Quadrans Blockchain

Yellow Paper - v0.2

Michele Battagliola[*], Andrea Flamini[†]
Riccardo Longo[‡], Alessio Meneghetti[§], Massimiliano Sala[¶]

February 18, 2021

Abstract

In this document we present the design of Quadrans, a blockchain platform for supply chains and IOT devices, capable of performing trustworthy and transparent operations. We aim to achieve scalability without losing security or decentralisation. Furthermore we use an innovative approach to the cryptographic layer of the ledger that gives back control to users allowing great flexibility, achieving also resiliency in case of disrupting cryptanalysis advancements. We also natively support Post-Quantum algorithms and enforce their use in block signing, while supporting lightweight encryption for IOT devices.

Contents

[*] michele.battagliola@unitn.it, *Department of Mathematics, University of Trento*
[†] andrea.flamini.1995@gmail.com, *Department of Mathematics, University of Trento*
[‡] riccardolongomath@gmail.com, *Department of Mathematics, University of Trento*
[§] almenegh@gmail.com, *Department of Mathematics, University of Trento*
[¶] maxsalacodes@gmail.com, *Department of Mathematics, University of Trento*

1 Introduction

The potential of distributed smart contracts and of the blockchain (a public data structure very resilient and openly auditable) has been apparent since the very beginning [But13; CBB16], and great interest has risen especially in particularly suitable sectors such as financial institutions and supply chains. Nowadays there are various platforms that offer interesting solutions specifically tailored to improve or facilitate supply chains, but this sector is huge and variegated, and there still are many problems not yet addressed.

Here we propose the technical specification for a decentralised platform for smart contracts [Cos+19b; Cos+19a] with a specific focus on the needs of Industry, complex supply chains, IOT devices, with significant designing efforts on the security of this platform and its cryptographic and protocol-related aspects. The proposed blockchain is called Quadrans Blockchain (QB) and consists of three sub-blockchains, the SynchroChain, the MasterChain and the ShardChain. QB supports two currencies: Quadrans Tokens (QDT) and Quadrans Coins (QDC). Users possessing QDT enjoy special priviledges, and are called TokenHolders.

Background

We started working taking into consideration the very long and articulated food supply chain because it starts from crop fields and it ends to restaurants and shops. Transparency in the processing and geographical indications schemes such as *DOC* and *IGP* can add substantial value to a product, and customers grow ever more conscious of the importance of the origin of what they eat. This means that there are tangible economic incentives in adopting technologies that enforce a controlled, secure and publicly auditable supply chain. In this scenario there are many delicate aspects that need to be taken care of: data needs to be tamper-proof and reliably authenticated and verified, some information may be confidential and only shared with specific partners, participation has to be adequately rewarded while preventing and discouraging malicious behaviour.

2 Algorithms and Parameters Flexibility

Quadrans is a blockchain designed to support interaction between users with peculiar and heterogeneous characteristics: from IOT devices, that power monitoring and automatic update of product status in the supply chain, to end consumers and companies. Such diversity calls for considerable levels of flexibility when choosing cryptographic primitives, because the computational resources at the disposal of each actor may differ quite a lot.

Moreover the realm of information security is in constant evolution, so it is of paramount importance to have a flexible and resilient approach, capable of adapting to novel discoveries and improvements. In fact technological advancements may render current solutions insecure or open the way to much more efficient alternatives, and scientific research could have disruptive cryptanalytic breakthroughs or discover powerful new approaches.

For these reasons we adopt a flexible approach to the selection and usage of cryptographic primitives and their parameters, favouring some standard choices to optimise efficiency but allowing each user to employ its own choice, in order to balance between security and computational cost, or even to avoid suspicious constructions (i.e. fear of backdoors).

2.1 Smart Contract Kernel

To manage this ductile design we employ a system of special smart contracts that act like a sort of *cryptographic kernel*. In particular they define and encode the available algorithms for each type of primitive, and validate and organise their parameters.

Quadrans allows the usage of any algorithm of a non-static pre-approved list, which initially contains our starting selection. A smart contract updatable every epoch is responsible to introduce new approved algorithms and maintain a sub-list of up to 63 standards that are efficiently encoded, from a total of up to 2^{16} different algorithms (a limit that we consider over-conservative). For every scheme in the list, there is a dedicated smart contract that defines the operating parameters of the algorithm.

In more detail the smart contract specifies three standard sets of parameters that correspond to just as many levels of security:

- *Basic*: suitable also for less powerful devices and interaction with legacy applications;
- *Intermediate*: a middle tier that strengthens the security while balancing efficiency;
- *Enhanced*: a higher-security level, that is fit for more sensitive and long-term information.

In addition to these standards, users are allowed to adopt their own parameters, so that usage can be furthermore tailored to specific needs (for example users can adopt their own elliptic curves), submitting them to the smart contract. In fact the same smart contract is used also to validate and assess proposed sets of parameters, to ensure that they satisfy a minimum level of security (that can be lower than the one of the basic parameters, to reach out to users that might have stringent constraints).

This approach optimises storage space and validation effort, since each set is stored and validated only once. Each smart contract in the kernel specifies a list of set of parameters and three indexes that identify which entries correspond to the three standards, so that the standard can easily be updated to adapt to cryptanalytic breakthroughs maintaining optimal encoding for these preferred sets. To preserve integrity and backwards compatibility the previous standard indexes are also shown by the smart contract, alongside the timestamp (epoch

block index) of the moment the outdated value has been superseded. At the beginning these standard indexes are `00000000`, `00000001`, `00000002`, so the first three entries in the parameter list are initialised with the starting choices for the three standards, later the list can be expanded up to $2^{32} - 1$ (more than four billion) entries than can therefore be indexed with a four-byte value.

The other information stored in the smart contract is a hash-link to the encoding specification that defines how to actually encode parameter sets and interpret related values (e.g. public keys for a signature scheme). Users can then submit their parameter set of choice to the smart contract, which will append it to the list (encoded according to the specification) if the entry is not a duplicate and if the parameters pass a series of tests that establish that they properly define a correct instance of the signature algorithm and sustain a minimum level of security.

2.2 Encoding

The encoding of algorithm choice has been developed so that the majority of cases require a single byte, that is called the *discerning byte*. In particular this byte encodes the index of the standard algorithm (in the six most significant bits), and the standard set of parameters (in the crumb composed by the two least significant bits). The crumbs 01, 10 and 11 correspond to the standard parameters associated to the aforementioned levels of security, namely the lower, the middle and the higher level; the crumb 00 signals a non-standard choice and therefore it must be followed by additional 4 bytes that specify the index of the chosen parameters in the smart contract list. Similarly the index 000000 for the standard algorithm signals a non-standard algorithm, so 2 additional bytes that specify the index of the chosen algorithm. A null byte indicates a non-standard algorithm with non-standard parameters, so is followed by 6 bytes: first 2 for the algorithm, then other 4 for the parameters.

So the *discerning byte* (or the following 2 bytes) identifies the smart contract that lists the parameters and specifies if a standard set is used; if that is not the case the index of the chosen set follows, so one can retrieve the value of the parameters from the smart contract. At this point algorithm and parameters are established, so the following bytes can be correctly interpreted referring to the specification defined in the smart contract (e.g. as a public key).

Example 1. Suppose that the following string is the hexadecimal representation of a public key:

```
0582006e9398a6986eda61fe91674c3a108c399475bf1e738f19dfc2db11db1d28
```

The first byte (`06`) identifies the scheme and therefore how to process the rest. Its binary representation is:

```
000001 01
```

The first (and most significant) six bits identify the first standard algorithm defined in the *digital signature algorithms* smart contract (e.g. ECDSA), the last (and least significant) two bits say that the basic-level standard is used (referring to the *ECDSA parameters* smart contract, e.g. the curve *secp256k1*). This means that what follows the first byte is the encoding of the public key, since the curve parameters are established and known, note also that they define the

length of the encoding, so the parser knows how many bytes to read. ECDSA's public key is a point of an elliptic curve, and the parameters used imply the quadratic residuosity of the second coordinate, so the remaining sixteen bytes of the public key are the first coordinate of the point.

3 Users

Users of Quadrans blockchain are identified by their address. Such an address is derived from a public key whose correspondent private key is owned by the user it identifies. Quadrans allows its users to choose from various digital signature algorithms for transaction signing, therefore the format of such public keys can vary but the format of the address is standardised for everyone. With "user" we always mean a hardware/software interacting with the Quadrans Blockchain, rather than the actual person who owns the hardware/software.

3.1 Digital Signatures

Quadrans exploits the flexible adoption of various digital signature algorithms as described in Section 2 to let each user balance security and computational cost, moreover we distinguish between transaction-signing and block-signing, requiring much higher levels of security for block signatures.

In fact we can assume that blocks are created and validated by fairly powerful users of the network that can surely afford to invest more resources in order to strengthen the long-term safety of the whole chain. Considering also the relative proportion between blocks and transactions (the former comprises hundreds of instances of the latter) we can employ signatures that are much more space-consuming for the blocks without debilitating the overall efficiency. For these reasons Quadrans blocks must be signed with Post-Quantum secure algorithms.

3.1.1 Available Digital Signature Algorithms

The initial selection of approved schemes is the following, where we present public-key length, signature length, and compare their combined length (w.r.t. a standard security level of 128 bit).

- `ECDSA` [JMV01] is the widespread standard, especially in the blockchain world. This means that implementations are widely available, even with optimisations for low-power devices. Both public keys and signatures are very short (32 bytes once the curve has been fixed), and together with its low computational cost this makes ECDSA an efficient signature. However the security is based on the difficulty of the discrete logarithm (DLOG) over the group of points of an elliptic curve, and Shor's algorithm for quantum computers breaks this assumption [Sho94]. Therefore this choice is not suitable for block signing, where a post-quantum secure algorithm is required.

- `EdDSA` [Ber+12; Ber+15] is another signature scheme based on elliptic curves, that is gaining advantage in terms of usage on the more classical ECDSA. The particular types of curves employed (the so called Twisted

Edwards curves [Ber+08]) make computations less susceptible to side-channel attacks and allow for further optimisation especially in batch signature verification. Another practical advantage is the deterministic nature of the signature that avoids the pitfalls of incorrect generation of random parameters, which have led to complete breaches of ECDSA signatures in the past [Sch15]. Key and signature sizes are equal to those using ECDSA, the computational cost is comparable or lower, and the security is based on the same mathematical assumption: so EdDSA is an interesting alternative to ECDSA, but is likewise unsuitable for block signing.

- `CRYSTALS-DILITHIUM` [Duc+18] is a lattice-based post-quantum signature algorithm. The primary advantage of Crystals-Dilithium over similar PQ proposals is the avoidance of Gaussian Sampling [MW17], which is a primitive easy to misuse [Bru+16] and that may lead to weakening attacks hard to detect. Public keys are a little more than 1 KB long, with signatures of roughly 2 KB. This is almost two orders of magnitudes more than elliptic curve-based (EC) schemes, but it is a price to pay for post-quantum security. This scheme has one of the lowest combined length of public key and signature among PQ algorithms.

- `FALCON` [Fou+18] is also a lattice-based post-quantum signature algorithm. It features small signatures and key sizes against other PQ schemes, but it uses Gaussian Sampling that has some potential issues (see above). Public keys are little less than 900 B, and signatures around 650 B: the shortest combined length among NIST round 2 candidates, but they are still an order of magnitude longer than EC signatures.

- `SPHINCS+` [Ber+19] is a post-quantum signature algorithm, this time based on hash functions. The construction makes few security assumptions (so it should be more resilient), and the implementation is quite similar to a scheme which has already been adopted internationally (XMSS [BDH11]). However implementations should account for fault attacks, that could allow signature forgery at a reasonably-low computational cost. Public keys are only 32 B long, however signatures are quite long (around 8 KB).

- `LUOV` [Beu+18] is a post-quantum signature algorithm, based on multivariate polynomials. It is an evolution of a well studied older scheme, and it is conservative in its margin of safety, which ensures that the algorithm can survive modest improvements to cryptanalysis. Like other multivariate schemes, it features small signatures but has large public keys: signatures are only around 250 B, and public keys are 1.5 KB long.

- `PICNIC` [Cha+17] is a post-quantum signature algorithm, based on a combination of block ciphers, hash functions, and zero-knowledge proofs. The scheme is designed to be extremely compact for hardware implementations, allowing for easy hardware acceleration to make the cipher lightweight on low-power devices. It is also nearly compatible with current X.509 certificate schemes [ITU19], and claims integrated tamper resistance. Similarly to SPHINCS+, public keys are very short (32 B), but signatures are definitely longer: around 12.5 KB (signature size is not fixed with this scheme).

- `RAINBOW` [DS05] is another multivariate post-quantum signature scheme. It has a simple mathematical construction, which eases a smooth and correct implementation, and it is a quite old scheme (first proposed in 2005). This means that its security has been widely researched and has withstood the test of time. It targets computational efficiency and small signature sizes, but has large key sizes: signatures are only 64 B, but public keys are around 58 KB.

The selection has been made by analysing the current standards and the state-of-the-art of digital signature algorithms, while referring in particular to the ongoing NIST selection regarding post-quantum algorithms [ST]. All post-quantum schemes in our list have advanced to the second round of the NIST selection. In defining our list we gave priority to low combined-length of public key and signature and variety on the underlying security assumptions, to improve resiliency against future unexpected attacks.

Besides the smart contract that defines parameter sets, Quadrans requires that for each digital signature algorithm there also is a smart contract that allows to verify any signature computed with that algorithm, so that it can be used as a reference implementation (although not optimised since it is general purpose for all parameters) and as last resource to verify signatures for wallet implementation that do not natively support that algorithm.

3.2 Addresses

User addresses in Quadrans are strings of hexadecimal characters that encode some bytes derived from the description of the user's authorised keys and common name, plus some other checksum bytes that help avoiding mistakes and clerical errors[1]. This means that users may check the formal validity and coherence of an address even offline.

3.2.1 Authorised Keys

Addresses support natively multi-signatures and even more complex access policies. Addresses embed information about the public keys whose corresponding private keys can withdraw from these addresses' balances. Simple addresses specify a single public key, and a signature verifiable with that key is sufficient to authorise any operation on behalf of these addresses. However more sophisticated addresses may specify various public keys (even of different signature protocols) and a policy that indicates which signatures are required to authorise transactions. For example an address may be associated to a public key for a post-quantum algorithm plus three other ECDSA keys and specify that authorisation may come through a single post-quantum signature OR at least two out of three ECDSA signatures.

[1] At this preliminary stage we define addresses as a simple string of 64 hexadecimal characters encoding the 256 bit digest of Keccak-256 (to achieve retro-compatibility with Ethereum [But13]) computed with the encoding of a single public key as input. However we intend to further study the matter and improve and expand the algorithm for address computation to achieve the goals presented in this section in a future release.

3.2.2 Common Name

Quadrans addresses embed a common name for their users. This avoids duplicated insertions of the same public key to the blockchain: once an address has been disclosed, all information (including the public key and the common name) are known to the community, hence further transactions can refer to address and common name only, without the need of specifying the entire public key.

Common names are not required to be fully human-readable: they can also encode IOT devices IDs in a way that only the owner may recognise and identify them, or even be random strings if some users wish to preserve their pseudo-anonymity.

4 Nodes

The Quadrans Blockchain is structured into three types of intertwined chains: a SynchroChain, a MasterChain, and multiple ShardChains (See Section 5). The Quadrans network is composed by two sets of active nodes:

- MasterNodes, which filter incoming transactions and are in charge of achieving a global consensus, maintaining the MasterChain and updating the Global State of the architecture, moreover they are also responsible of maintaining synchronization by managing the SynchroChain.
- Miners, which work in parallel on the ShardChains, validating transactions and running Smart Contracts, and are in charge of achieving a local consensus.

4.1 MasterNodes

To become a MasterNode, a TokenHolder needs to

- possess enough QDT;
- (participate and) win a PoS competition.

If a TokenHolder wins the PoS competition (possibly with the help of other TokenHolders) during epoch h, then it will be a MasterNode during epoch $h+2$. MasterNodes are in charge of:

- managing incoming transactions (they include the incoming transactions that pass a formal check into a Transaction Pool);
- achieving the Global Consensus:
  - validating Miners's work and constructing the MasterBlocks;
  - managing the Global State by aggregating the Local States;
- deciding the behaviour of the Quadrans Blockchain:
  - collecting bets from TokenHolders participating in PoS competitions;
  - collecting solutions of the PoW competitions from prospective Miners;

– deciding the number of Shards and other parameters, in order to optimise the workload of Miners, achieve the best possible throughput, and maintain strong security;

4.1.1 SynchroNodes

The SynchroNodes are MasterNodes that are in charge of managing the SynchroChain.

4.2 Miners

If a TokenHolder wins the PoW [MST20b; MST20a] competition during epoch h, then it will be a Miner during epoch $h + 2$.
Miners are in charge of managing ShardChains by:

- contacting MasterNodes to obtain the transaction pool;
- running Smart Contracts as specified by valid transactions;
- updating the Local State;
- creating ShardBlocks;
- reaching a local consensus on ShardBlocks not yet confirmed by the MasterChain.

5 Chain Structure

The Quadrans Blockchain is a set of blockchains, where Miners are divided into shards to work simultaneously on the computation of Smart Contracts, and therefore achieve a large throughput of executed transactions [Men+19]. The blockchains managed by Miners are called *ShardChains*. In addition, MasterNodes work on a higher-level chain, the *MasterChain*, collecting the states of ShardChains to achieve a global consensus. Finally, time is divided into slots of fixed length, and each slot is pre-assigned to a single Miner per ShardChain and to a single MasterNode. To coordinate the work on these chains and enhance safety, Quadrans has an extra chain, called *SynchroChain*, that marks the beginning of time-slots, time-stamps the other blocks, and defines the parameters of the ledger.

5.1 The SynchroChain

The consensus mechanisms that regulate the Quadrans blockchains rely heavily on time-related concepts, therefore it is necessary to coordinate and synchronize their execution. The SynchroChain acts as a Trusted Third Party that time-stamps the blocks on the other chains, and regulates the flow of time. In Quadrans time is discretized on two levels:

- *epochs* inside which parameters of the chains have fixed values, but these values can change from one epoch to the other;

- *time-slots* that regulate the production of blocks (one block per chain per time-slot). The number of time-slots in an epoch is a parameter, therefore different epochs can have a different number of time-slots.

The SynchroChain is controlled by SynchroNodes, that are chosen among the TokenHolders that are potential MasterNodes.

At the end of each time-slot the SynchroNodes produce a *SynchroBlock* that timestamps (and partially validates) all the blocks created during that time-slot. Consequentially the publication of a SynchroBlock marks the start of the next time-slot.

The last SynchroBlock of an epoch is called *EpochBlock* and contains additional information that regulates the operation on the chains:

- epoch parameters, fixing their values to be used in a future epoch (some values are dictated for the next epoch, some for the one after, and some for an epoch even further into the future);
- the information regarding the PoW competitions that regulate the ShardChain;
- the information regarding the PoS competition that regulates the MasterChain.

SynchroBlocks are referenced by the ShardBlocks created in the following time-slot, so it is necessary that the consensus on the SynchroChain is reached immediately, hence a BFT algorithm has been chosen [CL99].

Every SynchroBlock gives preliminary validation to the blocks of the other chains created in its time-slot (at most one MasterBlock and one ShardBlock per shard) by hash-linking them.

When an EpochBlock E_r is created during epoch $r-1$, MasterNodes add to their transaction pools the coinbase transactions corresponding to time-slots of Epoch r. Coinbase Transactions contain both the QDC to be created and the QDT that the TokenHolder has bet to become a MasterNode for Epoch r. These coinbase transactions are managed by Miners after the creation of the corresponding MasterBlocks. See Sections 5.2.
Linked to the EpochBlock there is a *Delta State*: information useful to reconstruct the current state. The knowledge of the state at the time of the previous EpochBlock and of the Delta State allows the correct reconstruction of the current state. This feature can be seen as the inclusion of check-points of the state, and it allows both fast verification of the correctness of the state and fast update of the state. Any new node can request only the Delta States of EpochBlocks and correctly obtain the current state.

5.1.1 SynchroBlocks

The structure of EpochBlock E_r is shown in Table 1.

5.2 ShardChains

The ShardChains are parallel blockchains where the Miners actually execute transactions and smart contracts. To manage the scalability, the number of ShardChains per Epoch (and therefore the degree of parallelization) is flexible. During epoch h there are N_h parallel ShardChains (a parameter specified in the EpochBlocks), and the Epoch lasts e_h time-slots, so during each Epoch a total number of $N_h \cdot e_h$ blocks can be created.

The selection of the TokenHolders that become Miners and create ShardBlocks during Epoch h is made through a PoW competition held during Epoch $h-2$, whose results are included in the EpochBlock E_{h-2}. In this way, Miners know in advance the time-slots in which they have to be active to create the assigned blocks. Each Miner active in a ShardChain is in charge of the creation of a maximum of $m_{\max,h}$ blocks. At each Epoch, the number of Active Miners is therefore at least $\frac{N_h \cdot e_h}{m_{\max,h}}$.

HEADER	
$H(\mathsf{Sy}^h_{i-1})$	hash-pointer to previous SynchroBlock
T_s	timestamp
R_i	root of the Merkle tree of DATA
$\mathtt{sig}_{\alpha_1}(\mathsf{Sy}^h_{i-1})$	
$\vdots$	signatures of the header (computed by SynchroNodes)
$\mathtt{sig}_{\alpha_k}(\mathsf{Sy}^h_{i-1})$	
DATA	
$H(\mathsf{Ma}^h_i)$	
$H(\mathsf{Sh}^h_{1,i})$	hash-pointers to the blocks created
$\vdots$	during ith time-slot of epoch h
$H(\mathsf{Sh}^h_{N_h,i})$	
EPOCH DATA	(only in EpochBlocks, i.e. if $i = e_h$)
`PoS DATA`	information that regulate
`PoW DATA`	the consensus on the other chains
`parameters`	value of parameters for following epochs

Table 1: SynchroBlock Sy^h_i created at the end of the ith time-slot of epoch h, by SynchroNodes $\alpha_1, \ldots, \alpha_k$. The signatures are formally put inside the header, since hash-pointers are defined as the hash value of a header.

Each ShardChain has the duty of managing transactions, that are divided into distinct pools according to their input address.
When Miners create blocks during their assigned time-slots, they update the local state, and then send a Delta State to the MasterNodes via an off-chain communication. The MasterNodes use this Delta State to update the Global State.

Remark 1. Miners have to be aware of the balance of each account whose address is managed by their own Shard (they can do this by keeping a copy of their ShardChain's local state), and have to check in the Global State for transactions coming from other Shards which may have changed the balance (they can do this by off-chain communication with MasterNodes).

5.2.1 ShardBlocks

The structure of ShardBlock S_{h,j_h} created by Miner m is shown in Table 2.

HEADER	
$H(\mathsf{Sh}^h_{\sigma,j})$	hash-pointer to ShardBlock $\mathsf{Sh}^h_{\sigma,j}$
$H(\mathsf{Ma}^h_s)$	hash-pointer to MasterBlock Ma^h_s
$H(\mathsf{Sy}^h_{i-1})$	hash-pointer to previous SynchroBlock
(x, K, c)	PoW data
R_{h,j_h}	Merkle root of executed transactions
$H(\Sigma^h_{\sigma,i})$	digest of the local state
$\mathtt{sig}_m(\mathsf{Sh}^h_{\sigma,i})$	signature of the header by Miner m
DATA	
$\mathtt{tx}_{h,j_h,1}$ $\mathtt{tx}_{h,j_h,2}$ $\vdots$	list of executed transactions
LOCAL STATE	

Table 2: ShardBlock $\mathsf{Sh}^h_{\sigma,i}$ created by Miner m during the ith time-slot of Epoch h, on the shard σ. $\mathtt{sig}_m(\mathsf{Sh}^h_{\sigma,i})$ is the signature of the first five entries of the header of the block $\mathsf{Sh}^h_{\sigma,i}$. The signature is formally put inside the header, since hash-pointers are defined as the hash value of a header.

5.3 The MasterChain

The MasterChain is a blockchain where the *MasterNodes* aggregate the computations of each ShardChain that have reached local consensus into a global state.

Since time is divided into Epochs of fixed length, during each Epoch h the MasterNodes create e_h blocks of the MasterChain. During its assigned time-slot, the active MasterNode creates a block of the MasterChain.
Each MasterBlock has the role of finalizing the State of the Quadrans Blockchain by putting together all local states updated by Miners during the creation of ShardBlocks. All information on the ShardStates are collected by MasterNodes via off-chain communication as soon as Miners create new ShardBlocks.
The consensus for the MasterChain is obtained by looking at the chain with larger weight.

5.3.1 MasterBlocks

The structure of MasterBlock M_t created by Miner α is shown in Table 3.

HEADER	
$H(\mathsf{Ma}_s^h)$	hash-pointer to MasterBlock Ma_s^h
$H(\mathsf{Sy}_{i-1}^h)$	hash-pointer to previous SynchroBlock
$H(\mathsf{Sh}_{1,i_1}^h)$	
$\vdots$	hash-pointers to finalized ShardBlocks
$H(\mathsf{Sh}_{N_h,i_{N_h}}^h)$	
$H(\Sigma_i)$	digest of the global state
T_i	timestamp
$\mathtt{sig}_\alpha(\mathsf{Ma}_i^h)$	signature of the header by Miner α
GLOBAL STATE	

Table 3: MasterBlock Ma_i^h created by MasterNode α during the ith time-slot of epoch h. The signature is formally put inside the header, since hash-pointers are defined as the hash value of a header.

References

[BDH11] 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*. Springer. 2011, pp. 117–129.

[Ber+08] Daniel J Bernstein et al. "Twisted Edwards curves". In: *International Conference on Cryptology in Africa*. Springer. 2008, pp. 389–405.

[Ber+12] Daniel J Bernstein et al. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89.

[Ber+15] Daniel J Bernstein et al. "EdDSA for more curves". In: *Cryptology ePrint Archive* 2015 (2015). URL: `https://eprint.iacr.org/2015/677`.

[Ber+19] Daniel J Bernstein et al. "The SPHINCS+ signature framework". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2129–2146.

[Beu+18] W Beullens et al. *LUOV signature scheme proposal for NIST PQC project (Round 2 version)*. 2018. URL: `https://github.com/WardBeullens/LUOV/raw/master/Supporting_Documentation/luov.pdf`.

[Bru+16] Leon Groot Bruinderink et al. "Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme". In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2016, pp. 323–345.

[But13] Vitalik Buterin. *Ethereum: a next generation smart contract and decentralized application platform*. `https://github.com/ethereum/wiki/wiki/White-Paper`. 2013.

[CBB16] Christopher D. Clack, Vikram A. Bakshi, and Lee Braine. "Smart Contract Templates: foundations, design landscape and research directions". In: *CoRR* abs/1608.00771 (2016).

[Cha+17] Melissa Chase et al. "Post-quantum zero-knowledge and signatures from symmetric-key primitives". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1825–1842.

[CL99] Miguel Castro and Barbara Liskov. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[Cos+19a] Davide Costa et al. *Introducing Quadrans*. 2019. URL: `https://quadrans.io/content/files/quadrans-light-paper-en.pdf`.

[Cos+19b] Davide Costa et al. *Quadrans Whitepaper*. 2019. URL: `https://quadrans.io/content/files/quadrans-white-paper-rev01.pdf`.

[DS05] Jintai Ding and Dieter Schmidt. "Rainbow, a new multivariable polynomial signature scheme". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2005, pp. 164–175.

[Duc+18] Léo Ducas et al. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.1 (Feb. 2018), pp. 238–268. DOI: `10.13154/tches.v2018.i1.238-268`. URL: `https://tches.iacr.org/index.php/TCHES/article/view/839`.

[Fou+18] Pierre-Alain Fouque et al. "Falcon: Fast-Fourier lattice-based compact signatures over NTRU". In: *Submission to the NIST's post-quantum cryptography standardization process* (2018). URL: `https://www.di.ens.fr/~prest/Publications/falcon.pdf`.

[ITU19] ITU. *X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. Oct. 2019. URL: `https://www.itu.int/rec/T-REC-X.509-201910-I/en`.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)". In: *International Journal of Information Security* 1.1 (2001), pp. 36–63.

[Men+19] Alessio Meneghetti et al. "A Survey on Efficient Parallelization of Blockchain-based Smart Contracts". In: *Annals of Emerging Technologies in Computing (AETiC)* 3.5 (2019).

[MST20a] Alessio Meneghetti, Massimiliano Sala, and Daniele Taufer. "A note on an ECDLP-based PoW model". In: *CEUR Workshop Proceedings Vol-2580 DLT 2020* (2020).

[MST20b] Alessio Meneghetti, Massimiliano Sala, and Daniele Taufer. "A Survey on PoW-based Consensus". In: *Annals of Emerging Technologies in Computing (AETiC)* 4.1 (2020).

[MW17] Daniele Micciancio and Michael Walter. "Gaussian sampling over the integers: Efficient, generic, constant-time". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 455–485.

[Sch15] Markus Schmid. *ECDSA-application and implementation failures*. 2015. URL: `https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Schmid.pdf`.

[Sho94] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on Foundations of Computer Science*. IEEE. 1994, pp. 124–134.

[ST] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization - Post-Quantum Cryptography*. URL: `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`.