transaction privacy, public verifiability, and complete, provably correct auditing. zkLedger supports a rich set of auditing queries which are useful to measure the financial health of a market. We developed a design using non-interactive zero-knowledge proofs to prove transactions maintain financial invariants and to support auditing. Our evaluation shows that zkLedger has reasonable performance for transaction settlement and auditing.

10 Acknowledgements

We thank Alexander Chernyakhovsky, Thaddeus Dryja, David Lazar, Ronald L. Rivest, C.J. Williams, and our shepherd and reviewers for helpful comments. The research leading to these results has received funding from: the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; and the Ethics and Governance of Artificial Intelligence Fund.

References

[1] Bank secrecy act of 1970, October 1970. 12 U.S.C. 103.

[2] Package btcec implements support for the elliptic curves needed for Bitcoin., July 2017. https://godoc.org/github.com/btcsuite/btcd/btcec.

[3] ABBE, E. A., KHANDANI, A. E., AND LO, A. W. Privacy-preserving methods for sharing financial risk exposures. *The American Economic Review 102*, 3 (2012), 65–70.

[4] ARASU, A., BLANAS, S., EGURO, K., KAUSHIK, R., KOSSMANN, D., RAMAMURTHY, R., AND VENKATESAN, R. Orthogonal security with cipherbase. In *CIDR* (2013).

[5] BAJAJ, S., AND SION, R. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering 26*, 3 (2014), 752–765.

[6] BAUMANN, A., PEINADO, M., AND HUNT, G. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS) 33*, 3 (2015), 8.

[7] BEN-SASSON, E., CHIESA, A., GENKIN, D., TROMER, E., AND VIRZA, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 90–108.

[8] BEN-SASSON, E., CHIESA, A., GREEN, M., TROMER, E., AND VIRZA, M. Secure sampling of public parameters for succinct zero knowledge proofs. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy* (2015), SP '15, pp. 287–304.

[9] BLUM, M., FELDMAN, P., AND MICALI, S. Non-interactive zero-knowledge and its applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988), STOC '88, pp. 103–112.

[10] BOGDANOV, D., TALVISTE, R., AND WILLEMSON, J. Deploying secure multi-party computation for financial data analysis. In *International Conference on Financial Cryptography and Data Security* (2012), Springer, pp. 57–64.

[11] BOWE, S., GABIZON, A., AND GREEN, M. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017.

[12] BOWE, S., GABIZON, A., AND MIERS, I. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017.

[13] BÜNZ, B., BOOTLE, J., BONEH, D., POELSTRA, A., AND MAXWELL, G. Bulletproofs: Short proofs for Confidential Transactions and more. In *Security and Privacy (SP), 2018 IEEE Symposium on* (2018), IEEE.

[14] BURKHART, M., STRASSER, M., MANY, D., AND DIMITROPOULOS, X. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network 1*, 101101 (2010).

[15] CASTRO, M., AND LISKOV, B. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.

[16] CECCHETTI, E., ZHANG, F., JI, Y., KOSBA, A., JUELS, A., AND SHI, E. Solidus: Confidential distributed ledger transactions via pvorm.

[17] CHAIN, I. Confidential assets. https://blog.chain.com/hidden-in-plain-sight-transacting-privately-on-a-blockchain-835ab75c01cb.

[18] CORRIGAN-GIBBS, H., AND BONEH, D. Prio: Private, robust, and scalable computation of aggregate statistics. *arXiv preprint arXiv:1703.06255* (2017).

[19] COUNCIL, F. R. Developments in audit 2016/2017 full report, 2017. http://www.frc.org.uk/getattachment/915c15a4-dbc7-4223-b8ae-cad53dbcca17/Developments-in-Audit-2016-17-Full-report.pdf.

[20] CRAMER, R., DAMGÅRD, I., AND SCHOENMAKERS, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings* (1994), pp. 174–187.

[21] DAGHER, G. G., BÜNZ, B., BONNEAU, J., CLARK, J., AND BONEH, D. Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, CO, 2015), ACM, pp. 720–731.

[22] Corda, 2017. https://github.com/corda/corda.

[23] Digital asset holdings, 2017. http://digitalasset.com.

[24] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union L119* (May 2016), 1–88.

[25] FELDMAN, A. J., ZELLER, W. P., FREEDMAN, M. J., AND FELTEN, E. W. Sporc: Group collaboration using untrusted cloud resources. In *OSDI* (2010), vol. 10, pp. 337–350.

[26] FIAT, A., AND SHAMIR, A. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of the 6th Annual International Cryptology Conference* (1987), CRYPTO '87, pp. 186–194.

[27] GARMAN, C., GREEN, M., AND MIERS, I. Accountable privacy for decentralized anonymous payments. Cryptology ePrint Archive, Report 2016/061, 2016. http://eprint.iacr.org/2016/061.

[28] GREENBERG, A. Fbi says it's seized $28.5 million in bitcoins from ross ulbricht, alleged owner of silk road. *Forbes 25* (2013).

[29] HERFINDAHL, O. C. *Concentration in the steel industry*. PhD thesis, Columbia University New York, 1950.

[30] LAMPORT, L., ET AL. Paxos made simple. *ACM Sigact News 32*, 4 (2001), 18–25.

[31] LI, J., KROHN, M. N., MAZIERES, D., AND SHASHA, D. E. Secure untrusted data repository (sundr). In *OSDI* (2004), vol. 4, pp. 9–9.

[32] MAHAJAN, P., SETTY, S., LEE, S., CLEMENT, A., ALVISI, L., DAHLIN, M., AND WALFISH, M. Depot: Cloud storage with minimal trust. *ACM Transactions on Computer Systems (TOCS) 29*, 4 (2011), 12.

[33] MAURER, U. Unifying zero-knowledge proofs of knowledge. *Proceedings of the 2nd International Conference on Cryptology in Africa* (2009), 272–286.

[34] MAXWELL, G. Confidential transactions. `https://people.xiph.org/~greg/confidential_values.txt` *(Accessed 8/2017)* (2015).

[35] MAXWELL, G., AND POELSTRA, A. Borromean ring signatures. `https://raw.githubusercontent.com/Blockstream/borromean_paper/master/borromean_draft_0.01_34241bb.pdf` *(Accessed 6/2017)* (2015).

[36] MEIKLEJOHN, S., POMAROLE, M., JORDAN, G., LEVCHENKO, K., MCCOY, D., VOELKER, G. M., AND SAVAGE, S. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference* (2013), ACM, pp. 127–140.

[37] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system, 2008.

[38] OBER, M., KATZENBEISSER, S., AND HAMACHER, K. Structure and anonymity of the bitcoin transaction graph. *Future internet 5*, 2 (2013), 237–250.

[39] ONGARO, D., AND OUSTERHOUT, J. K. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference* (2014), pp. 305–319.

[40] PAPADIMITRIOU, A., BHAGWAN, R., CHANDRAN, N., RAMJEE, R., HAEBERLEN, A., SINGH, H., MODI, A., AND BADRINARAYANAN, S. Big data analytics over encrypted datasets with seabed. In *OSDI* (2016), pp. 587–602.

[41] PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference* (1992), CRYPTO '91, pp. 129–140.

[42] POELSTRA, A., BACK, A., FRIEDENBACH, M., MAXWELL, G., AND WUILLE, P. Confidential assets, 2017. 4th Workshop on Bitcoin and Blockchain Research.

[43] POPA, R. A., REDFIELD, C., ZELDOVICH, N., AND BALAKRISHNAN, H. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 85–100.

[44] POPA, R. A., STARK, E., HELFER, J., VALDEZ, S., ZELDOVICH, N., KAASHOEK, M. F., AND BALAKRISHNAN, H. Building web applications on top of encrypted data using mylar. In *NSDI* (2014), pp. 157–172.

[45] REID, F., AND HARRIGAN, M. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[46] RON, D., AND SHAMIR, A. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security* (2013), Springer, pp. 6–24.

[47] SASSON, E. B., CHIESA, A., GARMAN, C., GREEN, M., MIERS, I., TROMER, E., AND VIRZA, M. Zerocash: Decentralized anonymous payments from Bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on* (2014), IEEE, pp. 459–474.

[48] SCHNORR, C.-P. Efficient signature generation by smart cards. *Journal of cryptology 4*, 3 (1991), 161–174.

[49] SCHUSTER, F., COSTA, M., FOURNET, C., GKANTSIDIS, C., PEINADO, M., MAINAR-RUIZ, G., AND RUSSINOVICH, M. Vc3: Trustworthy data analytics in the cloud using sgx. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 38–54.

[50] TU, S., KAASHOEK, M. F., MADDEN, S., AND ZELDOVICH, N. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment* (2013), vol. 6, VLDB Endowment, pp. 289–300.

[51] Zcash, 2017. `http://z.cash`.

[52] ZHENG, W., DAVE, A., BEEKMAN, J. G., POPA, R. A., GONZALEZ, J. E., AND STOICA, I. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI* (2017), pp. 283–298.

A Auditing Queries

Figure 5 is a list of the types of measurements zkLedger supports, including the estimated running time and the data beyond the measurement that is leaked. For example, as described in §5, computing transaction size variance requires leaking the mean transaction size and number of transactions per bank.

B Zero-knowledge proofs and privacy guarantees

To build our zero-knowledge protocols we rely on the following general result of Maurer (Theorem 3, [33]):

Theorem B.1. *Let $(H_1, \star)$ and $(H_2, \otimes)$ be two (not-necessarily commutative) groups and $f : H_1 \to H_2$ be a group homomorphism: $f(x \star y) = f(x) \otimes f(y)$. Let $\ell \in \mathbb{Z}$, $u \in H_1$, $\mathscr{C} \subset \mathbb{Z}$ be such that:*

1. $\gcd(c_1 - c_2, \ell) = 1$ *for all* $c_1, c_2 \in \mathscr{C}$ *(with* $c_1 \neq c_2$*), and*
2. $f(u) = z^\ell$.

There exists a 2-extractable Σ-protocol for language $\mathscr{L} := \{z : \exists w \text{ s.t. } z = f(w)\}$. Moreover, a protocol con-

Measurement	Time	Additional information leaked
Sum total of asset(s) per bank	$O(1)$	none
Outlier transactions per bank	$O(n)$	none
Concentration	$O(k)$	Sum totals per bank
Ratio holdings	$O(k)$	Sum totals per bank, number of transactions per bank
Mean transaction size per bank	$O(kn)$	Number of transactions per bank
Variance, skew, kurtosis	$O(kn)$	Mean per bank, number of transactions per bank
Real-time price averages	$O(kn)$	Number of transactions and average per bank over time period t

Figure 5: Types of supported auditing queries, their running time to audit based on the number of banks k and the number of rows in the ledger n, and a description of what information is leaked to the auditor.

sisting of s rounds is proof-of-knowledge if $1/|\mathscr{C}|^s$ *is negligible, and zero-knowledge if* $|\mathscr{C}|$ *is polynomially bounded.*

Using Theorem B.1 we can now unify the treatment of most of the zero-knowledge proofs used in our system. For example, the consistency proofs π^{C} rely on the following result:

Theorem B.2. *Let* $\mathbb{G}$ *be an order-r cyclic group and* g, h, pk *be any three elements of* $\mathbb{G}$. *There exists a* 2*-extractable* Σ*-protocol for language* $\mathscr{L}_{\mathsf{aux}} := \{(\mathsf{cm}, \mathsf{Token}) : \exists v, r \text{ s.t. } \mathsf{cm} = g^v h^r \wedge \mathsf{Token} = \mathsf{pk}^r\}$.

Proof. Consider $H_1 = \mathbb{Z}_r \times \mathbb{Z}_r$, defining the group operation to be component-wise addition, and let $H_2 = \mathbb{G} \times \mathbb{G}$, similarly defining group operation to be component-wise. Then $f(x,y) := (g^x h^y, \mathsf{pk}^y)$ is a group homomorphism between H_1 and H_2. Indeed, $f(x_1+x_2, y_1+y_2) = (g^{x_1+x_2} h^{y_1+y_2}, \mathsf{pk}^{y_1+y_2}) = f(x_1,y_1) \otimes f(x_2,y_2)$. Furthermore, setting $\ell = r$ and $u = (0,0)$ we have that for all $z \in H_2$ the following holds: $z^\ell = (1,1) = f(u)$. Therefore, we can apply Theorem B.1 and conclude that $\mathscr{L}_{\mathsf{aux}}$ has a 2-extractable Σ-protocol. □

To summarize, the three proofs in zkLedger (see Section 4.3) that relate commitments $\mathsf{cm}_i := g^{v_i} h^{r_i}$ and audit tokens $\mathsf{Token}_i := (\mathsf{pk}_i)^{r_i}$ have the following form:

- **Proof of Assets (π^{A}).** This proof consists of a new commitment $\mathsf{cm'}_i$, together with an audit token Token'_i, and a zero-knowledge proof asserting that either $\mathsf{cm'}_i$ is a re-commitment of the value in cm_i or a recommitment to the sum of the values in $\prod_{j=0}^{m} \mathsf{cm}_j$. To create this proof zkLedger relies on Theorem B.1 for constituent proofs; as these are Sigma-protocols we apply the standard OR-composition [20] to get the final disjunctive zero-knowledge proof. To prove that the commited value is in the range we use the range proofs in Confidential Assets [42]. We are investigating more recent proof systems (e.g. Bulletproofs [13]) to further reduce the proof size.
- **Proof of Balance (π^{B}).** In our implementation this proof is an empty string: the prover simply chooses the commitment randomness subject to condition $\sum r_i = 0$. With such a choice the auditor homomorphically adds the commitments and checks that this addition results in the neutral element of the group $\prod \mathsf{cm}_i = g^{\sum v_i} h^{\sum r_i} = g^0 h^0 = 1$.
- **Proof of Consistency (π^{C}).** We use two proofs derived from Theorem B.2 to assert that the randomness used in cm_i and Token_i are the same, and the randomness used in $\mathsf{cm'}_i$ and Token'_i are the same.

C Privacy in the combined system

Pedersen commitments provide information-theoretic privacy. In zkLedger Pedersen commitments are published together with authentication tokens and zero-knowledge proofs. We note that zero-knowledge proofs indeed don't spoil the information-theoretic privacy of committed values: the output of the zero-knowledge proof simulator is identical to the output produced by parties in the system. However, when combining Pedersen commitments and authentication tokens, the privacy guarantees become computational as we now explain.

The commitment, audit token, and public key triple $(\mathsf{cm}, \mathsf{Token})$ is of the form $(g^v h^r, \mathsf{pk}^r, \mathsf{pk}) = (g^v h^r, h^{\mathsf{sk}\cdot r}, h^{\mathsf{sk}})$, and these three values uniquely determine the v. That is, if an adversary could break the discrete logarithm problem, it could solve for sk, use that and value of Token to infer r, and finally recover v. That said, under the Decisional Diffie-Hellman (DDH) assumption, no information is leaked. Furthermore, the DDH assumption is widely assumed to hold in zkLedger's elliptic curve group.

Recall, that DDH holds if no polynomially-bounded adversary can distinguish between tuples of the form (h, h^a, h^b, h^{ab}) and (h, h^a, h^b, h^c) for a randomly chosen generator h and exponents a, b, c. Assume that a stateful adversary $\mathscr{A}_{\mathsf{zkL}}$, when given input (g, h, pk) is able to produce two values v_1 and v_2 such that it can distinguish commitments (and associated audit tokens) to v_1 from commitments (and audit tokens) to v_2, i.e. the adversary is able to distinguish the distributions $(g^{v_1} h^r, h^{\mathsf{sk}\cdot r}, h^{\mathsf{sk}})$

and $(g^{v_2}h^r, h^{\mathsf{sk}\cdot r}, h^{\mathsf{sk}})$. We now show how to use $\mathscr{A}_{\mathsf{zkL}}$ to construct an adversary $\mathscr{A}_{\mathsf{DDH}}$ breaking the DDH assumptions.

After receiving its challenge (h,x,y,z), where (x,y,z) is distributed either as (h^a,h^b,h^{ab}) or as (h^a,h^b,h^c), the adversary $\mathscr{A}_{\mathsf{DDH}}$ proceeds as follows. It samples a random generator g and calls $\mathscr{A}_{\mathsf{zkL}}$ on input (g,h,x), x now serving the role of the bank's public key. When $\mathscr{A}_{\mathsf{zkL}}$ returns two values v_1 and v_2, the DDH adversary $\mathscr{A}_{\mathsf{DDH}}$ picks a random $k \in \{1,2\}$ and prepares $\mathsf{cm}_k = g^{v_k}y$, $\mathsf{Token} = z$ and sends $(\mathsf{cm}_k, \mathsf{Token})$ to $\mathscr{A}_{\mathsf{zkL}}$. Finally, if $\mathscr{A}_{\mathsf{zkL}}$'s guess for k is correct, $\mathscr{A}_{\mathsf{DDH}}$ responds that the DDH challenge was of the form (h,h^a,h^b,h^{ab}) (i.e. a DDH quadruple), otherwise it responds that the DDH challenge was of the form (h,h^a,h^b,h^c) (i.e. a random quadruple).

Note that when $\mathscr{A}_{\mathsf{DDH}}$'s challenge is a DDH quadruple, the zkLedger adversary $\mathscr{A}_{\mathsf{zkL}}$ is run on a distribution it expects. In particular, all of its inputs are correctly formed with respect to $\mathsf{sk} = a$ and $r = b$. Whereas, when $\mathscr{A}_{\mathsf{DDH}}$'s challenge is a random quadruple, the inputs to $\mathscr{A}_{\mathsf{zkL}}$ have *information-theoretically* no information about the committed value: indeed, $\mathsf{Token} = h^c$ is unrelated to $\mathsf{cm} = g^v h^b$. Therefore, if the zkLedger adversary $\mathscr{A}_{\mathsf{zkL}}$ wins the commitment hiding game with non-negligible advantage, so does $\mathscr{A}_{\mathsf{DDH}}$ in the DDH game. Note that the proof extends to the multiple entry case by a standard hybrid argument.