Privacy-Preserving Noninteractive Compliance Audits of Blockchain Ledgers with Zero-Knowledge Proofs

Bertalan Zoltán Péter, Imre Kocsis

Department of Measurement and Information Systems,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics,
Műegyetem rkp. 3, H-1111 Budapest, Hungary
bpeter@edu.bme.hu, kocsis.imre@vik.bme.hu

Abstract: Privacy and auditability have been conflicting design requirements for blockchain-based distributed ledgers since the inception of the field. As purpose-built blockchains with permissioned consensus and client access are developing in a broad and diverse range of industries, a specific form of this dichotomy is emerging: the need to audit the handling of regulated on-ledger financial assets, such as central bank digital currencies, while preserving the privacy and confidentiality of transactions as much as possible. This paper proposes a novel, privacy-preserving, noninteractive-zero-knowledge-proof-based protocol for a blockchain-based distributed ledger, to prove conformance with fundamental compliance requirements to external auditing parties. We present an extendable implementation and demonstrate the practicality of the approach.

Keywords: blockchain; distributed ledger technology; zero-knowledge proofs; ZoKrates; central bank digital currency; audit; compliance

1 Introduction

Blockchain-based distributed ledger technology (DLT) [1] facilitates the creation of shared, distributed, ledger-like databases, the integrity of which is secured by some form of honest majority consensus across the parties operating the system. DLTs, especially smart contracts handling on-ledger financial instruments, are proving transformative in numerous industries [2-4].

Consortial – cross-organizational, and access-wise consortium-limited – blockchain systems are likely to become the subject of audits to ensure compliance with regulatory requirements. This is especially true now that digital forms of *fiat* currencies and fiat-backed assets are expected to appear on the ledgers of such

blockchains soon [5]. However, directly checking ledger contents often violates the privacy (or confidentiality) of the blockchain's users. Existing solutions supporting privacy and confidentiality are often not universal enough (e.g., because they are platform-specific or smart-contract-based), do not provide a straightforward method to specify requirements, or require additional architectural elements.

In this paper, we propose a novel approach[1] that allows the simple definition of blockchain monetary transfer audit requirements as computations in the procedural language of the ZoKrates [7] tool. Subsequently, any party with access to the blockchain can incrementally create zero-knowledge proofs (ZKPs) [8] of ongoing compliance against a periodically published series of block hash commitments that do not reveal any ledger data. The auditor can verify the proofs noninteractively, eliminating the need for the real-time participation of the blockchain's node operators. The auditee simply sends their proof of compliance to the auditor, who may verify it whenever they wish.

Figure 1 provides an overview of the proposed approach. In the *design* phase, audit requirements are algorithmized in ZoKrates. During the following *synthesis* step, the source code is compiled into a low-level representation called a *circuit* (see subsection 3.2 for more details), and the corresponding prover and verifier keys are generated. At the *operation* phase, the auditee generates proofs using their prover key, the circuit, and the necessary private and public inputs. The proofs are subsequently sent to the auditor party for verification, which can be done with the verifier key. Ledger contents are never revealed to the auditor. The auditee makes public commitments about the developing state of the blockchain, which enables the creation of measures against the consortium keeping "two sets of books."

The source code for our prototype implementation has been published on GitHub[2] and is freely available under the Apache 2.0 license.

The rest of this paper is organized as follows. The next section reviews some fundamental concepts our research builds on and compares our work with similar approaches. Then, in Protocol Design, we present our privacy-preserving audit design and prototype implementation. Finally, we discuss the performance of the prototype and the associated costs in section 4, followed by our conclusions in section 5.

[1] An initial version of the approach was presented at the faculty-level 2021 Scientific Student Competition of BME VIK and summarized in a PhD Minisymposium paper [6]. However, in comparison to this paper, that work used an *interactive* ZKP scheme, and the showcased implementation was only an early prototype.

[2] GitHub repository: `ftsrg/zkp_audit_zokrates`

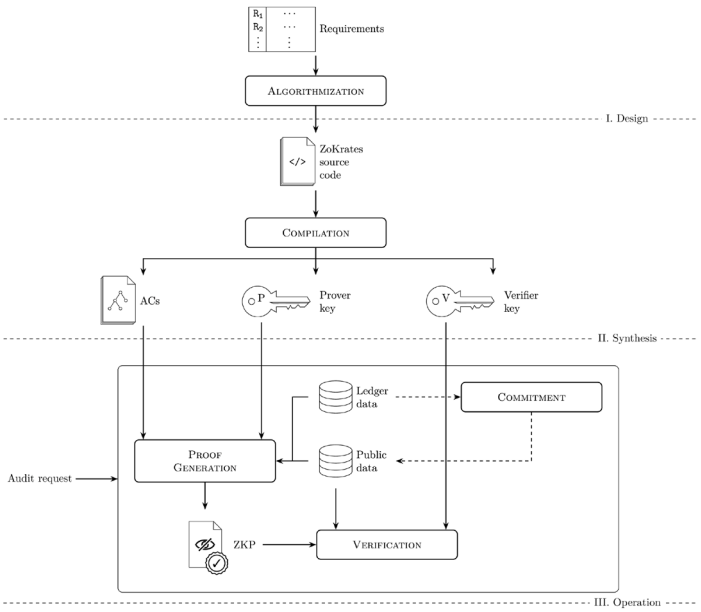

Figure 1
Overview of the process

2 Background

In this section, we introduce zero-knowledge proofs, the core technology that supports our proposal, followed by an elaboration on the relevance of this approach to central bank digital currency (CBDC) and bridging such systems with blockchains. We also compare our solution to similar works found in the literature.

2.1 Zero-Knowledge Proofs

Zero-knowledge proofs [8], or ZKPs for short, form a relatively new area of mathematics, which rely on cryptography to provide means to prove that a statement is true or that the prover knows a certain value that fits a set of criteria without revealing any information other than the proof (such as the fitting value) itself. For example, in self-sovereign identity applications, ZKPs facilitate one to present a so-called verifiable credential (like an ID card), proving that they are over the age of majority without revealing exactly how old they are [10]. In our context, we use ZKPs to prove compliance to requirements like “no transaction with a forbidden recipient address exists on the ledger” – without disclosing any ledger data.

The core idea originates from Goldwasser et al., who introduced *interactive zero-knowledge proofs* in 1989 [11]. Over time, several extensions have been developed, such as *noninteractive* proofs, *proofs of knowledge,* and *arguments of knowledge.* A zero-knowledge (ZK) proof of knowledge means that not only the *existence* of a witness (i.e., fitting value) is proven, but the prover *knows* a witness to the proof [12]. *Arguments* and *proofs* of knowledge are different (albeit sometimes used interchangeably in the literature) because arguments permit "proofs" of false statements that are computationally infeasible to find. In other words, ZK arguments are like ZK proofs but with *computational* soundness rather than *statistical* (see [13]).

Today, state-of-the-art ZKPs commonly found in literature and used in software are *succinct arguments* of knowledge, such as zk-STARKs (Zero-Knowledge Succinct Transparent ARguments of Knowledge) [14] and zk-SNARKs (Zero-Knowledge Succinct Noninteractive ARguments of Knowledge) [15]. *Succinctness* refers to the small size and easy verifiability of the generated proofs. *Transparency* means no trusted setup is necessary. ZKPs are used in numerous blockchain-related projects [16], such as:

- Zerocoin [17] and Zerocash [18] (and Zcash [19], its implementation), extending Bitcoin [20] with ZKP-based privacy
- fabZK [21], offering privacy-preserving and auditable smart contracts for Hyperledger Fabric [22]
- StarkNet[3] and zkSync[4], "ZK-Rollups" for Ethereum [23]
- Mina [24], an extremely lightweight cryptocurrency platform with ZK-powered smart contracts

2.2 Relevance to Central Bank Digital Currency Bridging

In recent years, central bank digital currency (CDBC), "*an electronic, fiat liability of a central bank that can be used to settle payments or as a store of value"* [25], has been a subject of active research worldwide [26]; in a survey done in 2020 by the Bank for International Settlements, the majority of central banks (CBs) around the world expressed that they are at least *exploring* CBDC [25]. At this point, there are no widely used production implementations yet, but several proof-of-concept and pilot deployments have been created[5].

CDBCs promise significantly decreased transaction processing time compared to many classic payment solutions (due to the potential elimination of intermediaries

[3] www.starknet.io (accessed on 2024-02-13)
[4] zksync.io (accessed on 2024-02-13)
[5] See, e.g., the Central Bank Digital Currency Tracker of the Atlantic Council: www.atlanticcouncil.org/cbdctracker (accessed on 2023-05-10)

involved), low transaction costs, and the possibility of *programmability* and the use of *legal tender* in smart contracts. Payment via cryptocurrencies has always been hindering the utilization of the true potential of smart contracts in various industries, and existing stablecoins [27] notwithstanding, the use of smart contracts in established industries needs proper legal tender in cases when financial transactions are also involved. The coming EU-regulated e-money [28] assets will be a part of the solution, but equally, we expect CBDCs to play a major role.

While the possibility of issuing CBDC on a distributed ledger – instead of a classic, centralized system – is being explored, recent developments indicate that most likely, (a) the authoritative CBDC ledger will host, at most, a very limited and controlled set of smart contracts, serving predominantly governmental purposes; and (b) directly or indirectly issuing CBDC to open, permissionless blockchains will not be pursued soon.

These hypotheses flow from recent developments in CBDC exploration. Project Bakong[6], e-Naira[7], Project Rosalind[8], and OpenCBDC [29] either do not or do not intend to support more than a minimal and controlled set of pre-approved smart contracts. E-krona[9] and e-hryvnia[10] plan smart contract support only in a later phase. We do not know of any specific platform design that supports the direct installation of arbitrary smart contracts.

Significant technical and policy arguments support detaching the authoritative CBDC ledger function from the function of CBDC-handling smart contracts. On the technical side, design for latency and throughput can be greatly complicated by smart contracts and contract usage profiles, which are "unknown" at system design time, especially for high-performance, cross-organizational blockchains [30] [31].

Additionally, with wide-scale smart-contract-based programmability, security concerns arise, which carry a significant (and, arguably, unnecessary) level of risk for the authoritative CBDC ledger [32-34]. Maintaining credible arguments of privacy and confidentiality for the ledger also becomes a matter of concern.

On the policy side, research has already mapped out the approaches for supplying smart contracts in DLTs with various forms of fiat-denominated (digital) money [5]. A critical insight is that "bringing money to" the smart contracts of a purpose-built DLT through techniques like bridging (introduced in the next section) is not only one of the viable options but is also far closer to actual rollout in practice than production CBDC platforms themselves, as for these techniques, CBDC as the source of money supply is only one of the options to enable DLT-internal settlement. (Such DLTs are beginning to appear in a regulated manner; see, e.g., the Kate Coin

6 bakong.nbc.gov.kh (accessed on 2024-02-13)
7 enaira.gov.ng (accessed on 2024-02-13)
8 www.bis.org/about/bisih/topics/cbdc/rosalind.htm (accessed on 2024-02-13)
9 www.riksbank.se/en-gb/payments--cash/e-krona/ (accessed on 2024-02-13)
10 bank.gov.ua/en/payments/e-hryvnia (accessed on 2024-02-13)

platform[11].) Thus, there seems to be little incentive for CBs to take up the risk of widely supporting direct CBDC-ledger programmability when the needs of the private sector can be supported in these established ways. From a more general policy perspective, detaching the authoritative CBDC ledger from the smart contract layer is also more in line with the two-layer (i.e., financial institutions in intermediary roles) CBDC approaches we see in the emerging platforms [35].

2.2.1 CBDC Bridging

Due to the reasons detailed above, a reasonable scenario for CBs to support CBDC programmability without uncontrolled smart contract installation on their CBDC ledgers is by *bridging* CBDC to an attached, *permissioned* blockchain (whose operators can be held accountable): a *sidechain.* In essence, this means allowing the locking of a certain amount of CBDC on the CB's ledger and *minting* (i.e., creating) an equivalent value of funds on the sidechain ledger. Note that bridging as a concept originates from the cryptocurrency world[12] but is directly applicable to CBDC-to-permissioned-distributed-ledger scenarios, too. For a recent survey and gap analysis of distributed ledger integration and interoperability, see [37]. Bridging CBDC to a consortial ledger enables smart-contract-encoded business rules to have irrevocable, atomic, and legally sound monetary side effects upon execution.

The bridged-out funds can be called *shadow CBDC* because they can be used much like the underlying CBDC on the bridged blockchain, in the knowledge that they are backed by CBDC on the original ledger. Shadow CBDC can be converted back to the original by *burning* (i.e., destroying) it on the sidechain and unlocking equivalent CBDC units on the CBDC ledger. Figure 2 illustrates CBDC bridging.

However, bridging also causes the locked CBDC to leave the CB's immediate supervision as transactions happen on the sidechain rather than through the CB. One way for the CB to ensure compliance is to audit the transactions on the sidechain.

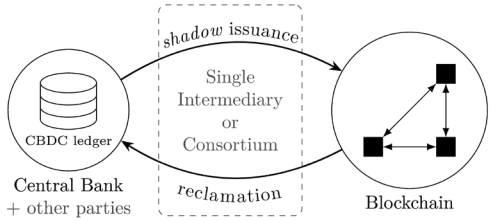

Figure 2
CBDC bridging [6]

11 newsroom.kbc.com/kbc-creates-a-first-in-europe-with-the-kate-coin-its-own-digital-coin-based-on-blockchain (accessed on 2024-02-13)

12 See, e.g., the 2-way peg protocol [36]: its core idea is to lock funds or assets on the main ledger and create equivalent funds or assets on the bridged ledger. The reverse of the process can be later performed to "convert" assets back to the main ledger.

2.2.2 Auditing Financial IT Systems

Auditing financial information technology (IT) systems is a well-established practice and one of the most effective tools for fraud detection [38]. Key audit types include those related to bookkeeping and accounting and compliance audits that ensure specific laws and regulations are met. The former verifies that the financial statements published by the auditee are sound, complete, and accurate by performing various tests and reviewing financial documents such as invoices. Compliance audits target one or more specific regulations and aim to establish adherence to them [39].

Regulations often embody *anti-money laundering* (AML) and *combating the financing of terrorism* (CFT) efforts. For instance, the 26/2020 (VIII 25) regulation of MNB (the central bank of Hungary) [40], in effect since 2021, describes how a financial screening system should be implemented to comply with the measures accepted by the European Union and the United Nations Security Council regarding AML/CFT.

As paper-based accounting is not common practice anymore and IT systems are used instead, there is a strong potential to automate the audit process. Artificial intelligence approaches have been recognized to be applicable [39], and "continuous auditing" [41] has been proposed using blockchain-based solutions [42].

Analyzing the records of the auditee to detect suspicious transactions, albeit a vital step in AML/CFT efforts [43], poses data privacy and confidentiality challenges. At the very least, in the EU, the auditing party may have to observe the General Data Protection Regulation (GDPR) [44], which imposes strict rules on data privacy[13]. Furthermore, any business ledger, distributed or not, may carry business-confidential information.

The privacy-preserving audit outlined in this paper allows the consortium operating the sidechain to prove to an auditing party that its ledger's transactions conform to regulatory requirements, without revealing any specific transaction data.

2.3 Related Work

Privacy-preserving, blockchain-based payment systems, such as Zcash [19] or Monero [45], have existed for several years, many of which are prepared for external transaction audits in platform-specific ways. Recent academic work has focused on resolving the contradiction between auditability/accountability and privacy. Several proposed approaches use smart contracts to implement a payment system layer above the blockchain network. For example, [46] presents a prototype implementation for efficient, privacy-preserving, and auditable token payments as

[13] The auditor and the regulator can be different parties.

Go chaincode for Hyperledger Fabric [22]. FabZK [21] extends Fabric to make smart contracts auditable without privacy infringement. However, the method requires transaction senders to create ZKPs, tokens, and commitments continuously – the proposal in this paper does not require any alterations to how the participating organizations use the blockchain. Some contributions are not extensions of existing systems but implementations on their own, such as MiniLedger [47] and PPChain [48]. [47] offers a thorough overview and comparison of several protocols and systems for auditable distributed payment systems.

Concerning CBDC-related applications, one conceptual approach to regulatory requirement compliance is to avoid the need for explicit audits altogether by platform guarantees. [49] uses ZKPs in a system similar to Zcash to provide cash-like CBDC while enforcing encumbrances, such as personal holding limits and thresholds on the "speed of money" by design. However, in such schemes, the regulatory party must either have a node in the system or trust the majority of the network operators to adhere to the requirement-enforcing rules of consensus.

Our approach differs from the known prior art in the following respects:

(1) Our approach is largely blockchain/DLT platform agnostic; the audit itself happens off-chain, with ledger data converted to the required format beforehand. Although targeted at consortial/private DLTs, it is also potentially applicable in permissionless settings.
(2) The compliance properties subject to audit are extendable as we utilize the ZoKrates language, which resembles generic procedural languages like JavaScript. The requirements simply need to be algorithmized and the program (circuit) compiled. Over time, audit programs can be created and retired freely. Future work will investigate the possibility of creating a domain-specific language (DSL) on top of ZoKrates to express properties to be audited.
(3) The organizations using the blockchain need not actively participate in the audit process, and creating proofs of compliance needs cooperation only from one blockchain node operator (who can change over time).

zkrpChain [50] and zkLedger [51] are the constructions that align the most with our work regarding goals and share core design ideas. However, neither offers the same flexibility as our approach. zkrpChain is built on smart contracts and is specific to Hyperledger Fabric. zkLedger focuses on auditing *banks,* which are blockchain nodes (not end users), and imposes additional requirements, such as the participants maintaining a so-called *commitment cache* and actively engaging in the audits.

In CBDC bridging, this would mean members of the sidechain consortium using the blockchain would have to contribute to the audits individually. Our protocol is universal: any criteria that can be expressed as a computation in ZoKrates can be verified. No complex interaction is needed from the participants; particularly in bridging scenarios, from the operators of the bridged chain.

This work is a direct continuation of [6], in which we presented an early version of our approach. A significant difference is that while we relied on an interactive zk-STARK protocol (with a prototype using Zilch [52]) in the previous work, here we present a zk-SNARK-based, noninteractive, asynchronous protocol. A corresponding prototype implementation has been created in ZoKrates [7]. Noninteractive proving suits the audit settings we predominantly target much better (see the subsection on relevance to CBDCs). ZoKrates enables a modular and extendable proving scheme (as presented in the next section).

3 Protocol Design

In this section, we define the audit protocol, which allows the auditee to prove that the current or a historical blockchain state adheres to the requirements imposed by the auditor.

The blockchain model in the following specification matches the one in our previous work [6]: in essence, transactions are *(source, destination, amount)* triples[14], and blocks consist of the previous block's header's hash value and a Merkle tree [57] constructed from the transactions in the block. The only extension to this model is the inclusion of an additional *index* value for each transaction in a block, which is to ensure that the Merkle tree's leaves are unique.

There are two basic aspects to consider: (1) how the conformance of a given state is proven and (2) how this proof is committed to the real blockchain state. The latter is crucial to ensure that the blockchain node operator is not secretly generating their proofs based on a different ledger, by "keeping two sets of books"; instead, they generate a proof from *private* and *public* inputs, where the data on-chain is *private* (this is a peculiarity of our use case, and is in contrast to the usual blockchain-related applications of ZKPs, where the on-chain data is usually the *public* input). Therefore, without precautions, the auditee could feed fabricated but otherwise valid data into the audit program and thus present a fake proof. We use cryptographic hashes to prevent this.

3.1 Audit Program Specification

Audit programs are computations that take some private and public inputs and optionally return an output. Such programs are used to express audit requirements. Refer to Figure 3 for a simple overview of audit programs.

[14] Where *source* and *destination* are Ethereum-style addresses (the last 20 bytes of the participant's public key in hexadecimal format; i.e., 40 hexadecimal characters)

The application of zk-SNARKs to specify computations increasingly relies on a series of transformations: e.g., from program logic through arithmetic circuits (ACs) [9] or rank-1 constraint systems (R1CSs) [15] [8] to Quadratic Arithmetic Programs (QAPs), which in turn form satisfaction problems. Some software frameworks offer a higher-level language that is automatically compiled into circuits under the hood. Our previous (interactive-proving) experiments were conducted using Zilch [52], which comes with an object-oriented, Java-like language for this purpose called ZeroJava. ZoKrates, the framework used in the prototype implementation for this paper, has a very generic procedural language as well, also named ZoKrates. Both languages can be used to define requirements with elementary programming knowledge easily.

These programs are considered public information: both the auditor and the auditee know how the computation is defined and any public inputs it takes. Private inputs are known only by the auditee. For example, an audit program that asserts that the recipients of all transactions are whitelisted parties could take a list of transactions as a private input, a whitelist (list of addresses) as a public input and return a Boolean value signifying whether every recipient was on the whitelist. In this trivial case, the audit program might consist of a linear search in the whitelist repeated for every transaction and a Boolean *flag* value returned at the end of the program.

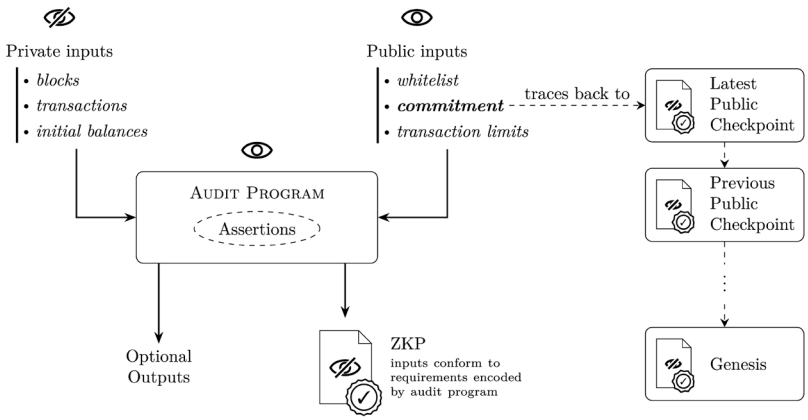

Figure 3
Audit programs and commitments

3.2 Proof Generation

Proofs are generated for a specific blockchain state (i.e., a range of blocks) and some specific criteria expressed in a ZoKrates program. Since it is rather impractical

[15] Simply referred to as *circuits* from now on; transformation between the formats is possible in both directions [7].

(partially due to the resource requirements of generating proofs) to verify every criterion in a single run, it is advisable to develop audit programs in a modular manner and execute them independently, possibly in parallel.

Zero-Knowledge Proof Generation in ZoKrates

ZoKrates [7] is a zk-SNARK [15] toolkit and, therefore, generates ZKPs based on zk-SNARK constructions, first introduced in subsection 2.1. The concrete schemes supported are Groth16 [53], GM17 [54], and Marlin [55]. Although a technical description of the underlying cryptographic primitives is out of scope for this paper, we summarize the proof generation process – see Figure 4 for an overview.

First, the desired computation is encoded in the simple, high-level DSL of ZoKrates. Typically, assertion statements are used to secure the validity of the resulting proofs. For example, a computation ensuring that a given address is eligible to receive funds might involve iterating over a list of allowed addresses and verifying that the given address matches one of the allowed ones in an assert statement. Valid proofs can only be generated by successful computations with no failing assertions.

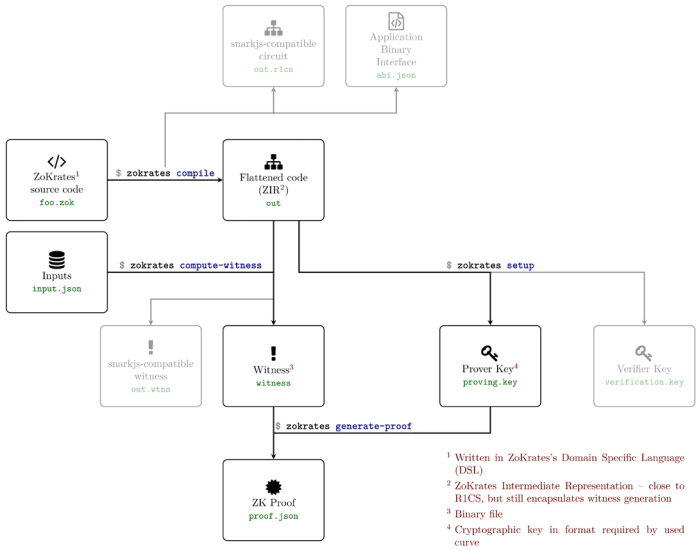

Figure 4
ZoKrates proof generation

Then, ZoKrates *compiles* the high-level code into a so-called *flattened* code: in simplified terms, an R1CS-compatible *circuit*[16]. For more complex programs, this

[16] The zk-SNARK scheme requires quadratic arithmetic programs (QAPs) [57] under the hood, but there are mappings from both ACs and R1CSs to QAPs [7]. QAPs implement the same logic but using polynomials.

can take a significant amount of time and consume substantial storage (in the magnitude of gigabytes). Based on the generated circuit, *prover* and *verifier keys* are generated (in a phase called *setup*). These constitute cryptographic material required to generate and verify proofs, respectively.

Based on the flattened code and the inputs to the computation (public and private), a *witness* can be created – computing a witness by executing the computation is equivalent to finding an assignment of variables that satisfies the constraints in the circuit [7]. Together with the *proving key,* a *proof* can be generated for the *witness.* To verify the proof later, the circuit and all public inputs are necessary.

3.3 Commitment to Real Blockchain State

There are fundamentally two ways to ensure that the prover has fed private input originating from the actual ledger into the program(s). We have chosen the second approach because it does not require architectural changes on the blockchain side.

3.3.1 Blockchain Node Managed by the Auditor

Some blockchain systems may allow the creation of *lightweight* nodes, which behave mostly the same as regular nodes, but instead of synchronizing entire blocks, they only consider block *headers.* This way, they cannot access concrete transaction details but only hashes and metadata such as the block height.

If, as part of the proof, the auditee shows that the root of the Merkle tree [57] built from the transactions in their private input (i.e., the supposed factual transaction data) is indeed the same as the one found in the actual block headers, the auditor can be confident that the probability of the proof being fake is insignificant.

A clear downside of this method is that the auditor must have its own node in the blockchain, and not every major blockchain implementation even offers such light nodes. For example, Hyperledger Fabric [22] did not have such a feature at the time of writing. Ethereum [23] does have light nodes, but practical applications of the process presented in this paper likely call for private, permissioned systems.

3.3.2 Commitment via Hash Traces and Checkpoints

If the auditor knows the contents of the *genesis* (the very first) block of the bridged chain, it is possible for them to trace back consecutive audit results to that block. For example, to audit the first block immediately following the genesis block, the audit program could take the hash of the genesis block's header as a public parameter and prove during the computation that the header of the audited block contains this hash value.

Of course, this does not guarantee that the transactions within this first block are, in fact, the transactions that are on the blockchain. It is technically possible for the auditee to maintain a separate phony blockchain with valid transactions that can be traced back to the commonly known genesis block through hash values. Therefore, the auditee must create a checkpoint at certain time or block-height-based intervals. This means verifying that a given block has a certain hash value in its header. The transactions were not faked if this hash matches the value reported in audits.

3.4 Complete Audit Process

Prior to the audit, the auditee converts the data on the blockchain to a format accepted by the audit protocol. This can happen continuously or in an ad-hoc manner when an audit is due. At this point, the compiled audit programs should be available to the auditee. They generate a ZKP for every audit program and thus prove the satisfaction of all requirements. The result is a set of files, which are then transferred to the auditor, who can use the original source code of the programs and the public inputs to verify these proofs.

3.5 Auditing a Representative Set of Requirements

As a prototype implementation, we have created three audit programs that together verify the following five simple requirements:

(1) For every transaction, the sender account has a sufficient balance to perform the transaction.
(2) The recipient of every transaction is whitelisted.
(3) After each transaction, the balance of the *sender's* account equals their balance immediately before the transaction *minus* the amount of funds transferred.
(4) After each transaction, the balance of the *recipient's* account equals their balance immediately before the transaction *plus* the amount of funds transferred.
(5) The hash value found in every block's header equals the root of the Merkle tree formed by the transactions in the previous block (except, of course, for the genesis block).

These form a representative, basic set of essential ledger data requirements, but it is trivial to include additional requirements. For example, requiring that no transaction's value exceeds a certain threshold could enforce cash-like encumbrances on the blockchain.

We have implemented the verification of these five requirements in three named audit programs:

- ***balances*** checks requirements (1), (3), and (4)

- ***whitelist*** checks requirement (2)
- ***merkle*** checks requirement (5)

The programs could take different input parameters (derived from the same ledger contents) since not all blockchain data is required for each computation. For example, to prove that transaction recipients are on the whitelist, merely a set of transactions is required. On the other hand, *balances,* the program that proves the valid change of account balances throughout several transactions, needs a consecutive list of those transactions and the initial account balances.

Figure 5 shows a simplified overview of how the above five requirements can be audited using the three programs. Note that the three proofs can be generated simultaneously from the same input data.

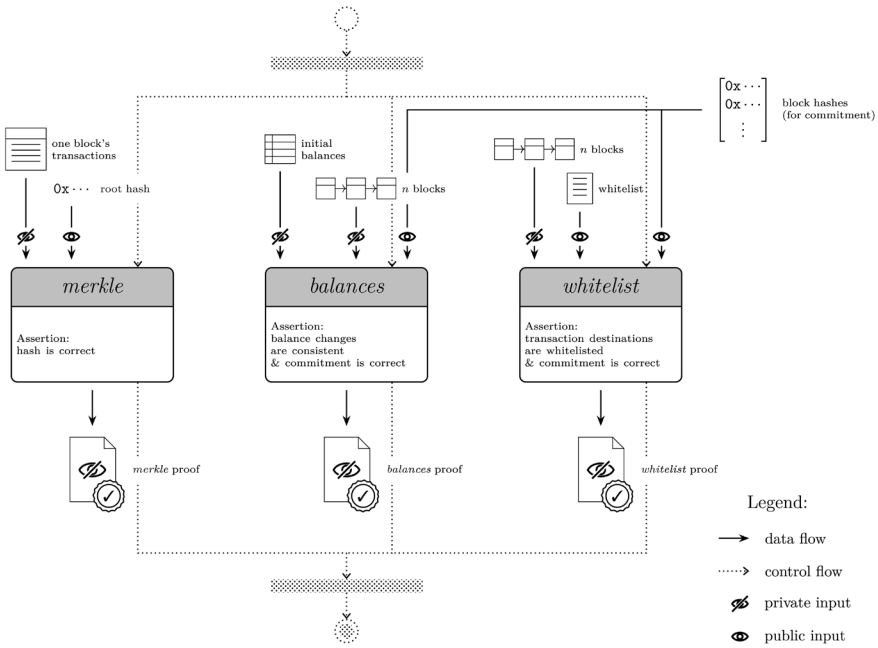

Figure 5
The three prototype audit programs

3.5.1 *balances:* verify the correct change of balances over time

The *balances* program takes as its input parameters a list of initial account balances, a list of blocks (both are private), and a commitment in the form of a list of transaction hashes (public). It then iterates over all transactions in all blocks, validating that the source account's balance is enough to cover the transaction and subsequently updating the accounts' last known balances. At each iteration, it also verifies that the transaction's hash equals the corresponding commitment hash.

3.5.2 *whitelist:* verify that every transaction's recipient is whitelisted

This program receives a list of blocks only known by the auditee and a public list of whitelisted accounts, plus the same commitment hash list as *balances.* It then asserts that for every transaction in every block, the recipient address can be found within the whitelist via a linear search. Like *balance*s*,* it compares each transaction's hash to the commitment values. It is theoretically possible to audit the contents of an entire blockchain ledger using this and the previous program simultaneously, but they are intended to be used for smaller segments with checkpoint states in between.

3.5.3 *merkle:* verify that the block hashes are valid

As the SHA2 [58] hash functions are not optimized for ZKPs, performing as few SHA256 computations within a single audit program as possible is desirable. In our implementation, the program can verify the correct hash value of the list of transactions of a single block. Upon receiving a (constant size) transaction array and a hash, it builds a Merkle tree from the transactions and compares its root to the input hash value, asserting that they are equal. With this program, a separate proof must be generated for each consecutive pair of blocks, but the process can be parallelized.

4 Performance & Cost Considerations

One of the problems with zk-SNARK [15] proofs today is that proof generation is very resource-intensive and may take a long time. In our experience, compiling ZoKrates source code is by no means an easy task for the computer either. For more complex programs (such as the one that computes unoptimized cryptographic hashes), exceedingly large memory and storage space was necessary. For eight transactions, building a Merkle tree [57] demanded somewhat more than 64 gigabytes of memory and 32 gigabytes of storage space for compilation. Hopefully, with the development of hashing algorithms more suitable for ZKPs, such hash computations will become more performant.

Nonetheless, the per-transaction costs are quite promising. The minimal Amazon EC2 virtual private server instance that we found was necessary to compile the source code and compute all three proofs was `r6a.4xlarge`, which – at the time the measurements were taken – cost around $1.09 per hour. In total, it took around 1.5 hours to generate the proofs, meaning that the per-transaction proving cost was approximately $0.016 (USD). The synthesis step, which includes compiling the code and generating the proving and verification keys, takes considerable time, but does not contribute to the operation cost, since it only has to be done once.

For our initial prototype, this cost result is encouraging; although not yet apt for microtransactions, it is well in the realm of practical feasibility for systems handling typically higher-value transactions (the cost is at the order of magnitude of normal bank transactions in many jurisdictions). Also, as we work with ZoKrates, a front-end to three different zk-SNARK schemes even today, we expect that with the rapid development of the ZKP field, our programs can become more resource-efficient without significant modifications. Finally, we chose SHA256 [58] as it is a very strong "default" hashing algorithm in the blockchain space; future research will evaluate the applicability of ZKP-friendly hashing algorithms, such as Pedersen hashes [19].

Figures 6 and 7 show the time and storage space required for the different phases of the three audit programs. The data was obtained on the aforementioned `r6a.4xlarge` AWS instance with an input of ten blocks, each containing ten transactions. The *balances* program took the longest to generate a proof at over four minutes, followed closely by the *merkle* program. The program that asserts whitelist membership was slightly faster. Due to the complexity of SHA256 computation, the *merkle* program took the longest to compile and consumed the most storage space by far. Since the zk-SNARKs generated by ZoKrates are succinct, the proof size is negligible, but the witness generated for the proof has a significant size (in the magnitude of hundreds of megabytes). It should be noted that the measured operation times for *merkle* apply to a single block of the input data, so the total time this program requires to prove conformance of the 10-block input is ten times the value seen on the bar chart.

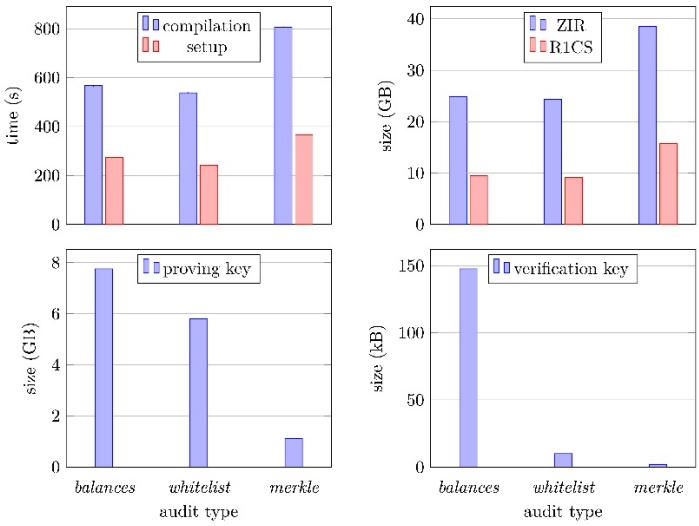

Figure 6
Time and space requirements of the prototype audit programs' synthesis

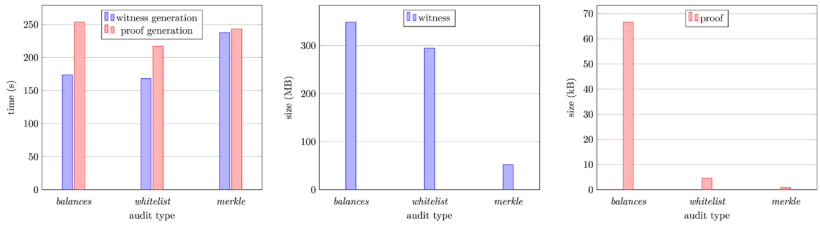

Figure 7
Time and space requirements of the prototype audit programs' operation

Conclusions

In the world of private blockchain systems and especially within the context of bridged CBDC, we expect the need for ledgers to undergo audits, in order to ensure compliance with regulations. However, this would often unnecessarily expose potentially sensitive on-ledger data to the auditors. With our proposed approach, harnessing the power of zero-knowledge proofs, it is possible for an auditee to prove compliance with predefined requirements without revealing sensitive information.

In the paper, we have presented a design for a privacy-preserving noninteractive audit protocol and a prototype implementation using ZoKrates. The prototype's performance is promising, but we also see several applicable optimizations. The proposed design has applications in cross-organizational distributed ledger systems and can also be extended to public blockchains.

Acknowledgements

This work was partially created under, and financed through, the Cooperation Agreement between the Hungarian National Bank (MNB) and the Budapest University of Technology and Economics (BME) in the Digitisation, artificial intelligence and data age workgroup.

The project supported by the Doctoral Excellence Fellowship Programme (DCEP) is funded by the National Research Development and Innovation Fund of the Ministry of Culture and Innovation and the Budapest University of Technology and Economics under a grant agreement with the National Research, Development and Innovation Office.

References

[1] M. Rauchs et al., "Distributed Ledger Technology Systems: A Conceptual Framework." 2018

[2] J. Al-Jaroodi and N. Mohamed, "Blockchain in Industries: A Survey," IEEE Access, Vol. 7, pp. 36500-36515, 2019

[3] Q. Zhu, S. W. Loke, R. Trujillo-Rasua, F. Jiang, and Y. Xiang, "Applications of Distributed Ledger Technologies to the Internet of Things: A Survey," ACM Comput. Surv., Vol. 52, No. 6, pp. 1-34, 2019

[4] T. Alladi, V. Chamola, R. M. Parizi, and K.-K. R. Choo, "Blockchain Applications for Industry 4.0 and Industrial IoT: A Review," IEEE Access, Vol. 7, pp. 176935-176951, 2019

[5] A. Bechtel, A. Ferreira, J. Gross, and P. Sandner, "The Future of Payments in a DLT-Based European Economy: A Roadmap," in The Future of Financial Systems in the Digital Age: Perspectives from Europe and Japan, in Perspectives in Law, Business and Innovation., Singapore: Springer, 2022

[6] B. Z. Péter and I. Kocsis, "ZKP-Based Audit for Blockchain Systems Managing Central Bank Digital Currency," presented at the 29th Minisymposium of the Department of Measurement and Information Systems, 2022, pp. 70-73

[7] J. Eberhardt and S. Tai, "ZoKrates - Scalable Privacy-Preserving Off-Chain Computations," in 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1084-1091

[8] ZKProof, "ZKProof Community Reference," Version 0.3, 2022

[9] H. Vollmer, "Arithmetic Circuits," in Introduction to Circuit Complexity: A Uniform Approach, Berlin, Heidelberg: Springer, 1999

[10] R. Greene, "Self-Sovereign Identity and the Decentralized, Consent-Based Model," Blockchain Law, No. 8, 2022

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof-Systems," in Proceedings of the seventeenth annual ACM symposium on Theory of computing, 1985, pp. 291-304

[12] U. Fiege, A. Fiat, and A. Shamir, "Zero Knowledge Proofs of Identity," in Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987, pp. 210-217

[13] J. Thaler, "Proofs, Arguments, and Zero-Knowledge," SEC, Vol. 4, Nos. 2-4, pp. 117-660, 2022

[14] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, Transparent, and Post-Quantum Secure Computational Integrity." 2018

[15] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture," presented at the 23rd USENIX Security Symposium (USENIX Security 14) 2014, pp. 781-796

[16] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A Survey on Zero-Knowledge Proof in Blockchain," IEEE Network, Vol. 35, No. 4, pp. 198-205, 2021

[17] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," in 2013 IEEE Symposium on Security and Privacy, 2013, pp. 397-411

[18] E. Ben Sasson et al., "Zerocash: Decentralized Anonymous Payments from Bitcoin," in 2014 IEEE Symposium on Security and Privacy, 2014, pp. 459-474

[19] D. E. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification," Version 2023.4.0 [NU5] 2023

[20] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008

[21] H. Kang, T. Dai, N. Jean-Louis, S. Tao, and X. Gu, "FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric," in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) 2019, pp. 543-555

[22] E. Androulaki et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in Proceedings of the Thirteenth EuroSys Conference, 2018, pp. 1-15

[23] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2019

[24] J. Bonneau, I. Meckler, and V. Rao, "Mina: Decentralized Cryptocurrency at Scale," 2020

[25] C. Boar and A. Wehrli, "Ready, Steady, Go? - Results of the Third BIS Survey on Central Bank Digital Currency," Bank for International Settlements, BIS Paper 114, 2021

[26] P. Fáykiss, B. I. Horváth, G. Horváth, N. Kiss-Mihály, Á. Nyikes, and A. Szom-bati, "Transformation of Money in the Digital Age," Polgári Szemle, Vol. 17, No. Spec., pp. 124-140, 2021

[27] I. Fiedler and L. Ante, "Stablecoins," in The Emerald Handbook on Cryptoassets: Investment Opportunities and Challenges, Emerald Publishing Limited, 2023

[28] I. Hallak, "Markets in crypto-assets (MiCA)" 2023

[29] J. Lovejoy et al., "A High Performance Payment Processing System Designed for Central Bank Digital Currencies" 2022

[30] J. A. Chacko, R. Mayer, and H.-A. Jacobsen, "How To Optimize My Blockchain? A Multi-Level Recommendation Approach," Proceedings of the ACM on Management of Data, Vol. 1, No. 1, pp. 1-27, 2023

[31] A. Klenik and A. Pataricza, "Adding Semantics to Measurements: Ontology-Guided, Systematic Performance Analysis," Acta Cybernetica, Vol. 26, No. 2, pp. 175-293, 2023

[32] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential Risks of Hyperledger Fabric Smart Contracts," in 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE) 2019, pp. 1-10

[33] W. Zou et al., "Smart Contract Development: Challenges and Opportunities," IEEE Transactions on Software Engineering, Vol. 47, No. 10, pp. 2084-2106, 2021

[34] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart Contract Security: A Software Lifecycle Perspective," IEEE Access, Vol. 7, pp. 150184-150202, 2019

[35] Bank for International Settlements (BIS) "Lessons learnt on CBDCs," 2023

[36] S. A. Back et al., "Enabling Blockchain Innovations with Pegged Sidechains," 2014

[37] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends," ACM Comput. Surv., Vol. 54, No. 8, pp. 1-41, 2021

[38] A. Seetharaman, M. Senthilvelmurugan, and R. Periyanayagam, "Anatomy of Computer Accounting Frauds," Managerial Auditing Journal, vol. 19, no. 8, pp. 1055–1072, 2004

[39] G. Barta, "Mesterséges intelligencia módszerek alkalmazása az informatikai rendszerek biztonsági auditjában," PhD dissertation, Hungarian University of Agriculture and Life Sciences, 2021

[40] Magyar Nemzeti Bank (the central bank of Hungary), 26/2020 (VIII. 25.) MNB rendelet. 2021

[41] Z. Rezaee, R. Elam, and A. Sharbatoghlie, "Continuous Auditing: The Audit of the Future," Managerial Auditing Journal, Vol. 16, No. 3, pp. 150-158, 2001

[42] H. Han, R. K. Shiwakoti, R. Jarvis, C. Mordi, and D. Botchie, "Accounting and Auditing with Blockchain Technology and Artificial Intelligence: A Literature Review," International Journal of Accounting Information Systems, Vol. 48, p. 100598, 2023

[43] X. Luo, "Suspicious Transaction Detection for Anti-Money Laundering," IJSIA, Vol. 8, No. 2, pp. 157-166, 2014

[44] European Parliament and Council of the European Union, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016, Vol. 119. 2016

[45] K. M. Alonso and J. H. Joancomartí, "Monero - Privacy in the blockchain." 2018

[46] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-Preserving Auditable Token Payments in a

Permissioned Blockchain System," in Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, 2020, pp. 255-267

[47] P. Chatzigiannis and F. Baldimtsi, "MiniLedger: Compact-Sized Anonymous and Auditable Distributed Payments," in Computer Security – ESORICS 2021, 2021, pp. 407-429

[48] C. Lin, D. He, X. Huang, X. Xie, and K.-K. R. Choo, "PPChain: A Privacy-Preserving Permissioned Blockchain Architecture for Cryptocurrency and Other Regulated Applications," IEEE Systems Journal, Vol. 15, No. 3, pp. 4367-4378, 2021

[49] J. Gross, J. Sedlmeir, M. Babel, A. Bechtel, and B. Schellinger, "Designing a Central Bank Digital Currency with Support for Cash-Like Privacy" 2021

[50] S. Xu et al., "zkrpChain: Towards Multi-Party Privacy-Preserving Data Auditing for Consortium Blockchains based on Zero-Knowledge Range Proofs," Future Generation Computer Systems, Vol. 128, pp. 490-504, 2022

[51] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers," presented at the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018, pp. 65-80

[52] D. Mouris and N. G. Tsoutsos, "Zilch: A Framework for Deploying Transparent Zero-Knowledge Proofs," IEEE Transactions on Information Forensics and Security, Vol. 16, pp. 3269-3284, 2021

[53] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in Advances in Cryptology – EUROCRYPT 2016, 2016, pp. 305-326

[54] J. Groth and M. Maller, "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs," in Advances in Cryptology – CRYPTO 2017, 2017, pp. 581-612

[55] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS," in Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2020, pp. 738-768

[56] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic Span Programs and Succinct NIZKs without PCPs," in Advances in Cryptology – EUROCRYPT 2013, 2013, pp. 626-645

[57] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in Advances in Cryptology — CRYPTO '87, 1988, pp. 369-378

[58] National Institute of Standards and Technology, "Secure Hash Standard (SHS)" U.S. Department of Commerce, 2015