MiniLedger: Compact-sized Anonymous and Auditable Distributed Payments

Panagiotis Chatzigiannis and Foteini Baldimtsi*

pchatzig@gmu.edu and foteini@gmu.edu
George Mason University

Abstract. While privacy preserving distributed payment schemes manage to drastically improve user privacy, they come at the cost of generating new regulatory concerns: in a private ledger the transactions cannot be subject to any level of auditing, and thus are not compatible with tracing illegal behaviors.
In this work we present MiniLedger, a distributed payment system which not only guarantees the privacy of transactions, but also offers built-in functionalities for various types of audits by any external authority. MiniLedger is the *first* private and auditable payment system with storage costs independent of the number of transactions. To achieve such a storage improvement, we introduce pruning functionalities for the transaction history while maintaining integrity and auditing. We provide formal security definitions and a number of extensions for various auditing levels. Our evaluation results show that MiniLedger is practical in terms of storage requiring as low as 70KB per participant for 128 bits of security, and depending on the implementation choices, can prune 1 million transactions in less than a second.

1 Introduction

One of the main issues with distributed ledger-based (or else blockchain) payment schemes (e.g. Bitcoin) is the lack of privacy. All transaction information - including transacting parties' public keys and associated values - are permanently recorded on the public blockchain/ledger, and using side-channel information these keys can be clustered and eventually linked to real identities [10,47]. Even if we look at applications of permissioned blockchain types [8,5,29,6] for electronic payments, with transaction data shared between a controlled set of financial institutions in order to reduce verification and reconciliation costs, the privacy problem persists. The financial data of a customer at institution A should not be shared with institution B as also dictated by recent regulation such as the European General Data Protection Regulation [3].

A number of solutions have been suggested to solve the privacy issues of distributed ledgers by hiding both the transaction graph and its associated assets

* The authors have been supported by the National Science Foundation (NSF) under Grant 1717067, an IBM Faculty Award and Facebook Research Award.

and amounts from public view, thereby offering strong privacy guarantees equivalent to cash [12,21,31,57], however, while privacy is a fundamental right, the need for *auditing mechanisms* is required to ensure compliance with laws and regulation [2,40] as done in traditional payment systems via auditing companies (i.e. Deloitte, Pricewaterhouse Coopers, Ernst and Young, and KPMG) [4]. Constructing payment schemes that satisfy both privacy and auditability *at the same time*, is a rather challenging problem since these properties are often conflicting. The challenge becomes even harder when one takes *efficiency and scalability* into account. In particular, one of the most common approaches to solve the scalability issue, that of pruning old/unneeded transactions from the ledger, directly hurts auditability, as an auditor cannot possibly audit data that no longer exists in the ledger.

A limited type of accountability already existed in traditional, "Chaumian" private electronic cash [15] where when a coin was double-spent, the identity of the owner could be revealed. The first attempt to build accountability in private, distributed payments was given in [36] which extends Zcash [12] by adding auxiliary information to coins, used exclusively by a designated trusted authority for accountability purposes (i.e., regulatory closure, spending limits and tracing tainted coins). While this is a nice step towards building accountability, it is very limited in the types of allowed auditing and gives too much power to the trusted authority. More recently, a notable auditability approach was given in the setting of distributed private payments for an authorized set of participants: zkLedger [51] offered an interesting solution for auditing by consent which can be executed by any third party and at any point of the protocol for various audit functions. Unfortunately, this solution is not practical due to very high storage costs (linear in the number participants *and* to the total number of performed transactions in the system), while it also suffers from the use of strong assumptions such as out-of-band communication between participants.

Our contributions. We present MiniLedger: the *first space efficient*, distributed private payment system that allows an authorized set of participants to transact with each other, while also allowing for a wide set of auditing by consent operations by *any* third party auditor. We provide formal, game based definitions and a construction that relies upon a number of cryptographic primitives: a consensus protocol, semi-homomorphic encryption, compact set representation techniques (cryptographic accumulators) and non-interactive zero-knowledge proofs (NIZKs).

At a high level, MiniLedger consists of n Banks transacting with each other through a common transaction history, or else a ledger L which is maintained by a consensus mechanism (orthogonal to our work). The ledger is modeled as a two-dimensional table with n columns, one for each participating Bank, and rows representing transactions. Whenever Bank B_j wishes to send funds of value v to another B_k, it creates a n-sized vector containing (semi)homomorphic encryptions and NIZK proofs which is appended in L. B_j encrypts the value that is sent to each participating Bank in the system using each receiving Bank's public key, i.e. the encrypted values would be v for B_k, $-v$ for B_j and 0 for

any other Bank. These encryptions provide privacy in MiniLedger since they hide values as well as the sender and recipient of each transaction, while still allowing all participating Banks to decrypt the value that corresponds to them and to compute their total assets at any point. This overcomes the need for any out-of-band communication between Banks which created security issues in previous works (ref. Section 4.2). Finally, the included ZK proofs guarantee that transactions are valid without revealing any information.

MiniLedger provides auditability *by consent*. Any third party auditor with access to L can ask audit queries to a Bank and verify the responses based on the public information on L. The simplest audit is to learn the value of a cell in L, i.e. the exact amount of funds a Bank received/sent at any point. This basic audit can be used to derive more complex audit types as we discuss in Section 5.2, such as transaction history, account balance, spending limit etc., without disclosing more information to the auditor than needed.

Space Efficiency. The main innovation of MiniLedger lies in the maintenance and storage of L. In previous auditable schemes (such as zkLedger [51]) the *full* L needs to be stored at *any time* and by *all* participants. The challenge in MiniLedger design was compacting the ledger while maintaining security and a wide set of auditing functionalities. As noted above, completely erasing transaction information would make auditing impossible (since an auditor cannot possibly audit something that no longer exists). MiniLedger employs a smart type of transaction pruning: participating Banks can prune their own transaction history by computing a provable, *compact representation* of their previously posted history and broadcast the resulting digest to the consensus layer. Once consensus participants agree to a pruning operation (i.e. verify the digest as a valid representation of the Bank's history), that history can be erased from L and thus by all system participants (except the pruning Bank itself which always need to store its own transaction history locally). Auditing is still possible since a compact digest of transaction information is always stored in L and the Bank under audit can prove that the revealed values correctly correspond to the digest. As a result, the size of L in MiniLedger can be nearly constant (i.e. independent of the number of transactions that ever happened).

Our compact transaction history representation can lead to multiple additional benefits (besides obviously reduced storage requirements). First, a compact L makes addition of new system participants (i.e. Banks) much more efficient (typically, new parties need to download the whole L requiring large bandwidth and waiting time). Then, although the structure of L does not allow for a very large number of participating Banks n (as the computation cost of a single transaction is linear in n), the compactness of L allows augmenting MiniLedger with more fine-grained types of auditing and enabling audits in a client level (instead of a Bank level). We present MiniLedger+, an extension that serves a much larger user base in Section 5.1.

Finally, we implement a prototype of the transaction layer of MiniLedger and evaluate its performance in terms of transaction costs, pruning costs and size of L which we estimate to be as low as 70KB of storage for each Bank. We show

that transaction computation cost, for a system with 100 Banks, takes about 4 sec, while transaction auditing is less than 5 ms, independent of the number of Banks. Transaction computation costs increase linearly to the number of Banks (as in zkLedger) but by optimizing the underlying ZK proofs we achieve some small constant improvement. Although the linear transaction computation cost might still seem high, we note that using our MiniLedger+ extension, a small number of Banks can serve a very large user base. We perform experiments on two different types of pruning instantiations, one using Merkle trees [48] and one using batch RSA accumulators [13]. Both result in pruning measurements that are independent of the number of participating Banks. Our experiments show that we can prune 1 million transactions in less than a second using Merkle trees and in about 2 hours using the RSA accumulator, and can perform audits in milliseconds in the same transaction set. We also show that we can audit multiple transactions at a time more efficiently with the RSA accumulator, and can create audit openings for all transactions in less than a millisecond with a single exponentiation, assuming pre-computation of the necessary witness. As we show in Section 6, the above trade-offs between the two instantiations suggest that the eventual choice is up to the deployment use-case of MiniLedger.

Related Work. We present an non-exhaustive overview of related works.
Anonymous distributed payment systems. Zcash [12], and its fully-developed digital currency, is a permissionless protocol hiding both transacting parties and transaction amounts using zero knowledge proofs. Other notable systems are CryptoNote and the Monero cryptocurrency [57], based on decoy transaction inputs and ring signatures to provide privacy of transactions within small anonymity sets, and Quisquis [31] which provides similar anonymity level to Monero but allows for a more compact sized ledger. Zether [16] is a smart contract based payment system which only hides transaction amounts. Mimblewimble [34] uses Confidential Transactions [45] to hide transaction values in homomorphic commitments, and prunes intermediate values from the blockchain after being spent (which might be insecure in other UTXO systems such as Bitcoin), improving its scalability. In the permissioned setting, Solidus [21] allows for confidential transactions in public ledgers, employing Oblivious RAM techniques to hide access patterns in publicly verifiable encrypted Bank memory blocks. This approach enables users to transact in the system anonymously using Banks as intermediaries.
Adding auditability/accountability. A number of Zcash extensions [36,41,49] proposed the addition of auxiliary information to coins to be used exclusively by a designated, trusted authority for accountability purposes. While this allows for a number of accountability functionalities, it suffers from centralization of auditing power. Additionally, all such works inherit the underlying limitations of Zcash such as the need for trusted setup and strong computational assumptions. Traceable Monero [43] attempts to add accountability features on top of Monero. In a similar idea to Zerocash, a designated "tracing" authority can link anonymous transactions with the same spending party and learn the origin or destination of a transaction. The tracing authority's role can again be distributed among several

	Record	Anon.	Audit	Perm	Prune
Zcash [12,36]	UTXO	✔	$✔^T$	O	×
Monero [57,43]	TXO	$✔^S$	$✔^K$	O	×
Quisquis [31]	Hybrid	$✔^S$	×	O	✔
MW [34]	UTXO	×	×	O	✔
Solidus [21]	Accnt	$✔^S$	×	C	×
zkLedger [51]	Accnt	✔	✔	C	×
PGC [25]	Accnt	×	✔	O	×
Zether [16]	Accnt	option	×	O	×
MiniLedger	Accnt	✔	✔	C	✔

Table 1: Confidential payment schemes comparison. By $✔^S$ we denote set anonymity, $✔^T$ auditing through a TP and $✔^K$ through "view keys" (which reveal all private information of an account). By O: permissionless and C: permissioned we refer to the set of parties that participate in the payment scheme and not the underlying consensus.

authorities to prevent single point of failure and distribute trust. PRCash [59] aims to achieve accountability for transaction volume over time. A regulation authority (can be distributed using threshold encryption) issues anonymous credentials to the system's transacting users. If transaction volume in a period exceeds a spending limit, the user can voluntarily deanonymize himself to the authority to continue transacting. PRCash however only focuses on this specific audit type. zkLedger presented a unique architecture for implementing various interactive audit types in a permissioned setting, but its linear-growing storage requirements in terms of transactions make it unpractical for real deployment. Additionally, it assumes transaction values are communicated out-of-band, creating an attack vector that could prevent participants from answering audits. Fundamentally, the requirement of communicating values out-of-band defeats the whole purpose of its construction. We discuss details of these shortcomings in Appendix C.6. We also note an extension to zkLedger using private swaps [22] for supporting asynchronous submission of transactions, which however is orthogonal to our work.

Finally, some works attempt to provide auditability and accountability in an organization rather than in an account level. For instance, [28,30] propose accountability mechanisms for cryptocurrency exchanges, enabling them to prove their solvency based on Merkle Trees and range proofs. A standard has been recently proposed for such functionalities in [23]. Also [33,39] proposed accountability solutions for general public records using public ledgers or blockchains.

In Table 1 we summarize properties of private payment schemes and refer the reader to [24] for a systematization of knowledge on auditable and accountable distributed payment systems.

Prunable and stateless blockchains. Given the append-only immutability property for most ledgers, the concern for ever-growing storage requirements in blockchains was stated even in the original Bitcoin whitepaper [50], which considered *pruning* old transaction information without affecting the core system's properties. Ethereum [58], being an account-based system, supports explicit support of "old state" pruning as a default option, and defers to "archival" nodes for any history queries. Coda (Mina) [14] is a prominent example of a stateless (succinct) blockchain, which only needs to store the most recent state with

recursive verifiability using SNARKs. Accumulators and vector commitments have also been proposed to maintain a stateless blockchain [13,26]. All such approaches however might negatively impact auditability and are therefore not directly applicable in our setting.

2 Preliminaries

We first define the notation we will be using throughout this work.By λ we denote the security parameter and by $z \leftarrow \mathcal{Z}$ the uniformly at random selection of an element z from space $\mathcal{Z}$. A probabilistic polynomial-time (PPT) algorithm B with input a and output b is written as $b \leftarrow B(a)$. By $:=$ we denote deterministic computation and by $a \rightarrow b$ we denote assignment of value a to value b. We denote a protocol between two parties A and B with inputs x and y respectively as $\{\mathsf{A}(x) \leftrightarrow \mathsf{B}(y)\}$. By $(\mathsf{pk}, \mathsf{sk})$ we denote a public-private key pair and by $[x_i]_{i=1}^{y}$ a list of elements $(x_1, x_2, ..., x_y)$. By $x \parallel y$ we denote concatenation of bit strings x and y. We denote a matrix M with m rows and n columns as M_{mn} and a $i-$th row and j-th column cell in the matrix as (i, j).

We now provide a brief description of all the cryptographic building blocks used in the following sections:

Digital Signatures. We consider an existentially unforgeable under an adaptive chosen-message attack digital signature scheme [38] $(\mathsf{SignGen}, \mathsf{Sign}, \mathsf{SVrfy})$.

ElGamal Encryption Variant. MiniLedger uses a variant of ElGamal encryption (called "twisted ElGamal" (TEG) [25]). Compared to standard ElGamal, it requires an additional group generator (denoted by h below) in the public parameters pp, which makes it possible to homomorphically add ciphertexts c_2 and c_2' generated for *different* public keys pk and pk' and intentionally leak information on the relation of encrypted messages m and m' as we discuss below. This variant only works when message space is small due to the need for a lookup table on decryption. TEG is secure against chosen plaintext attacks and works as follows:

- $\mathsf{pp} \leftarrow \mathsf{SetupTEG}(1^\lambda)$: Outputs $\mathsf{pp} = (\mathbb{G}, g, h, p)$ where g, h are generators of cyclic group $\mathbb{G}$ of prime order p.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenTEG}(\mathsf{pp})$: Outputs $\mathsf{sk} \leftarrow \mathbb{Z}_p$, $\mathsf{pk} = h^{\mathsf{sk}}$.
- $(c_1, c_2) \leftarrow \mathsf{EncTEG}(\mathsf{pk}, m)$: Sample $r \leftarrow \mathbb{Z}_p$, compute $c_1 = \mathsf{pk}^r, c_2 = g^m h^r$ and output $C = (c_1, c_2)$
- $m \leftarrow \mathsf{DecTEG}(\mathsf{sk}, (c_1, c_2))$: Compute $g^m = c_2 / c_1^{(1/\mathsf{sk})}$ and recover m from a look-up table (assuming that the message space is relatively small).

TEG encryption is additively homomorphic:

$$\mathsf{EncTEG}(\mathsf{pk}, m_1)\mathsf{EncTEG}(\mathsf{pk}, m_2) = \mathsf{EncTEG}(\mathsf{pk}, m_1 + m_2).$$

Also if $(c_1, c_2) \leftarrow \mathsf{EncTEG}(\mathsf{pk}, m)$ and $(c_1', c_2') \leftarrow \mathsf{EncTEG}(\mathsf{pk}', m')$, then $c_2 c_2'$ contains an encryption of $m + m'$. This implies if $c_2 c_2' = 1$, then any external observer can deduce that $m = -m'$ (for properly chosen r, r').

Commitment schemes. A non-interactive commitment takes as input public parameters pp, message m and randomness r and outputs value $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{pp}, m, r)$ such that, on one hand, reveals no information about the message (*hiding* property) but, on the other hand, it is hard to find (m', r') such that $\mathsf{Com}(\mathsf{pp}, m, r) = \mathsf{Com}(\mathsf{pp}, m', r')$, when $m' \neq m$ (*binding* property). We use Pedersen commitments [53] which are additively homomorphic and allow efficient zero-knowledge proofs . Note that ciphertext c_2 from the above twisted ElGamal variant is essentially a Pedersen commitment, which is perfectly hiding and computationally binding[1]. A Pedersen commitment is constructed as follows:

- $\mathsf{pp} \leftarrow \mathsf{ComGen}(1^\lambda)$ Outputs $\mathsf{pp} = (\mathbb{G}, g, h, p)$ where g, h are generators of cyclic group $\mathbb{G}$ of prime order p.
- $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{pp}, m, r)$ On pp, a message $m \in [1...p]$ and randomness $r \in [1...p]$, outputs a commitment $\mathsf{cm} = g^m h^r$.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{cm}, m, r)$ A verifier given a commitment cm and an opening (m, r) returns a verification bit b.

Zero-knowledge proofs. A zero-knowledge (ZK) proof is a two-party protocol between a prover P, holding some private data (or else *witness*) w for a public instance x, and a verifier V. The goal of P is to convince V that some property of w is true i.e. $R(x, w) = 1$, for an NP-relation R, without V learning anything more. The types of ZK proofs used in MiniLedger are:

1. ZK proof on the opening of a commitment: Using Camenisch-Stadler notation [18] (used throughout the paper): $ZKP : \{(w, r) : X = g^w h^r \bmod n\}(X, g, h, n)$ where (X, g, h, n) are the public statements given as common input to both parties, and (w, r) is the secret witness.
2. ZK proof of knowledge of discrete log: $ZKP : \{(x) : X = g^x \bmod n\}(X, g, n)$.
3. ZK proof of equality of discrete logs: $ZKP : \{(x, r, r') : X = g^x h^r \bmod n, Y = g^x h^{r'} \bmod n\}(X, Y, g, h, n)$.
4. ZK range proof that a committed value v lies within a specific interval (a, b): $ZKP : \{(v, r) : X = g^v h^r \bmod n \wedge v \in (a, b)\}(X, g, h, n)$. Such proofs can also be used to show that the value v is positive or does not overflow some modulo operation. Most well-known construction families for range proofs include square decomposition, multi-base decomposition (used by zkLedger [46,54]) and the recent Bulletproofs [17].

ZK proofs can be composed as follows: (1) AND composition $\pi_1 \wedge \pi_2$ which can be easily constructed by sequential or parallel proving of underlying assertions, and (2) OR composition $\pi_1 \vee \pi_2$ which can be constructed by proving knowledge for the one and simulating knowledge for the other, without revealing which of the two is actually proved and which is simulated. We also note the ZK proofs used in MiniLedger are public coin and can be converted to non-interactive using the FS heuristic [32]. We refer to Appendix A for the formal properties of ZK proofs.

[1] We note that zkLedger [51] uses Pedersen commitments but overlooks the connection with twisted ElGamal. A proper use of twisted ElGamal in zkLedger can lead to optimizations as discussed in detail in Appendix D.

Cryptographic Accumulators. Accumulators allow the succinct and binding representation of a set of elements S and support constant-size proofs of (non) membership on S. We focus on *additive* accumulators where elements can be added over time to S and to *positive* accumulators which allow for efficient proofs of membership(in MiniLedger extensions we use *universal* accumulators which also allow for non-membership proofs). We consider *trapdoorless* accumulators in order to prevent the need for a trusted party that holds a trapdoor and could potentially create fake (non)membership proofs. Finally we require the accumulator to be *deterministic*, i.e. always produce the same representation given a specific set. An accumulator typically consists of the following algorithms [11]:

- $(\mathsf{pp}, D_0) \leftarrow \mathsf{AccSetup}(\lambda_{acc})$ generates the public parameters and instantiates the accumulator initial state D_0.
- $\mathsf{Add}(D_t, x) := (D_{t+1}, \mathsf{upmsg})$ adds x to accumulator D_t, which outputs D_{t+1} and upmsg which enables witness holders to update their witnesses.
- $\mathsf{MemWitCreate}(D_t, x, S_t) := w_x^t$ Creates a membership proof w_x^t for x where S_t is the set of elements accumulated in D_t. $\mathsf{NonMemWitCreate}$ creates the equivalent non-membership proof u_x^t.
- $\mathsf{MemWitUp}(D_t, w_x^t, x, \mathsf{upmsg}) := w_x^{t+1}$ Updates membership proof w_x^t for x after an element is added to the accumulator. $\mathsf{NonMemWitUp}$ is the equivalent algorithm for non-membership.
- $\mathsf{VerMem}(D_t, x, w_x^t) := \{0, 1\}$ Verifies membership proof w_x^t of x in D_t.

The basic security property of accumulators is *soundness* (or else *collision-freeness*) which states that for every element *not* in the accumulator it is infeasible to prove membership.

We utilize two types of accumulators: (a) the additive, universal RSA accumulator [13] and (b) additive, positive Merkle Trees [48]. We note that RSA accumulator can become trapdoorless if a trusted party (or an MPC protocol) is used to compute the primes for the modulo n, or a public RSA challenge number (i.e. from RSA Labs) is adopted. We also note that we will apply batching techniques in element additions and membership proofs [13]. In Section 6 we discuss the trade-offs between the two options for different implementation scenarios.

Consensus. A consensus protocol (denoted by CN) allows a set, S_{CN}, of distributed parties to reach agreement in the presence of faults. For MiniLedger we assume that the agreement is in regards to data posted on a ledger L and participation in the consensus protocol can be either *permissioned* (i.e. only authenticated parties have write access in the ledger) or *permissionless*(i.e. any party can write in the ledger as in Blockchain systems like Ethereum, Bitcoin etc). Consensus protocols that maintain such a fault-tolerant ledger and their details (e.g. participation credentials, incentives, sybil attack prevention etc.) are out of the scope of this paper and can be done using standard techniques [8,19]. For our construction, we assume a consensus protocol: $\mathsf{Conscus}(x, L) := L'$ which allows all system participants given some input value x and ledger state L, to agree on a new ledger L'. We also assume it satisfies the following two fundamental properties [35]: (a) Consistency: An honest node's view of the ledger on some round j is a prefix of an honest node's view of the ledger on some round

$j + \ell, \ell > 0$. (b) Liveness: An honest party on input of a value x, after a certain number of rounds will output a view of the ledger that includes x.

3 MiniLedger Model

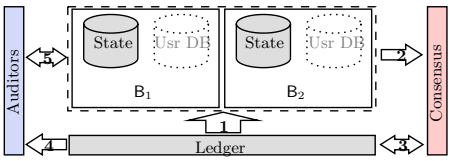


Fig. 1: MiniLedger overview. "State" is a private database for each Bank and UsrDB is an optional private databases for Bank's clients. Banks read from Ledger to create or prune transactions (1) and forward the transaction creation or pruning output to consensus (2). Consensus verifies and updates the Ledger (3). Auditor read Ledger (4) and interact with Banks to audit transactions (5).

We consider the following system participants: a Trusted Party TP, a set of consensus participants S_{CN}, a static set of n Banks with IDs defined by $[\mathsf{B}_j]_{j=1}^n$ (known to everyone) and an arbitrary number of Auditors A. Each Bank has a key pair $[(\mathsf{pk_j}, \mathsf{sk_j})]_{j=1}^n$ and an initial asset value $[v_j]_{j=1}^n$. Banks maintain an internal state $[st_j]_{j=1}^n$. We denote transactions by tx_i where i represents the transaction's index. We store transactions in a public ledger L maintained by a consensus layer CN and stored by all banks.

We summarize the role of each participant in MiniLedger and provide the architecture overview in Figure 1:

- TP is a trusted entity which runs an one-time setup to instantiate the system public parameters and verifies the initial assets of each Bank. TP could be replaced by an MPC protocol (i.e [37]) executed by the Banks.
- Banks generate transactions tx that transfer some of their assets to one or more other Banks, while hiding the value and the transacting parties. Transactions are sent to the consensus layer CN (via an anonymous communications protocol, i.e Tor) and if valid are appended on L. Banks can prune their transaction history on L and "replace" it by a digest D. The pruning Bank needs to send D to CN (incentives for the Bank to perform the pruning operation are orthogonal to our construction) and is responsible[2]to keep a copy of the pruned transactions in its private database "State" . If D is valid, CN participants update L by deleting the pruned transactions and replacing them by D.
- Auditors by observing the ledger, can audit the Banks at any point for any set of transactions through interactive protocols. Auditors should be able to

[2] Failure to locally store transaction history can lead to audit failures.

audit the value of a single transaction or a subset of transactions, whether these transactions are still in L or have been pruned.

We now state the assumptions required in MiniLedger and then describe our security and privacy goals.

Assumptions. We focus on the transaction layer and consider issues with underlying consensus and network layers and their mitigations orthogonal to this work. Specifically, we assume the fundamental consensus properties, as defined in Sec. 2, always hold. On network level, we assume a malicious Bank cannot block another Bank's view (Eclipse attacks). In addition we assume that transactions are sent to *all* Banks using anonymous communication channels to preserve anonymity of the sending and the receiving Bank(s). We *do not* require out-of-band communication between Banks.

The "race conditions" problem in CreateTx(), where different transactions might be created concurrently, is considered orthogonal to our work. We can either assume all transactions are submitted to consensus for verification in a synchronous manner (i.e. no "mempool" functionality), or can adopt an existing solution that uses an atomic exchange protocol, proposed for zkLedger in [22]. Finally, for the sake of simplicity, we assume the set of participating Banks is static but is easy to extend our system to dynamically add/remove Banks. We also assume the Random Oracle model to convert our ZK proofs to non-interactive.

Security Goals. MiniLedger should satisfy the following properties (formally defined in a game-based fashion in Appendices B.1, B.2):

Theft prevention and balance: When spending, we require that a) transaction is authorized by sending Bank, b) after spending, Bank's balance decreases by the appropriate amount and c) sending Bank cannot spend more than its total assets.

Secure pruning: Ledger pruning, which is executed in a user base, outputs a digest that a) is only created by the respective Bank, b) contains the correct transactions in the correct order, and c) does not contain bogus transactions. Ultimately secure pruning prevents tampering with ledger history.

Ledger correctness: The ledger only accepts valid transactions and pruning operations.

Correct and Sound Auditability: An honest Bank following the protocol can always answer audits correctly and convince an Auditor, while an Auditor always rejects false claims from a malicious Bank.

Privacy: The ledger hides both the identities of transacting parties and values of transactions from any external observer (except auditors who learn specific information during the auditing protocol).

4 MiniLedger Construction

Overview. We consider n Banks that transact with each other in an anonymous and auditable way by posting data in a common ledger L (a two-dimensional table with n columns, one for each participating Bank, and a number of rows

which represent transactions). The ledger is maintained by consensus participants, who verify every submitted transaction, and is stored by all Banks. The Banks could be running consensus themselves, or outsource this operation to any external set of consensus parties.

For each tx_i, the sending Bank (i.e. the transaction creator) creates a whole row in L which includes twisted ElGamal encryptions $C_{ij} = (c_1, c_2)$ that hide the transferred value v_{ij} that corresponds to each cell (i, j). For instance, if we assume that there's only one receiving Bank in a transaction, the sending Bank would compute an encryption of $-v$ for its own cell, an encryption of v for the receiver cell, and a number of encryptions of 0 for the rest of the cells. This makes the transmitted values indistinguishable to any external observer due to ElGamal IND-CPA security(assuming the sender uses different randomness values for each encryption). Additionally, the sending Bank accompanies each encryption with a NIZK proof π to prevent dishonest Bank behavior as discussed in details below. This specific ledger structure allows an external auditor to audit for a value sent/received by a Bank at any given point, with the Bank replying with a value v and a ZK proof π^{Aud} for its claim. This basic audit protocol can be extended to more complex queries (such as total assets held by a Bank or if a transaction exceeds a set limit) as we explain in Section 5.2.

A straightforward implementation of such a transaction table, as done in zkLedger, leads to a ledger L that grows linearly to the number of posted transactions. This makes schemes like zkLedger hard to adopt in practice, since every single participant would have to store a table of size n times the total number of transactions that have ever occurred. Besides storage costs, the overall computational performance would also degrade even more over time.

Reducing storage costs. The main idea for MiniLedger, is that each Bank B_j periodically initiates a pruning operation, which prunes (or "compacts") all transactions in its corresponding column on L (this is in contrast to typical consensus pruning, where nodes may be offline and have their transaction history in the public ledger pruned without their consent). When a Bank performs a pruning operation, it has to publish a digest D containing the pruned transactions. The consensus layer will first verify that D is indeed a valid digest, i.e. contains the transactions to be pruned, and then, come to an agreement about the pruning operation. Note that B_j is still responsible for maintaining a private copy of *all* its pruned transactions, however, there are great storage savings for the public version of the ledger L which *everyone* in the system has to maintain. In other words, with each Bank pruning its own transaction history, the whole ledger is effectively "sharded" to all Banks, where each Bank is responsible for maintaining a correct copy of its own history, while the public ledger only contains the compact representation D_j of each B_j's transaction history (as well as a few recent transactions that might have not been pruned yet). We note that the cost of a pruning operation depends on the number of transactions to be pruned but is independent of the number of participating Banks n and can be amortized based on the pruning frequency.

When B_j is audited for a pruned (i.e. not publicly visible) transaction value v_{ij}, it would have to present the needed data to the auditor by recovering it from its private copy of its transaction history. In addition, it would have to prove to the auditor not only that this data is contained in D, but also that it had been posted on the specific row i (i.e. maintain ordering).

We implement this pruning operation using cryptographic accumulators since they achieve a short, constant size representation of D. We require schemes which are (a) *additive*, i.e. have an update functionality that enables a Bank to prune additional transactions and update an already published D by adding the newly pruned ones, (b) *positive*, i.e. allow for proofs of membership but also capable of providing a "position"/"ordering" proof, and (c) *trapdoorless*, i.e. nobody has a trapdoor to create fake proofs of membership.

4.1 Our Construction

For our construction we assume the following building blocks: the variant of the ElGamal encryption ($\mathsf{SetupTEG}, \mathsf{GenTEG}, \mathsf{EncTEG}, \mathsf{DecTEG}$), an EU-CMA signature scheme ($\mathsf{SignGen}, \mathsf{Sign}, \mathsf{SVrfy}$), an additive, positive cryptographic accumulator ($\mathsf{AccSetup}, \mathsf{Add}, \mathsf{MemWitCreate}, \mathsf{MemWitUp}, \mathsf{VerMem}$)with additional properties as discussed above, the Pedersen commitment scheme (ComGen, Com, Open), a consensus protocol $\mathsf{Conscus}$ and a NIZK proof system. The phases of MiniLedger work as follows:

Setup: Setup can be executed with the help of a trusted third party or via an MPC protocol amongst Banks.

1. $\mathsf{SysSetup}\{\mathsf{TP}(1^\lambda) \leftrightarrow [B_j(v_j)]_{j=1}^n\}$. This interactive protocol is executed between TP and a set of n Banks. TP sets as κ the number of bits that can represent a value and verifies the initial asset value v_j for each Bank . TP generates the public parameters for the accumulator by running $\mathsf{AccSetup}()$, the key parameters of the ElGamal variant encryption scheme by executing $\mathsf{SetupTEG}()$ (which are also used for the Pedersen commitment scheme), the consensus protocol parameters by running $\mathsf{TPCSetup}$, and the joined set of parameters denoted as pp is sent to all Banks. Each Bank generates an ElGamal key pair $(\mathsf{pk}_{\mathsf{B}_j}, \mathsf{sk}_{\mathsf{B}_j})$ through $\mathsf{GenTEG}()$ and sends $\mathsf{pk}_{\mathsf{B}_j}$ to TP. Finally, TP encrypts the initial values of each Bank by running $C_{0j} = (c_1^{(0j)}, c_2^{(0j)}) \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_j}, v_j)$[3]. Then, it initializes a "running value total" which starts as $Q_{0j} = C_{0j}$ and will hold the encryption of the total assets of each Bank at any point. The vector $[Q_{0j}, C_{0j}]_{j=1}^n$ consists of the "genesis" state of the ledger L along with the system parameters pp containing the key parameters and all Bank public keys. At any point, the ledger L is agreed by the consensus participants and we assume that all Banks (whether participating in consensus layer or not) store $L.\mathsf{pp}$ and L are default inputs everywhere below.

Transaction creation:

[3] To simplify notation, from now on we will drop the superscripts from the two parts of Elgamal ciphertext, i.e., we will simply write $C_{0j} = (c_1, c_2)$.

2. $\mathsf{tx}_i \leftarrow \mathsf{CreateTx}(\mathsf{sk}_{\mathsf{B_k}}, [v_{ij},]_{j=1}^n)$. This algorithm is run by Bank B_k wishing to transmit some (or all) of its assets to other Banks in L. For each B_j in L (including itself), B_k executes $C_{ij} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B_j}}, v_{ij})$ and computes $Q_{ij} \rightarrow Q_{(i-1)j} \cdot C_{ij}$. In order to prove *balance*, similarly to [51], B_k should pick randomness values for the ElGamal variant encryptions such that $\sum_{j=1}^n r_{ij} = 0$. Then, the sending Bank B_k generates a NIZK π_{ij} $\forall j \in (1, ..n)$ which proves the following (the exact description of π_{ij} can be found in Appendix D):
Proof of Assets: Shows that *either* a) B_j is receiving some value ($v_{ij} \geq 0$), *or* b) B_j is spending no more than its total assets ($\sum_{k=1}^i v_{kj} \geq 0$) and within the valid range after transaction execution, while proving knowledge of its secret key $\mathsf{sk_j}$ showing it authorized the transfer. In both cases, an auxiliary commitment $\mathsf{cm_{ij}}$ is used which commits to either v_{ij} or $\sum_{k=1}^i v_{kj}$, so the proof includes a single range proof for the commitment value to reduce computational costs, as the range proof is the most computationally expensive part of π.
Proof of Consistency: Ensures consistency for the encryption randomness r in c_1 and c_2 in both cases of the previous sub-proof, which guarantees correct decryption by Bank k.
The transaction $\mathsf{tx}_i = [C_{ik}, \mathsf{cm_{ik}}, \pi_{ik}, Q_{ik}]_{k=1}^n$ is sent to consensus layer CN.
3. $\mathsf{VerifyTx}(\mathsf{tx}_i) := b_i$. Verify all ZK proofs $[\pi_j]_{j=1}^n$, check that $\prod_{j=1}^n c_2^{(ij)} = 1$ (proof of balance) and that $Q_{ij} = Q_{(i-1)j} \cdot C_{ij}$. On successful verification output 1, else output $\perp$.

Transaction pruning:

4. $(D_{\beta j}, st'_j, \sigma_j) \leftarrow \mathsf{Prune}(st_j)$ This algorithm is executed by B_j when it wishes to prune its transaction history of depth $q = \beta - \alpha$ and "compact" it to an accumulator digest $D_{\beta j}$, where α is the latest digest and β is a currently posted row number (usually a Bank will prune everything between its last pruning and the latest transaction that appeared in L). It parses C_{ij} from each tx_{ij}. It fetches its previous digest $D_{\alpha j}$ (if $\alpha = 1$, sets $D \rightarrow D_{\alpha j}$ as the initial accumulator value where A is defined from pp). Then $\forall C_{ij}, i \in [\alpha, \beta]$ it consecutively runs accumulator addition $\mathsf{Add}(D_{(i-1)j}, (i \parallel C_{ij}))$ (note the inclusion of index i which preserves ordering of pruned transactions in D_j). Finally it stores all transaction encryptions $[i, C_{ij}]_{i=\alpha}^{\beta}$ to its local memory, updates st_j to st'_j, computes $\sigma_j \leftarrow \mathsf{Sign}(D_{\beta j})$ and sends $D_{\beta j}, \sigma_j$ to CN. Note that $D_{\beta j}$ does not include proofs π, and pruning breaks proofs of balance in rows for all Banks. Still "breaking" these old proofs is not an issue, as they have already been verified.
5. $\mathsf{PruneVrfy}(D_{\beta j}, \sigma_j) := b_j$ On receipt of $D_{\beta j}$, locally executes $\mathsf{Prune}()$ for the same transaction set to compute $D'_{\beta j}$. If $D'_{\beta j} = D_{\beta j}$ (given the accumulator is deterministic) outputs 1, else outputs $\perp$.
We note that after a *successful* pruning operation (i.e. one that is agreed upon in consensus layer), all system participants that store L can delete all existing data in cells $(i, j) \forall i < \beta$ and just store $D_{\beta j}$ along with the latest $Q_{\beta j}$.

Consensus protocol: This is handled in the consensus layer CN with its details orthogonal to our scheme. Similar to typical blockchain consensus, participants will only update L with a new tx or D if this is valid according to the corre-

sponding vefirication algorithms (i.e. in Bitcoin, consensus participants validate transactions before posting them in L).

6. $\mathsf{Conscus}(\mathsf{tx}$ or $D) := L'$. Runs the consensus protocol among S_{CN} to update the ledger with a new tx or pruning digest D after checking their validity. If consensus participants come to an agreement, L is updated to a new state L'.

Auditing: Our auditing protocols below include a basic audit for a value v (that has either been pruned or not) and a set's sum of such values (which might be all past transactions, thus auditing Bank's total assets). These audits are interactive and require the Bank's consent. MiniLedger can support additional audit types and/or non-interactive audits as we discuss in Section 5.2.

7. $\mathsf{Audit}\{\mathsf{A}(C_{ij}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j. In this basic audit, A audits B_j for the value v_{ij} of a specific transaction tx_{ij} (that has not been pruned from L so far), encrypted as C_{ij} on the ledger L. B_j first decrypts the encrypted transaction through $\mathsf{DecTEG}()$ and sends v_{ij} to A, as well as a NIZK $\pi^{\mathsf{Aud}} : \{(\mathsf{sk_j}) : c_2/g^{v_{ij}} = (c_1)^{1/\mathsf{sk_j}}\}(c_1, c_2, v_{ij}, \mathsf{pk_j}, g, h)$. Then A accepts the audit for v_{ij} if π^{Aud} successfully verifies.

8. $\mathsf{AuditSum}\{\mathsf{A}([C_{ij}]_{i=\alpha}^{\beta}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j. Here A audits B_j for the sum of the values $\sum_{k=\alpha}^{\beta} v_{kj}$ for transactions $\mathsf{tx}_{\alpha j}$ up to $\mathsf{tx}_{\beta j}$ (that have not been pruned from L so far). This protocol is a generalization of the $\mathsf{Audit}\{\}$ protocol outlined above, (with $\mathsf{Audit}\{\}$ having as inputs $(\prod_{i=\alpha}^{\beta} C_{ij})$ and $\prod$ denoting direct product for ciphertexts c_1, c_2), because of ElGamal variant additive homomorphism. Note that although in this protocol the transactions are assumed to be consecutive for simplicity, its functionality is identical if the transactions are "isolated". Also if indices $\alpha = 1$ and β equals to the most recent transaction index (and no pruning has happened in the system), the audit is performed on the Bank's total assets.

9. $\mathsf{AudPruned}\{\mathsf{A}([(i,j)]_{i=\alpha}^{\gamma}, [C_{ij}]_{i=\gamma}^{\beta}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j, where transactions $[\mathsf{tx}_{ij}]_{i=\alpha}^{\gamma}$ have been pruned from L (and thus the auditor only knows their indices and nothing else), and transactions $[\mathsf{tx}_{ij}]_{i=\gamma}^{\beta}$ which are still public in L (i.e. not pruned) and thus the auditor still sees their encryptions. This protocol generalizes $\mathsf{AuditSum}\{\}$. It allows the auditor to audit B_j for: (a) specific transactions (pruned or not) and, (b) sums of assets (pruned or not). For case (a), besides auditing a transaction with index in $[\gamma, \ldots, \beta]$ which is still in L, the auditor can also audit B_j for a specific transaction that has been pruned from L (i.e. ask: "Which was the value of the i-th transaction?"). The Bank would respond with the corresponding C_{ij} and depending on the underlying accumulator used, B_j would also provide a proof that C_{ij} is a member of its pruned history D_j with index i. For case (b), an auditor can audit the total (or a range of) assets of B_j no matter what transaction information of B_j remains on L. Auditing total assets works as follows: B_j fetches the stored transaction encryptions $[C_{ij}]_{i=\alpha}^{\gamma}$ from its local memory st_j, computes $[w_{ij}]_{i=\alpha}^{\gamma} \leftarrow \mathsf{MemWitCreate}(D_j, [C_{ij}]_{i=\alpha}^{\gamma}, st_j)$ [4] . Then A reads D_j from

[4] When using batch RSA accumulator as we discuss later, we don't send a set of witnesses but a single witness for all encryptions.

	B_1	...	B_n
tx_1 ... tx_9	$D_{9,1}, Q_{9,1}$		...
tx_{10}	$C_{10,1} = (c_1 = \mathsf{pk}_1{}^{r_{10,1}},\ c_2 = g^{v_{10,1}} h^{r_{10,1}})$ $\pi_{10,1}, \mathsf{cm}_{10,1}, Q_{10,1}$		...
tx_{11}	$C_{11,1} = (c_1 = \mathsf{pk}_1{}^{r_{11,1}},\ c_2 = g^{v_{11,1}} h^{r_{11,1}})$ $\pi_{11,1}, \mathsf{cm}_{11,1}, Q_{11,1}$		...

(a) Ledger state before pruning, assuming B_1 had pruned before at tx_{10}.

	B_1	...	B_n
tx_1 ... tx_{11}	$D_{11,1}, Q_{11,1}$		...
tx_{12}	$C_{12,1} = (c_1 = \mathsf{pk}_1{}^{r_{12,1}},\ c_2 = g^{v_{12,1}} h^{r_{12,1}})$ $\pi_{12,1}, \mathsf{cm}_{12,1}, Q_{12,1}$		...

(b) Ledger state after B_1 prunes at tx_{12}. Digest $D_{11,1}$ represents $C_{10,1}, C_{11,1}$ and ciphertexts that were represented in $D_{9,1}$.

Table 2: MiniLedger Architecture and Pruning.

L and executes $\mathsf{VerMem}(D_j, (i \parallel C_{ij}), w_{ij}) \forall i \in (\alpha, \gamma)$, outputting $[b_{ij}]_{i=\alpha}^{\gamma}$. For every i, if $b_{ij} == 1$ it executes the $\mathsf{Audit}\{\}$ protocol with C_{ij} as input.
10. $\mathsf{AudTotal}\{\mathsf{A}(Q_{ij}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is equivalent to $\mathsf{Audit}\{\}$ for auditing B_j's total assets $\sum_{i=1}^{m} v_{ij}$ instead of a single v_{ij}.

We informally state the following theorem for the security of our scheme and provide proof sketches in Appendix B.3.

Theorem 1 *Assuming the security of the ElGamal variant, Pedersen commitments, NIZK, Accumulators and Consensus properties,* MiniLedger *construction satisfies our security definitions as given in Appendix B.2.*

4.2 Discussion and Comparisons

Although MiniLedger architecture resembles zkLedger [51], there exist crucial differences that make MiniLedger superior both in terms of efficiency and security. We give an overview below, and a thorough analysis in Appendix C.
Storage. As already discussed, MiniLedger by leveraging consensus properties applies a pruning strategy which achieves $O(n)$ storage requirements for L, compared to $O(mn)$ for zkLedger (where m is the total number of transactions ever happened, and is a monotonically increasing value).
Security. MiniLedger does not require any out-of-band communication, as all needed information is communicated through the ledger using encryptions. On the other hand, zkLedger assumes if a Bank is actually receiving some value in a transaction, it should be notified by the sending Bank and also learn the associated value (which was hidden in the commitment) through an out-of-band channel. zkLedger however, does not require receiving Banks to be directly informed on the randomness (i.e. commitment $\mathsf{cm_{ij}}$ is never opened), since they can still answer the audits correctly using the audit tokens, provided that it knows its total assets precisely. This assumption is very strong and can potentially lead to attacks, such as the *"unknown value"* attack where a malicious sending Bank informs the receiving Bank on a wrong value (or does not inform it at all), which then prevents the receiving Bank from answering audits or even participate in the system. More importantly, with transaction values communicated

out-of-bank, the randomness could be included with them as well. This would make the system trivial and defeat the purpose of most of its architecture, as the ledger would consist of just Pedersen commitments and proofs of assets. In this version the above attack would not work assuming all Banks are *always* online and verify the openings in real time, which is also a very strong assumption.
Computation. MiniLedger optimizes ZK proof computation over zkLedger by combining disjunctive proof of assets and proof of consistency into a single proof, giving an efficiency gain of roughly 10% in space and computation. (We note that this optimization could also benefit zkLedger as discussed in Appendix D.)Additionally, while zkLedger's computation performance degrades over time (as the monotonically-increasing ledger requires more operations to construct transactions), MiniLedger through the running totals Q achieves steady optimal performance.
On setup parameters. We argue that even with the use of a TP, the trust level is rather low. The parameters of ElGamal are just random generators (similar to Pedersen commitments in zkLedger) and for certain accumulator instantiations (such as Merkle trees) there is no trapdoor behind the parameter generation. Finally, the consensus setup essentially consists of choosing trapdoorless parameters (i.e. block specifics etc) and the set of participating parties. Thus, the only trust placed in TP is to pick a valid set of participants – something that all participants can check, exactly as in zkLedger. In comparison to zkLedger (given that ElGamal parameters are the same as Pedersen commitment parameters), the only additional setup is that of the accumulator which as discussed, for certain instantiations can be completely trapdoorless.

5 MiniLedger Security and Extensions

5.1 Adding Clients for Fine-grained Auditing

The compact nature of MiniLedger allows for fine-grained auditing where Banks represented on the ledger can serve as an intermediary for a set of clients, allowing them to exchange values using their Banks as intermediaries. We overview this system which we call MiniLedger+ below and provide its detailed construction in Appendix E.

Table 3: MiniLedger+ public ledger L. The extra information to be stored is denoted in blue color.

	B_1	...	B_j	...
...				
tx_i			$C_{ij} = (c_1 = \mathsf{pk}_1{}^{r_{ij}}, c_2 = g^{v_{ij}} h^{r_{ij}})$ $\pi_{ij}, \mathsf{cm}_{\mathsf{ij}}, D_{ij}, Q_{ij}, R_{ij}, H_{ij}$	

Protocol overview. Each Bank B_j maintains a *private* ledger of clients L_{B_j} (denoted as "UsrDB" in Figure 1), independent of the public ledger L. For each client m, B_j stores its transactions in encrypted format. These clients can be dynamically added or removed from L_{B_j}. Inspired by [21], each client is uniquely

and publicly associated with a single Bank as $f(\mathsf{pk}_{\mathsf{jm}}) = \mathsf{B}_j$ where f is a well-defined mapping of public keys pk to Banks.

When a client of B_s wishes to transfer value v to another client in the system, she creates a transaction that includes a transaction id, encryptions of the recipient client's pk, the receiver's Bank B_r and v, as well as appropriate NIZKs to prove consistency with the protocol. This information is recorded on the private ledger L_{B_s}. Then, B_s, constructs a transaction on L that transfers v to B_r. In addition to the transaction information required for standard MiniLedger, *each* cell will also contain an encryption of the recipient client's pk under $\mathsf{pk}_{\mathsf{B}_r}$, the transaction id, and NIZKs that prove that the correct value is sent to the correct Bank. All this information will be concatenated and represented by R_{ij} in L (as shown in Table 3). B_s also computes a digest H of the constructed transaction using a collision-resistant hash function $H()$ and includes it in all cells of the respective row. Note that information across rows in R, H is redundant but necessary to preserve ledger indistinguishability. Finally, the receiving B_r will perform the reverse steps to allocate the value to its client. B_s might elect to aggregate many transactions of its clients into a single one on L (recall that a transaction can have multiple receiving Banks, details in Appendix E.4).

In MiniLedger+, monetary values are not owned by the clients themselves, but are co-managed with the client and his respective Bank (as it happens in the actual banking system), meaning that an honest Bank having total assets Σv represented in L, will always have them distributed internally to its clients. As a result, Σv reflected in a column in L should always match the sum of value sums for all clients in L_{B_j}. However since L_{B_j} is private, the mechanism ensuring that this invariant holds will be *reactive*, in contrast with the main ledger L where the mechanism is proactive. In other words, an auditor would have to perform regular audits of the Banks to ensure they allocate the funds to their clients correctly and follow the protocol. In Appendix E we describe this audit in detail, as well as the extended threat model of MiniLedger+.

Auditors in MiniLedger+ can also perform audits at a client level as follows. Auditor first asks B_j to present its private L_{B_j} and fully validates its correctness and consistency with L as before. Then, the auditor performs audits on clients in a similar fashion as with Banks in standard MiniLedger (i.e. audit single transactions, sums etc.). Note that although the client is responsible for answering its audits correctly, B_j would be responsible for any inconsistencies between L_{B_j} and L. We provide more details in Appendix E.3.

Comparisons. MiniLedger+ has minimal storage overhead in L compared to MiniLedger. Note that the additional entries R and H in each column can still be pruned as in MiniLedger, thus maintaining a ledger of constant size. The additional computation costs associated with this extension are linear in the number of clients a Bank has, but can be made more efficient by aggregating techniques. We provide an overview of these costs in Table 7 in Appendix E.

Solidus [21] has a similar Bank-Client architecture but does not hide the identity of transacting parties and does not offer auditability functionalities while employing expensive ORAM operations. RSCoin [29] also has a similar paradigm of

"mintettes" associated with clients but without any privacy guarantees. Finally, [9] proposes a privacy-preserving payment UTXO-based system (as opposed to account-based in our work), which also supports fine-grained auditing in a user level. In this work, there exists a powerful designated auditor for each user, who can decrypt all user's transactions, making system-wide deanonymization possible in case auditors collude. The auditing costs are much greater compared to MiniLedger+ and the protocol is overall much more complex while it does not support different audit types such as total assets or value thresholds.

5.2 Additional Types of Audits

As shown in Section 4.1, MiniLedger basic audit functionality $\mathsf{Audit}\{\}$ is on the value v_{ij} of specific transaction tx_{ij}. Several more audit types can be constructed which reduce to that basic audit. We discuss some of those below, and provide more details for audit extensions in Appendix F.2.

Full transaction audit: For an auditor to learn the full details of a transaction (sender, receiver and values), they would have to audit the entire row (i.e. perform n audits on v_{ij} $\forall j$).

Statistical audits: Audits such as average or standard deviation are supported by utilizing "bit flags" to disregard zero-value transactions, proved for correctness in zero knowledge.

Value or transactions exceeding limit: Utilizing appropriate range proofs, an auditor can learn if a sent or received value exceeds some limit t. Multiple range proofs can show a Bank has not exceeded the limit over a time period.

Transaction recipient: While a Bank doesn't know (and therefore cannot prove) where a *received* value came from (unless learning it out-of-band as in zkLedger), for *outbound* transactions the Bank can keep an additional record of its transaction recipients in its local memory. As an example, for proving in tx_i that the Bank really sent v_{ij} to B_j, it could send this claim to the auditor who in turn would simply then audit B_j to verify this claim.

Client audits: Audits in a client level (e.g. statistical audits or transaction limits) can be performed similar to the respective audits in a Bank level, however the auditor needs first to learn and verify the Bank's private ledger L_B as discussed in Section 5.1. A special audit would be learning if a client has ever sent assets to another client pk. This audit requires transactions to include an additive universal accumulator, with each sender adding the end client recipient's pk to the accumulator, while also providing its Bank a ZK proof of adding the correct key. The audit is a proof of (non-) membership for that pk.

6 Evaluation

We implement a prototype of the transaction layer of MiniLedger in Python using the *petlib*[5] library to support cryptographic operations over the secp256k1

[5] `https://github.com/gdanezis/petlib`

elliptic curve. We use the *zksk* library [44] for the ZK and implement range proofs using the Schoenmakers' Multi-Base Decomposition method [55][6]. The measurements were performed on Ubuntu 18.04 - i5-8500 3.0 GHz CPU - 16GB RAM using a single thread[7]. As we focus on the transaction layer, we do not include measurements of the underlying consensus and network layers as they are orthogonal to our work. We do however discuss the potential additional costs in Section 6.

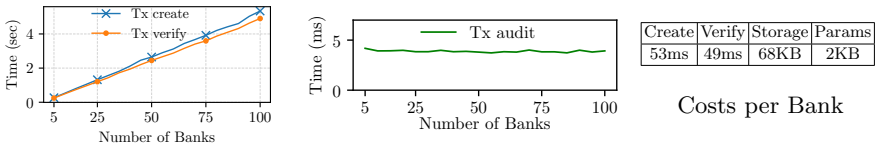
Accumulator Instantiation. A critical implementation choice is how to instantiate the accumulator needed for the pruning operations. For efficiency reasons, we require schemes with constant size public parameters and no upper bound on the number of accumulated (i.e pruned) elements. We only consider schemes that have at most sublinear computation and communication complexity (in the number of pruned elements) for opening/proving a single transaction to the auditor and where the auditor's verification cost is also at most sublinear.

We first consider Merkle trees [48]. Assuming a Bank prunes q transactions, the Merkle root provides $O(1)$ representation in terms of storage with $O(1)$ public parameters. Opening and verification complexity of Merkle proofs for a single transaction audit involves $O(\log q)$ hashing operations. However although hashes are relatively cheap operations, the over-linear verification complexity might be a concern when auditing a series of transactions. Finally, it should be noted that Merkle trees only support membership proofs.

We then consider the batch-RSA accumulator [13]. Given that all RSA typeaccumulators can only accumulate primes p, we use a deterministic prime mapping hash function (as in [13]) to enable accumulation of arbitrary inputs. The batch-RSA accumulator has $O(1)$ storage for its digest with $O(1)$ public parameters as well. Proving membership for a single element p in the standard RSA accumulator, requires the prover computing a witness w equal to the primes' product in the accumulator without p (an $O(q)$ operation as shown in Figure 4), and the verifier checking that $(g^w)^p = A$ where A is the current state of the accumulator. However, in batch-RSA, the prover can reduce computation costs by "batching" these operations for a set of elements $(p_1, p_2, ...)$ which are in the accumulator and provide a Proof of Knowledge of Exponent $\pi^{PoKE} : \{(x) : (g^w)^x = A\}(w, g, A)$ on the product $x = p_1 p_2 ...$ convincing a verifier without sending x (which is typically large) and thus reducing communication costs. In our setting we use batch-RSA in auditing in order to improve computational efficiency of membership proofs (when being audited for multiple transactions) and we note that the Bank under audit does not need to include a PoKE, as the auditor needs to recompute the prime mapping of the ciphertexts he is given by the Bank anyway (this is a trade-off between PoKE computation cost for the Bank and higher communication cost between Bank and Auditor). Through these techniques, batch-RSA achieves same complexity $O(q)$ as when

[6] By using twisted ElGamal [25], MiniLedger is fully-compatible with Bulletproofs [17] which can further reduce its concrete storage requirements.

[7] A basic implementation of MiniLedger is available at `https://github.com/PanosChtz/Miniledger`



Create	Verify	Storage	Params
53ms	49ms	68KB	2KB

Costs per Bank

Fig. 2: Transaction creation, verification and auditing costs.

proving membership of a single element (while Merkle Trees have $O(\ell \log q)$ complexity for ℓ elements). Consequently, the basic pruning operations Prune() and PruneVrfy() are about two orders of magnitude more expensive compared to Merkle trees as we show in Figure 3. However they are efficient when auditing large amounts of transactions especially if auditing the total sum. Then, batching allows for negligible computation costs for the proving Bank, and negligible audit verification cost for a single transaction (which can enable audit extensions as discussed in Appendix E). Thus, choice between Merkle trees and batch-RSA accumulators ultimately relies on the use-case requirements.

Finally, an alternative approach is the use of Vector Commitments [13,20,42]. However these constructions although offering additional properties, either have linear or quadratic public parameter costs [20,42] or are more expensive when proving and verifying membership compared to Merkle trees or RSA accumulators, as we show on Table 8 of Appendix G.

For our batch-RSA implementation, we use the SHA-256 hash function and the Miller - Rabin primality test for hashing to prime numbers, and we use an RSA-3072 modulo to maintain the same level of security [52]. We decouple the witness computation cost from the proof of membership cost for the Bank, as the Bank might elect to pre-compute the witnesses before its audit (assuming however that it does not prune again until the audit, since that would require the witnesses to be recomputed again). Note the auditor needs to run the hashing to prime mapping function again for all audited values (i.e. the auditor cannot rely on the "honesty" of a Bank presenting pre-computed prime numbers for its pruned transactions). For our Merkle tree implementation, we adopt a relevant python library[8], and we use SHA-256 as the underlying hash function.

Transaction Creation, Verification and Auditing. Every MiniLedger transaction includes an ElGamal ciphertext C, a commitment cm, a NIZK π and a running total Q for *each* Bank. Naturally, this results in linearly-increasing computation costs in terms of number of Banks as shown in Figure 2 for both transaction creation and verification. Note that storing the running total Q leads to constant transaction creation and verification computational costs (for fixed number of Banks), making total assets auditing much more efficient. In contrast, zkLedger's growing ledger size also implies linearly-increasing NIZK proof verification costs, as the verifier would need to compute the product of all transaction elements for each Bank(applying our running total technique to zkLedger could

[8] `https://github.com/vpaliy/merklelib`

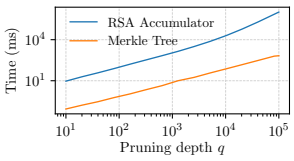


Fig. 3: Pruning computation cost

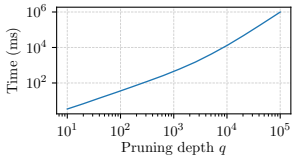


Fig. 4: RSA witness Generation cost

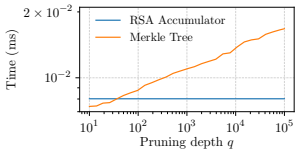


Fig. 5: Audit open cost for one tx

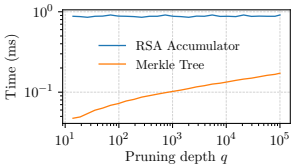


Fig. 6: Audit verify cost

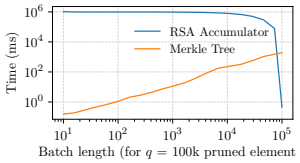


Fig. 7: Batch audit open costs

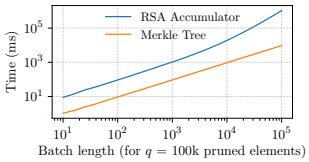


Fig. 8: Batch audit verify costs

have analogous benefits). The transaction creation and verification costs are 53ms and 49ms respectively (for a single cell in L)and are roughly comparable with [51]. Note that although we cannot directly compare different implementations, in Table 6 we show that our more efficient NIZK requires less expensive operations.

Auditing any single value on the ledger takes about 4 ms as shown in Figure 2. This is the cost for the complete auditing protocol, namely the decryption and proving cost for the Bank and the verification cost for the Auditor. In contrast to [51], the auditing cost is constant without being impacted either by the number of Banks or the number of past transactions.

Transaction Pruning. We evaluate the computation requirements of the pruning operation which involves executing Prune() and PruneVrfy() to create the digest D_j. Our results in Figure 3 show it is possible to prune and verify about 1 million transactions in less than a second using Merkle trees and in about 2 hours using RSA accumulator. Note prime number multiplication costs dominate the total costs (which also include hashing to primes and an exponentiation) when the pruning depth becomes large. We also stress that these computation requirements are *independent* of the number of Banks n in the system.

For transaction auditing in AudPruned{} interactive protocol, auditing opening and verification costs are shown in Figures 5 and 6 respectively. As previously discussed, we do not include the RSA accumulator's witness creation costs (which can be pre-computed) and are shown in Figure 4.

For auditing sums of values (i.e. "batch" auditing), the associated costs for opening and verifying a 100K transaction digest are shown in Figures 7 and 8 respectively, with x-axis representing the number of audited transactions. Note that for auditing 10^5 transactions (i.e. the whole sum), RSA accumulator opening is significantly cheaper compared to Merkle trees, as the audited Bank would only need to retrieve the respective transactions from its local memory (which

implies nearly O(1) cost) and send them to the auditor (who would in turn need to recompute all primes and perform the exponentiation of their product).

Based on our evaluation results and the discussion above, the choice between Merkle tree and batch RSA accumulator depends on use-case. Merkle trees fit a system expected to incorporate sparse audits on individual transactions, while RSA accumulator is preferred on deployments with frequent auditing on many transactions at a time (e.g. sums of assets or value thresholds over a time period).

Storage Costs. The storage cost for L has a $64n$-byte lower bound for the ElGamal variant encryptions (which represent the running total Q), plus the needed storage for each digest D and the system's pp, assuming all n Banks have pruned their transaction history and the ledger is made of a single row. Although during the system's operation where transactions are continuously appended on the ledger, its actual size will be more than that lower bound, enforcing frequent pruning operations through appropriate incentives (or penalties) will keep the size of the ledger close to its minimum. These savings in storage costs are huge compared to [51] where all Banks would have to store a ledger of size $O(nm)$ where m is the total number of transactions that have happened since the system's genesis.

Concretely, in our implementation each transaction's communication and storage cost is 68KB per Bank, which includes the ElGamal variant encryption, the auxiliary commitment, the NIZK and the running total. Note that we provide the actual memory footprint of our implementation (which relies on the underlying libraries' efficiencies) and not the theoretical lower bounds. A MiniLedger instantiation including the necessary public parameters, one transaction and a digest requires only 70KB of storage per Bank.

Network and Consensus Costs. As discussed in Section 3, MiniLedger focuses in the transaction layer, thus consensus layer costs depend on the exact instantiation choice. *Any* consensus protocol can be plugged to MiniLedger as long as it satisfies the basic properties of consistency and liveness. Although we consider consensus orthogonal to our implementation, we do recognize that its choice (along with network latency) ultimately affects transaction throughput. All benchmarks performed so far focus on metrics independent from consensus. Providing a full implementation of MiniLedger including a consensus layer is out of scope, we note previous works [31,51,59,25] also take a similar approach on evaluation and do not include consensus measurements. For instance, zkLedger [51] evaluation only takes network latency into account which is not useful without considering consensus costs (consensus is needed to guarantee agreement on L at any time). To showcase an implementation scenario, we discuss below how MiniLedger could be implemented using existing systems in the consensus layer and also provide some rough cost estimations.

We chose Hyperledger, one of the most prominent distributed operating systems for permissioned blockchains, to provide some estimated consensus measurements. Using Hyperledger Fabric with Kafka [1] as a permissioned consensus layer requires at least 0.5 seconds to complete a full consensus operation with 4 peers and 256-bit ECDSA [8]. We previously discussed that Banks

can store the ledger themselves and/or also have a "consensus verifier" role in our system (recall that while they could run consensus themselves, they do not have to, as we decouple consensus from Banks allowing any consortium of parties for ledger maintenance). Thus, Banks could act as Hyperledger "clients", "peers" and "orderers" simultaneously, which would impact performance especially with a PBFT-family consensus algorithm. For simplicity and efficiency, we consider Banks only acting as "clients", outsourcing ledger storage and consensus operations to an arbitrary number of "peer" and "orderer" nodes respectively. These numbers are entirely dependent on the use-case and does not affect MiniLedger performance or scalability. This separation between Banks and consensus participants is quite natural. As another example, Diem [6], uses similar architecture with MiniLedger (decoupled permissioned consensus and provider-intermediated transactions [7]) but has different goals in terms of privacy and auditability.

Banks	Peers	Tx/s	Network
10	80	21	LAN
100	4	2	WAN

Table 4: Consensus costs

Based on the Hyperledger evaluation, we derive conservative estimations of the expected transaction throughput, shown in Table 4. These estimations are more than sufficient for intra-Bank transactions in a deployed system (recall than any number of client-to-client transactions in MiniLedger+ can be aggregated in a single MiniLedger transaction). MiniLedger could also be imported in Diem, however the expected transaction throughput is lower as it uses a Byzantine-tolerant consensus [60].

Although permissioned consensus generally seems more fitting to MiniLedger, permissionless consensus could also be utilized. For instance, MiniLedger could be implemented on top of an Ethereum smart contract, where Banks would be responsible to pay the respective gas fees. While technically any Proof-of-X consensus could be used, the underlying game-theoretic aspects should also be considered.

Fine-grained Audit Extension. For MiniLedger+ we need to store extra information on L, namely R_{ij} and H_{ij} for each entry. For consistency we use the same 256-bit hash function as in our RSA accumulator. R_{ij} has a total size of 512 bits, (excluding the size of id), still pruning makes a ledger with same lower bound possible as before. In Appendix E we derive the computation overhead for the sending Bank to create R_{ij} roughly equivalent to $\mathsf{CreateTx}()$ (in terms of number of clients and Banks respectively), showing computation costs are doubled compared to basic MiniLedger. As the Bank only needs to include additional information in D for each transaction, the effect on pruning costs for L is negligible.

7 Conclusion

We present MiniLedger, the first *private and auditable payment system with storage independent to the number of transactions*. MiniLedger utilizes existing cryptographic tools and innovates on the meticulous design of optimized ZK proofs to tackle important scalability issues in auditable, private payments. Ad-

ditionally, we provide the first formal security definitions for auditability and secure pruning in private and auditable payment systems. We achieve huge storage savings compared to previous works that store information for each transaction ever happened. Using our pruning techniques, the overall MiniLedger size can be impressively compacted to 70KB per Bank, no matter how many transactions have ever occurred. Note that our storage and computation costs could be further improved, e.g. by using Bulletproofs [17] (instead of Schoenmakers multi-base decomposition [55]), more efficient programming languages (e.g. Rust) and libraries, or by utilizing CPU parallelization. However, as in related systems [6,51] our goal is not to support "thousands" of Banks, but an arbitrary number of clients as discussed in Section 5.1, which does not affect the computation/storage costs in the public ledger. MiniLedger can currently serve a small consortium of Banks (e.g. the world's Central Banks) with an *arbitrary* number of clients, or build a hierarchy of a large number of Banks and clients in accordance with MiniLedger+. Evaluating MiniLedger in such a large scale or achieving its properties in a permissionless setting are interesting directions for future work.

References

1. Apache kafka, https://kafka.apache.org/
2. Privacy coins face existential threat amid regulatory pinch, https://www.bloomberg.com/news/articles/2019-09-19/privacy-coins-face-existential-threat-amid-regulatory-crackdown
3. 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 pp. 1–88 (2016)
4. Developments in audit 2016/2017 (2017), https://www.frc.org.uk/getattachment/915c15a4-dbc7-4223-b8ae-cad53dbcca17/Developments-in-Audit-2016-17-Full-report.pdf
5. Corda: A distributed ledger (2019), https://www.corda.net/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf
6. The libra blockchain (2020), https://developers.libra.org/docs/assets/papers/the-libra-blockchain/2020-05-26.pdf
7. Libra roles and permissions (2020), https://lip.libra.org/lip-2/
8. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Oliveira, R., Felber, P., Hu, Y.C. (eds.) Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, April 23-26, 2018. pp. 30:1–30:15. ACM (2018)
9. Androulaki, E., Camenisch, J., De Caro, A., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B.: Privacy-preserving auditable token payments in a permissioned blockchain system. Cryptology ePrint Archive, Report 2019/1058 (2019), https://eprint.iacr.org/2019/1058
10. Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in Bitcoin. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 34–51. Springer, Heidelberg (Apr 2013). https://doi.org/10.1007/978-3-642-39884-1_4
11. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: 2017 IEEE European Symposium on Security and Privacy, Paris, France, April 26-28, 2017. pp. 301–315. IEEE (2017)
12. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
13. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 561–586. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_20
14. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352 (2020)
15. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_26

16. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Heidelberg (Feb 2020)
17. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018)
18. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052252
19. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186 (1999)
20. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_5
21. Cecchetti, E., Zhang, F., Ji, Y., Kosba, A.E., Juels, A., Shi, E.: Solidus: Confidential distributed ledger transactions via PVORM. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 701–717. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134010
22. Centelles, A., Dijkstra, G.: Extending zkledger with private swaps `https://cdn2.hubspot.net/hubfs/6034488/privateledger.pdf`
23. Chalkias, K., Lewi, K., Mohassel, P., Nikolaenko, V.: Distributed auditing proofs of liabilities. Cryptology ePrint Archive, Report 2020/468 (2020), `https://eprint.iacr.org/2020/468`
24. Chatzigiannis, P., Baldimtsi, F., Chalkias, K.: Sok: Auditability and accountability in distributed payment systems. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12727, pp. 311–337. Springer (2021). https://doi.org/10.1007/978-3-030-78375-4_13
25. Chen, Y., Ma, X., Tang, C., Au, M.H.: PGC: Decentralized confidential payment system with auditability. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part I. LNCS, vol. 12308, pp. 591–610. Springer, Heidelberg (Sep 2020). https://doi.org/10.1007/978-3-030-58951-6_29
26. Chepurnoy, A., Papamanthou, C., Zhang, Y.: Edrax: A cryptocurrency with stateless transaction validation. Cryptology ePrint Archive, Report 2018/968 (2018), `https://eprint.iacr.org/2018/968`
27. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_19
28. Dagher, G.G., Bünz, B., Bonneau, J., Clark, J., Boneh, D.: Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 720–731. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813674
29. Danezis, G., Meiklejohn, S.: Centrally banked cryptocurrencies. In: NDSS 2016. The Internet Society (Feb 2016)
30. Doerner, J., Shelat, A., Evans, D.: Zeroledge: Proving solvency with privacy (2015)
31. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: Galbraith, S.D., Moriai, S. (eds.) ASI-

ACRYPT 2019, Part I. LNCS, vol. 11921, pp. 649–678. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_23
32. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
33. Frankle, J., Park, S., Shaar, D., Goldwasser, S., Weitzner, D.J.: AUDIT: Practical accountability of secret processes. Cryptology ePrint Archive, Report 2018/697 (2018), https://eprint.iacr.org/2018/697
34. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of Mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 657–689. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_22
35. Garay, J.A., Kiayias, A.: SoK: A consensus taxonomy in the blockchain era. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 284–318. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-40186-3_13
36. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (Feb 2016)
37. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (May 1999)
38. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
39. Goldwasser, S., Park, S.: Public accountability vs. secret laws: Can they coexist? Cryptology ePrint Archive, Report 2018/664 (2018), https://eprint.iacr.org/2018/664
40. Heasman, W.: Privacy coins in 2019: True financial freedom or a criminal's delight? (Jan 2020), https://cointelegraph.com/news/privacy-coins-in-2019-true-financial-freedom-or-a-criminals-delight
41. Jiang, Y., Li, Y., Zhu, Y.: Auditable zerocoin scheme with user awareness. In: Proceedings of the 3rd International Conference on Cryptography, Security and Privacy, Kuala Lumpur, Malaysia, January 19-21, 2019. pp. 28–32 (2019)
42. Lai, R.W.F., Malavolta, G.: Subvector commitments with application to succinct arguments. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 530–560. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_19
43. Li, Y., Yang, G., Susilo, W., Yu, Y., Au, M.H., Liu, D.: Traceable monero: Anonymous cryptocurrency with enhanced accountability. IEEE Transactions on Dependable and Secure Computing (2019). https://doi.org/10.1109/TDSC.2019.2910058
44. Lueks, W., Kulynych, B., Fasquelle, J., Bail-Collet, S.L., Troncoso, C.: zksk: A library for composable zero-knowledge proofs. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 50–54 (2019)
45. Maxwell, G.: Confidential transactions (2015), https://people.xiph.org/~greg/confidential_values.txt
46. Maxwell, G., Poelstra, A.: Borromean ring signatures (2015), https://github.com/Blockstream/borromean_paper/blob/master/borromean_draft_0.01_34241bb.pdf
47. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: Characterizing payments among men with no names. Commun. ACM **59**(4), 86–93 (Mar 2016)

48. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2_32
49. Naganuma, K., Yoshino, M., Sato, H., Suzuki, T.: Auditable zerocoin. In: 2017 IEEE European Symposium on Security and Privacy Workshops. pp. 59–63 (2017)
50. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
51. Narula, N., Vasquez, W., Virza, M.: zkledger: Privacy-preserving auditing for distributed ledgers. In: 15th USENIX Symposium on Networked Systems Design and Implementation. pp. 65–80. USENIX Association, Renton, WA (Apr 2018)
52. National Institute of Standards and Technology: Recommendation for Key Management: NIST SP 800-57 Part 1 Rev 4. USA (2016)
53. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
54. Poelstra, A., Back, A., Friedenbach, M., Maxwell, G., Wuille, P.: Confidential assets. In: Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., Sala, M. (eds.) FC 2018 Workshops. LNCS, vol. 10958, pp. 43–63. Springer, Heidelberg (Mar 2019). https://doi.org/10.1007/978-3-662-58820-8_4
55. Schoenmakers, B.: Interval proofs revisited. In: Workshop on Frontiers in Electronic Elections (2005)
56. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. Journal of Cryptology **15**(2), 75–96 (Mar 2002). https://doi.org/10.1007/s00145-001-0020-9
57. Van Saberhagen, N.: Cryptonote v 2.0 (2013), `https://cryptonote.org/whitepaper.pdf`
58. Wood, G.: Ethereum: A secure decentralized generalised transaction ledger (2021), `https://ethereum.github.io/yellowpaper/paper.pdf`, accessed: 2021-02-14
59. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: PRCash: Fast, private and regulated transactions for digital currencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 158–178. Springer, Heidelberg (Feb 2019)
60. Zhang, J., Gao, J., Wu, Z., Yan, W., Wu, Q., Li, Q., Chen, Z.: Performance analysis of the libra blockchain: An experimental study. CoRR **abs/1912.05241** (2019), `http://arxiv.org/abs/1912.05241`

A Zero-knowledge Proof properties

The needed properties of a zero-knowledge proof are Completeness, Soundness and Zero-Knowledge,defined formally as follows.

- *Complenetess:* If $R(x, w) = 1$ and both players are honest V always accepts.
- *Soundness:* For every malicious and computationally unbounded P^*, there is a negligible function $\epsilon(\cdot)$ s.t. if x is a false statement (i.e. $R(x, w) = 0$ for all w), after P^* interacts with V, $\Pr[V \text{ accepts}] \leq \epsilon(|x|)$.
- *Zero Knowledge:* For every malicious PPT V^*, there exists a PPT simulator $\mathcal{S}$ and negligible function $\epsilon(\cdot)$ s.t. for every distinguisher D and $(x, w) \in R$ we have $|\Pr[D(\text{View}_{V^*}(x, w)) = 1] - \Pr[D(\mathcal{S}) = 1]| \leq \epsilon(|x|)$.

B MiniLedger Security

B.1 Scheme Definitions

We define MiniLedger for a static set of n Banks with IDs defined by $[\mathsf{B}_j]_{j=1}^n$. Each Bank has a key pair $[(\mathsf{pk}_\mathsf{j}, \mathsf{sk}_\mathsf{j})]_{j=1}^n$ and an initial asset value $[v_j]_{j=1}^n$. Banks maintain an internal state $[st_j]_{j=1}^n$. We assume that the set of participating Banks IDs, $[\mathsf{B}_j]_{j=1}^n$, is known to all system participants.

MiniLedger is composed of the following protocols:

- $\mathsf{SysSetup}\{\mathsf{TP} \leftrightarrow [\mathsf{B}_j]_{j=1}^n\}$: executed between TP and a set of Banks $[\mathsf{B}_j]_{j=1}^n$ (or by Banks through an MPC protocol). It verifies the initial values of Banks and outputs the system parameters pp, and an initial ledger L with total system value $v_T = \sum_{j=1}^n v_j$. (We assume that pp and L are default inputs everywhere below.)
- $\mathsf{CreateTx}(B_j, B_k, v)$: run by a Bank B_j, outputs transaction tx which transfers assets v to Bank B_k[9].
- $\mathsf{VerifyTx}(\mathsf{tx})$: run by any party, verifies the validity of tx and outputs a verification bit b.
- $\mathsf{Prune}(st_j)$: run by a Bank B_j, outputs a digest D containing a "compact" representation of its transaction history and updates the Bank's state to st'_j.
- $\mathsf{PruneVrfy}(D)$: run by any party, verifies the validity of digest D and outputs a verification bit b.
- $\mathsf{Conscus}(\mathsf{tx} \text{ or } D)$: is run by all consensus participants in S_{CN}. On input a transaction tx or a pruning digest D, the consensus participants will verify the transaction/digest using the corresponding algorithms and update the ledger to L'.
- $\mathsf{Audit}\{\mathsf{A} \leftrightarrow \mathsf{B}_j\}$: executed between a Bank B_j and an auditor A, where the auditor audits a specific transaction (or a set of them) the Bank reveals the value(s) v of that transaction (or set) to the auditor including a proof π^{Aud} of correct presentation of the value(s) v.

[9] Here we assume a single receiving Bank for notation simplicity, but this algorithm can be easily extended to support multiple receivers.

- $\mathsf{AudVrfy}(v, \pi^{\mathsf{Aud}})$: executed by an auditor A to verify the validity of the proof π^{Aud} based on the provided value v.

B.2 Security Definitions

To define security we first describe the oracles provided to an adversary $\mathcal{A}$. We assume a challenger CH maintains a corrupted Bank list T_c and performs some "bookkeeping" for honest Banks $\notin T_c$ on oracle queries, where honest Banks have total assets v^{CH}. This bookkeeping includes the following: a) for $\mathsf{CreateTx}()$, it appends the output transaction tx to the ledger L, and b) for $\mathsf{Prune}()$, it replaces the honest Bank's transaction history with digest D on L and updates the Bank's state to st'_j. Note that for security definitions below that involve auditability, we only consider the basic audit on a transaction value - for brevity we omit security definitions that involve more complex auditing.

- $\mathcal{O}_c(\mathsf{B}_j)$: $\mathcal{A}$ corrupts Bank B_j and takes full control of it. $\mathcal{O}_{corr}$ records B_j to T_c. This oracle captures that $\mathcal{A}$ can corrupt honest Banks.
- $\mathcal{O}_{tx}((\mathsf{B}_j, -v_j), [\mathsf{B}_k, v_k])$: $\mathcal{A}$ queries $\mathcal{O}_{tx}$ to create a transaction which transfers v_j from Bank j (where $j \notin T_c$) to recipient Bank(s) B_k. If Bank j has at least assets v_j, $\mathcal{O}_{tx}$ executes $\mathsf{CreateTx}()$ outputting tx_i and runs bookkeeping, else it outputs $\perp$. This oracle captures that $\mathcal{A}$ can direct honest Banks to make specific (but valid) transactions of its choosing.
- $\mathcal{O}_{prn}(\mathsf{B}_j)$: $\mathcal{A}$ queries $\mathcal{O}_{prn}$ to prune the transaction history of $\mathsf{B}_j \notin T_c$. $\mathcal{O}_{prn}$ generates digest D and runs bookkeeping. This oracle captures that $\mathcal{A}$ can prune the transaction history of an honest Bank.

Definition 1 (Theft prevention and balance) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}, v_t = \sum_{j=1}^n v_i; \\ \mathsf{tx}^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}}(\mathsf{pp}) : \\ \left\{ (\mathsf{B}_j \notin T_c) \vee (v_t^* \neq v_t) \vee (\mathsf{B}_j \in T_c, \mathsf{B}_k \notin T_c, v^* > v^{\mathcal{A}}) \right\} \wedge \\ \mathsf{VerifyTx}(\mathsf{tx}^*) = 1 \end{array}\right] \leq negl(\lambda)$$

where tx^* is a transaction spending v^* from B_j to B_k, $v^{\mathcal{A}}$ the adversary's total assets before tx^* and v_t, v_t^* the system's total value before and after tx^* respectively.

This property captures the requirements that: only the owner of the assets can spend them, a transaction results to a decrease of sender's assets by the value represented in the transaction and that the sender cannot spend more than its total assets.

Definition 2 (Secure pruning) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}$:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\};\\ D^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}}(\mathsf{pp}):\\ \left\{\begin{array}{l}\mathsf{B}_j \notin T_c \vee\\ (\exists \mathsf{tx}_{\mathsf{B}_j} \in L \wedge \mathsf{tx}_{\mathsf{B}_j} \notin D^*) \vee\\ (\exists \mathsf{tx}^*_{\mathsf{B}_j} \notin L \wedge \mathsf{tx}^*_{\mathsf{B}_j} \in D^*)\end{array}\right\} \wedge\\ \mathsf{PruneVrfy}(D^*) = 1\end{array}\right] \leq negl(\lambda)$$

where $\mathsf{tx}_{\mathsf{B}_j}$ denotes a transaction where B_j is involved in, and $\mathsf{tx} \in D^*$ denotes representation of a transaction in the digest.

In the above experiment, $\mathcal{A}$ wins if he creates a digest D^* on behalf of an honest Bank B_j, or if D^* *either* does not contain a transaction $\mathsf{tx}_{\mathsf{B}_j}$ that exists in L [10] *or* contains a transaction $\mathsf{tx}^*_{\mathsf{B}_j}$ that does not exist in L. This property captures the requirement that digest outputs are only created by the respective Banks, that pruning operations contain the correct transactions in the correct order, and do not contain bogus transactions.

Definition 3 (Ledger Correctness) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}$:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\};\\ \mathsf{tx}^* \backslash D^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}}(\mathsf{pp}):\\ \left\{\begin{array}{l}\mathsf{VerifyTx}(\mathsf{tx}^*) = 0 \wedge \mathsf{tx}^* \in L\\ \mathsf{PruneVrfy}(D^*) = 0 \wedge D^* \in L\end{array}\right\}\end{array}\right] \leq negl(\lambda)$$

This property captures the requirement that only valid transactions or pruning operations are accepted on the ledger.

Definition 4 (Correct Auditability) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ and for any honest auditor A:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\};\\ \mathcal{A}^{\mathcal{O}}(\mathsf{pp}):\\ \exists \mathsf{B}_j \notin T_c \wedge \exists \mathsf{tx}_{\mathsf{B}_j} \wedge\\ \mathsf{Audit}\{\mathsf{A}(\mathsf{tx}_{\mathsf{B}_j}) \leftrightarrow \mathsf{B}_j(v)\} := \pi^{\mathsf{Aud}} \wedge\\ \mathsf{AudVrfy}(\pi^{\mathsf{Aud}}) := 0\end{array}\right] \leq negl(\lambda)$$

where $\mathsf{tx}_{\mathsf{B}_j}$ denotes a transaction where B_j is involved in and v is input of $\mathsf{CreateTx}()$ *which outputs* $\mathsf{tx}_{\mathsf{B}_j}$.

In the above experiment, after $\mathcal{A}$ has made polynomial number of queries to the oracles (which include generating transactions and pruning them), $\mathcal{A}$ wins if any honest Bank is unable to correctly answer some audit. This property captures the requirement that any Bank following the protocol should always be able to answer audits correctly.

[10] $\mathcal{A}$ also wins if the transaction is indeed in D^* but out of order.

Definition 5 (Sound Auditability) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ and for any honest auditor* A*:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\};\\ \mathcal{A}^{\mathcal{O}}(\mathsf{pp}):\\ \mathsf{B}_j \in T_c \wedge \exists \mathsf{tx} \wedge\\ \mathsf{Audit}\{\mathsf{A}(\mathsf{tx}) \leftrightarrow \mathsf{B}_j(v^*)\} := \pi^{\mathsf{Aud}} \wedge\\ v^* \neq v \wedge\\ \mathsf{Vrfy}(\pi^{\mathsf{Aud}}) := 1\end{array}\right] \leq negl(\lambda)$$

where v is input of $\mathsf{CreateTx}()$ *which outputs* tx *and v^* is any arbitrary value.*

In the above experiment, after $\mathcal{A}$ has made polynomial number of queries to the oracles (which include generating transactions and pruning them), $\mathcal{A}$ wins if it succeeds on cheating an auditor when a corrupted Bank is audited for a transaction. This property captures the requirement that an auditor cannot accept false audit claims.

Definition 6 (Privacy) *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}\\ b \in \{0,1\}, L_b, L_{1-b} \leftarrow \mathsf{CH};\\ \text{StartLoop}\\ \mathsf{tx}, \mathsf{tx}' \leftarrow \mathcal{A}^{(\mathcal{O}_{tx}, \mathcal{O}_{prn})_b, (\mathcal{O}_{tx}, \mathcal{O}_{prn})_{1-b}}(\mathsf{pp})\\ \mathsf{CH}(\mathsf{tx}, \mathsf{tx}') : \mathsf{VerifyTx}(\mathsf{tx}), \mathsf{VerifyTx}(\mathsf{tx}')\\ \mathsf{CH} : \mathsf{tx} \rightarrow L'_b, \mathsf{tx}' \rightarrow L'_{1-b}\\ \text{End Loop}:\\ b' \leftarrow \mathcal{A},\\ b = b'\end{array}\right] \leq \frac{1}{2} + negl(\lambda)$$

In the above experiment, the challenger CH instantiates two ledgers L and L' in the setup phase and flips a bit. Then $\mathcal{A}$ queries two instances of oracles $\mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ (each uniquely associated with a ledger) a polynomial number of times. However $\mathcal{A}$ is only allowed to query the same oracle type (i.e. it cannot query $\mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ the same time). For each query set, the challenger updates the two ledgers as chosen by b in the setup phase. Finally $\mathcal{A}$ guesses b. The privacy property captures the requirement that the ledger hides both transacting parties and the associated values.

B.3 Security Proofs

We now informally argue about the security of Theorem 1.

Lemma 1 MiniLedger *satisfies Theft prevention and balance under the assumptions of ZK soundness, and hardness of the discrete logarithm problem.*

Proof. We distinguish the following two cases:
Case 1 (theft prevention):

Case 1a: In case $\mathcal{A}$ can derive a secret key from a public key (i.e. discrete logarithm hardness assumption does not hold), this immediately enables theft of funds.

Case 1b: In the case $\mathcal{A}$ can break ZK soundness, when outputting tx^* can either a) For $\pi^* = \pi_L^* \vee \pi_R^*$ in tx^*, $\mathsf{Ver}(\pi_R^*) = 1, R(x, \mathsf{sk}) = 0$ (i.e. output the π_R^* component of the OR proof π without knowing sk, or b) violate the range proof component of the proof π, i.e. $\mathsf{cm} = g^{v'} h^{r''} \wedge v' \in [0, 2^k$, $\mathsf{Ver}(\pi_R^*) = 1, v < 0$ ($v > 2^k$ is equivalent to $v < 0$), thus permitting committing to a negative value under an honest Bank (which effectively spends from that Bank).
Case 2 (balance):

Case 2a: Winning the game by changing the systems's total value (i.e. having instantiated a system with total value v_t, construct a verifiable tx^* which then results into system's total value $v_t^* \neq v_t$): A verifiable tx^* implies $\prod_{j=1}^{n} c_2^{(ij)} = 1 \implies g^{v_1+v_2+..+v_n+e} h^{r_1+r_2+..+r_n} = 1$, where $e \neq 0$, which implies $\mathcal{A}$ can find $r^* = r_1 + r_2 + .. + r_n \neq 0$ such that $h^{r^*} = 1/g^{v_t^*}$, which contradicts the hardness of discrete logarithm problem.

Case 2b: Winning the game by spending more than the total assets: In the case $\mathcal{A}$ can break ZK soundness, when outputting tx^* he can violate the range proof component π_r of the proof π^*, i.e. for π_r : $\mathsf{cm} = g^{v'} h^{r''} \wedge v' \in [0, 2^k$, $\mathsf{Ver}(\pi_R^*) = 1, v < 0$ ($v > 2^k$ is equivalent to $v < 0$). This enables committing to a negative sum of assets under a corrupted Bank, which implies spending more than its total assets.

Lemma 2 MiniLedger *satisfies secure pruning assuming Accumulator soundness, EU-CMA signatures and consistency of the consensus layer.*

Proof. **Case 1:** $\mathcal{A}$ can win the game in Definition 2 by pruning on behalf of an honest Bank. This can only succeed if $\mathcal{A}$ manages to sign on behalf of that Bank, which directly contradicts signature unforgeability.

Case 2: $\mathcal{A}$ wins the game by adding an arbitrary transaction into a digest D on behalf of a corrupted Bank. This implies either breaking accumulator soundness, as the adversary would be able to successfully prove membership for a non-accumulated element, or consensus consistency (recall that the digest is created by the consensus participants as well).

Case 3: $\mathcal{A}$ wins the game by omitting a transaction in the digest D, which would also contradict accumulator soundness as in the previous case.

Lemma 3 MiniLedger *satisfies Ledger Correctness assuming consistency of the consensus layer.*

Proof. This follows directly from the assumed consensus properties.

Lemma 4 MiniLedger *satisfies Correct Auditability assuming ZK soundness.*

Proof. For Correct Auditability we define the following game:

1. Setup: CH runs $\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}$ and sends pp to $\mathcal{A}$, which include $[\mathsf{pk}_{\mathsf{B}_j}, C_{0j}, Q_{0j}]_{j=1}^n$.
2. Query: $\mathcal{A}$ queries $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}, \mathcal{O}_{Aud}$ and CH answers the queries as in Appendix B.2, maintaining the ledger L.
3. Output: For an honest Bank j, $\mathcal{A}$ outputs a transaction $\mathsf{tx}_{\mathsf{B}_j}$ where the Bank fails to convince an honest Auditor A that the value of $\mathsf{tx}_{\mathsf{B}_j}$ is v.

Assume $\mathcal{A}$ wins the above game with non negligible probability. Then $\mathcal{B}$ would break ZK soundness as follows: On input of $\pi^* : \{c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r\}$, $\mathcal{B}$ runs $\mathsf{SysSetup}$, however it replaces pk_1 with pk and sets as g, h the output of $\mathsf{SetupTEG}$. Then forwards (pp, L) to $\mathcal{A}$, with $\mathcal{B}$ providing oracle access to $\mathcal{A}$ for $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{Aud}$ and $\mathcal{O}_{prn}$. For $\mathcal{O}_{tx}$, $\mathcal{B}$ outputs tx_i to $\mathcal{A}$ that includes $[\pi]_{k=1}^n$ which also ensures consistency for randomness r between each pair of ciphertexts c_1 and c_2. However in each $\mathcal{O}_{tx}$ query, $\mathcal{B}$ adds $c_{2,1} = g^v h^r, c_{1,1} = \mathsf{pk}^r$ to each update of L. When $\mathcal{A}$ queries $\mathcal{O}_c$ for B_1, $\mathcal{B}$ aborts. When $\mathcal{A}$ outputs a $\mathsf{tx}_{\mathsf{B}_j}$, if $j \neq 1$, $\mathcal{B}$ aborts. Else if $j = 1$, with $\mathsf{Audit}\{\mathsf{A}(\mathsf{tx}_{\mathsf{B}_1}) \leftrightarrow \mathsf{B}_1(v)\} := \pi^{\mathsf{Aud}} \wedge \mathsf{AudVrfy}(\pi^{\mathsf{Aud}}) := 0$, implies that B_1 fails to decrypt correctly to answer the audit, thus $c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^{r'}$ breaking ZK soundness.

Lemma 5 MiniLedger *satisfies Sound Auditability assuming ZK soundness, Accumulator soundness and consistency of the consensus layer.*

Proof. **Case 1:** tx is still in L. An $\mathcal{A}$ breaking ZK soundness can convince CH for some false v^*.

Case 2: tx is pruned and represented in the digest D in L. An $\mathcal{A}$ breaking Accumulator soundness can prove membership to CH for an arbitrary tx^* and then proceed with auditing on that tx^*.

Lemma 6 MiniLedger *satisfies Privacy assuming IND-CPA security of ElGamal encryption variant, Pedersen commitment hiding and Zero-knowledgeness of NIZKs.*

Proof. We describe a sequence of hybrid experiments as follows:

Hybrid 0:

1. Setup: CH runs $\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}$, flips a bit $b \in \{0, 1\}$ and makes two identical initialized ledgers L_b, L_{1-b}. CH sends pp to $\mathcal{A}$, which include $[\mathsf{pk}_{\mathsf{B}_j}, C_{0j}, Q_{0j}]_{j=1}^n$ and the two ledgers L_b, L_{1-b}.
2. Query: $\mathcal{A}$ for ledgers L_b, L_{1-b} makes queries $\mathcal{O}_{tx}, \mathcal{O}_{prn}$ where each oracle has two separate instantiations uniquely associated with a ledger. CH answers the queries as in Appendix B.2, verifying transactions or prune operations and maintaining the ledgers L_b, L_{1-b} accordingly. $\mathcal{A}$ is restricted to make simultaneous oracle queries of the same type on the ledgers.
3. Output: $\mathcal{A}$ outputs a bit b' and wins if $b = b'$.

Hybrid 1: Same as Hybrid 0 but now CH when answering queries to $\mathcal{O}_{tx}$, simulates the ZK proofs π_{ik} in each $\mathsf{tx}_i = [C_{ik}, \mathsf{cm}_{\mathsf{ik}}, \pi_{ik}, Q_{ik}]_{k=1}^n$.

Hybrid 2: Same as Hybrid 1 but now CH when answering queries to $\mathcal{O}_{tx}$ replaces ciphertexts C_{ik} and running totals Q_{ik} with random strings.

Hybrid 3: Same as Hybrid 2 but now CH when answering queries to $\mathcal{O}_{tx}$ replaces commitments $\mathsf{cm_{ik}}$ with random strings.

Corollary 1 *Hybrids 0 and 1 are indistinguishable.*

Proof. Immediately follows from the zero-knowledge property of π.

Corollary 2 *Hybrids 1 and 2 are indistinguishable.*

Proof. Immediately follows from the IND-CPA property of ElGamal encryption variant.

Corollary 3 *Hybrids 2 and 3 are indistinguishable.*

Proof. Immediately follows from the hiding property of Pedersen commitments.

C zkLedger Analysis

C.1 zkLedger Description

Here we describe the zkLedger protocol [51], which MiniLedger is inspired from. It consists of n Banks that are transacting with each other, as shown in Table 5. Transactions tx_i, are appended as rows to ledger L. In each tx_i, the sending Bank creates a whole row in L which includes Pedersen commitments $\mathsf{cm_{ij}}$ that hide the transferred value v_{ij} that correspond to each cell (i, j). For instance, if we assume that there's only one receiving Bank in a transaction, the sending Bank would compute a commitment to $-v$ for its own cell, a commitment to v for the receiver cell, and a number of commitments to 0 for the rest of the cells. This makes the transmitted values indistinguishable to any external observer due to Pedersen commitment's hiding property (assuming the sender uses different randomness values for each commitment). Although transaction values are hidden, an external auditor can audit for the sum of the values held by a Bank at any given point (which can be extended to more complex queries such as statistical data on transaction values). zkLedger prevents dishonest Bank behavior through the following ZK proofs:

Proof of Assets π^A In each transaction cell (i, j), π^A_{ij} is an OR proof which ensures that either a) $v_{ij} \geq 0$ and within some valid range (via a range proof) *or* b) if B_j is transferring value v_{ij}, π^A_{ij} ensures that the overall asset balance of B_j, i.e, $\sum_{k=1}^{i} v_{kj}$, remains ≥ 0 and within some valid range (via a range proof), and also B_j authorizes the transfer (by proving knowledge of its private key $\mathsf{sk_j}$). Given that range proofs are the most expensive parts of π^A, zkLedger includes an auxiliary commitment $\mathsf{cm'_{ij}}$[11] in each cell which is either a re-commitment to v_{ij} *or* a commitment to $\sum_{k=1}^{i} v_{kj}$. This makes it possible to only require a single range proof to the committed value in $\mathsf{cm'_{ij}}$ and only use the OR proof to prove knowledge of $\mathsf{sk_j}$ if B_j is the sending Bank.

[11] zkLedger also mentions an additional auxiliary $\mathsf{Token'}$, which is redundant.

	B_1	...	B_j	...	B_n
...					
tx_i			$\mathsf{cm_{ij}} = g^{v_{ij}} h^{r_{ij}}$ $\mathsf{Token_{ij}} = \mathsf{pk_j}^{r_{ij}}$ $\pi^A_{ij}, \pi^C_{ij}, \mathsf{cm'_{ij}}, \mathsf{Token'_{ij}}$		
...					

Table 5: zkLedger construction overview.

Proof of Balance π^B This proof ensures that no values are created or destroyed in a transaction, i.e. the sum of values in a row i must satisfy $\sum_{k=1}^{n} v_{ik} = 0$. To prove this, the sending Bank needs to pick the randomness values r used in the commitments such that $\sum_{k=1}^{n} r_{ik} = 0$ holds. Consequently, the product of all commitments in a row will satisfy $\prod_{k=1}^{n} \mathsf{cm_{ik}} = 1$ which makes up the proof π^B [12].

Audit Functionality and Proof of Consistency π^C As previously discussed, the basic query of an Auditor is the sum of assets v_j for a Bank B_j at any given point. To allow such an audit, each ledger entry (i, j) includes an audit token $\mathsf{Token_{ij}} = \mathsf{pk_j}^{r_{ij}} = (h^{r_{ij}})^{\mathsf{sk_i}}$. When B_j is under audit for its total assets v_j, after ith rows were appended in L, it first reveals the claimed v_j to the auditor (who already has access to L). Then, B_j proves in ZK that $\prod_{k=1}^{i} \mathsf{cm_{kj}}/g^{v_{kj}} = (\prod_{k=1}^{i} \mathsf{Token_{kj}})^{\mathsf{sk_j}}$. The ZK proof hides the secret key of B_j while it proves that if the claimed value for total assets v_j is correct, the above ZK statement holds. We note that the role of $\mathsf{Token_{ij}}$ is essential to "cancel out" the randomness r_{ij} of commitments for auditing, since B_j never learns r_{ij}'s. Thus, the proof of consistency π^C_{ij} shows in ZK that the randomness in each commitment $\mathsf{cm_{ij}}$ is the same with corresponding audit token $\mathsf{Token_{ij}}$. (If B_j knew openings of all commitments, then $\mathsf{Token_{ij}}$ would not be necessary as it would just prove in ZK that it knows a set of v_{ij}'s and $r'_{ij}s$ such that $\prod_{k=1}^{i} \mathsf{cm_{kj}} = g^{v_j} h^{r_j}$ where $r_j = \sum_{k=1}^{i} r_k j$.)

We provide a detailed analysis of the above proofs π^A and π^C below.

C.2 Disjunctive Proof of Equality of Committed Values (π^A)

As discussed above, the sending Bank when constructing a zkLedger transaction tx_i must provide a proof of assets π^A_{ij} (We omit indices i, j in the rest of this section for simplicity). π^A would prove in zero-knowledge that the Bank is either receiving some value (including a value of zero) *or* the value it spends does not result in a negative balance while also proving knowledge of its secret key. This OR composition of ZK proofs is implemented as in [27]. By denoting $\hat{\mathsf{cm}} =$

[12] In zkLedger it is implied that a separate proof of balance is needed for each cell in L. However, it is sufficient for a verifier to check that the commitment's product in the row equals to 1 so the proof π_B does not have to be explicitly included in the ledger.

$\prod_{i=1}^{m} \mathsf{cm_{ij}} = g^{\Sigma v_i} h^{\Sigma r_i} = g^{\hat{v}} h^{\hat{r}}$ and $\hat{\mathsf{Token}} = \prod_{i=1}^{m} \mathsf{Token}_{ij} = \mathsf{pk_j}^{\Sigma r_i} = \mathsf{pk}^{\hat{r}}$, π^A is expressed as $ZKP\{(v, r, v', r', \hat{v}, \mathsf{sk}) : \{[(\mathsf{cm}' = g^v h^{r'} \wedge \mathsf{cm} = g^v h^r) \vee (\mathsf{cm}' = g^{\hat{v}} h^{r'} \wedge \hat{\mathsf{cm}} = g^{\hat{v}} \hat{\mathsf{Token}}^{1/\mathsf{sk}} \wedge \mathsf{Token} = (h^r)^{\mathsf{sk}})] \wedge [\mathsf{cm}' = g^{v'} h^{r'} \wedge v' \in [0, 2^k]]\} 0(\mathsf{cm}, \mathsf{cm}', \hat{\mathsf{cm}}, \hat{\mathsf{Token}}, \mathsf{Token}, g, h, \mathsf{pk})$.

We can break π^A into two separate proofs as $\pi^{A1} \wedge \pi^{A2}$ as follows. In π^{A1}, the prover shows that the value in cm' is the same as in cm (Bank is receiving) or the value in cm' is the same as the sum of commitments in the column $\hat{\mathsf{cm}}$ and it knows sk (Bank is spending). In π^{A2}, the prover shows that the value v' in the auxiliary commitment cm' is positive and within valid range, which also prevents overspending. In [51], the range proof is implemented as described in Confidential Assets [54] using Borromean ring signatures [46], where each bit in the value is represented as a ring signature between zero or one. The details of proof π^{A1} are shown in Figure 9.

Computational costs The prover's total cost for π^{A1} as above is 1 prime-order exps and 4 prime-order multi-exps. For π^{A2} (range proof) is κ prime-order exps and κ prime-order multi-exps. The verifier's total cost for π^{A1} is 6 prime-order exps and 4 prime-order multi-exps. For π^{A2} (range proof) is 2κ prime-order multi-exps.

Storage costs A non-interactive version of π^A would require to store 1 exps and 4 multi-exps, as well as 8 exponent values.

C.3 Proof of Equality of Discrete Logs (π^C)

π^C is made of two identical proofs of consistency π^{C1} and π^{C2} for both cm and cm'_i respectively. Each proof would be as follows:
$ZKP : \{(v, r) : \mathsf{cm} = g^v h^r \wedge \mathsf{Token} = \mathsf{pk}^r\}(\mathsf{cm}, \mathsf{Token}, g, h, \mathsf{pk})$

- P chooses random values q_1, q_2
- P computes $R_1 = g^{q_1} h^{q_2}$ and $R_2 = \mathsf{pk}^{q_2}$ and sends R_1, R_2 to V
- V picks e at random and sends to P
- P computes $z_1 = q_1 + ev$ and $z_2 = q_2 + er$, sends z_1 and z_2 to V
- V checks if $R_1 \mathsf{cm}^e = g^{z_1} h^{z_2}$ and $R_2 \mathsf{Token}^e = \mathsf{pk}^{z_2}$

Completeness: Straightforward to verify.

Special Soundness: The extractor completes the protocol with transcript (R_1, R_2, e, z_1, z_2). Then rewinds to step 2 and gets transcript $(R_1, R_2, e', z'_1, z'_2)$. Now the extractor can compute $\frac{z_1 - z'_1}{e - e'} = \frac{q_1 + ev - q_1 - e'v}{e - e'} = v$ and $\frac{z_2 - z'_2}{e - e'} = \frac{q_2 + er - q_2 - e'r}{e - e'} = r$. This guarantees the equality of the values in R_1 and R_2 because q_1 which contains x is used by the extractor in both X and Y, and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

HVZK: The simulator $\mathcal{S}$ on input of statement $(\mathsf{cm}, \mathsf{Token}, g, h, \mathsf{pk})$ randomly chooses (z_1, z_2, e) and outputs the transcript $(g_1^{z_1} h^{z_2} / g_1^{ve} h^{re}, g_2^{q_1} / g_2^{ve}, e, z_1, z_2)$ which is perfectly indistinguishable from a honestly-executed protocol transcript (R_1, R_2, e, z_1, z_2).

$ZKP\{(v, r, v', r', \hat{v}, \mathsf{sk}) : \{(\mathsf{cm}' = g^v h^{r'} \wedge \mathsf{cm} = g^v h^r) \vee (\mathsf{cm}' = g^{\hat{v}} h^{r'} \wedge \hat{\mathsf{cm}} = g^{\hat{v}} \hat{\mathsf{Token}}^{1/\mathsf{sk}} \wedge \mathsf{Token} = (h^r)^{\mathsf{sk}})\}(\mathsf{cm}, \mathsf{cm}', \hat{\mathsf{cm}}, \hat{\mathsf{Token}}, \mathsf{Token}, g, h, \mathsf{pk})$

Left part of OR proof is True: (running simulator $\mathcal{S}$ for the right part) - witnesses: v, r, r'

- P chooses $\chi, \psi_1, \psi_2, q, s_1, s_2, t, e_2$ at random
- P computes:

$$R_1 = g^q h^{s_1} \qquad R_1' = g^q h^{s_2}$$
$$R_2 = \frac{g^\chi h^{\psi_1}}{g^{e_2 \hat{v}} h^{e_2 r'}} \qquad R_2' = \frac{g^\chi \hat{\mathsf{Token}}^{\psi_2}}{g^{e_2 \hat{v}} \hat{\mathsf{Token}}^{e_2/\mathsf{sk}}}$$
$$R_2'' = \frac{h^t}{h^{e_2 r \mathsf{sk}}}$$

  and sends $R_1, R_1', R_2, R_2', R_2''$ to V.
- V picks e at random and sends to P.
- P computes

$$e_1 = e - e_2 \qquad z_1 = q + v e_1 \qquad z_2 = s_1 + r' e_1$$
$$z_3 = s_2 + r e_1 \qquad z_4 = \chi \qquad z_5 = \psi_1$$
$$z_6 = \psi_2 \qquad z_7 = t$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 (\mathsf{cm}')^{e_1} \qquad g^{z_1} h^{z_3} = R_1' (\mathsf{cm})^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 (\mathsf{cm}')^{e_2} \qquad g^{z_4} \hat{\mathsf{Token}}^{z_6} = R_2' (\hat{\mathsf{cm}})^{e_2}$$
$$h^{z_7} = R_2'' (\mathsf{Token})^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_2, R_2', R_2''), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_2, R_2', R_2''), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, e_2^*))$. Now the extractor can compute $v = \frac{z_1 - z_1^*}{e_1 - e_1^*}$, $r' = \frac{z_2 - z_2^*}{e_1 - e_1^*}$ and $r = \frac{z_3 - z_3^*}{e_1 - e_2^*}$. This guarantees the equality of the v values in cm and cm' because q which contains v is used by the extractor in both R_1 and R_1', and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

Right part of OR proof is True: (running simulator $\mathcal{S}$ for the left part) - witnesses: $\hat{v}, r, r', \mathsf{sk}$

- P chooses $\chi, \psi_1, \psi_2, q, s_1, s_2, t, e_1$ at random
- P computes:

$$R_1 = \frac{g^\chi h^{\psi_1}}{g^{e_1 v} h^{e_1 r'}} \qquad R_1' = \frac{g^\chi h^{\psi_2}}{g^{e_1 v} h^{e_1 r}}$$
$$R_2 = g^q h^{s_1} \qquad R_2' = g^q \hat{\mathsf{Token}}^{1/t}$$
$$R_2'' = h^{s_2 t}$$

  and sends $R_1, R_1', R_2, R_2', R_2''$ to V.
- V picks e at random and sends to P.
- P computes

$$e_2 = e - e_1 \qquad z_1 = \chi \qquad z_2 = \psi_1$$
$$z_3 = \psi_2 \qquad z_4 = q + \hat{v} e_2 \qquad z_5 = s_1 + r' e_2$$
$$z_6 = 1/t + e_2/\mathsf{sk} \qquad z_7 = s_2 t + r \mathsf{sk}\, e_2$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 (\mathsf{cm}')^{e_1} \qquad g^{z_1} h^{z_3} = R_1' (\mathsf{cm})^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 (\mathsf{cm}')^{e_2} \qquad g^{z_4} \hat{\mathsf{Token}}^{z_6} = R_2' (\hat{\mathsf{cm}})^{e_2}$$
$$h^{z_7} = R_2'' (\mathsf{Token})^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_2, R_2', R_2''), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_2, R_2', R_2''), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, e_2^*))$. Now the extractor can compute $\hat{v} = \frac{z_4 - z_4^*}{e_2 - e_2^*}$, $r' = \frac{z_5 - z_5^*}{e_2 - e_2^*}$, $\mathsf{sk} = \frac{e_2 - e_2^*}{z_6 - z_6^*}$ and $r = \frac{z_7 - z_7^*}{\mathsf{sk}(e_2 - e_2^*)}$. This guarantees the equalities of $\hat{v}$ in cm', $\hat{\mathsf{cm}}$ and of sk in $\hat{\mathsf{cm}}$, Token because q which contains $\hat{v}$ is used by the extractor in both R_2 and R_2' and t which contains sk is used by the extractor in both R_2' and R_2'', and because these relations are hard (based on the hardness of the Discrete Logarithm assumption).

HVZK: The simulator $\mathcal{S}$ on input of statement $(\mathsf{cm}, \mathsf{cm}', \hat{\mathsf{cm}}, \hat{\mathsf{Token}}, \mathsf{Token}, g, h, \mathsf{pk})$ randomly chooses $\chi, \psi_1, \psi_2, q, s_1, s_2, t, e_1, e$ and outputs the transcript $((\frac{R_1}{(\mathsf{cm}')^{e-e_2}}, \frac{R_1'}{(\mathsf{cm})^{e-e_2}}, \frac{R_2}{(\mathsf{cm}')^{e_2}}, \frac{R_2'}{(\hat{\mathsf{cm}})^{e_2}}, \frac{R_2''}{(\mathsf{Token})^{e_2}}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2))$ which is perfectly indistinguishable from a honestly-executed protocol transcript $((R_1, R_1', R_2, R_2', R_2''), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, e_2))$.

Fig. 9: zkLedger's disjunctive proof of equality of committed values.

Computational costs The prover's total cost for *each* of π^{C1} and π^{C2} as above is 1 prime-order exps and 1 prime-order multi-exps. The verifier's total cost is 3 prime-order exps and 1 prime-order multi-exps.

Storage costs A non-interactive version of this proof would require to store 1 exps and 1 multi-exps, as well as 2 exponent values.

C.4 Proof of Audit π^{Aud}

During a Bank's audit, the Bank provides a value v to the auditor, along with the a ZKP π^{Aud}, equivalent to the basic MiniLedger audit described in Section 4.1.

The prover's computational cost during an audit is 2 prime-order exps, while the verifier's computational cost is 4 prime-order exps.

C.5 ZK Proofs Cost Analysis

We now provide an outline for the computation and storage costs of zkLedger's ZK proofs, assuming n participating Banks in L. Also since we provide this analysis for comparison purposes with MiniLedger in the next section, we omit the computation and storage costs of the range proof in our overall comparison as these are specific to the range proof construction (range proof costs would be equivalent to MiniLedger as well).

Transaction (prover's) computation cost A Bank when generating a transaction needs to compute the following for each Bank i in the ledger:

- Create π^{A1}: n prime-order exps and $4n$ prime-order multi-exps
- Create π^{C1}: n prime-order exps and n prime-order multi-exps
- Create π^{C2}: n prime-order exps and n prime-order multi-exps

Total costs are $3n$ prime-order exps and $6n$ prime-order multi-exps.

Verifier's computation cost

- π^{A1}: $6n$ prime-order exps and $4n$ prime-order multi-exps
- π^{C1}: $3n$ prime-order exps and n prime-order multi-exps
- π^{C2}: $3n$ prime-order exps and n prime-order multi-exps

This totals to $12n$ prime-order exps and $6n$ prime-order multi-exps.

Auditing computation costs In π^{Aud}, the prover's cost providing a ZKP to the auditor is 2 prime-order exps, while the auditor's cost is 4 prime-order exps.

Storage costs Each zkLedger transaction is associated with the following communication/storage costs for the needed ZK proofs: n proofs of assets π^{A1} (n exps and $4n$ multi-exps elements plus $8n$ exponent values) and $2n$ consistency proofs π^{C} ($2n$ exps and $2n$ multi-exps elements plus $4n$ exponent values). The totals are $3n$ exps, $6n$ multi-exps and $12n$ exponent values.

C.6 Vulnerabilities and Shortcomings

Out-of-band Communication and Unknown Value Attacks zkLedger assumes that a Bank wishing to send some value v to another Bank would inform the receiving Bank accordingly through an out-of-band channel. The problem with this assumption is two-fold. First, a malicious Bank B^* could prevent another Bank B_R in the system from answering audits successfully, by sending some value v^* and either claiming that it sent some other value $v \neq v^*$, or not informing B_R at all. Although there is a mechanism in place to protect B_R against unknown randomness values r (through the audit tokens), no such mechanism exists for the values v themselves. In such a case, B_R, not knowing v^*, would not be able to convince A since it would claim that the hidden value in a commitment cm is v.

Secondly, given the requirement for communicating values v out-of-band, which implies that all Banks need to be online at all times, there is no reason not to communicate the whole commitment opening instead. In this case the above attack would not work since each Bank would verify its respective opening in real time. In case B_R detected any type of malicious behavior from B^* in a transaction, it would alert other Banks or system participants. However this now trivializes zkLedger, as there is no reason for using the audit tokens and proof of consistency π^C. The ledger would consist of just commitments and proof of assets π^A. During audits, the Bank would just prove in Zero Knowledge that the hidden value is indeed v, or simply open the commitment to the auditor.

Degrading performance Besides zkLedger's evident drawback of $O(mn)$ storage costs, a monotonically-growing ledger size also impacts the computation costs for NIZKs. More specifically, π^A shown in detail above requires the verifier to make m multiplications to compute $\hat{\mathsf{cm}} = \prod_{i=1}^{m} \mathsf{cm}_{\mathsf{ij}} = g^{\Sigma v_i} h^{\Sigma r_i}$ for *each* Bank (recall the verifier does not know which Bank is spending). Although multiplication is considered a relatively cheap operation, these costs will eventually add up. An performance degrade will also occur when auditing total assets for the same reason. MiniLedger's "running total" technique would mitigate these issues.

D MiniLedger Zero Knowledge Proof

In MiniLedger we improve on zkLedger's proofs π^A and π^C by combining them as a single disjunctive [27] proof π with proving consistency for ElGamal encryptions in both cases as follows (note this optimization is also applicable to zkLedger).

$ZKP\{(v, v', r, r', r'', \hat{v}, \hat{r}, \mathsf{sk}) : [(\mathsf{cm} = g^v h^{r'} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r) \vee (\mathsf{cm} = g^{\hat{v}} h^{r'} \wedge \hat{c_2} = g^{\hat{v}} \hat{c_1}^{1/\mathsf{sk}} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r \wedge \mathsf{pk} = h^{\mathsf{sk}})] \wedge [\mathsf{cm} = g^{v'} h^{r''} \wedge v' \in [0, 2^k]]\}(\mathsf{cm}, c_1, c_2, \hat{c_1}, \hat{c_2}, g, h, \mathsf{pk})$

The details for the above proof are shown in Figure 10 where we omit the range proof part for notation simplicity as before. Our optimization results in the following total computational and storage costs:

$ZKP\{(v, r, r', \hat{v}, \hat{r}, \mathsf{sk}) : (\mathsf{cm} = g^v h^{r'} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r) \vee (\mathsf{cm} = g^{\hat{v}} h^{r'} \wedge \hat{c_2} = g^{\hat{v}} \hat{c_1}^{1/\mathsf{sk}} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r \wedge \mathsf{pk} = h^{\mathsf{sk}})\}(\mathsf{cm}, c_1, c_2, \hat{c_1}, \hat{c_2}, g, h, \mathsf{pk})$

Left part of OR proof is True: (running simulator $\mathcal{S}$ for the right part) - Witnesses: v, r, r'

- P chooses $\chi_2, \chi_3, \psi_1, \psi_3, \psi_4, q, s_1, s_2, t, e_2$ at random
- P computes:

$$R_1 = g^q h^{s_1} \qquad R_1' = g^q h^{s_2} \qquad R_1'' = \mathsf{pk}^{s_2}$$
$$R_2 = \frac{g^{\chi_2} h^{\psi_1}}{g^{e_2 \hat{v}} h^{e_2 r'}} \qquad R_2' = \frac{g^{\chi_2} \hat{c_1}^{\psi_3}}{g^{e_2 \hat{v}} \hat{c_1}^{e_2/\mathsf{sk}}}$$
$$R_2'' = \frac{g^{\chi_3} h^{\psi_4}}{g^{e_2 v} h^{e_2 r}} \qquad R_2''' = \frac{\mathsf{pk}^{\psi_4}}{\mathsf{pk}^{e_2 r}} \qquad R_2^{(4)} = \frac{h^t}{h^{e_2 \mathsf{sk}}}$$

  and sends $R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}$ to V.
- V picks e at random and sends to P.
- P computes

$$e_1 = e - e_2 \qquad z_1 = q + v e_1$$
$$z_2 = s_1 + r' e_1 \qquad z_3 = s_2 + r e_1$$
$$z_4 = \chi_2 \qquad z_5 = \psi_1 \qquad z_6 = \psi_3$$
$$z_7 = \chi_3 \qquad z_8 = \psi_4 \qquad z_9 = t$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 \mathsf{cm}^{e_1} \quad g^{z_1} h^{z_3} = R_1' c_2^{e_1} \quad \mathsf{pk}^{z_3} = R_1'' c_1^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 \mathsf{cm}^{e_2} \quad g^{z_4} \hat{c_1}^{z_6} = R_2' \hat{c_2}^{e_2} \quad g^{z_7} h^{z_8} = R_2'' c_2^{e_2}$$
$$\mathsf{pk}^{z_8} = R_2''' c_1^{e_2} \qquad h^{z_9} = R_2^{(4)} \mathsf{pk}^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, z_8^*, z_9^*, e_2^*))$. Now the extractor can compute $v = \frac{z_1 - z_1^*}{e_1 - e_1^*}$, $r' = \frac{z_2 - z_2^*}{e_1 - e_1^*}$ and $r = \frac{z_3 - z_3^*}{e_1 - e_1^*}$. This guarantees the equality of values v and r in cm, c_2 and c_1 because q, s_2 which contain v and r respectively are used by the extractor in all R_1, R_1' and R_1'', and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

Right part of OR proof is True: (running simulator $\mathcal{S}$ for the left part) - Witnesses: $\hat{v}, v, r', \mathsf{sk}, r$

- P chooses $\chi_1, \psi_1, \psi_2, q_1, q_2, s_1, s_2, t, e_1$ at random
- P computes:

$$R_1 = \frac{g^{\chi_1} h^{\psi_1}}{g^{e_1 v} h^{e_1 r'}} \qquad R_1' = \frac{g^{\chi_1} h^{\psi_2}}{g^{e_1 v} h^{e_1 r}} \qquad R_1'' = \frac{\mathsf{pk}^{\psi_2}}{\mathsf{pk}^{e_1 r}}$$
$$R_2 = g^{q_1} h^{s_1} \qquad R_2' = g^{q_1} \hat{c_1}^{1/t} \qquad R_2'' = g^{q_2} h^{s_2}$$
$$R_2''' = \mathsf{pk}^{s_2} \qquad R_2^{(4)} = h^t$$

  and sends $R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}$ to V.
- V picks e at random and sends to P.
- P computes

$$e_2 = e - e_1 \qquad z_1 = \chi_1 \qquad z_2 = \psi_1$$
$$z_3 = \psi_2 \qquad z_4 = q_1 + \hat{v} e_2 \qquad z_5 = s_1 + r' e_2$$
$$z_6 = 1/t + e_2/\mathsf{sk} \qquad z_7 = q_2 + v e_2$$
$$z_8 = s_2 + r e_2 \qquad z_9 = t + \mathsf{sk}\, e_2$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 \mathsf{cm}^{e_1} \quad g^{z_1} h^{z_3} = R_1' c_2^{e_1} \quad \mathsf{pk}^{z_3} = R_1'' c_1^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 \mathsf{cm}^{e_2} \quad g^{z_4} \hat{c_1}^{z_6} = R_2' \hat{c_2}^{e_2} \quad g^{z_7} h^{z_8} = R_2'' c_2^{e_2}$$
$$\mathsf{pk}^{z_8} = R_2''' c_1^{e_2} \qquad h^{z_9} = R_2^{(4)} \mathsf{pk}^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, z_8^*, z_9^*, e_2^*))$. Now the extractor can compute $\hat{v} = \frac{z_4 - z_4^*}{e_2 - e_2^*}$, $v = \frac{z_7 - z_7^*}{e_2 - e_2^*}$, $r' = \frac{z_5 - z_5^*}{e_2 - e_2^*}$, $\mathsf{sk} = \frac{e_2 - e_2^*}{z_6 - z_6^*}$ and $r = \frac{z_8 - z_8^*}{e_2 - e_2^*}$. This guarantees the equality of values $\hat{v}, r$ in $\mathsf{cm}, \hat{c_2}$ and c_2, c_1 respectively because q_1 and s_3 which contain $\hat{v}$ and r resepctively are used by the extractor in all R_2, R_2' and R_2'', R_2''', and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

HVZK: The simulator $\mathcal{S}$ on input of statement $(v, r, r', \hat{v}, \hat{r}, \mathsf{sk})$ randomly chooses $\chi_1, \chi_2, \chi_3, \psi_1, \psi_2, \psi_3, \psi_4, t, e_2, e$ and outputs the transcript $((\frac{R_1}{\mathsf{cm}^{e-e_2}}, \frac{R_1'}{c_2^{e-e_2}}, \frac{R_1''}{c_1^{e-e_2}}, \frac{R_2}{\mathsf{cm}^{e_2}}, \frac{R_2'}{\hat{c_2}^{e_2}}, \frac{R_2''}{c_2^{e_2}}, \frac{R_2'''}{c_1^{e_2}}, \frac{R_2^{(4)}}{\mathsf{pk}^{e_2}}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ which is perfectly indistinguishable from a honestly-executed protocol transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$.

Fig. 10: MiniLedger ZK proof π (without range proof)

Transaction (prover's) computation cost The proving costs for π would be $3n$ prime-order exps and $5n$ prime-order multi-exps.

Verifier's computation cost $11n$ prime-order exps and $5n$ prime-order multi-exps

Auditing costs for π^{Aud} Same as in zkLedger.

Storage costs Each MiniLedger transaction is associated with the following communication/storage costs for the needed ZK proofs: $3n$ exps and $5n$ multi-exps elements as well as $10n$ exponent values.

In Table 6 we provide a comparison between the computational and storage costs between zkLedger and MiniLedger for their respective ZK proofs.

	zkLedger			MiniLedger		
	Prime-order exps	Prime-order multi-exps	Exponents	Prime-order exps	Prime-order multi-exps	Exponents
Transaction cost	$3n$	$6n$	-	$3n$	$5n$	-
Verification cost	$12n$	$6n$	-	$11n$	$5n$	-
Storage cost	$3n$	$6n$	$12n$	$3n$	$5n$	$10n$
Auditee cost	2	-	-	2	-	-
Auditor's cost	4	-	-	4	-	-

Table 6: zkLedger and MiniLedger ZK proof costs per transaction

E MiniLedger+ construction and Fine-grained Audit Algorithms

As discussed, we can take advantage of the compactness of MiniLedger to allow for fine-grained auditing in a client level, where Banks are now acting as intermediaries for their client transactions. In this section we provide the detailed construction for MiniLedger+.

Let client pk_{11} associated with Bank B_1 wishes to transfer some of his funds v to a client pk_{21} associated with Bank B_2. The client would then construct a transaction tx as follows:

C1. Construct $id = s \parallel \tau$ where s is a nonce and τ is a timestamp
C2. $C_1 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{11}, v)$
C3. $C_2 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B1}}, v)$
C4. $C_3 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B1}}, \mathsf{B}_2)$
C5. $C_4 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B2}}, \mathsf{pk}_{21})$
C6. Compute ZKP π_1 for $C_4 \wedge f(\mathsf{pk}_{21}) = \mathsf{B}_2$ that proves consistency in c_4 for $\mathsf{B}_2, \mathsf{pk}_{21}$
C7. Compute ZKP π_2 for $C_3 \wedge C_4$ proving consistency for $\mathsf{B}_2, \mathsf{pk}_{\mathsf{B2}}$
C8. Compute ZKP π_3 for $C_1 \wedge C_2$ proving consistency for v

C9. Sign all the previous outputs as σ
C10. Construct $\mathsf{tx}_c : (id, C_1, C_2, C_3, C_4, \pi_1, \pi_2, \pi_3, \sigma)$ and send it to B_1.

B_1 after receiving tx will perform the following actions:

S1. $\mathsf{Vrfy}(\sigma)$
S2. $\mathsf{Vrfy}(\pi_1 \wedge \pi_2 \wedge \pi_3)$
S3. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B1}}, C_2) := v$
S4. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B1}}, C_3) := \mathsf{B}_2$
S5. Verify that $v \geq \Sigma v$ for client pk_{11} (has the assets to transfer)
S6. Append tx in its private table under pk_{11}
S7. Construct a transaction tx_i in L, where C_{i2} (under B_2's column) transfers v to B_2
S8. Compute ZKP π_4, π_5 for C_{i2}, C_2, C_3, proving consistency for v and B_2 (i.e. proving it sent the correct value to the correct Bank)
S9. Assign $id \parallel C_4 \parallel \pi_4 \parallel \pi_5$ to all entries R_{ij} of row i in L
S10. Assign $id \parallel H(\mathsf{tx}_c)$ to all entries H_{ij} of row i in L

At this point, the sending client pk_{11} by observing the new entries H_{ij} can verify that its transaction has been correctly processed by its Bank B_1. Finally the receiving Bank B_2 would perform the following which completes the transaction:

R1. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_{i2}) := v$
R2. Parse R_{i2} and perform $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4) := \mathsf{pk}_{21}$
R3. $C_1' \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{21}, v)$
R4. $C_2' \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v)$
R5. Compute ZKP π_1' for C_4, C_1' proving consistency for pk_{21}
R6. Compute ZKP π_2' for C_{i2}, C_1', C_2' proving consistency for v
R7. Construct $\mathsf{tx}_{c'} : (id', C_1', C_2', \epsilon, \pi_1', \pi_2', \epsilon, \sigma')$ where ϵ are empty strings and σ' is a signature on $\mathsf{tx}_{c'}$ by B_2
R8. Construct an empty transaction in L, where all entries are empty strings ϵ except $H := id' \parallel H(\mathsf{tx}_{c'})$.

Note that only the receiving Bank B_2 will be able to decrypt C_4 and learn the final recipient pk_{21}, as other Bank's decryptions will fail. This ensures that noone except the original sending client learns both the sender and the end recipient of a transaction. Also an external observer still cannot learn the sending or receiving Bank, as the indistinguishability properties of MiniLedger are maintained (which is not true in a similar system, Solidus [21]).

E.1 Assumptions and Threat model

For the above extension we assume the assumptions and threat model described in Section 3 regarding the public ledger L. We now assume however that the auditors behave honestly and do not collude with anyone, as they are the entities safeguarding the scheme's security. Similarly as before, we assume an anonymous broadcast functionality between clients and Banks to preserve anonymity.

	Without aggregation		With aggregation (aggregation factor θ)	
	Prime-order exps	Prime-order multi-exps	Prime-order exps	Prime-order multi-exps
Client cost	6	$6+q$	6	$6+q$
Sending Bank cost	$9+q+4n$	$5+q+7n$	$10+q+4n/\theta$	$6+q+(2+7n)/\theta$
Receiving Bank cost	6	5	$6+1/\theta$	$4+3/\theta$

Table 7: Fine-grained audit extension computation costs overview (normalized aggregation costs).

Otherwise Banks may try to manipulate their private tables and present altered tables to auditors, steal money from their users, make transactions without their permission or make them fail audits. Clients can also assumed to be malicious - they might try to make steal money either from another client or from another Bank in the system, hide their assets, provide false information to audits or try to make a system participant (Bank or client) fail future audits. Banks and/or clients can also freely collude (without however breaking consensus properties).

Finally, while from the Client-Bank relation it is natural for the Bank to know all its clients and their values, the Bank can only learn transactions where their client is a sender or receiver but does not learn the end receiver or the initial sender client respectively.

E.2 Auditing Banks

Although honest behavior of the Banks can only be verified reactively, it can still be held accountable if any of its clients "complain" about loss of funds. Therefore the auditor, either at random or triggered by such a complain, will perform the following protocol with the Bank:

AB1. Auditor sends an audit query to Bank
AB2. The Bank presents the encrypted L_{B_j} to the auditor, as well as a ZKP π_Σ that the sum of sums of all its clients is equal to the Bank's sum in L
AB3. The auditor verifies π_Σ and also verifies that each tx_c in L_{B_j} is signed by its respective client and can be matched with an existing digest $H(\mathsf{tx}_c)$ in L.

E.3 Auditing Clients

An auditor A can also request audits from individual clients, providing that the corresponding Bank presents its encrypted L_{B_j} to A. As in the basic MiniLedger, A can request an audit either for specific client transactions or for the sum of a client's assets in L_{B_j}. The prover can be either the Bank or the client, proving its answers in ZK. The auditing protocol for a client would work as follows:

AC1. For auditing client $\mathsf{pk}_{\mathsf{jm}}$, A makes the request from B_j.
AC2. B_j presents L_{B_j} to A.

AC3. Auditor A fully validates L_{B_j}, similarly as in AB3. above.
AC4. On successful verification (i.e. verify π_Σ and that each tx_c signed and matched with a digest $H(\mathsf{tx}_c)$) , A requests value v from the client (or Bank) (the audit could be on a specific tx, or the total assets Σv of the client represented by the product of the respective ciphertexts)
AC5. Client (or Bank) sends v to A, along with a ZKP that the respective ciphertext encrypts v (or the ciphertexts' product if audited for total assets)

An interesting extension family in the client auditing protocol is a client proving that it was *not* involved in tx_c with value v, or has *not* transacted with another client $\mathsf{pk_r}$. At a high level, the first can be achieved with a Bank accumulating $id \parallel v$ in L_{B_j} along with the needed zero-knowledge proofs, and provide the auditor a non-membership proof of $id \parallel v$. The second audit requires the client accumulating the above ciphertexts C_4 (and similarly with a ZK proof for honest accumulation). To prevent tampering with data, hashes of accumulators and proofs should be regularly posted on the Bank's private ledger (and in turn, hashes to the main ledger). These extensions require a universal accumulator (as the RSA accumulator), since Merkle trees do not provide such functionality.

E.4 Aggregating Transactions

The above protocol requires a transaction to be posted in L for *each client* transaction, which negatively impacts the overall scalability. However the Bank can defer creating the transaction in L by waiting for more transactions of its clients to be posted on its local "mempool". Then the sending Bank can fulfill multiple transaction requests of its clients with a single transaction in L, since that transaction can accomodate multiple recipient Banks.

While aggregating transactions for recipients belonging to different Banks is simple by including multiple entries in R and H, the protocol needs to be modified to accommodate multiple transactions when two or more recipients belong to the same Bank. Similar to the previous example, let two clients pk_{11} and pk_{12} associated with Bank B_1 wishing to transfer some of their funds v_1 and v_2 to clients pk_{21} and pk_{22} respectively, associated with Bank B_2. First, both clients would construct their transactions $\mathsf{tx}^{(1)}, \mathsf{tx}^{(2)}$ as before. B_1 would then work through steps S1 to S6 as above for each client separately. The rest of its steps would be as follows:

S7. Construct a transaction in L, where C_{i2} under B_2's column transfers $v = v_1 + v_2$ to B_2
S8. Compute ZKP π_4, π_5 for $C_{i2}, (C_2^{(1)} + C_2^{(2)}), (C_3^{(1)} \wedge C_3^{(2)})$, proving consistency for v and B_2 (i.e. proves from homomorphic addition of $C_2^{(1)} + C_2^{(2)}$ it sent the correct value, and separately proves with each $C_3^{(1)}, C_3^{(2)}$ that it sent to the correct Bank)
S9. Compute $C_2^{(1')} = \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_1)$ and $C_2^{(2')} = \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_2)$
S10. Compute ZKP π_6 that proves consistency between $(C_2^{(1)} \wedge C_2^{(1')})$ and $(C_2^{(2)} \wedge C_2^{(2')})$ for v_1, v_2 respectively

S11. Assign $id^{(1)} \parallel c_4^{(1)} \parallel \pi_4 \parallel \pi_5 \parallel \pi_6$ and $id^{(2)} \parallel c_4^{(2)} \parallel \pi_4 \parallel \pi_5 \parallel \pi_6$ to R. [13] Note that B_1 cannot "mix and match" since it would fail the audit because of tx in its table.
S12. Assign $id^{(1)} \parallel H(\mathsf{tx}^{(1)}) \parallel id^{(2)} \parallel H(\mathsf{tx}^{(2)})$ to H.

Then Bank B_2 would perform the following steps:

R1. Decrypt v from ciphertext C_{i2} in L
R2. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4^{(1)}) := \mathsf{pk}_{21}$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4^{(2)}) := \mathsf{pk}_{22}$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_2^{(1')}) := v_1$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_2^{(2')}) := v_2$
R3. Compute $C_1^{(1')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{21}, v_1)$ and $C_1^{(2')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{22}, v_2)$
R4. Compute $C_2^{(1')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_1)$ and $C_2^{(2')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_2)$
R5. Compute $\pi_1^{(1')}$ for $C_4^{(1)}, C_1^{(1')}$ proving consistency for pk_{21} & $\pi_1^{(2')}$ for $C_4^{(2)}, C_1^{(2')}$ proving consistency for pk_{22}
R6. Compute $\pi_2^{(1')}$ for $C_1^{(1')}, C_2^{(1')}$ proving consistency for v_1, $\pi_2^{(2')}$ for $C_1^{(2')}, C_2^{(2')}$ proving consistency for v_2 and $c_2^{(k2)}, (C_1^{(1')} \cdot C_2^{(1')}), (C_1^{(2')} \cdot C_2^{(2')})$ proving consistency for v
R7. Construct $\mathsf{tx}^{(1')} : (id^{(1')}, C_1^{(1')}, C_2^{(1')}, \epsilon, \epsilon, \pi_1^{(1')}, \pi_2^{(1')}, \epsilon, \sigma^{(1')})$ where ϵ are empty strings and $\sigma^{(1')}$ is a signature on $\mathsf{tx}^{(1')}$ by B_2 (similarly for $\mathsf{tx}^{(2')}$)
R8. Construct an empty tx in L, setting all entries as ϵ except $H := id^{(1')} \parallel H(\mathsf{tx}^{(1')}) \parallel id^{(2')} \parallel H(\mathsf{tx}^{(2')})$.

Note that transaction aggregations do not intefere with audits either in the Bank or in the client level, since all transactions are still "recorded" in the hashtable.

Another interesting case is when a client requests some monetary value to be transferred to another client belonging to the same Bank, which would constitute an "internal" transaction for that Bank. The Bank's assets in L would not change and a separate transaction in L would not be needed, however the Bank would still post the respective digests in L to ensure correct auditing. Such transactions can of course be aggregated with external transactions in L as discussed above.

E.5 Security Analysis

As in basic MiniLedger, malicious Banks cannot steal, hide or manipulate assets in L. In MiniLedger+ however, a malicious Bank could trivially manipulate its private table (e.g. change its clients' values) since that table is not directly observable by an external verifier. However malicious behavior would eventually be detected in an audit as the private table won't be consistent with the added column data in L. In the case the Bank attempts to move funds

[13] To prevent leaktzage of information that some Bank receives values for two of its clients, the protocol could enforce posting dummy values equal to the maximum number of clients some Bank has in the system .

internally without client authorization (debiting one of its clients and crediting another), even though the balance sum in L_{B_j} would match the balance reflected in L, this unauthorized transaction still wouldn't be matched to an entry in L and would fail the audit, thus exposing the Bank of malicious behavior against its own clients. For the same reason, a Bank cannot "omit" a client transaction in L_{B_j}, as this would also result in a total asset mismatch.

In a client audit, the client or the respective Bank might attempt to provide false answers to the auditor, even by trying to collude. Here the auditor would have verified the validity of L_{B_j} first, so any subsequent audits would be executed on transactions that have been verified to be valid by having matched them with the public ledger L.

E.6 Cost Analysis for MiniLedger+ without Aggregation

Assuming an instantiation with the ElGamal variant encryption scheme, its computation costs are as follows (here we denote $c_1 \to X$ and $c_2 \to Y$):

Client costs A client needs to construct tx which includes the following computation costs: Compute ciphertexts C_1, C_2, C_3, C_4: 4 prime-order exps and 4 prime-order multi-exps. Compute π_1: Assuming mapping function $f()$ for a client public key $\mathsf{pk_C}$ and Bank's j id B_j is derived from verifying a client's signature $\sigma_{\mathsf{sk_C}}(\mathsf{pk_C}, \mathsf{B}_j)$ and assuming B_j has q clients in total, π_1 would be an OR zero-knowledge proof of knowledge of randomness r of ciphertext Y for all q client public keys in the ElGamal variant encryption, which would be q prime-order multi-exps. Compute π_2: Since receiving Bank is known to the sending Bank, π_2 would just consist of a PoK of randomness for ciphertext Y of C_3 and for ciphertext X of C_4, which are 2 prime-order exps. Compute π_3: 2 prime-order multi-exps.

Sending Bank costs A Bank on receiving tx from its client has the following computation costs: Verify π_1: q prime-order exps and q prime-order multi-exps. Verify π_2: 4 prime-order exps. Verify π_3: 2 prime-order exps & 2 prime-order multi-exps. Decrypt for C_2, C_3: 2 prime-order exps. MiniLedger $\mathsf{CreateTx}()$ costs. Compute π_4: 2 prime-order multi-exps.Comp. π_5: 1 prime-order exp & 1 prime-order multi-exp.

Receiving Bank costs A Bank receiving value from a transaction in L has the following computation costs: Decrypt from transaction in L: 1 prime-order exp. Decrypt c_4: 1 prime-order exp. Encrypt C_1', C_2': 2 prime-order exps and 2 prime-order multi-exps. Compute π_1': Proof would be constructed in a similar way to π_2, 2 prime-order exps. Compute π_2': Is a proof of consistency of v for Y ciphertexts of C_1', C_2' and C_{i2} which costs 3 prime-order multi-exps.

E.7 Cost Analysis for MiniLedger+ with Aggregation

Let us now consider the case a sending Bank aggregates θ transactions where all recipients belong to the same receiving Bank B_j (aggregations for recipients

among different Banks have additive costs as in section E.6). For comparison, we outline the *normalized* computation costs per client transaction as follows:

Client costs Same costs as without aggregation.

Sending Bank costs Verify π_1: q prime-order exps and q prime-order multi-exps. Verify π_2: 4 prime-order exps. Verify π_3: 2 prime-order exps and 2 prime-order multi-exps. Decrypt C_2, C_3: 2 prime-order exps. MiniLedger $\mathsf{CreateTx}$ cost divided by θ. Compute π_4: $2/\theta$ prime-order multi-exps (due to homomorphic additive property). Comp. π_5: 1 prime-order exp & 1 prime-order multi-exp. Comp. $C_2^{(i')}$: 1 prime-order exp & 1 prime-order multi-exp. Compute π_6: 2 prime-order multi-exps

Receiving Bank costs Decryption costs from zkLedger transaction divided by θ. Decrypt $C_4^{(i)}$ and $C_2^{(i)}$: 2 prime-order exps. Encrypt $C_1^{(i')}, C_2^{(i')}$: 2 prime-order exps and 2 prime-order multi-exps. Compute $\pi_1^{(i')}$: 2 prime-order exps. Compute $\pi_2^{(i')}$: Is a proof of consistency of v for Y ciphertexts of $C_1^{(i')}, C_2^{(i')}$ and $c_2^{(i2)}, \prod C_1^{(i')}, \prod C_2^{(i')}$ which costs $2 + 3/\theta$ prime-order multi-exps.

F Additional Audit Types and Modifications

F.1 Audit Without Consent

All audit functionalities described in Section 4 are interactive and require the Bank's consent. We could enable non-interactive audits by including an encryption of π^{Aud} and its statement for each transaction cell under a pre-determined trusted auditor's public key (which preserves privacy). However since $\mathsf{AudPruned}\{\}$ is always interactive (and cannot be converted to a non-interactive protocol because it needs the Bank's input from its memory state), audit without consent can only take place on non-pruned data. Because of this inherent limitation, the only possible type of audit without consent for pruned transactions is $\mathsf{AudTotal}\{\}$. Then the statement of NIZK π^{Aud} would be $(Q_j, \sum v_j, \mathsf{pk_j}, g, h)$.

With MiniLedger taking this approach, an alternative direction for ledger "compacting" could be pruning each transaction right away (without consent from Banks) in a similar fashion to CODA [14]. Since MiniLedger is account-based, keeping running totals Q after each transaction execution would be sufficient. The ledger would only consist of a single row of running totals representing each Bank's total assets. After a transaction is broadcasted by a Bank, it would first be checked for validity as before and all running totals would be updated as $Q \cdot C \to Q'$, ensuring optimal $O(n)$ ledger size (of course without Banks needing to keep local memory state).

Finally, to remove the above trusted auditor requirement we can utilize threshold encryption [56], where a coalition of t out of n Auditors could audit any Bank in the ledger, even without its consent. The protocol would now require from a Bank to encrypt the value v_{ij} under the designated threshold

encryption public keys. Then t Auditors will be able to audit any Bank's total assets.

F.2 Additional Audit Types

From our "basic" audit on a value v in L, we can derive several more audit types that reduce to a basic audit. We outine some below, however note this list is not exhaustive. We also note again that these audits can still be executed even for pruned data.

Statistical Audits This audit type category is similar to zkLedger's. This requires a sending Bank committing to a bit flag b in each cell in L, which indicates if that Bank praticipates in that transaction (i.e. $b = 1$ if $v \neq 0$, and $b = 0$ otherwise), accompanied with a NIZK to enforce correctness. "Statistical" audits involve queries that require only non-zero value transaction consideration, as zero-value encryptions would skew the result. For instance, to query the average transacted value for a Bank over a period, the auditor would need to query all the Bank column cells that correspond to that period. The Bank would then reply for all those cells as in the basic audit, with the addition of a reply to the bit flag. Then the auditor after verifying the audit replies, would compute the average value from cells with a bit flag of 1. This category is also applicable to MiniLedger+ in an identical manner.

Value compared to some limit. To query if a Bank sent or received a value less or over an amount t, the audited Bank simply needs to provide a range proof $\pi_r : \{v : (v \geq t)\}$. To preserve correctness, the value v needs to be associated with the hidden value in cm in π (included in Figure 10). As with the basic audit, this proof can be provided either proactively (i.e. posted on L) or reactively (i.e. provided to the auditor during audit).

Limit over time The auditor might want to learn if a Bank's transactions have exceeded a value over a time period (e.g. if Bank has received over $1M over a week). We can create a conjunction of the two audits previously discussed to create such an audit (i.e. audit on average value over time combined with range proof on that average value). To prevent skewing the result in case the Bank has both sent and received values in that period, additional range proofs are needed to prove if a value is positive or negative and included in the overall limit audit. Also note that the notion of "time" in MiniLedger is equivalent to transaction rows, which can include auxiliary timestamp information.

Transaction recipient. The goal of this audit type is for a sending Bank to prove the recipients for one of its transactions. If that transaction has not pruned parts, the Bank can simply reply with the list of receivers and then the auditor would need to audit each Bank in the transaction row to verify this. However it is likely that at least one Bank has pruned its respective cell in that transaction. In this case, the Bank should keep a record of its transaction recipients in its

local memory for each of its *outbound* transactions, and reply to the auditor accordingly. Bit flags used in the Statistical Audits discussed before can also be utilized to make this audit type more efficient.

Client audits in MiniLedger+ To execute audits in a client level, the auditor first needs to fetch the client transactions from the Bank's private ledger, and verify their validity as outlined in Appendix E.3. From that point, the auditor can perform all audits in a client level in a similar fashion to the respective audits in a Bank level. For instance, to learn if some MiniLedger+ client exceeded a value transaction threshold within a time period or over a number of transactions, this audit can be executed by selecting the client's transactions from the Bank's private table that happened within this period by their *id*'s. The audit would then be on the sum of the values represented by the product of the respective ciphertexts, and the client would produce a range proof for that ciphertext product as above. and select those with the appropriate timestamp.

A special useful audit would be to learn if a MiniLedger+ client has sent assets to some specific client pk or not. The transactions would need to be augmented with an additive universal accumulator, with each sender adding the end client recipient's pk to the accumulator, while also providing its Bank a ZK proof of adding the correct public key. During an audit, the client would have to prove membership (or non membership) to the auditor. An important note is that the receiving client *does not* directly learn the original sender of a specific transaction in-band, which implies the above approach cannot work for a client to prove if he has *received* (or not) assets from another client.

G Choosing a Construction for Digest D

In Section 6 we discussed our options for instantiating the accumulator used in pruning. Here we provide a summary of comparisons between possible options in Table 8.

D	$\lvert D\rvert$	$\lvert\mathsf{pp}\rvert$	time (PruneVrfy)	Up-dat-able	Dyn. size	time(Prune)	time(Open) time(BOpen)	$\lvert\pi\rvert$ $\lvert\hat{\pi}\rvert$	time(Verify) time(BVerify)
Merkle Tree	$O(1)$	$O(1)$	$O(q)\mathbb{H}$	✔	✔	$O(q)\mathbb{H}$	$O(\log q)\mathbb{H}$ $O(\ell \log q)\mathbb{H}$	$O(\log q)$ $O(\ell \log q)$	$O(\log q)\mathbb{H}$ $O(\ell \log q)\mathbb{H}$
Catalano-Fiore CDH [20]	$O(1)$	$O(q)$	$O(q)\mathbb{G}$	✔	×	$O(q)\mathbb{G}$	$O(q)\mathbb{G}$ $O(\ell q)\mathbb{G}$	$O(1)$ $O(\ell)$	$O(\lambda)\mathbb{G}$ $O(\ell\lambda)\mathbb{G}$
Lai-Malavolta [42]	$O(1)$	$O(q^2)$	$O(q)\mathbb{G}$	×	×	$O(q)\mathbb{G}$	$O(q)\mathbb{G}$ $O(\ell q)\mathbb{G}$	$O(1)$ $O(1)$	$O(\lambda)\mathbb{G}$ $O(\ell\lambda)\mathbb{G}$
Boneh-Bunz-Fisch VC [13]	$O(1)$	$O(1)$	$O(\lambda)\mathbb{G} +$ $O(kq \log q)\mathbb{F}$	✔		$O(kq \log q)\mathbb{G}$	$O(kq \log q)\mathbb{G}$ $O(kq \log q)\mathbb{G}$	$O(1)$ $O(\lambda)$	$O(\lambda)\mathbb{G} + O(k \log \ell)\mathbb{F}$ $O(\lambda)\mathbb{G} + O(k\ell \log \ell)\mathbb{F}$
Batch-RSA accumulator [13]	$O(1)$	$O(1)$	$O(\lambda)\mathbb{G} +$ $O(q)\mathbb{F} + O(q)\mathbb{H}$	✔	✔	$O(\lambda)\mathbb{G} +$ $O(q)\mathbb{F} + O(q)\mathbb{H}$	$O(q)\mathbb{F}$ $O(q)\mathbb{F}$	$O(1)$ $O(1)$	$O(q)\mathbb{F}$ $O(q)\mathbb{F}$

Table 8: Data structure D comparison. q: number of pruned transactions, k: # bits, λ: security parameter, $\mathbb{F}$: group multiplications, $\mathbb{H}$: hash operations, $\mathbb{G}$: group exponentiations. Costs for $\mathsf{Open}(), \pi$ and $\mathsf{Verify}()$ are for a **single** transaction audit, while costs for $\mathsf{BOpen}(), \hat{\pi}$ and $\mathsf{BVerify}()$ are for an ℓ**-batched** transaction audit.