A public blockchain solution permitting secure storage and deletion of private data —Draft

Dr. Björn Stein[1], Konstantin Kuznecov[1], Sangseop Lee[2] and Dr. Jürgen Müller[3]

[1]Lition Foundation
[2]Korea Blockchain Association
[3]SAP Innovation Centre

August 14, 2018

Abstract

We present the design for a blockchain network and minimum requirements for a governing agreement among a privileged subset of the nodes' operators to ensure that sensitive and private data can be handled and securely deleted on demand. The guiding design criteria are based on an operational blockchain application and include data minimization under the constraint of providing fault tolerance, postquantum security, privacy of sensitive data, a provision to delete all occurrences of sensitive data, and the freedom to join as a (non-privileged) node without any special provisions or legal obligations.

1 Blockchain technologies

Distributed systems are subject to the CAP theorem asserting that consistency, availability, and partition-tolerance cannot all be simultaneously achieved. Although the CAP theorem was proven to be correct, [Gilbert and Lynch, 2002] practical applications continue to show ways that come close to simultaneously achieving these three desirable qualities. Blockchain technologies do so in a probabilistical fashion in that data validity is ultimately never final. This way, partitions are avoided (as they are temporary). Without (persisting) partitions, the CAP theorem does not require that a choice must be made between availability and consistency.

Blockchain applications have reached a significant audience with Bitcoin [Nakamoto, 2009], a cryptocurrency built on top of a distributed consensus algorithm using the hashcash [Back, 2002] proof-of-work algorithm as a competition to determine an orderer. Unlike other Byzantine fault tolerant consensus algorithms, Bitcoin and other blockchain technologies can typically operate with an unknown number of participants and in a trustless fashion that is radically different from the threshold fraction of honest nodes required by merely Byzantine fault tolerant consensus algorithms.

Both the ability to operate with an unknown number of nodes and being trustless are originally achieved by determining a temporary orderer (or successful "miner") by a proof-of-work algorithm, ensuring the correctness of its submissions (or the "mined block") by validation in each node, and further validating blocks by treating as valid only the nodes in the longest path (the "blockchain") from the root in a tree of such blocks to the latest submission in this path. There is hence no ultimate guar-

antee of data finality (competing leaves can, in theory, always grow to become the longest blockchain) but only a (usually quickly increasing) confidence in blocks as one path gets increasingly longer and competing paths are increasingly abandoned from being grown.

Blockchain technology not only enables cryptocurrencies, in which integer quantities (of "coins" or "tokens" or a subdivision thereof) is managed in a way that each such quantity has an "owner" by whom it can exclusively be redistributed, that its sum is somehow constrained (e.g. to a total, growing but limited supply of Bitcoins in the blockchain technology of the same name), and these properties are automatically enforced by the underlying technology in a way that even conspiring nodes cannot violate them unless they reach a threshold of "mining power" (e.g. a majority of proof-of-work power in Bitcoin called a "51 percent attack"). It also enables a generalization of this scheme known as smart contract, [Szabo, 1997] a concept invented prior to the realization in Bitcoin. Perhaps best-known for its suitability for smart contracts is the blockchain technology Ethereum. [Buterin, 2014]

There is a natural way to provide security against quantum computers: Employ postquantum cryptography. This may become possible e.g. with a future but already discussed upgrade [Buterin, 2015] to Ethereum allowing participants to choose the cryptographic algorithms used for their data and smart contracts.

Other developments include permissioned blockchains which have a sense of identity beyond an "address" or hash of a public key that any participant can choose simply by generating a private key. One advantage is that "private transactions" with restricted data visibility can be implemented by encoding data in away that only a given set of node can decode it; an example for this capability is Quorum [Nielsen et al., 2016] which extends Ethereum in this regard. Permissioned blockchains can be based on existing identity infrastructure involving certificate authorities and hence can build on established standards for addressing security concerns, e.g. by key rotation upon key compromise or even in regular intervals. This provides a practical advantage for implementing requirements that a certified identity shall be able to exercise certain powers not available to other participants. This can include restricting the set of validating nodes to a known number and hence enabling conventional consensus algorithms to be employed; an example of such capability is the Hyperledger blockchain [Jagadeesan et al., 2017a,b].

One topic of this paper is a solution to a perhaps neglected aspect of blockchain technology, compatibility with ideas of data protection. This may seem a contradiction in terms as blockchain technology has historically relied on public visibility and large-scale replication of data ("transactions") yet data protection is centered around ideas of the minimalization of data, the restriction of its visibility to the point of maintaining a notion of ownership, and ultimately the ability to control data and data flow, including the feasibility of verifiable deletion of sensitive data. Indeed the obvious approaches such as limiting transactions with sensitive data to a small number of permissioned nodes with which special legal arrangements about the deletion of data is made is a poor compromise: The limitation to few nodes in itself loses an aspect of the security behind blockchains—consider that a threshold-reaching conspiracy requires fewer conspiring nodes. The provision to delete transaction necessarily loses an entire pillar, that of a provable chain of (hash-linked) blocks, unless a provision is made to e.g. retain a cryptographic hash of the deleted data.

Realizing data protection ideals is not just a

virtue but has become a business requirement for many applications. We argue that this justifies the creation of yet another blockchain technology centered around its requirements. This is, as explained in the preceding paragraph, necessarily a compromise but we detail a solution retaining key features of both data protection and blockchain technology. To include provisions for deletion of data, this solution necessarily reaches beyond the realm of a pure computer implementation: An algorithm cannot prove that it has not retained an extra copy of a given data. Legal agreements between parties operating such an algorithm must be in place and can be supported by an algorithm that makes it computationally at least as cheap to actually forget (or, better yet, overwrite) some data than to retain it.

Beyond provisions for data protection, our blockchain design features scalability to high speed, high throughput, and low operating cost. These are traditionally hard issues for blockchains as their trustless nature usually requires every node to verify every transaction. That is a bottleneck for throughput and, if the computational effort across every node was priced (which it often is not), a cost factor. Worse yet, blockchain technologies traditionally built on proof-of-work schemes that worsen the cost aspects, often by many orders of magnitude.

2 Consistency, nodes' power, and private data

In any blockchain technology it is critical that a node checks every single transaction in the blockchain lest it mistakenly identifies an invalid path (involving blocks that were not only ordered but augmented with some invalid transaction) as the correct blockchain to follow. Whilst such a situation, if affecting only one or a few nodes, would heal itself, it is problematic if it affects a majority of mining power and thus perpetuates. It could be argued that non-mining nodes can operate in a "light" manner where they do not verify the validity of all transactions in the blockchain, thereby relying on the assumption that the majority (by mining power) of miners do not follow such an approach and hence relinquishing the trustless nature of most blockchain technologies.

The security sacrifice associated with light nodes would seem to limit the throughput and the maximum data size in the blockchain of a trustless blockchain technology to that feasible on the least powerful non-light (or "full") node. This is not necessarily true: Instead of forming a single blockchain, provisions for side chains can be made. By subscribing only to a limited number of side chains, less powerful nodes can still ascertain the validity of chains relevant to their transactions. Approaches to achieve better scaling than traitional blockchain solutions include using directed acyclic graphs ("tangles") instead of a single blockchain (e.g. in Iota [Popov, 2017]) to achieve the required partial ordering to decide an execution order between potentially conflicting transactions. Recent developments in a similar direction include the multi-blockchain framework polkadot [Wood, 2017] and the novel consensus algorithms Hashgraph [Baird, 2016] and Algorand [Gilad et al., 2017, Chen and Micali, 2016, Chen et al., 2018].

A problem without known solutions is the limited verifiability of private data, see e.g. the criticism in de Vilaca Burgos et al. [2017]. If such private data is only visible to a limited set of nodes, only these can calculate the outcome of a transaction acting on it. This statement is not self-explanatory because homomorphic encryption might enable an exception. Homomorphic encryption refers to cryptographic systems that allow certain operations to be performed on encrypted data without reveal-

ing the data that is encrypted. However, a simple argument shows that homomorphic encryption cannot provide a key element to typical smart contracts which often need to assert that an integer quantity is sufficiently large (e.g. that a balance is sufficient to make a payment). Enabling a party to make such determinations as well as continuing to perform a related subtraction (e.g. lower the balance by the payment amount) would naturally enable said party to learn if the original value is twice, thrice, etc., of the value it is compared against (e.g. how large, approximately, the balance is), thus undermining the desired privacy. The only way forward with this idea is to employ zero knowledge proofs of the execution of an entire smart contract, merely verifying the encrypted outcome of an algorithm rather than conditions (like an encrypted number being larger than a threshold) and actions (such as substracting from this number) separately, but this is extremely resource intensive despite recent progress in inventing schemes that could one day be used in this fashion, e.g. ZK-STARK [Ben-Sasson et al., 2018], bulletproofs [Bünz et al., 2017], and NIZKPoK [Katz et al., 2018].

3 Public proofs involving private data

We propose to solve the security issue of private data (and private transactions) in a blockchain technology by providing a way to publicly prove the correctness of a transaction involving private data without revealing the private data. This proof is of a probabilistic nature. It is faster and shorter than non-interactive zero knowledge proofs (NIZKPs) but sufficiently similar to replace them.

NIZKPs are best introduced by first considering an interactive zero knowledge proof to prove knowledge of a function. Imagine an entity wanting to demonstrate that it has learnt to reverse a trap door (hash) function yet it is unwilling to share the secret how this can be achieved. We would trust it to have this ability if it can repeatedly demonstrate it for data provably randomly generated, in particular, if we interactively challenge it to reveal the trap door function's input data for output data we supply. This is an interactive zero knowledge proof. Any interactive zero knowledge proof can be transformed into a non-interactive zero knowledge proof by supplying a suitable algorithm to generate the challenges from data that the prover is unable to fully control (otherwise the prover could effectively choose to avoid certain challenges that would reveal its inability to achieve what it intends to prove being able to perform). It must be pointed out that there are reasons [Unruh, 2015, 2017] to question that at least one such transform, the Fiat-Shamir transform [Fiat and Shamir, 1986], despite being based on hash functions and thus suggesting postquantum security, does indeed offer postquantum security in the general case.

Our proposal is not trustless. In fact, it requires trust in two aspects that are partially at odds with each other. One is that there is a set of verifier nodes (or at least one verifer node) that the executor of a transaction with private data trusts to keep this data safe. The other is that every node in the network trusts that not all members of this set of nodes conspire. To recognize the potential for contradiction, consider these requirements in a restated form: The set of verifier nodes must be trusted by the executor, to keep its private data secret and to delete it upon demand, but then again not be trusted to such an extent that all of them might be suspected, by any other node, of conspiring to forge data.

The first aspect of required trust, in handling private data, is exercised by the executor of a transaction: It forwards private input data (along with several sets of fake data that

ideally should be indistinguishable from the real data for the recipients) associated with the smart contract to be executed to the set of verifier nodes. Input data shall be encompass any parameter passed to a smart contract's function, that function, the current state of (the subset of accessed) data managed by the smart contract (including one or more nonces to blind its hashed value), and one or more nonces to blind output data to be calculated. The set of verifier nodes form a gossip network over which this data is shared in encrypted form and they independently calculate the resulting output data. Then each of them issues an endorsement indicating the nonce-blinded hashed values of each input and output data. This endorsement proof gets shared over a wider gossip network that includes all nodes of the network (or those subscribed to the corresponding side-chain, to be discussed in section 9). If a quorum is reached without a proof of the opposite (by revealing the unhashed input data) by one of the verifier nodes being mined into a block, the executor can reveal a proof of the correctness of one of the hashed input data sets by providing a path to a global (or side-chain wide) Merkle root. This will be mined into an ensuing block. The presence of both an unopposed quorum of endorsements and the Merkle paths proving one of the input data sets as having been present is our public proof about private data which we call an endorsement proof. The data flow is depicted in figure 1.

In case sufficiently many unoppossed endorsement proofs have been mined into a block, it is evident that this many nodes of the verifier set have checked that the input and output data revealed to them is indeed correctly processed according to the smart contract. We define a quorum, to be explicitly given when specifying the set of verifier nodes, as the threshold above which this shall suffice for all nodes to trust that their information about it, blinded hashes of the input and output data, is calculated correctly. To move forward, the executor node then has a proof that a then specified one among the input data sets corresponds to the current state of the private data referenced. This is possible by supplying a Merkle path from a global Merkle root updated by every node to the real input data set. Note that this path may lead to more than one node (in which case already all data in the input and output data set consisted of more than a single hash). Upon finding such a block mined (and the required quorum of verifier assertions in earlier blocks), every node updates its global Merkle root according to the blinded hash of the real output data.

Conflicts may arise when two smart contract invocations change data under the same Merkle root. This is essentially always the case when an intervening (and data changing) smart contract is finalized whilst another smart contract is in the first phase of this two-phase (test data and signed invocation submission and revelation of the input data's existance) because we propose using a single Merkle tree. Hence, even if the smart contract invocations can proceed in arbitrary order because they do not write to the same data locations, the Merkle path proving the input data's presence for the second invocatin must be updated. This updating is trivially possible for any node observing new blocks in the blockchain because the Merkle tree for the output data (which must likewise be updated) together with the original input data's Merkle tree contain the hashes (or contents) of every relevant node. The actual updating is performed by each mining node.

The reason we call this NIZKP-like construct not just an endorsement but a (probabilistic) endorsement proof relates to the idealization that the verifier nodes behave memory-less and to the example we gave for an interactive zero knowledge proof. Within this idealization, we probe at least one verifier node's ability (and

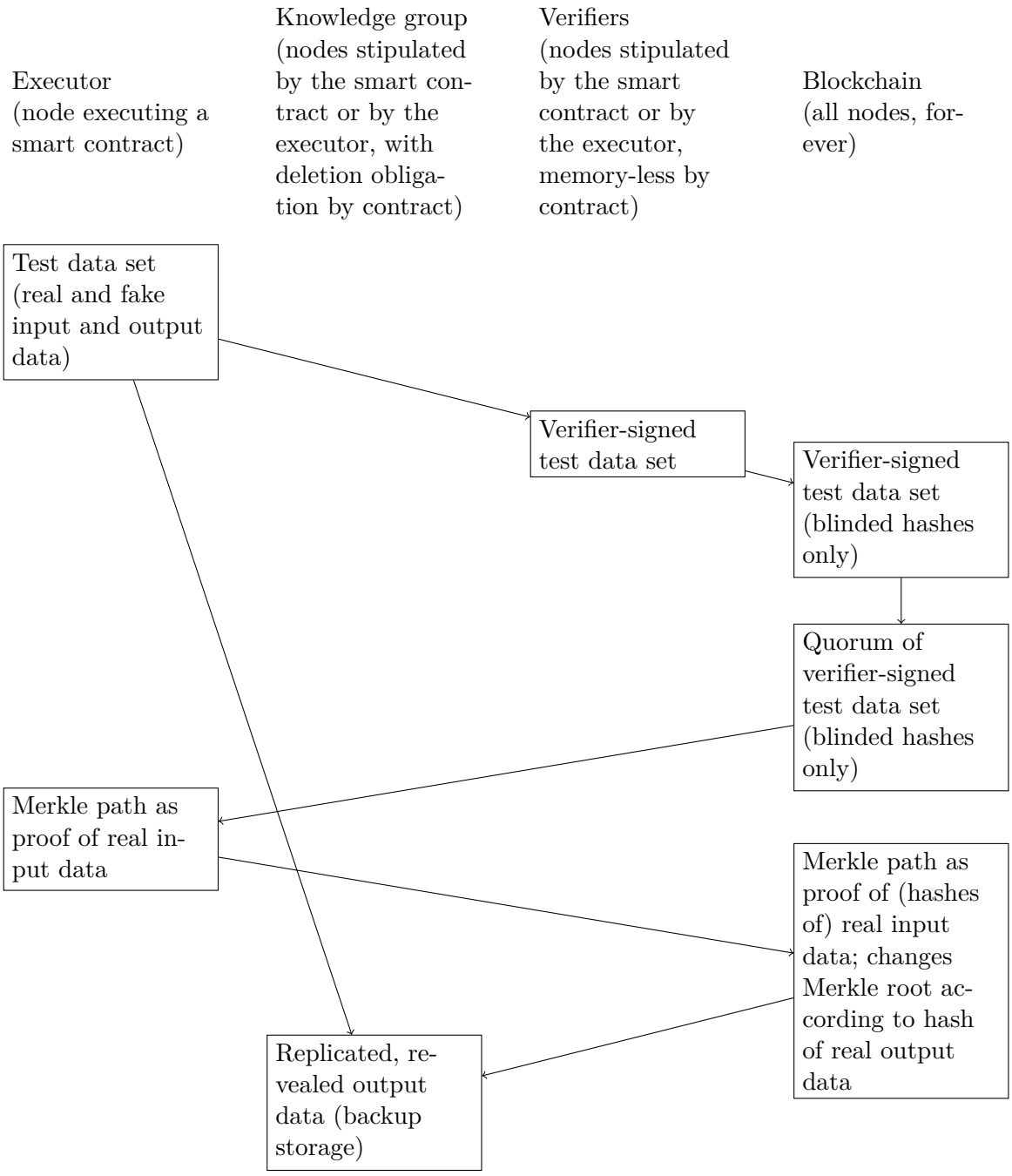

Figure 1: High-level communication.

willingness) to accurately calculate the (hash of the) output data of a transaction with given input data. Given sufficiently many (s) successfull executions we trust in it producing the correct output also for the correct data. The number s need not be nearly as large as for a proof with a remaining probability of achieving it by luck on a cryptographically low level if correct operation is incentivized (or incorrect one penalized) in a way that spans more than one endorsement proof.

This idealization requires enforcement beyond the world of computer code: A contractual agreement, including checks and appropriate penalties, to indeed run verifier nodes in the prescribed, memory-less manner, may be necessary. To partially alleviate this necessity, we propose a computer code based mechanism, to be detailed later, to penalize verifier nodes found to have produced even just a single incorrect output data hash. Since it it vitally important to the integrity of the blockchain technology that this indeed never occurs unchecked, we suggest that fault- and glitch-tolerant programming should be employed, and errors could be penalized to a maximum effect such as simultaneous exclusion of the offending node and forfeiture of coins or tokens held by it.

4 High-Level Realization

The high-level outline of the preceeding section leaves open the question how various proofs can be implemented. We propose to organize storage into a Merkle-Tree where the two branches at each level correspond to one bit in a large address space composed of rather large bitstrings that encode at least the set of verifier nodes, the set of nodes authorized to access data in a purely reading or also in a writing fashion, along with an integer actually describing an address into the address space described by the rest of the bitstring.

Using a Merkle tree solves the problem of proving the existence of a encrypted prior state (of a sub-address space) if the root of the overall Merkle tree is a consistent, distributed property. This is achieved by using the old and new Merkle root as the contents of each block in the blockchain, along with the novel proofs of each transaction and their old and new Merkle paths as proof of inclusion.

Using endorsement proofs as well as Merkle paths as proof of prior inclusion of data into the current Merkle root, the communication (and blockchain data inclusion) depicted in figure 1 becomes possible.

5 Hashes and Signatures

The security of our proposal obviously depends, apart from the required trust and enforcement issues already detailed, on the security of the hashes, signatures, and encryption employed. As we only employ encryption in transport, and envision to exclusively use the ubiquitous standard TLS (transport layer security) which we assume to acquire quantum resistance encryption methods in the future, we shall focus on hashes and signatures. Hashes are typically postquantum safe because the best attack against a quasirandom function is Grover's algorithm [Grover, 1996], lowering the security level of a perfect hash function to half its classical security level or, equivalently, increasing the bit length requirement to twice the requirement for resistance against merely classically equipped attackers.

To maintain postquantum security in the most obvious way, we opt for a hash based scheme for signatures. Hash based signature schemes can provide essentially unlimited signing capabilities yet such schemes (e.g. SPHINCS [Bernstein et al., 2015] or XMSS [Buchmann et al., 2011a] for which a

merely informational internet engineering task force's request for comments, RFC 8391, exists) have overheads, are somewhat complicated and, for a cryptographic scheme to be deployed, still relatively novel.

To also be reasonably secure against implementation issues or yet to be discovered attacks, we choose the simple and well-examined Winternitz one-time signature (WOTS, see e.g. Buchmann et al. [2011b]) scheme that offers a simple compression of the basic Lamport signature scheme [Lamport, 1979] which could be seen as an example of the most secure approach to signing as secure signing in general is only possible if and only if there is a secure hashing scheme [Rompel, 1990]. Using a one-time signature scheme implies constant key rotation: Each signature must include the publication of a new public key and it must have provisions to sign at least two messages (the key change and a payload message).

We propose to allow a signer to sign arbitrarily many payload messages with a single WOTS in order to allow many transactions (from many nodes) to be processed by each verifier with a single key change. To enable this, the signer actually signs a Merkle root over the individual payloads. This allows the complete signature that includes all payload messages to be shortened to only the payload message(s) of interest by keeping these, the root signature, and a Merkle path to each retained payload message.

It is critically important that signing twice using the same private key is avoided: Not only would such a double-signing using a one-time signature scheme weaken its security but tolerating it would allow a miner to simultaneously work on more than one fork of the blockchain. This is a problem if mining is not limited by computing power as it is in proof-of-work schemes. The less computationally expensive alternative that also is less prone to mining power concentration is proof-of-stake.

6 Proof-of-stake

Proof-of-stake schemes work by allocating mining power according to the stake that a miner has in the blockchain. Usually, using a cryptocurrency implemented by the blockchain technology, either the current balance or the integral of balance over time is chosen where the balance refers to the amount of cryptocurrency held by the miner's account or address. Proof-of-stake schemes are not the only alternative to proof-of-work schemes; in particular, the Byzantine consensus algorithm Algorand [Chen and Micali, 2016, Gilad et al., 2017, Chen et al., 2018] promises to deliver a computationally even cheaper alternative. Due to its novelty and significant implementation effort, subtlety, and resulting security uncertainty, we consider it only for a future, improved version of the blockchain technology presented here which originally shall employ a proof-of-stake scheme. The successful miner shall create a signature over the mined block, a proof of stake, and its key change. This data is then propagated via a gossip protocol.

Proof-of-stake schemes are not automatically secure. The foremost problem lies in the ability of miners to expend mining effort on more than one branch of the blockchain. There is an incentive for it as branches that are behind have a nonzero probability of still becoming the longest chain; if all nodes behave according to this incentive and mine on multiple (or even all known) sidechains, the usual and highly desirable assumption that this nonzero probability is at least practically negligible no longer holds. Hence a mechanism must be provided that at least discourages, and ideally completely eliminates, such behavior of mining on branches that should be abandonned. One such approach is to penalize mining on more than one branch. This incidentally corresponds to never using one-time-signatures more than one time which would be required

whenever a node wishes to increase the number of branches on which it mines. We propose to introduce proofs of conviction with which any node that becomes aware of another node's wrongdoing (such as reusing keys to issue more than one one-time-signature using the same key) can reliably report this wrongdoing to all other nodes, allowing actions such as evicting the convicted node from further participation.

We propose to allow a potential miner to mine a block whenever the hash of the concatenation of its public address and an ever increasing block number is smaller than its stake measured in a special kind of tokens to be discussed later in section 11. This implies that, on average, there will be one minr per block number but also that some block numbers will have no or multiple miners. This can be handled by assigning a block height that is a function of this hash value and the token balance of the miner; for the following discussion, we assume that a step function is chosen that is always 1 if the miner is authorized to mine and 0 otherwise. We call this function's sum over all blocks of a blockchain the height of a blockchain. One might worry that this allows forks to be grown retroactively: Blocks for which two or more miners exist might have one of them mine their block very late, creating a fork in a seemingly established, already longer blockchain. However, in the typical case that this established blockchain has a mined block for almost all (or at leat more than half) of the ensuing block numbers, it is impossible to grow the new fork to the same height within the same range of block numbers: The majority of potential miners is already comitted to the new block and would have to issue a second WOTS for the same public key which, as discussed above, is prohibited in an effective manner by proofs of conviction. Hence an established blockchain is secure against deep forks if it fulfills a certain condition. For the example block height function, this condition is that the majority of block numbers must be used in blocks; nodes should monitor this ratio and refuse or delay mining on forks where this is not given. Note that the actual condition is a bit more complicated: Not an absolute majority but a majority over all known forks is sufficient and it is necessary to lower the bound according to known forks to prevent mining from stalling in the instance that many forks exist.

7 Proofs of conviction

There are two kinds of possible misbehaviors that need to be revealed in a provable fashion by proofs of conviction. One is the issuance of false endorsement proofs. The other is the reuse of a key for issuing a signature.

An incorrect endorsement proof can be revealed by repeating it and revealing the relevant set of input and output data that leads to those hashes contained in it that do not respond to the correct output data. The correct output data of the smart contract invocation in question can then be calculated by every node by executing the smart contract. Game-theoretically, this behavior comes with no risk to data privacy if an incorrect endorsement is associated with a punitive event (because then verifier nodes are incentivized to work correctly, ideally to the point of using fault- and hence glitch-tolerant programming techniques). This punitive action ideally includes exclusion from the network. The proof of conviction described above should automatically trigger the application of this punitive effect by all honest nodes; for example, it could trigger the immediate inclusion in a certificate revocation list local to each node (and, eventually, also in a global one managed by the certificate authority). We will clarify what we mean by certificate and certificate authority in the following section.

The other kind of conviction proof consists

of two (maximally reduced) signatures using the same public key. As verifying that the same key is used reduces to calculating the public key from each signature, no further information is required (the public address to be excluded by inclusion in a certificate revocation list is contained in each signature, whose validity for the signed message must naturally also be checked). The included message can be maxiamally reduced by only including the public address and the Merkle root of the signed statements; details such as the signed key change or other signed message components are not required, should be pruned to reduce the computational load on other nodes, and must be pruned in order for the conviction proof to be recognized as valid (any ambiguity about this matter would create a complicated situation where nodes effectively follow different branches of an immediate fork in node membership that could later develop into a real blockchain fork; the followed branch would then depend on what choice is implemented in their software).

8 TLS, certificates and groups of nodes

To ensure a transition path towards postquantum transport security, without knowing of a scheme we trust to certainly provide it in an efficient and overall compelling way, we propose to use the standard transport layer security (TLS) for communication between nodes: We expect TLS to develop a future encryption (and authentication) option offering postquantum security overall. In case of privileged information, a node must first identify the node to which we are connected without being succesptible to a man-in-the-middle attack. This is only possible using a certificate authority (CA) system underlying this TLS communication. In order for this system to operate as trustless as possible, we propose that every node can become a CA to be trusted with regard to certifying the identity of nodes allowed to receive sensitive data owned by this node. This provides redundancy: In case there are multiple owners of the same data, each can act as the relevant CA (and each must be recognized as such by other nodes).

We hence propose that any node authorized to view private data can get a certificate from the CA that owns this data where the certificate obtained identifies the authorized node by an identifier we call its public address. This public address is assigned upon generating the first WOTS by a node and is the public key corresponding to this one-time signature: It is self-assigned but cannot be forged (without key compromise). It persists in the transport encryption side by the CA-issued certificate and in the blockchain side by repetition in each signature (which is only accepted as valid if the WOTS' public key is identical to it or if its signed statements include a previous signature featuring the same public address).

Each CA shall also act as a resolver that nodes subscribing to a side-chain with data owned by the CA can use to join a gossip protocol among other such nodes. This solves the problem of peer discovery on a side-chain in the following way. Initially, each potential CA persists into the mainchain its self-signed certificate along with the internet protocol (IP) address where it can be reached; these self-advertisements are repeated whenever the IP address changes. A node wishing to subscribe to this side-chain reads this certificate and the corredponding IP address and initialized a TLS connection to it, checking the certificate used and aborting if it does not correspond to that found in the blockchain. If the node is to take up a privileged role (i.e. become a member of the knowledge group or a verifier node), it also communicates a certificate signing request (CSR) to the CA. This communica-

tion indirectly becomes part of the blockchain because—to prove its public address and hence identity—the requesting node must sign this request with a WOTS and hence must include a key rotation statement that must be included into the blockchain.

9 Side chains and global address space

The previous sections focused on the idea of a single blockchain. This does not scale because even the weakest node will have to process all transactions in order to maintain the trustless nature of the blockchain. We now extent the notion of a single blockchain to organizing blocks into largely independent strands of separate blockchains that we will refer to as side-chains. In each side-chain we maintain different linked chains of key rotations such that an actor has different public keys for each side-chain.

Each smart contract has, associated with it, a tree of subset blockchains we call side-chains. This tree is incorporated into a global address space where its root resides at a global address given by the hash of the data of which the smart contract consists (we envision that even identical smart contracts will be able to have unique hashes due to the ability of appending a nonce to the data representing the code of which the smart contract consists). It is a ternary tree. This is chosen for ready representation as a Merkle-like tree where each node has an associated hash value formed as the hash of the ordered concatenation of its 0-associated child node, a different Merkle root (of its associated side-chain if present or no data otherwise), and that of its 1-associated child node. Below the Merkle root sits an ordinary Merkle tree, giving rise to addresses that, written in a ternary numeral system, only uses the digits 0 and 1 except in exactly one place where it has the digit 2 to mark the transition between the Merkle-like and the true Merkle tree spanned by the unified (global) address space.

This organization allows giving Merkle (or rather Merkle-like) paths as proof of data presence and hence allow applying endorsement proofs in the same way as already discussed for a single blockchain. It does, however, bring the complication that a transaction might access data of more than one side-chain. This can be solved by allowing to act as verifiers only the intersection of the sets of verifiers of each of the side-chains involved. Sensible smart contract design should chose the verifier sets and especially their quorum conditions such as to ensure that the resulting subsets reach a quorum, ideally comfortably (i.e. with a sufficient margin) to allow for fault-tolerance. The process of making a multiple side-chains spanning transaction is as follows. First the executor node signs a suitable test data set with each of the required signatures and key rotations, one from every side-chain involved. Then the set of verifier nodes calculate and sign the resulting output data hashes. Finally, the executor node publishes a revelation statement that finalizes the transaction when mined.

10 Data storage and deletion

Having public proofs about private data, it is possible to organize a blockchain where only transaction executors and knowledge groups can actively access private data (and sets of verifier nodes become aware of it) yet every node can verify that all transactions of a blockchain were executed correctly. We solve the problem that non-verifier nodes outside of the knowledge group associated with private data must eventually identify the identity of private data used as input to smart contracts and that present as output of an earlier, valid

smart contract execution by sharing a blinded hash of the data with all nodes. The blinding is performed by always prepending the data with a randomly generated nonce of the same bitsize as the hash. The hash is shared by incorporating it into the blockchain.

The members of a knowledge group communicate amongst each other the contents of a distributed key value store that we call a temporary unhasher. It resolves a hash to the nonce and data used to create it or to the information that this data has been deleted. The unhasher is semi-immutable in that the stored data can be overwritten once but only with an empty value (i.e. a value having a length of zero bits) that indicates deletion. The communication to update the unhasher shall be performed in a postquantum secure manner; we envision the usage of TLS in the expectation that this standard will be amended to permit the (exclusive) use of postquantum secure cryptography.

Each local unhasher operated by a node can hence, for each address, be described as a state machine with states "unknown," "known" (nonce and data), and "deleted." When encountering messages about a certain state that are exchanged (transport encrypted by TLS) between nodes of a knowledge group, between the locally present state and the communicated state, the one that appears later in the list of states wins out (see table 1).

To ensure compliance with local laws (such as the EU's GDPR or some countries' requirement to store personal information of citizens only inside their respective country), the nodes forming the knowledge group and the set of verifiers must be choosen accordingly. In particular, all verifier nodes must have committed (e.g. by contractual agreement) to not store the unblinded fake and real data or their hashes at all. The member nodes of knowledge groups must have committed to a similar agreement to honor the prescribed state machine in the sense that deleted data is indeed approriately deleted and not maintained as a copy, backup, or similar.

State	Message that triggers a transition to this state
deleted	deletion request (signed by anyone with write access to the data)
known	valid transaction (with endorsement proof, revealed and blinded input data as incorporated into the blockchain), further accompanied with unblinded output data for this address
unknown	none (initial state)

Table 1: Possible states of each address in a local unhasher (key-value store), in order of decreasing priority. The state with highest priority wins if different nodes know different states. Otherwise, state transition only occur as described in the text.

11 Naïve token example

It is trivial to implement a cryptocurrency or token with full public visibility using our scheme. Despite the fact that we call this a naïve (or public) token, it fills an important role: Balances to be used in the proof-of-stake scheme must be public anyways, as must any transfer thereof that is to be performed automatically as a penalty by a convition proof. In order to avoid the added complexity of defining a native token or cryptocurrency for this purpose, we propose to standardize a token of this public kind to serve that purpose.

Having such a privileged token, we propose to also use it for a similar purpose as "gas" is used in Ethereum, namely to incentivize nodes to collaborate towards executing and mining smart contract invocations. The realization is necessarily slightly more complex in Ethereum

Field	Description
ID	The byte 00_{16} indicating that this message is a signature.
Address	A hash-sized public address for which this signature can sign. For the first ever signature, this is equal to the public key associated with the WOTS used; for all subsequent signatures (that must involve a key rotation) by the same node, it must remain the same.
Statements	A Merkle tree where each node is given in one of three ways, and should include the previous signature (reduced to be over a Merkle root): 1. The byte 00_{16} indicating that the node's two children follow. 2. The byte 01_{16} indicating that a hash follows (a pruned branch). 3. The byte 02_{16} indicating that data (a nonce of hash-length and a data structure) follows.
WOTS	A Winternitz one time signature using up to 2^8 repetitions of the hash function.

Table 2: Data structure "signature."

Field	Description
ID	The byte 01_{16} indicating that this message is a key change.
public key	(A hash of) the public key for the next WOTS.

Table 3: Data structure for statement "key change."

as we wish to come closer to rewarding all participating nodes than by just setting aside an amount of tokens for the miner. We propose to address this by allowing the executor of a smart contract invocation to include more than one invocation, with at least some of them assigning tokens to the set of verifier nodes and to a pseudo-address from which the contract allows the successful miner to transfer tokens towards its balance.

The realization can exactly mirror what one would do to implement e.g. a ERC20 or ERC223 token in Ethereum: One creates a smart contract using an address space that includes every node as a node of the knowledge group and in which a hashmap or similar data structure maps public address to an integer denoting the amount of tokens held by that address. This smart contract must, at a minimum, define one function to transfer tokens from the caller's address to another address given as a parameter.

12 Private token example solving the real-time gross settlement problem

Usually, one might be reluctant to reveal one's balance, certainly not on a per-transaction basis that, depending on the nature of the transactions, might reveal very personal information or allow linking the transaction to a personal identity. If one adds to the usual finality of cryptocurrency transfers a near-perfect bank secrecy (which is imperfect only by allowing a regulator to see the balances and transfers), one arrives as what is known in the financial world as the real-time gross settlement problem where the technical term real-time roughly translates as faster than historical inter-bank money transfers and guaranteed to be within the same working day. An investigation into the usefulness of permissioned blockchain tech-

Field	Description
ID	The byte 02_{16} indicating that this message is a test data set.
Bit count	An unsigned 32-bit integer indicating the number of bits in the ensuing prefix. The length of the prefix as stored in the data structure is rounded up to the next full byte.
Prefix	The address prefix indicating the (ideally tightest) bound to which access by the smart contract function to be executed is limited. Putting this data early in the data structure allows the extraction of the knowledge group and the set of verifier nodes even before the full data structure has been read or transmitted.
Address	The address of the smart contract to be invoked.
Invokation	A special address space that is filled with, in this order starting at address zero, a hash of the invoked function's name, the value of its first parameter, that of the second parameter, etc. This is saved or transmitted in the same format as each of the data sets, see there.
Set count	A byte indicating the total number of input data sets (which should include the real data set).
Data sets	A concatenation of all data sets, each of which is a Merkle tree of the same format as used in signatures, see table 2, with data given in non-hashed form if and only if this data structure is used in communication with nodes in the knowledge group or the set of verifier nodes. Otherwise, of the entire Merkle tree only the root hash is included. In either case, only the actually referenced data is included, yielding a different root hash than used for keeping track of the blockchain's overall status.

Table 4: Data structure "Test data set" (for phase 1 of the two-phase transaction commitment).

nology for this purpose under the additional requirement to become fault-tolerant has been performed by de Vilaca Burgos et al. [2017] with the result that neither Hyperledger or Quorum fully meets the requirement due to an inability, at least in the chosen approach, to prevent double spends in the event of a failure of a single node operated by a financial regulator.

We demonstrate by construction that our proposed blockchain technology is able to solve the real-time gross settlement problem. In this example, as in the one cited, each bank shall operate exactly one node and, unlike in the cited example, multiple nodes are operated by agencies trusted or contractually obliged to keep the secrecy of transactions (e.g. a banking regulator, non government organizations, a consortium organization jointly owned by all participating banks, etc.). The initial setup includes a smart contract with functions to be detailed as they are called in this example.

Initially, all banks (and any one joining in the future) starts out with a zero balance. This is realized in the smart contract by keeping a balance in an address space to which only the individual bank is in the knowledge group and treating missing data (in state "unknown") as zero. Alternatively, any other initial token distribution can be trivially implemented, for example to mirror the individual banks' posessions at the day of inception of the smart con-

Field	Description
ID	A byte equal to 03_{16} identifying this as a block to be incorporated into the blockchain.
Prior	The hash of the previous block.
Authorization	
Content signatures	Concatenated signatures (of statements with ID type 4 and 5) that form this block which is valid if and only if it is validly signed, the prior refers to a valid block, the authorization is valid, every signature is valid, every content signature is maximally reduced to exactly one statement of ID type 4 or 5, and every such signature is relevant in the sense that the blockchain's status would be different if it was not present.

Table 5: Data structure of the statement "block found."

Field	Description
ID	A byte equal to 04_{16} identifying this as an endorsement proof.
(Rest)	The same data structure as contained in the statement "Test data set," see table 4.

Table 6: Data structure of the statement "endorsement proof."

Field	Description
ID	A byte equal to 05_{16} identifying this as an executor's revelation.
Remainder 1	The same data structure as contained in the statement "Test data set," see table 4, except that the field "set count" must be one and the Merkle tree given as "data set" here containing the full data corresponding to the prefix-identified side-chain's current state. The Merkle root will hence be already known to other nodes.
Remainder 2	The analogous Merkle path not for the input but for the output data, again in the context of the global Merkle root.

Table 7: Data structure of the statement "executor's revelation." Note that in general, it will be signed by a miner and not by the executor because the miner may need to update the Merkle path according to changes made by transactions revealed earlier.

Field	Description
ID	A byte equal to 06_{16} identifying this as a smart contract (and side-chain) genesis.

Table 8: Data structure of the statement "genesis."

Field	Description
ID	A byte equal to 07_{16} identifying this as a CA certificate.
size	An unsigned 32-bit integer equal to the number of bytes in the ensuing certificate.
certificate	The certificate data, as standardized by TLS.

Table 9: Data structure of the statement "CA certificate."

Field	Description
ID	A byte equal to 08_{16} identifying this as a CSR.
size	An unsigned 32-bit integer equal to the number of bytes in the ensuing CSR.
CSR	The certificate signing request's data, as standardized by TLS.

Table 10: Data structure of the statement "Certificate signing request."

tract. In the following, it is assumed that at least one participant has been initialized with a non-zero balance; if this is not the case, a function (that shall not be detailed further) of the smart contract must be implemented to arrange to supply banks with tokens. Balances will be implemented as unsigned integers and hence can never become negative.

A bank wishing to initiate a transfer of some of its tokens to another bank shall call a function of the smart contract that accesses, after checking for sufficient balance, another knowledge group formed of the two banks involved in the transaction. The set of verifier nodes can be formed from a subset of the trusted agencies augmented by the two banks themselves; the smart contract function shall, as all smart contract functions, check that the verifier set includes sufficiently many of the trusted agencies. In detail, the initiating bank signs a transaction that includes the called function and a dataset containing, among decoys/fakes, the blinded hashes of the real input and output data. The output data, in this private address space shared between the two banks, shall resemble a check: It contains the amount to be transferred, the benefiary node (of the recipient bank), and a signature of the sending node (of the originator bank).

Cashing this check is done by the recipient node by calling another function of the smart contract. This function first asserts that the check is appropriately signed and lists the caller as beneficiary node. If this is the case, it increases the caller's private balance (in its private address space) accordingly (with checking for overflows optional if it is otherwise guaranteed that the total token supply cannot cause an overflow).

This scheme affords near-maximum privacy: The existence and originator of a transaction is shared between banks and otherwise only with watchdog nodes that have entered a legal agreement about implementing their nodes without memory of such information. It has the potential to offer hyper-realtime settlement and is as secure as the unwillingness of all involved actors (at least one bank and the minimum set of verifiers the smart contract acceots) to all collude at the risk that a single whistleblower among them can have colluders banned from further participating in this system (plus the further penalty of loosing all public tokens). Unfortunately, we are unaware of a way to design a system that generalizes the loss of public tokens to the loss of the private tokens that are normally used in this example.

13 Conclusion

Given the lack of existing solutions to satisfy the requirements experienced in a real, commercially productive blockchain application, Lition has designed a proprietary blockchain solution. This new blockchain to solve these shortcomings has been described in this document. Lition has presented the design for a blockchain network and minimum requirements for a governing agreement among a privileged subset of the nodes' operators to ensure that sensitive and private data can be handled and securely deleted on demand and even connected to smart contracts for deletion. The guiding design criteria of postquantum security for data integrity, data minimization under the constraint of providing fault tolerance, privacy of sensitive data, a provision to delete all occurrences of sensitive data, and the freedom to join as a (non-privileged) node without any special provisions or legal obligations have been described, along with the novel approach to solve the security issue of private data (and private transactions) in a blockchain technology by providing a technique to publicly prove the correctness of a transaction involving private data without revealing the private data. This proof is not fully trustless but is of a probabilistic nature and similar to non-interactive zero knowledge proofs (NIZKPs).

References

Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002. ISSN 0163-5700. doi: 10.1145/564585.564601. URL http://doi.acm.org/10.1145/564585.564601.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. whitepaper, 2009, 2009. URL https://bitcoin.org/bitcoin.pdf.

Adam Back. Hashcash-a denial of service counter-measure, 2002. URL http://www.hashcash.org/papers/hashcash.pdf.

Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997. URL http://firstmonday.org/ojs/index.php/fm/article/view/548/469.

Vitalik Buterin. A next-generation smart contract and decentralized application platform—Ethereum whitepaper, 2014. URL https://web.archive.org/web/20161021061647/https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

Vitalik Buterin. EIP 101 (Serenity): Currency and crypto abstraction, 2015. URL https://github.com/ethereum/EIPs/issues/28.

Patrick Mylund Nielsen et al. Quorum whitepaper, 2016. URL https://web.archive.org/web/20170807171314/https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf.

Ram Jagadeesan, Mic Bowman, Binh Nguyen, Hart Montgomery, Stan Liberman, Murali Krishna, Vipin Bharathan, Alexander Yakovlev, Todd Little, Nathan George, and Tracy Kuhrt. Hyperledger architecture, volume 1, 2017a. URL https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

Ram Jagadeesan, Mic Bowman, Binh Nguyen, Hart Montgomery, Stan Liberman, Murali

Krishna, Vipin Bharathan, Alexander Yakovlev, Todd Little, Nathan George, and Tracy Kuhrt. Hyperledger architecture, volume 2, 2017b. URL https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf.

Serguei Popov. The tangle, 2017. URL https://iota.org/IOTA_Whitepaper.pdf.

Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework—whitepaper, 2017. URL https://github.com/w3f/polkadot-white-paper/blob/master/PolkaDotPaper.pdf.

Leemon Baird. Hashgraph consensus: Fair, fast, byzantine fault tolerance, 2016. Swirlds, Inc. Technical Report SWIRLDS-TR-2016.

Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5085-3. doi: 10.1145/3132747.3132757. URL http://doi.acm.org/10.1145/3132747.3132757.

Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016. URL https://arxiv.org/abs/1607.01341.

Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. ALGORAND agreement: Super fast and partition resilient Byzantine agreement. Cryptology ePrint Archive, Report 2018/377, 2018. URL https://eprint.iacr.org/2018/377.

Aldenio de Vilaca Burgos, Jose Deodoro de Oliveira Filho, Marcus Vinicius Cursino Suares, and Rafael Sarres de Almeida. Distributed ledger technical research in Central Bank of Brazil, 2017. URL http://www.bcb.gov.br/htms/public/microcredito/Distributed_ledger_technical_research_in_Central_Bank_of_Brazil.pdf.

Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptol. ePrint Arch., Report 2018/046, 2018. URL https://eprint.iacr.org/2018/046.

Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066, 2017. URL https://eprint.iacr.org/2017/1066.

Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. Cryptology ePrint Archive, Report 2018/475, 2018. https://eprint.iacr.org/2018/475.

Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 755–784. Springer, 2015.

Dominique Unruh. Post-quantum security of Fiat-Shamir. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–95. Springer, 2017.

Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and

signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer, 2015.

Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS—A practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011a.

Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the Winternitz one-time signature scheme. In *International Conference on Cryptology in Africa*, pages 363–378. Springer, 2011b.

Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394. ACM, 1990.

A Blockchain Compactification

It may be possible to reduce the size of the blockchain history by storing most aspects of the endorsement proofs, the repeated verifier signatures, in a different location that does not need to be kept for the entire history: A single verifier signature is sufficient to reconstruct the history of (hashed and blinded) private data, provided that the identity of every such verifier signature and the absence of a proof of conviction is also saved with the blockchain history. We estimate that this will reduce the storage requirement for a node wishing to keep a full history by more than an order of magnitude (consider, for example, that every data set is typically accompanied by two decoy data sets and a quorum of five verifier nodes is typically stipulated which translates into three times five data sets in the complete endorsement proof of which only one would need to be kept in the blockchain history for completed transactions).

This scheme can be taken to the extreme: It already implies that such a node can only recostruct the history of the blockchain network's state but no longer prove it to other nodes. For the same result, it would be sufficient to merely record every smart contract execution's outcome. In particular, by keeping the executor's signature and one verifier's signature, it can still be proven that the state is not forged to assume an arbitrary state (although it is no longer possible to prove that individual transactions did or did not succeed in changing the state).