Proof of Assets in the Diem Blockchain

Panagiotis Chatzigiannis[1] and Konstantinos Chalkias[2]

[1] George Mason University
[2] Novi Financial / Facebook Research

Abstract. A great challenge for distributed payment systems is their compliance with regulations, such as anti-money laundering, insolvency legislation, countering the financing of terrorism and sanctions laws. After Bitcoin's MtGox scandal, one of the most needed auditing functionalities for financial solvency and tax reporting purposes is to prove ownership of blockchain reserves, a process known as Proof of Assets (PoA). This work formalizes the PoA requirements in account-based blockchains, focusing on the unique hierarchical account structure of the Diem blockchain, formerly known as Libra. In particular, we take into account some unique features of the Diem infrastructure to consider different PoA modes by exploring time-stamping edge cases, cold wallets, locked assets, spending-ability delegation and account pruning, among the others. We also propose practical optimizations to the byte-size of PoA in the presence of light clients who cannot run a full node, including skipping *Validator* updates, while still maintaining the 66.67% Byzantine fault tolerance (BFT) guarantee.

Keywords: Diem blockchain · solvency · tax reporting · light clients.

1 Introduction

During the last decade, many distributed payment systems have emerged as an alternative to centralized banking. While blockchains were initially in anarchy, the need for regulation became apparent to ensure their compliance with laws, regulations and tax reporting. One aspect of such requirement is Proof of Assets (PoA), a fundamental part for proving financial solvency on behalf of custodial wallets [21, 23], also known as Virtual Asset Providers (VASPs). Briefly, it is a cryptographic evidence that the organization possesses sufficient assets which, combined with its proved liabilities, offers the so called Proof of Solvency (PoSolv). The need of such proofs became even more apparent after infamous cryptocurrency exchange collapses, such as MtGox [24, 36] or even the recent withdrawals suspension from OKEx [4].

Our work focuses on practical PoA solutions in the Diem blockchain, however parts of our proposal apply to other systems as well. As regulatory compliance, transparency and fund safety are among the top priorities for Diem [12],

Panagiotis Chatzigiannis did part of this work during an internship at Novi Financial/Facebook Research.

PoA should be an important feature to achieve a safer wallet ecosystem. Diem's unique hierarchical account model differs from other blockchains and allows for several different PoA types that are not possible in other platforms. Our goal is to formalize and explore many different types of asset proofs in the Diem blockchain. Additionally, as we will show, Diem's PoA, in combination to Know-Your-Customer (KYC) identity verification, can also be useful to mitigate tax evasion, something that is not straight-forward in other blockchains where one can deny ownership of an address. In Diem, wallet addresses are pinned to particular entities, and thus VASPs cannot hide their owned assets on purpose.

In the following, we provide a summary of related work, a detailed analysis of PoA variants handcrafted to Diem's design, and finally practical recommendations for proof compression, aiming to make it more friendly for light (potentially mobile) clients.

1.1 Related work

Bitstamp's Proof of Reserves [1] was one of the first attempts to provide evidence of a custodial wallet's total assets through an interactive protocol with a third party auditor. The process was to prove account-key ownership by signing over a provided random message; briefly, the ability to sign over a challenge string implies control and ownership of the account(s).

Provisions [23] presented a protocol based on zero-knowledge (ZK) proofs to prove assets, as part of a more general scheme to prove solvency. Its focus was to hide which accounts are owned by the audited entity. Briefly, the organization would form an anonymity set by adding random accounts from the public blockchain to those it already controls, and then prove (in ZK) that it knows a set of private keys that add up to or exceed some amount. Unfortunately, Provisions' custom ZK protocol cannot work with *hashed* public keys (which account for the majority of today's on-chain addresses), or with privacy-preserving cryptocurrencies (such as ZCash [31]) and its protocol's efficiency is linear to the size of anonymity set; thus, it cannot practically apply to most of today's blockchains. However, it is still considered the most sophisticated PoA solution to this day[3].

MProve [25] implemented a PoA algorithm tailored to Monero [41]. Since ring signature obfuscation does not allow for directly applying the Provisions PoA, its approach was to prove that the key images of the addresses controlled by the organization have not previously appeared on the blockchain. As PoA protocols are susceptible to collusion, MProve provides a proof of non-collusion as well by leveraging the one-time nature of key images. Unfortunately, this exposes the sender's identity when these key images are spent, potentially enabling tracing of transactions which breaks Monero's advertised privacy guarantees.

Wang et al. [42] proposed a scheme for a buyer proving assets to a vendor before finalizing a deal, using the transaction's details as a "challenge", which

[3] One could use a generic ZK system; however proving costs might be prohibitive in practice for Diem's ZK-unfriendly Pure-Ed25519-with-SHA512 signatures (including multi-sig); even with the latest recursive ZK proof schemes [26].

however is limited to a "buyer-vendor" use-case without any privacy characteristics. More importantly, it does not preserve the prover's privacy against the verifier (or regulator) as strongly as Provisions.

Blockstream's proof of reserves [38] consists of signing an "invalid" Bitcoin transaction for each owned Unspent Transaction Output (UTXO). This transaction cannot be published to the blockchain, however still degrades the organization's privacy against the auditor. A similar approach is followed by Kraken cryptocurrency exchange [3]. The main advantage of this method is that hardware security module (HSM) or cold wallet implementations do not need an extra logic for signing PoA payloads and thus, it is directly backwards compatible with existing custodial wallets.

Ethereum [7] proposed a different payload format when signing a message other than a valid transaction[4]. The purpose of this distinction is to ensure that one should not accidentally sign a transaction masqueraded as a message nonce. In our PoA case, this prevents an auditor from maliciously picking a hash of a transaction as an audit-nonce, which if signed, it could be submitted on chain without the user knowing. Also Iconomi's proof of reserves [6] proved key ownership to Deloitte (auditor) through either signed nonces or predefined transactions from the proving addresses.

Finally, a recent work [22] provided definitions and systematization for several payment systems, including those offering PoA functionalities, and compared them in terms of their properties and efficiency.

2 Diem Architecture

2.1 Keys and Accounts

Diem [12] is an account-based blockchain payment system, currently maintained by a permissioned set of *Validators* which participate in its BFT-based consensus protocol [14]. Although there are no built-in privacy preserving protocols for its account states and transactions, due to its permissioned nature, all public queries (including blockchain correctness verifications) are proxied through *full nodes*, which have the same view of the blockchain as *Validators*, but without participating in consensus. Compared to traditional cryptocurrencies, Diem provides the following features:

- *Authentication* keys, known as *auth_keys*, are hashed versions of account public keys, however they can be rotated independently as a proactive or reactive measure to defend against possible key loss. This means that unlike Ethereum, a key rotation does not imply change of address.
- Diem natively supports single Pure Ed25519 [32] or threshold multi-sig (k-*out-of-*n up to $n = 32$) *auth_keys*.
- There exists the concept of *withdraw capability*, where the permission to spend can be *delegated* to a different account. This implies that the spending key does not necessarily reside in the state of each address.

[4] Ethereum's message signing uses a flag prefix, to ensure an invalid transaction: *sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message))*.

– It also supports the *key-rotation capability* where one can give permission to other accounts to update their *auth_keys*. This is useful for wallets where one can still refer to another *cold* address to gain access back to their account in case of accidental *hot auth_key* key loss.
– Account roles define the account owner's authority in the system. A unique characteristic of Diem is its hierarchical role-based access control [10]. Unlike Bitcoin and Ethereum, especially for VASPs, there exist a KYC-ed parent and child accounts as shown in Fig 1.

2.2 Hierarchical model

For the purposes of this work, we focus on Diem roles most commonly related to PoA: ParentVASP and ChildVASPs. A ParentVASP represents the primary account of a regulated custodial wallet, while multiple ChildVASPs can be created by ParentVASP accounts[5]. In Diem, a PoA will be requested from the ParentVASP , and these proofs should include all of their children's assets as well. Although not privacy-preserving, due to the well-defined linkability of the accounts belonging to the same entity, hiding owned addresses is not possible for KYC-ed VASPs. In Section 4.1 we provide details about the PoA related Diem data structures.

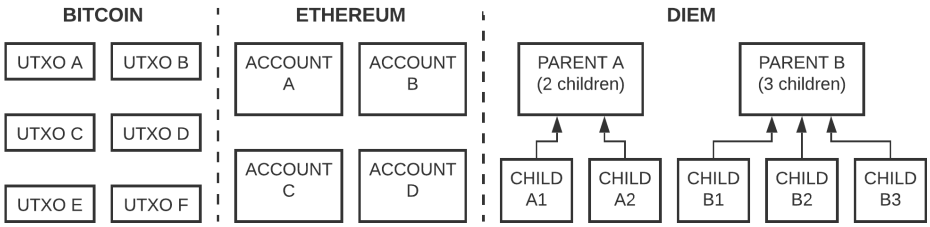

Fig. 1. Address structure in different blockchains.

As an account-based system, Diem associates each account A with a value v_j at each block j. We denote by A^P and A^C accounts with ParentVASP and ChildVASP roles, respectively. An A^P can be linked to n accounts $A^C_1, A^C_2, ...A^C_n$. There is a relation F which maps each child to its parent account, i.e. $F(A^{C_i}) = A^P$. Note however that no inverse relation exists in Diem, i.e. the parent's state does not include a relation $F^{-1}(A^P) = [A^{C_1}, A^{C_2}, ..., A^{C_n}]$. This was probably a design decision to not allow parent account states growing indefinitely when more children are added, because for a large n that map would require significant storage space.

However, the data structure for A^P *does* include the cardinality n, a very important property to later ensure no child is missing in the proofs. Note that although a ParentVASP can create ChildVASPs, this does not necessarily mean

[5] Note that in Diem a ChildVASP is not allowed to have any other children itself.

that it controls the keys of its children, and ChildVASPs can transact independently. Of course, nobody prevents wallets from reusing the same key in multiple accounts or apply a BIP32 deterministic key derivation [33]. That said, the hierarchy is mainly enforced for KYC-ed account linking and splitting the risk of a key compromise attack; it also allows for different key and asset management policies, such as cold, warm and hot wallets or transaction sharding and parallelization[6].

2.3 Diem Proof of Assets

Generally, a PoA in Diem implies showing that a ParentVASP account is in possession of assets of some specific currency(ies) value. However, there is a subtle distinction on how to actually show this. One could merely use existing blockchain data structures, and sum the values of a ParentVASP and all of its ChildVASP accounts, based on account *ownership*. While straightforward, this proof does not provide key possession guarantees at the time of the auditing taking place. For instance, account holders might have lost access to their keys, which would make them unable to spend their assets. Therefore, we distinguish between the following two PoA types for a query on account A^P for a block j:

Soft PoA: This proof is non-interactive, and a user (a third party auditor or even a light client) can obtain it at any time and for any block j via a series of blockchain requests to potentially untrusted nodes. Its simple goal is to provably present the total balance for all accounts belonging to the audited entity. No proof-of-knowledge of the spending key is required (and thus the name *soft*), however the parent account is linked with the KYC-ed entity; no other entity can claim this address's balance, and thus some applications might tolerate *soft* proofs. We highlight that this is only possible in Diem due to its hierarchical identity-address binding which makes collusion more difficult and traceable; a *WalletA* cannot just temporarily borrow its private key to a *WalletB* (an on-chain transaction should happen posing the risk of being censored). Such a proof is constructed by showing the following:

1. Given a genesis or any known checkpoint state with Merkle root r_G, prove that the Merkle root r_j is valid (see Section 4.1 for details on these data structures). In practice, the auditor will pick the block j for which the PoA is needed. Note that in Diem, this can be shown using a series of epoch change proofs to get the validator-set at block j.
2. For r_j, provide Merkle inclusion proofs for both parent A^P and its children $A^C_1, .., A^C_n$ account states.
3. All related account state balances (i.e. $A^P, A^C_1, .., A^C_n$) sum up to a value V. In PoSolv, this V is typically compared against proofs of liabilities [21].
4. $F(A^C_i) = A^P, \forall i \in (1, ..n)$, where n is the cardinality in account state A^P. This ensures that no child is accidentally or purposely omitted from the list.

[6] While typical account-based systems require a sequential id to prevent replay attacks, Diem's hierarchical model enables parallelization at the entity level, due to each child maintaining its own sequential id.

Hard PoA: A *hard* PoA is requiring a key-ownership proof on top of *soft* proofs, usually via signing. To prevent replay attacks, the protocol should require each account to sign over some random challenge. Note that it is acceptable to sign with the *auth_key* (or a delegated key via withdraw capabilities), valid at a requested block in the past or the most recent one. We refer to these two types as *dated*-hard PoA and *live*-hard PoA, respectively (further discussed in section 4).

3 Implementation considerations

3.1 What message to sign?

As previously discussed, *hard* PoA simulates a proof of key-possession by signing over a challenge r to prevent replays. This can either be a "special" hard PoA transaction, included as metadata, or even run off-chain. Options for r include any combination of the following:

- a random string interactively provided by the auditor.
- the hash of the block (or state snapshot) at height $(j-1)$. Note that Diem has the concept of transaction *version*, which is a monotonically increasing integer for all of the on-chain transactions. The latter means that one can even take a snapshot at the middle of the block, but typically, when we refer to height we imply the *version* number of the last transaction in a block.
- the latest Bitcoin block or from other robust proof-of-work blockchains (thus, use an external reference for randomness). However that would require running a mini light client as a smart contract or trust an Oracle service that could verify correctness of the external seed input.
- the output of a distributed randomness generation protocol (such as RandHound [39]), which can even be run by Diem *Validators* at each block.
- other publicly verifiable sources of randomness which embed timestamp information [29], such as the closing stock prices in the stock market, weather conditions in major cities etc, ideally with the use of verifiable delay functions (VDFs) [16].

However, some of the above randomness sources are susceptible to collusion attacks. For instance, the auditor and the ParentVASP might collude on the provided randomness in advance, or consensus *Validators* might agree to form a predictable block in Diem (this might be tolerated by the BFT assumption). Therefore we prefer a combination of external verifiable randomness and the RoundHound protocol ran from *Validators* which can offer better transparency guarantees. In short, we need a verifiable random and fresh challenge to ensure that the prover could not have predicted and pre-signed it long ago.

While hard PoA could also be automated to be executed at some predetermined times, the above randomness or challenges need to be unpredictable to prevent misbehavior. Note however that unpredictability is weaker than being "bias-proof", a property required by other use-cases (e.g. lottery protocols). For

instance, in a lottery protocol an attacker's goal could be to increase the probability of outputting a string that ends in 0. However in the case of hard PoA, biasing the result in this way would have no benefit for the attacker as we're only interested on signing over a fresh unpredictable challenge. More information on what data to-be-signed offers the above freshness and unpredictability properties is provided in section 4.3.

3.2 Various PoA considerations

Account state pruning: Many blockchain systems (including Diem) conserve space by pruning old account states, but still keeping the state's hash to preserve the system's security. Therefore, if the latest blockchain height is m and a PoA is requested for some height $j < m$, the full account state containing a balance v_j might not be available on-chain. In this case, the account's state would have to be recovered by a full-node who maintains the full history. Validating the provided pruned state is easy; we just check if the state's hash-output equals the blockchain-maintained hash value for this account.

Cold wallets and valet keys: Hard PoA might be cumbersome when air-gapped wallets are involved, as performing such an operation would require bringing keys out of cold storage. The process sometimes requires expensive ceremonies, i.e. when the key resides in HSM modules or physical vaults, or when it is split between several parties. A possible approach to improve usability could be a) embedding PoA operations in HSM or b) using valet keys as defined in [5].

Incentives: When proving solvency, malicious auditees might collude to temporarily prove assets *greater* than some value that represents their off-chain liabilities. On the other hand, other auditees might try to *hide* assets on purpose (e.g for tax evasion purposes). This would be a problem in any system other than Diem, where the auditee could simply claim loss or non-knowledge of some key, and complex blockchain analysis techniques (e.g. clustering) would have to be deployed to prevent such behavior. However Diem's hierarchical, KYC-ed account model mitigates this.

Locked assets: In our model we do not consider locked on-chain assets, i.e., for future atomic swaps or side-chain smart contracts (locked assets are not supported by Diem yet). In fact, proving solvency by taking locked assets into account is an open research challenge in every blockchain, as discussed in the recent ZKProof 2020 workshop [21].

4 Diem-specific implementation considerations

4.1 Primitives and soft PoA implementation in Diem

Sparse Merkle Trees: Recall a Merkle tree [35] is a binary tree constructed by a collision-resistant hash function h, providing logarithmic proofs with logarithmic complexity. Sparse Merkle trees share the same philisophy, however

tree-leaves do not contain the accumulated elements themselves but serve to form an "index" of the element along with its path to the root. This enables them to provide proofs of non-membership, where non-accumulated elements can simply end to a placeholder value to maintain tree balance. However as these class of Merkle trees are intractably large, we can also represent them by omitting sub-trees that only contain placeholder values. Diem uses a variant of Sparse Merkle trees (Jellyfish Merkle trees [27]) which enables shorter Merkle proof sizes while still providing collision resistance.

In Diem, transactions are accumulated in a Merkle tree, which in turn contains roots of sparse Merkle trees that represent the state of all accounts as the transaction gets executed [11, 12]. The top Merkle tree root defines the block hash and is signed by the *Validators* participating in the consensus (at least 66.7% of them should sign) as transactions are processed and account states are modified accordingly. We describe specific data structure format in Diem below.

Diem Data Structures [8, 9]: In Diem, account states are represented as an `AccountStateBlob` which includes, among others, the address, balance for each currency and account role (i.e., ParentVASP or ChildVASP). These account states are stored in a sparse Merkle tree called `TransactionInfo`. In turn, this sparse Merkle tree's root hash `state_root_hash` represents all of the accounts' global state at the end of a specific transaction.

In turn, the *most recent* `TransactionInfo` root in an blockchain version, along with the epoch number corresponding to the current *Validator* set and a timestamp, are encapsulated in a `BlockInfo` data structure. This data structure along with a hash value of the consensus Quorum Certificate is encapsulated in a `LedgerInfo` Merkle Tree. Note that a version's most recent Transaction (e.g. transaction T4 in Figure 2), effectively defines the global state of all accounts for that version.

Finally, `LedgerInfo` along with consensus signatures by the current *Validator* set is encapsulated in a `LedgerInfoWithSignatures` data structure, making it acceptable by anyone trusting Diem's BFT assumptions.

Proofs: A core object for implementing Diem soft PoA is the `AccountStateProof` data structure. This contains a sparse Merkle tree proof (`SparseMerkleProof`) for a `TransactionInfo` object, which in turn is verified by a `TransactionInfoWithProof` proof for the Merkle tree. A second crucial element is `EpochChangeProof`, which includes the list of signatures involved in *Validator* set updates. Through these built-in proof functionalities in Diem, we implemented a proof-of-concept for executing soft PoA [40].

4.2 Random challenge consistency

It is recommended, especially when BIP32 [33] is applied or the same key is used between accounts, that r should be the same across all signed messages to minimize proof size. However, as keys in Diem are rotated regularly (discussed

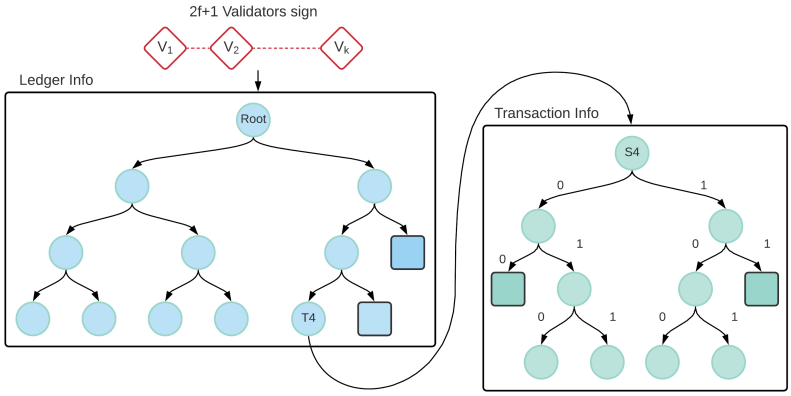

Fig. 2. Diem data structure overview.

in Section 2.1), there are two options for signing a hard PoA for time[7] t: a) use authentication key that was valid at a past instance t, and b) use the most recent authentication key (which will be linked to the key at t using a chain of rotations). While both PoA types are acceptable for proving asset control at t, the latter version is stronger as it also shows key control for a more recent time $t+\Delta$. A reason for picking a slightly older t might be to reduce the probability of wallet collusion; if one doesn't know for which t they will be audited, temporarily borrowing private keys from other wallets is riskier. It is highlighted though that wallet providers might have deleted old account keys, and thus a t closer to the current time/block should be preferred, unless there is a reason not to, e.g. for proving assets exactly at the end of a calendar year.

In any case, we refer to the above two hard PoA types as *dated*-hard PoA and *live*-hard PoA respectively. We mention that especially for cold wallets, it is advised the auditee signs and rotates the keys simultaneously to ensure some additional (although not complete [19]) post-quantum security, due to publishing hashed keys.

4.3 Signed block hashes as randomness

In the previous section we discussed that block hashes can be used as a randomness source to sign a hard PoA message, preferably in combination with other randomness sources. Specifically in Diem, to prevent an attacker from manipulating this source, we would pick the root of the `LedgerInfo` tree that includes $2f + 1$ Validator signatures(thus a `LedgerInfoWithSignatures` object), where f denotes the upper bound of Byzantine Validators. Therefore to manipulate this information, an attacker would need to also subvert more than f Validators

[7] We assume t is in the past. While it could be possible to make a PoA request for some time in the future in advance, this enables several collusion attack vectors which we do not discuss in this paper.

which in turn would break the assumption of Byzantine Fault Tolerance. Note that a dishonest leader could in theory selectively pick any $2f + 1$ signature combination when all Validators sign, but fortunately this does not give any advantage as we are interested in a fresh and unpredictable, but not necessarily unbiased, challenge.

4.4 Accurate timestamping

Diem's blocks use monotonically increasing timestamps. This implies that (unlike other blockchains) one could use a time reference t instead of a block-height j. However, using well-defined block-heights is advisable for PoA purposes. In addition, all PoA elements should be consistent for a specific block, with the proof showing the total assets for a *snapshot* of the *same* blockchain height (or timestamp) across all ParentVASP and ChildVASPs. If a variation in height was allowed, malicious provers could move assets among their accounts in neighboring blocks and falsely claim assets greater that those actually owned.

Also, as mentioned before, Diem uses "versions" rather than "blocks-heights", with each transaction resulting in a unique, incremental version. Therefore, as each block has subsequently a *range* of versions, the account states in the latest version in a block need to be retrieved [27]. This can be implemented through appropriate GetVersionByTimestamp() and GetStateByVersion() functionalities, which would return the blockchain version for some specific timestamp and the blockchain state for some version respectively. Note that as shown in Figure 2, the latest transaction in an epoch should be considered (transaction T4 in the Figure) for all account state proofs, and prove that the immediate next transaction belongs to the next epoch. This ensures that account proofs are provided after all transactions in the block have been considered.

4.5 Compression

Signature compression: Signatures and public keys account for the largest part of a PoA payload. Actually, there exist three types of signatures:

1. *Validator* signatures over the block data.
2. account signatures for every transaction in a block.
3. key-possession-proof signatures for each auditee key (potentially delegated).

In PoA we are interested in the first and third signature types. Compression can be achieved though various techniques, but some of them require a Diem protocol update and thus, they cannot be applied directly. Examples include having the *Validators* running interactive multi-sig protocols, such as Musig2 [37] and FROST [34], or supporting BLS signatures [17], which allows for aggregation to a single signature (although we still need the public keys). Solutions not requiring a protocol update include the SNARKs [30], STARKs [26] or the recent non-interactive EdDSA half-aggregation [20]. However, for auditee signatures

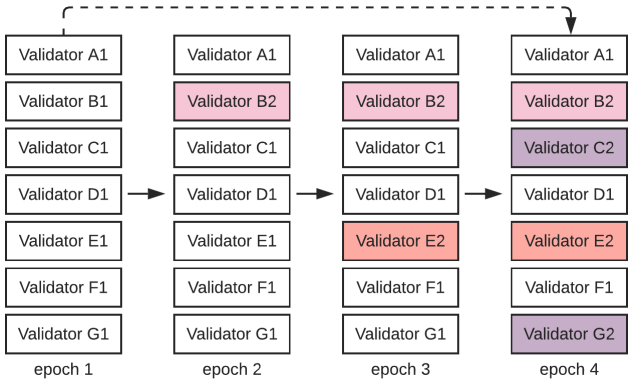

Fig. 3. Epoch skipping optimization.

over a challenge, the prover, who controls all of the keys, can simulate a Musig2 in-the-head or apply the Schnorr batching technique of [28].

Epoch proof compression: At the moment, Diem's epoch-change proofs are sent in raw format, without tackling duplication between epochs. We present an easy to implement partial compression method without advanced ZK protocols.

Normally, to verify epoch changes, where at least one *Validator* rotates its key, we have to verify all intermediate epochs from the last known checkpoint. However we can skip epoch verifications if less than 1/3 *Validators* have been updated, and only require to "jump" to an epoch where a sufficient number of *Validators* have changed (concept shown in Fig 3). Note however that this optimization is incompatible with long range attack prevention [13]; still, this might be tolerable in some threat models. We can further optimize epoch proofs by only considering the required $2f + 1$ signatures (omit the rest) along with their key rotation operations, even when all *Validators* have signed.

4.6 Multiple currencies

Note that Diem supports several currencies, and asset proofs might be required across all of them. Our recommendation is that PoA should run per currency (but again for the same height). Converting all currencies to a single one using the current exchange rate is not advised for PoSolv purposes [21], as there are examples of extreme volatility (i.e., the case of Swiss franc cap removal on Jan 15, 2015 [18]).

4.7 PoA transaction type

In general, as an alternative to a carefully signed message that is distinguishable from a regular transaction by design [2], transferring some amount (or even a zero amount) to a (designated) address would also work for PoA purposes, especially if one wants on-chain PoA recording. In Diem, a hard PoA could

also be executed through a special transaction type, with the sole purpose of signing a message, however such "NO-OP" transactions are not yet implemented. Fortunately, Diem allows sending funds to self, which is one way to implement hard PoA. Another option would be to send some amount to a pre-determined "sink" account. Such approach has the advantage of consolidating all associated PoA events and making them easier to track.

4.8 Withdraw capability

As discussed in Section 2.1, Diem has the unique functionality of granting the capability of spending to other accounts and smart contracts [15]. This delegation mechanism introduces additional complexity when proving assets. Because of withdraw capabilities, the PoA message signing should happen on-chain, which would make sure the smart-contract logic that involve withdrawal capabilities would execute. Otherwise, off-chain verifiers would need a copy of the current blockchain's state and be able to simulate running a transaction in this copy, which would make the whole process expensive or cumbersome. Another issue is related to potentially incompatible implementations of custom withdraw capability logic (smart-contract), because currently there is no enforcement of requiring additional metadata (which in our case is required to attach the random challenge).

5 Conclusion

We presented several considerations for implementing proof of assets in the Diem blockchain. By taking advantage of Diem's native hierarchical account structure, two major policies of asset proofs have been analyzed: *soft* PoA, which can be executed at any time without interaction with account holders, and *hard* PoA which provide extra assurance that account holders are in control of their keys. However the latter requires a more carefully planned, coordinated interaction.

All of our proofs rely on widely-used cryptographic primitives with standard assumptions (i.e. signatures and Merkle proofs). We discuss several edge-cases that should be taken into account when designing PoA protocols in a hieararchical, KYC-ed, account-based blockchain system (e.g. timestamping and consistency), and propose practical solutions, including options for fresh and unpredictable proof of key-possession challenges. Finally, we propose easy to implement optimizations (e.g. signature and epoch change proof compression), while still remaining compatible to the underlying Diem blockchain.

Acknowledgements

We thank Philip Hayes, David Wolinsky, Alden Hu and Valeria Nikolaenko for their implementation contributions, Riyaz Faizullabhoy for custody related hints, Dmitry Korneev and Adeniyi Abiodun for their input on needed regulation and compliance, and finally Sam Blackshear and Tim Zakian for enabling the required Move-language api regarding parent-children Diem account linking.

References

1. Bitstamp proof of reserves, https://www.bitstamp.net/s/documents/Bitstamp_proof_of_reserves_statement.pdf
2. Ethereum wiki (archive.org), https://web.archive.org/web/20190613115908if_/https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign
3. Kraken proof of reserves, https://www.kraken.com/en-us/proof-of-reserves-audit
4. Okex suspends withdrawals, says key holder not available due to cooperation with investigation, https://www.coindesk.com/okex-suspends-withdrawals
5. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. Real World Crypto 2016, https://rwc.iacr.org/2016/Slides/Provisions%20talk%20RWC.pdf
6. Proof of solvency technical overview (2018), https://medium.com/iconominet/proof-of-solvency-technical-overview-d1d0e8a8a0b8
7. Ethereum wiki (2019), https://web.archive.org/web/20190613115908if_/https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign
8. Diem authenticated data structure specification (2021), https://github.com/diem/diem/blob/main/specifications/common/authenticated_data_structures.md
9. Diem data structures specification (2021), https://github.com/diem/diem/blob/main/specifications/common/data_structures.md
10. Diem roles and permissions (2021), https://dip.diem.com/dip-2/
11. Diem storage module (2021), https://github.com/diem/diem/tree/master/storage
12. Amsden, Z., Arora, R., Bano, S., Baudet, M., Blackshear, S., Bothra, A., Cabrera, G., Catalini, C., Chalkias, K., Cheng, E., Ching, A., Chursin, A., Danezis, G., Giacomo, G.D., Dill, D.L., Ding, H., Doudchenko, N., Gao, V., Gao, Z., Garillot, F., Gorven, M., Hayes, P., Hou, J.M., Hu, Y., Hurley, K., Lewi, K., Li, C., Li, Z., Malkhi, D., Margulis, S., Maurer, B., Mohassel, P., de Naurois, L., Nikolaenko, V., Nowacki, T., Orlov, O., Perelman, D., Pott, A., Proctor, B., Qadeer, S., Rain, Russi, D., Schwab, B., Sezer, S., Sonnino, A., Venter, H., Wei, L., Wernerfelt, N., Williams, B., Wu, Q., Yan, X., Zakian, T., Zhou, R.: The libra blockchain (2020), https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf
13. Azouvi, S., Danezis, G., Nikolaenko, V.: Winkle: Foiling long-range attacks in proof-of-stake systems. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 189–201 (2020)
14. Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., Sonnino, A.: State machine replication in the libra blockchain. The Libra Assn., Tech. Rep (2019)
15. Blackshear, S., Wilsion, B., Zakian, T.: Diem improvement proposal 11 (2021), https://dip.diem.com/dip-11/
16. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual international cryptology conference. pp. 757–788. Springer (2018)
17. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001)
18. Breedon, F., Chen, L., Ranaldo, A., Vause, N.: Judgement day: Algorithmic trading around the swiss franc cap removal (2018)

19. Chalkias, K., Brown, J., Hearn, M., Lillehagen, T., Nitto, I., Schroeter, T.: Blockchained post-quantum signatures. In: 2018 IEEE Blockchain. pp. 1196–1203. IEEE (2018)
20. Chalkias, K., Garillot, F., Kondi, Y., Nikolaenko, V.: Non-interactive half-aggregation of eddsa and variants of schnorr signatures. CT-RSA (2021)
21. 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
22. Chatzigiannis, P., Baldimtsi, F., Chalkias, K.: Sok: Auditability and accountability in distributed payment systems. Cryptology ePrint Archive, Report 2021/239 (2021), https://eprint.iacr.org/2021/239
23. 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
24. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014, Part II. LNCS, vol. 8713, pp. 313–326. Springer, Heidelberg (Sep 2014)
25. Dutta, A., Vijayakumaran, S.: Mprove: A proof of reserves protocol for monero exchanges. In: 2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019. pp. 330–339. IEEE (2019). https://doi.org/10.1109/EuroSPW.2019.00043, https://doi.org/10.1109/EuroSPW.2019.00043
26. Gabizon, A., Gurkan, K., Jovanovic, P., Konstantopoulos, G., Oines, A., Olszewski, M., Straka, M., Tromer, E.: Plumo: Towards scalable interoperable blockchains using ultra light validation systems. ZKProof (2020)
27. Gao, Z., Hu, Y., Wu, Q.: Jellyfish merkle tree (2021), https://developers.diem.com/papers/jellyfish-merkle-tree/2021-01-14.pdf
28. Gennaro, R., Leigh, D., Sundaram, R., Yerazunis, W.: Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 276–292. Springer (2004)
29. Gjermundrød, H., Chalkias, K., Dionysiou, I.: Going beyond the coinbase transaction fee: Alternative reward schemes for miners in blockchain systems. In: Proceedings of the 20th Pan-Hellenic Conference on Informatics. pp. 1–4 (2016)
30. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
31. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification. GitHub: San Francisco, CA, USA (2016)
32. Josefsson, S., Liusvaara, I.: RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA) (Jan 2017). https://doi.org/10.17487/RFC8032
33. Khovratovich, D., Law, J.: Bip32-ed25519: Hierarchical deterministic keys over a non-linear keyspace. In: 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 27–31. IEEE (2017)
34. Komlo, C., Goldberg, I.: Frost: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852 (2020), https://eprint.iacr.org/2020/852

35. 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)
36. Moore, T., Christin, N.: Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 25–33. Springer, Heidelberg (Apr 2013)
37. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261 (2020), https://eprint.iacr.org/2020/1261
38. Roose, S.: Standardizing bitcoin proof of reserves, https://blockstream.com/2019/02/04/en-standardizing-bitcoin-proof-of-reserves/
39. Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy. pp. 444–460. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.45
40. – blinded –: Diem proof of assets (2020), https://www.dropbox.com/s/wdjq47l7dds98wf/proofassets.rs
41. Van Saberhagen, N.: Cryptonote v 2.0 (2013), https://cryptonote.org/whitepaper.pdf
42. Wang, H., He, D., Ji, Y.: Designated-verifier proof of assets for bitcoin exchange using elliptic curve cryptography. Future Gener. Comput. Syst. **107**, 854–862 (2020). https://doi.org/10.1016/j.future.2017.06.028, https://doi.org/10.1016/j.future.2017.06.028