ZKP-Based Audit for Blockchain Systems Managing Central Bank Digital Currency

Bertalan Zoltán Péter, Imre Kocsis
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
bpeter@edu.bme.hu, ikocsis@mit.bme.hu

***Abstract*—Central Bank Digital Currency (CBDC) systems are being developed around the world and production solutions can be expected in the near future. Should a central bank allow handling of CBDC on a ledger that is not under its supervision (via platform bridging), it may wish to specify certain conformance requirements regarding the transactions. We propose a novel audit scheme based on Zero-Knowledge Proofs, which allows the operator of the bridged ledger to prove its compliance to such requirements, without revealing details about the transactions (such as the exact participants, the direction of the transfer, or the transferred value). This scheme aims to resolve the conflict between banks having to audit how CBDC is used on the bridged blockchain and consortia trying to keep sensitive data private.**

***Index Terms*—blockchain, audit, central bank digital currency, zero-knowledge proof**

I. Introduction

Central Banks (CBs) around the world are actively researching the possibilities of implementing their own Central Bank Digital Currency (CBDC), with some institutions already testing prototypes and pilots [1]. Considering the popularity and success of cryptocurrencies and blockchains equipped with various smart contracts, we can safely assume that once a production-grade CBDC emerges, there is soon going to be a real demand to make it usable on private as well as public blockchain networks. For example, a consortium might maintain their own blockchain network, where they wish to use CBDC as currency. A large motivating factor is that the CBDC system most likely will not allow the installation of arbitrary smart contracts (or universal programmability of any form), a feature widely used by cryptocurrencies today.

There is already a solution known in the cryptocurrency world for the integration of platforms that do not support smart contracts and those that do support them, in the form of platform *bridging*. Bridging is essentially the process of allowing some units (in our case, CBDC) to be locked on one ledger, while so-called *shadow* or *wrapped* units of equivalent value are created on the other ledger (the *side-chain*). The shadow can later be converted back into the original instrument on the first ledger, possibly by a new owner.

However, such bridged CBDC would still be digital money which has to adhere to certain Know Your Customer / Anti Money Laundering (KYC/AML) requirements established by law. A simple example is requiring CBDC (or its shadow) to only be owned by persons or organizations who are allowed (ie whitelisted) at all times. In other words, digital cash cannot get into the hands of forbidden parties. To prove that these requirements are satisfied, the transactions done with the bridged CBDC will most likely be subject to audits. Unfortunately, the straightforward option of the auditors simply reading the ledger contents is not viable since the information in the ledger may be highly confidential. The bank is not interested in all of the data in the ledger, merely in compliance with the requirements.

In this paper, we present a novel audit scheme for the bridged CBDC scenario, which makes it possible for the CB to verify conformance to arbitrary requirements in such a way that potentially confidential information stored on the blockchain is not revealed. Our specific contributions are the following:

- We establish and formalize a minimal blockchain model for our purposes.
- We define five elementary audit criteria, which must be met by the transactions on the blockchain.
- We define a complete audit scheme, which allows a bridged blockchain to prove conformance to these criteria.

The rest of this paper is organized as follows. We further elaborate on CBDC bridging in the subsection below, followed by comparing our approach to other recent work in section II. Section III introduces our simplified blockchain model. Building on this formalization, we present an audit model and protocol in section IV, which we have implemented as a prototype. Finally, section V summarizes our results and plans for future work.

CBDC Bridging

CBDC is a form of digital fiat money issued by CBs, just like physical banknotes that have existed for hundreds of years. It is similar to other forms of digital money in the sense that it requires maintaining a *ledger* of transactions or at least a registry of current account balances [2]. How this ledger is accessed, structured, and maintained is a design choice.

A blockchain is essentially a data structure consisting of a cryptographically linked list of *blocks* that contain *records.*

The research was supported within the framework of the Cooperation Agreement between the National Bank of Hungary and BME.

DOI 10.3311/MINISY2022-018

While cryptocurrencies traditionally store their ledger in blockchains, CBDC implementations may call for a centralized ledger solution, not a distributed one. Nevertheless, blockchain systems are still relevant in the context of CBDC when we consider the possibility of 'transferring' CBDC to an external blockchain network (sometimes called a *side-chain*) where it may be used similarly to cryptocurrencies.

The usual definition of CBDC does not imply universal programmability by users. In the context of Distributed Ledger Technology (DLT) and blockchains, this means that CBDC implementations will likely not support arbitrary smart contracts. On the other hand, smart contracts are prevalent in today's cryptocurrencies and are drivers of innovation in several areas. However, the fact that they cannot handle digital cash (as opposed to cryptocurrencies or stablecoins) remains largely unaddressed. One way to integrate smart contracts with CBDC is to adopt the existing *bridging* technology for integrating different blockchain platforms. Figure 1 offers a simple architectural overview of such a bridged CBDC scenario.

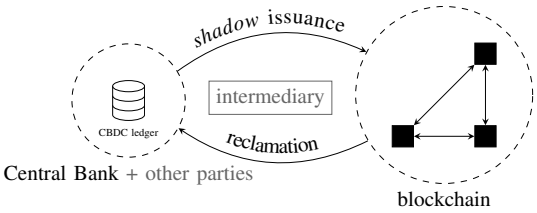


Fig. 1: CBDC bridging

The question is: in this setup, can the CB enforce requirements and conduct periodic or on-demand audits on the consortial blockchain *without* gaining access to sensitive information in the process? In this paper, we focus on an audit model and protocol based on a relatively new family of cryptographic algorithms, Zero-Knowledge Proofs (ZKPs), which essentially allow the bank to verify – or, from another perspective, the consortium to *prove* – that the requirements are satisfied without seeing into any of the transactions, or even their number.

II. Related Work

zkrpChain [3] and *zkLedger* [4] are somewhat similar works, but neither target nor solve the exact same problem. As its name suggests, the former focuses on *range proofs*, which are not readily applicable to the verification of criteria such as being on a whitelist (set membership). The implementation mainly uses smart contracts. On the other hand, our proposal is universal because it allows arbitrary computations to verify the requirements. Thus, anything be expressed as a program (source code) can be verified.

zkLedger requires the auditees to actively participate in the audits and maintain a synchronized *commitment cache*. In the audit scheme outlined in this paper, however, the participating organizations of the side-chain need not actively take part in audits; a single blockchain node suffices.

Additionally, neither of these solutions offers an easily programmable interface to define the audit criteria. We have implemented a prototype in a ZKP system that allows the expression of the requirements as a simple, procedural program (rather than arithmetic circuits, for example).

III. Blockchain and Requirement Model

To rigorously define an audit protocol, we must first establish what we mean by the blockchain to which CBDC is bridged.

A. Blockchain

The blockchain is an infinitely growing sequence of blocks, where each block consists of a block header (denoted by B) and a block body (denoted by T). The header contains data that makes this construction a blockchain, and the body is simply a sequence of transactions organized as a *Merkle tree* [5] which belong to the block. Figure 2 is a visual representation of a segment of the block sequence. We index block headers and bodies as B_i and T_i respectively, denoting the header and body of the i-th block (ie the block at height i).

Thus, the entire blockchain $\mathfrak{B}$ at a given height (ie block count) h can be expressed as a sequence of ordered pairs of B_i and T_i: $\mathfrak{B} = ((B_0, T_0), \ldots, (B_h, T_h))$.

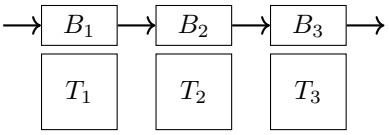
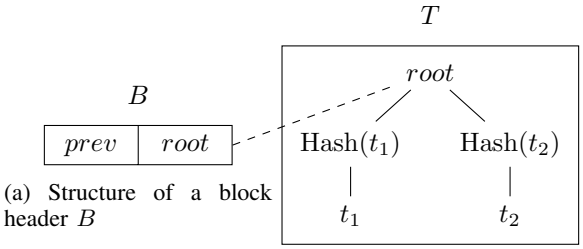


Fig. 2: The block sequence of the blockchain model

1) Blocks: Each block consists of a header and a body. The header is an ordered pair of the *hash* of the previous block's header ($prev$) and the root of the Merkle tree that contains all the transactions in the block ($root$). The body is the sequence of transactions stored in a *Merkle tree*, whose top hash is $root$. Figure 3 illustrates the individual block headers and bodies. The *genesis* block's $prev$ value is nil. We denote the sequence of all blocks in the blockchain $\mathfrak{B}$ by $\mathbb{B}_{\mathfrak{B}}$.



(a) Structure of a block header B

(b) Structure of a block body T

Fig. 3: Visualization of block headers and bodies in the blockchain model

2) Transactions: Transactions (denoted by t) are ordered triples formed by their *source* s, their *receiver* r and the *amount* a of funds transferred.

To further simplify the model, let us assume that each block contains exactly two transactions:

$$\forall (B = (root, prev), T = (t_1, t_2)) \in \mathfrak{B} \\ : root = \mathrm{Hash}\left(\mathrm{Hash}(t_1) + \!\!+ \mathrm{Hash}(t_2)\right)$$

'Hash' denotes an arbitrary collision-resistant hash function and $+\!\!+$ is the concatenation operator (applied to binary values).

The sequence of transactions in a blockchain $\mathfrak{B}$ – denoted by $\mathbb{T}_{\mathfrak{B}}$ – is understood as the sequence of all transactions in all blocks: $\mathbb{T}_{\mathfrak{B}} = \{t \in T : (B, T) \in \mathfrak{B}\}$.

We also define $\mathcal{S}(t)$, $\mathcal{R}(t)$, and $\mathcal{A}(t)$ to denote the sender, receiver, and amount of a transaction t respectively.

3) Accounts: Accounts represent the senders and receivers of transactions. In our model, they are simply identified by integer values: $\mathbb{A} \neq \varnothing \subseteq \mathbb{N}$. Account 0 is the *genesis account*, which always exists. We assume that the set of accounts is constant for the lifetime of the blockchain.

Every account has its *balance*, which is a non-negative integer. When the blockchain is created, the genesis account's balance becomes the *total balance* in the blockchain. In other words, there is a fixed amount of units ('money') in the system at all times.

We denote the set of all accounts in a blockchain $\mathfrak{B}$ by $\mathbb{A}_{\mathfrak{B}}$.

4) Genesis: Genesis refers to the creation of a blockchain. The following two parameters are involved when creating a blockchain $\mathfrak{B}$: the total balance $\$_{\mathfrak{B}}$, and the final set of accounts $\mathbb{A}_{\mathfrak{B}}$.

B. Requirements

We have collected five basic requirements to which blockchain state is expected to conform, verified during an audit protocol. We want to ensure that for all transactions in any given block

(1) the sender has sufficient balance to spend;
(2) the receiver is allowed to receive funds (ie is whitelisted);
(3) the balance of the receiver after the transaction equals their balance before the transaction *plus* the transferred funds;
(4) the balance of the sender after the transaction equals their balance before the transaction *minus* the transferred funds;
(5) the hash of the block header (found in the next block's header) is indeed the hash of the block's header concatenated with the root of the Merkle tree that contains the transactions in the block.

Based on our formalized blockchain model, we can express these requirements succinctly, as seen in Table I.

t^i_j denotes the jth transaction in the ith block B_i. $\mathcal{P}(t)$ is the transaction immediately before t:

$\mathcal{P}(t^i_j) = \begin{cases} t^i_{j-1} & : j > 1 \\ t^{i-1}_N & : \text{otherwise} \end{cases}$, where N is the (constant) number of transactions in each block (defined to be 2 for simplicity). $\mathcal{B} : \mathbb{A} \times \mathbb{T} \to \mathbb{N}$ with $\mathcal{B}(u, t^i_j) =$ the balance of account u *after* transaction t^i_j is a function defined to simplify expressing account balances.

req.	formalization
(1)	$\forall t^i_j \in \mathbb{T} : \mathcal{B}\left(s, \mathcal{P}(t^i_j)\right) \geq \mathcal{A}(t^i_j)$
(2)	$\forall t^i_j \in \mathbb{T} : \mathcal{S}(t^i_j) \in WhiteList$
(3)	$\forall t^i_j \in \mathbb{T} : \mathcal{B}(r, t^i_j) = \mathcal{B}\left(r, \mathcal{P}(t^i_j)\right) + \mathcal{A}(t^i_j)$
(4)	$\forall t^i_j \in \mathbb{T} : \mathcal{B}(s, t^i_j) = \mathcal{B}\left(s, \mathcal{P}(t^i_j)\right) - \mathcal{A}(t^i_j)$
(5)	$\forall B_i = (prev_i, root_i), B_{i-1} = (prev_{i-1}, root_{i-1}) \in \mathbb{B}$ $: prev_i = \mathrm{Hash}(B_{i-1}) = \mathrm{Hash}(prev_{i-1} + \!\!+ root_{i-1}) \quad (i > 0)$

TABLE I: Formalized requirements

$$\mathcal{B}(u, t^i_j) = \sum_{n=0}^{i-1} \left(\sum_{\theta \in T_n : \mathcal{R}(\theta) = u} \mathcal{A}(\theta) - \sum_{\theta \in T_n : \mathcal{S}(\theta) = u} \mathcal{A}(\theta) \right) \\ + \sum_{k=1}^{j} [\mathcal{R}(t^i_k) = u]\, \mathcal{A}(t^i_k) - \sum_{k=1}^{j} [\mathcal{S}(t^i_k) = u]\, \mathcal{A}(t^i_k)$$

where $[P]$ is an *Iverson-bracket* expression, ie $[P]$ is 1 if P is true and 0 otherwise. $WhiteList \subseteq \mathbb{A}$ is a set of allowed transaction senders, which may change over time.

IV. Audit Scheme

During an audit, the CB must be able to verify conformity to the requirements without directly accessing ledger contents. As we established earlier, the latter may not be compatible with the confidentiality requirements of the consortium using the blockchain. Our audit scheme relies on ZKPs to allow the consortium to prove that the ledger state is valid in *zero-knowledge*. In other words, the bank (the auditor) learns nothing more in the process other than that the requirements are satisfied.

ZKPs are cryptographic constructions that allow a Prover to prove the truth of a statement to a Verifier in such a way that no information is revealed in the process other than whether the statement is true. They are relatively new in mathematics and cryptography but already have several applications in cryptocurrencies where privacy is paramount. For example, *Zerocoin* [6] and its successor, *Zerocash* [7], heavily rely on ZKPs.

As shown by [8], it is possible to generate a Non-Interactive ZKP for arbitrary *computations* done on a von Neumann architecture. This essentially means that a program or algorithm itself can be the subject of a Zero-Knowledge Proof. For our purposes, the audit protocol can be expressed as a computation that verifies our requirements in zero-knowledge.

Several ZKP systems and software implementations exist. In our work, we used *Zilch* [9], mainly because it allowed rapid prototyping thanks to its Java-like procedural language in which computations can be expressed. *Zilch* internally relies on Zero-Knowledge Succinct Transparent ARguments of Knowledge (zk-STARKs).

Our audit protocol design is *interactive*, meaning that the CB and the blockchain engage in an online, synchronous exchange of messages, during which the satisfaction of the requirements by the ledger state is proven. The interactive audit flow is visualized in Figure 4.

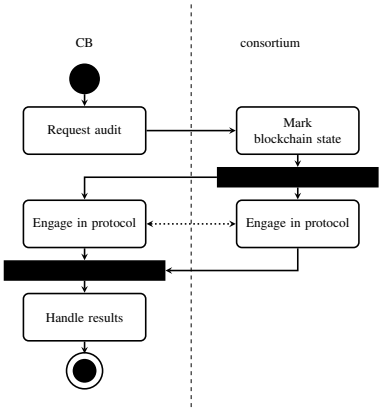


Fig. 4: Interactive audit

A crucial problem to solve is how to ensure that the state which forms the subject of the audit is the actual ledger state, and not data fabricated by a malicious consortium. A simple solution would be for the CB to maintain their own so-called *lightweight* or *header* node on the blockchain, which does not have access to block bodies (ie transactions), only headers. Thus, no information is leaked by the block headers themselves. In such a setup, an export of the marked state would be committed to the blockchain state in the sense that it contains the hashes of all blocks (except perhaps the last one), which are also recomputable for each block from the transactions.

The rest of this section describes an audit process in our model. Let us start with an arbitrary state of the side-chain and an incoming audit request from the CB. The current state is marked: the audit is performed on the data that existed at the time of the audit request. In the meantime, the blockchain can still be operational; blocks can be appended, smart contracts can be executed. From this point, the required steps are the following:

1) If required, we convert the blockchain state into a format that can be fed into the audit algorithm. In this process, we trim all unnecessary information and essentially generate a *view* of blockchain, which is just a sequence of transactions.
2) We open a communications channel between the CB and a node on the blockchain to convey audit information.
3) The two parties engage in an Interactive Zero-Knowledge Protocol, during which the verification algorithm is executed as a computation, and successful termination signifies a successful audit. This algorithm must be previously agreed upon by both parties and is itself public. The concrete arguments of the algorithm are only known by the blockchain node.
4) Whether successful or not, the CB takes note of the event.

It is up to the CB to decide how the results of a non-compliant audit are handled: they may require a more in-depth audit to give the consortium a chance to come clean about the situation at the cost of exposing their private data, or they may simply record the violation.

More information, including the verification protocol as pseudocode and the *Zilch* prototype can be found in [10].

V. Conclusion and Further Work

In our work, we have presented a complete audit model and protocol for a bridged CBDC scenario using ZKPs. We have created a prototype implementation using *Zilch*, which showed promising results during ad-hoc testing with hand-crafted data. After improving on the less refined parts of our implementation, we would like to put it to the test by evaluating its performance on data similar in volume to what is expected on a real consortial network.

An exciting challenge is how bridged CBDC that is processed by smart contracts can be handled. Tackling this problem would take us one step closer to real-life applications. We also plan to consider the transactions' cryptographic signatures, verifying them during audits.

Looking further, the audit protocol outlined in this paper is not specialized to CBDC systems whatsoever. The prospects of applying the methodology in the broader area of cross-organizational integrations are certainly worth looking into.

References

[1] "CBDC tracker," https://cbdctracker.org, accessed: 2022-01-11.
[2] C. Boar, H. Holden, and A. Wadsworth, *Impending arrival - a sequel to the survey on central bank digital currency*, ser. BIS Papers. Bank for International Settlements, 04 2020, no. 107. [Online]. Available: https://ideas.repec.org/b/bis/bisbps/107.html
[3] S. Xu, X. Cai, Y. Zhao, Z. Ren, L. Wu, H. Zhang, L. Du, and Y. Tong, "zkrpchain: Privacy-preserving data auditing for consortium blockchains based on zero-knowledge range proofs," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 656–663.
[4] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 04 2018, pp. 65–80. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/narula
[5] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.
[6] 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.
[7] E. Ben-sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 05 2014, pp. 459–474.
[8] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. USA: USENIX Association, 2014, pp. 781–796.
[9] 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.
[10] B. Z. Péter, "ZKP-based audit for blockchain systems managing central bank digital currency," 2021, *Scientific Student Competition* Report. [Online]. Available: https://tdk.bme.hu/VIK/inform/ZKPalapu-audit-digitalis-jegybankpenzt-kezelo