OMAX

Yellow Paper

Introduction:

OMAX is not only an emerging cryptocurrency platform that is built on OMAX Blockchain but also has features of transparency, traceability and public which we believe is crucial for the long-term feasibility and decentralization. OMAX is a complete all in one ecosystem that contains the staking and NFT marketplace under an umbrella. OMAX will use its own blockchain technology to make its NFT marketplace Unique. OMAX main goal is to make decentralized (or non-custodial) trading easier and integrate to eCommerce platform and build the use case. OMAX is an EVM compatible blockchain, The EVM is the Turing complete virtual machine that allows the deployment and execution of smart contracts via transactions within a blockchain. which means all the products and functionalities of the Ethereum and Binance chains will be used in OMAX i.e remix, solidity versions in addition to our own functions. Users can create tokens, NFT market place and write smart contracts on OMAX Blockchain. With the addition of this feature it now offers a lower-cost alternative with similar features, faster transaction times, and support. The OMAX Blockchain is a unique, sovereign, unlike other blockchain, that supports smart contracts and time consuming. The purpose of this architecture was to keep OMAX Blockchain's high throughput while including smart contracts into its ecosystem.
OMAX, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to gradually change it into some current state.

$$\sigma t+1 \equiv X(\sigma t, T)$$

where X is the OMAX state transition function, X allows components to carry out arbitrary computation, while σ allows components to store arbitrary state between transactions. Transactions are collated into blocks; blocks are chained together using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block and an identifier for the final state. OMAX has an intrinsic currency, OMX. One OMX is defined as being 10^18 Omi.

Design Principles

Standalone Blockchain:

OMAX is an independent blockchain rather than a layer-2 solution in terms of technology. The majority of OMAX's core technical and business functions should be self-contained, so they can continue to work even if system is down for a short time.

OMAX Compatibility:

OMAX has chosen to be compatible with the existing EVM protocol in order to take benefit of the rather developed apps and community. This means that most dApps, ecosystem components, and tooling's will work with OMAX with little or minimal adjustments; OMAX nodes will have similar (or somewhat lesser) hardware requirements and expertise to run and maintain. OMAX should be able to keep up with future EVM protocol improvements thanks to the implementation.

Consensus and Validator Quorum:

Consensus protocols play a fundamental role in the blockchain technology as they have the responsibility to ensure that the chain of blocks replicated amongst the nodes is consistent. The type of network and environment assumptions made when designing a consensus protocol influence how the blockchain performs once deployed in a real environment and network. Some of the key performance metrics that are heavily influenced by consensus protocols are:
(i) throughput or number of transactions per second,
(ii) latency or time taken from when a transaction is submitted to the 1 system to when the transaction is included in a block and
(iii) robustness or what type of attacks the protocol can withstand.

The OMAX consensus protocol is designed to achieve the following objectives based on the aforementioned design principles: Latency time should be competitive comparative to existing blockchain network, i.e. 700 milliseconds or less. It takes a short amount of time to certify the finality of transactions, such as 250 seconds. The block reward is collected through transaction fees and will be paid in OMAX. There is no inflation of the native token OMAX. It tries to be as compatible with the EVM compatible system as feasible. It enables the governance of contemporary proof-of-stake blockchain networks.
Another way to classify consensus protocols is by the technique used to prevent an attacker from conducting a Sybil attack which consists in one node being able to gain power in the system by creating multiples pseudonymous identities. Typically, creating a new digital identity that can be used to interact with a blockchain is quite cheap as it just requires generating a random private key and the related public key. One of the most widely used and famous techniques for preventing Sybil attacks is proof of work. Proof-of-work (PoW) requires node to spend compute effort in solving a hard cryptographic puzzle before being able to propose a block to be added to the blockchain. Proof-of-Stake (PoS) is another quite well known technique where the right to propose new blocks is given according to the amount of stake owned. In contrast, in Proof-of Authority, or PoA, Sybil attacks are prevented by conferring the right to create new blocks only to a defined set of nodes. Within the IBFT 2.0 protocol, the nodes with the right to create new blocks and ensuring blockchain consistency are called validators. OMAX chain recommends combining (PoS) and (PoA). Proof of Staked Authority is incredibly efficient. It saves a significant amount of energy and hardware resources OMAX implements , Proof of Staked Authority (PoSA) consensus algorithm, which are involved in transaction validation and block validation. This protocols are used when participants are known to each other and there is a level of trust between them. In IBFT networks, transactions and blocks are validated by approved accounts, known as validators. Validators take turns creating the next block. Existing validators propose and vote to add or remove validators. IBFT has immediate finality. When using IBFT , there are no forks and all valid blocks are included in the main chain.

Given a system of n components, t of which are dishonest, and assuming only point-to-point channel between all the components. Whenever a component A tries to broadcast a value x, the other components are allowed to discuss with each other and verify the consistency of A's broadcast, and eventually settle on a common value y. IBFT uses a pool of validating nodes (Validators) operating on network to determine if a proposed block is suitable for addition to the chain.

One node of the Validators is arbitrarily selected as the Proposer and is responsible for constructing a block at the block-interval and sharing said block with the group. If a super-majority of the Validators deem the block to be valid it is added to the blockchain. At the completion of the consensus round, the Validators may select a new Proposer which will be responsible for providing the candidate Block at the next block interval. The consensus mechanism is a synchronised state machine which is responsible for ensuring all Validators append the same block to the chain at the same height. If a block fails to insert, the Proposer is changed, and the process starts anew. To ensure only one block can be appended to the state machine, IBFT prevents changing the proposed block once a super-majority of validators have agreed to its insertion (but not performed said work), this process is referred to as 'Block Locking'. The IBFT consensus mechanism offers system stability provided less than 1/3 of the validating nodes are behaving incorrectly (either due to being compromised or due to faulty code). I.e. to tolerate F faulty nodes the validation group must contain at least 3F + 1 nodes (more than this does not increase system integrity).

State Machine States:

- **Awaiting Proposal.** Validator is waiting for a new block to be supplied by the current proposer. If the validator is the proposer for this round, they prepare the proposed block and transmit it in a pre-prepare message.
- **Preparing.** Has received a (valid) proposed block and notified validator-peers; is now waiting for validator-peers to notify their acceptance of the block.
- **Ready.** Has received validator-peer's acceptance of block, and is waiting for them to be a in a similar position. At this stage the proposed block has been 'locked-in', and cannot be replaced until an attempt at insertion has been conducted. • Round Change. The round timed out before consensus was reached or the block failed to insert. Wait for all validators to agree on the next round number.

Transitions:

1. **Awaiting Proposal → Preparing.** On reception of a new block (Preprepare message) from the proposer (i.e. the block is valid in its content, as is its proposed chain insertion point).
2. **Awaiting Proposal → Round Change.** The received proposal was not a valid block according to a given set of rules (e.g. invalid proposer, incorrect round numbering).
3. **Preparing → Ready.** On reception of 2F+1 notifications (Prepare message) from validator-peers indicating the proposed block is suitable for insertion.
4. **Ready → Awaiting Proposal.** On reception of 2F+1 notifications (Commit message) from validator-peers indicating they are ready to append the block to the chain. On transition, the process of appending the block to the chain is performed (success).
5. **Ready → Round Change.** As per Ready->Awaiting Proposal, however, block insertion has failed.
6. **Round Change → Awaiting Proposal.** 2F+1 of validators agree on the next round number to be used.

In IBFT there is no specific "client" which sends out requests and waits for the results. Instead, all of the validators can be seen as clients. Furthermore, to keep the blockchain progressing, a proposer will be continuously selected in each round to create block proposal for consensus.

IBFT uses 3-phase consensus, PRE-PREPARE, PREPARE, and COMMIT. The system can tolerate at most of F faulty nodes in a N validator nodes network, where N = 3F + 1. Before each round, the validators will pick one of them as the proposer, by default, in a round robin fashion. The proposer will then propose a new block proposal and broadcast it along with the PRE-PREPARE message. Upon receiving the PRE-PREPARE message from the proposer, validators enter the state of PRE-PREPARED and then broadcast PREPARE message. This step is to make sure all validators are working on the same sequence and the same round. While receiving 2F + 1 of PREPARE messages, the validator enters the state of PREPARED and then broadcasts COMMIT message. This step is to inform its peers that it accepts the proposed block and is going to insert the block to the chain. Lastly, validators wait for 2F + 1 of COMMIT messages to enter COMMITTED state and then insert the block to the chain.

Blocks in Istanbul BFT protocol are final, which means that there are no forks and any valid block must be somewhere in the main chain. To prevent a faulty node from generating a totally different chain from the main chain, each validator appends 2F + 1 received COMMIT signatures to extraData field in the header before inserting it into the chain. Thus blocks are self-verifiable and light client can be supported as well. However, the dynamic extraData would cause an issue on block hash calculation. Since the same block from different validators can have different set of COMMIT signatures, the same block can have different block hashes as well. To solve this, we calculate the block hash by excluding the COMMIT signatures part. Therefore, we can still keep the block/block hash consistency as well as put the consensus proof in the block header.

Consensus states:

Istanbul BFT is a state machine replication algorithm. Each validator maintains a state machine replica in order reach block consensus.

States:

- **NEW ROUND:**

Proposer to send new block proposal. Validators wait for PRE-PREPARE message.

- **PRE-PREPARED:** A validator has received PRE-PREPARE message and broadcasts PREPARE message. Then it waits for 2F + 1 of PREPARE or COMMIT messages.
- **PREPARED:** A validator has received 2F + 1 of PREPARE messages and broadcasts COMMIT messages. Then it waits for 2F + 1 of COMMIT messages.
- **COMMITTED:** A validator has received 2F + 1 of COMMIT messages and is able to insert the proposed block into the blockchain.
- **FINAL COMMITTED:** A new block is successfully inserted into the blockchain and the validator is ready for the next round.
- **ROUND CHANGE:** A validator is waiting for 2F + 1 of ROUND CHANGE messages on the same proposed round number.

Validation Group Membership:

The members of the validation group may change over time through a voting mechanism. Members can be added or removed through a majority (Floor(N/2) + 1) vote; each vote is captured in the Block Header.

Each node in the network (including non-validating nodes) is responsible for tracking the vote tally for each validator to determine the current Validators and ensure signatures on mined blocks fall within the expected group.

Given each vote is contained in the Block Header, only the Proposer for a given round is able to cast a vote. Thus it is important, if nodes are to be added/removed in a timely fashion, that the Proposer role be updated on a regular basis.

Once a node reaches majority votes, they immediately join/leave the validator group. IBFT recognises a Voting Epoch, which defines a point at which all votes which have not yet reached a majority are removed, forcing the voting tally to be restarted. This implies when tallying votes, Validators need only start at the most recent epoch. By default, the Voting Epoch occurs every 30,000 blocks.

Votes define a change of state (i.e. candidates get voted in, validators get voted out), not voting for a given node implies the Validator does not wish the node to change state (an explicit vote is not required to maintain the status quo).

Key Advantages of IBFT

- **Immediate block finality**

IBFT allows one block for each block height since it can be added by elected block producer. This feature eradicates the possibility of any forking, uncle blocks and provides immediate confirmation of executed transactions.

Reduced time between blocks

Since every block producer gets its chance to add block deterministically, the effort required to write and validate blocks is reduced significantly (as compared to PoW), greatly increasing the throughput of the chain.

- **High data integrity and fault tolerance**

IBFT based Blockchain deals with a consortium of BPs. This ensures that only trust-worthy BPs contribute who maintain the integrity of every new block. Additionally, a two-third majority of these BPs are required to sign any block prior to adding it to the blockchain, making forgery in block producing extremely difficult. As mentioned earlier, every member in this consortium of block producer gets its chance. That essentially means, that no block producer will get a chance to hold the power for a long time. This removed the chance of even any faulty block producer to influence the ledger for a long time.

- **Operationally flexible**

The consortium of BPs can be updated in time, ensures trustworthy BPs remain the member of the consortium.

System Model.

The IBFT protocol relies on the Ethereum ÐΞVp2p protocol for the delivering of all protocol messages. We model the gossip network as an eventually synchronous network, where there exists a point in time called global stabilisation time (GST), after which the message delay is bounded by a constant, Δ. Before GST there is not bound on the message delay, and we admit messages being lost.

Failure Model.

We consider a Byzantine failure mode system, where Byzantine nodes can behave arbitrarily. In contrast, honest nodes never diverge from the protocol definition. We denote the maximum number of Byzantine nodes that an eventually synchronous network of n nodes participating in the consensus protocol can be tolerant to with f(n). the relationship between the total number of nodes, n, and the maximum number of Byzantine nodes can be expressed as follows: $f(n) \equiv [(n-1)/3]$

Cryptographic Primitives.

The IBFT protocol uses the Keccak hash function variant to produce blocks. Keccak hash function is collision resistant. The IBFT 2.0 protocol relies on the Elliptic Curve Digital Signature protocol to sign transactions. the signatures generated for two distinct messages are different with high probability. The unforgeability property ensures that Byzantine nodes, even if they collude, cannot forge digital signatures produced by honest nodes.

We denote message m signed by validator v as $(m)\sigma v$.

Protocol Description.

Each IBFT node maintains a local copy of the blockchain where the first block, called genesis block, is the same for all nodes. Each block B added to the blockchain must be cryptographically linked to another block in the blockchain, Bp which is commonly defined as the parent of block B, and, conversely, B is defined as the child of Bp. In IBFT, starting from the genesis block, the next block to be added to the local blockchain maintained by a node is the child of the latest block that was added to the blockchain

The IBFT protocol is modelled as running sequential instances of what we call the IBFT-2.0- block-finalisation-protocol, where the objective of the h-th instance of the IBFT-2.0-block-finalisation-protocol is to decide which block, and consequently which set of transactions, are to be added at height h of the blockchain maintained by any IBFT node.

Each instance of the IBFT-2.0-block-finalisation-protocol is organised in rounds and in each round one of the validators is given the responsibility to propose a block for the height associated with the specific instance of the IBFT-2.0-block-finalisation-protocol that the validator is running. Once agreement is reached, the IBFT-2.0-block-finalisation-protocol creates a finalised block which includes the block and additional information that allows any node, even nodes that did not participate in the IBFT-2.0-block-finalisation-protocol, to verify that agreement on the block included in the finalised block was correctly reached.

In practice, each IBFT node adds finalised blocks to its local blockchain, In this way, any node joining the network at any point in time, when synching its local blockchain with its peers, receives all the information required to verify that agreement was indeed reached correctly on each block that it receives, even on those created before it joined the network.

when a new finalised block is received by a node v, v executes the following operations:

1.verifies if the finalised block received is for the next expected chain height, i.e h_v;
2.if so, it verifies if the finalised block received is a valid finalised block.
3.if both verifications pass, then:
3.1. v adds the finalised block to its local blockchain;
3.2. if v is a validator for the current instance of the IBFT-2.0-block-finalisation-protocol, then v stops that instance;
3.3. v advances the next expected block height, hv, by 1;

if v is a validator for the IBFT-2.0-block-finalisation-protocol instance for the new value of hv, then v starts that instance.

Algorithm 1: IBFT 2.0 protocol for IBFT node v

Functions:

Quorum(n)≡[2n/3]

isValidFinalisedBlock(FB : (FB_{EB}, FB_{FP} : (FB_{FPr}, FB_{FPcs})), *v*)≡

size0f(FP_{cs}**)** ≥ *Quorum*($n_{h,v}$) **and**

any *cs* ∈ FP_{cs} **such that:**

EthAddressRecover(KEC((FB_{EB}, FB_{FPr})), *cs*) **in** validators(*chain*$_v$[0 : h_v − 1])

Initialisation:

chain$_v$[0] ← *genesis block*

h_v ← 1

if *v* **in** validators(*chain*$_v$[0 : h_v − 1]) **then start** the h_v-th instance of the IBFT-2.0-block-finalisation-protocol

Upon Blocks:

upon ⟨FINALISED-BLOCK, $\overline{FB}$: ($\overline{FB_{EB}}$, $\overline{FB_{FP}}$)⟩ **in** *receivedMessages*$_v$ **do**

if blockHeight($\overline{FB_{EB}}$) = h_v **then**

if *isValidFinalisedBlock*($\overline{FB}$), *v*) **then**

chain$_v$[h_v] ← $\overline{FB}$

if *v* **in** validators(*chain*$_v$[0 : h_v − 1]) **then stop** the h_v-th instance of the IBFT-2.0-block-finalisation-protocol

h_v ← h_v + 1

if *v* **in** validators(*chain*$_v$[0 : h_v − 1]) **then start** the h_v-th instance of the IBFT-2.0-block-finalisation-protocol

end

end

upon (

$\langle\langle$PROPOSE, h_m, *, *$\rangle_{\sigma\ \ sender}$, *, *$\rangle$ **in** *receivedMessages*$_v$ **or**

$\langle\langle$PREPARE, h_m, *, *$\rangle_{\sigma\ \ sender}$, *, *$\rangle$ **in** *receivedMessages*$_v$ **or**

$\langle\langle$COMMIT, h_m, *, *$\rangle_{\sigma\ \ sender}$, *, *$\rangle$ **in** *receivedMessages*$_v$ **or**

$\langle\langle$ROUND-CHANGE, h_m, *, *$\rangle_{\sigma\ \ sender}$, *, *$\rangle$ **in** *receivedMessages*$_v$

or

) **and**

$\overline{h_m} > h_v$ **and**

$sender \in \overline{\text{peers}_v}$ **and** $\text{expectedHeight}_v[\overline{sender}] < \overline{hm}$

do

$\text{expectedHeight}_v[\overline{sender}] \leftarrow \overline{hm}$

send ⟨GET-BLOCKS, h_v, h_m⟩

As described by the function isValidFinalisationBlock(FB, v), an IBFT 2.0 finalised block FB is defined valid if and only if all of the following conditions are met:

- it contains at least Quorum(n) ≡ [2n/3] different commit seals, where n is the number of validators for instance h of the IBFT-2.0-block-finalisation-protocol, i.e. n ≡ n(chainv[0 : h − 1]);
- each commit seal corresponds to the signature of one of the validators over the Ethereum block and the round number included in the finalisation proof.

Analysis of the IBFT-2.0-block-finalisation-protocol

IBFT-2.0-block-finalisation-protocol provides optimal Byzantine-fault-tolerant Safety. the local blockchains of all honest nodes are identical until finalised block with height h − 1. Therefore, since the set of validators for the h-th instance of the IBFT-2.0-block-finalisation-protocol is a function of the local blockchain until the block with height h − 1, this set is identical amongst all honest validators. We denote the total number of validators for the h-th instance of the IBFT-2.0-block-finalisation-protocol with nh.

Honest validators running the h-th instance of the IBFT-2.0-block-finalisation-protocol move to a round higher than the current one when one of the following events occurs:

- the round timer for the current round expires;
- they receive Quorum(nh) Round-Change messages for a round higher than the current one sent by distinct validators;
- they receive a valid Proposal message for a round higher than the current one.

If validator v starts round r at time t, then v will move to a round r' > r by the time t + RoundTimerTimeout(r).

Let srh,v(r) denote the time at which validator v starts round r of instance h.

Let v_f be the first honest validator of instance h to start instance h. The following relation is verified for any honest validator v of instance h and round r such that both v_f and v start round r at some point while in instance h:

$$sr_{h,v_f}(r) \leq sr_{h,v}(r)$$

Let v_f be the first honest validator of instance h to start instance h. The following condition is always verified:

$$nonForcedRoundStart_{h,v_f}(r) = sr_{h,v_f}(r)$$

Let v_ℓ be the last honest validator to start instance h of the IBFT-2.0-block-finalisation-protocol. The following relation is verified for any honest validator v of instance h and round r such that both v_ℓ and vstart round r at some point while in instance h:

$$nonForcedRoundStart_{h,v_\ell}(r) \geq sr_{h,v}(r)$$

The following equation holds for any validator v of the h-th instance of the IBFT-2.0-block-finalisation-protocol provided that validator v starts round r at some point while in instance h.

$$\begin{aligned} nonForcedRoundStart_{h,v}(r) &\equiv si_{h,v} + \Sigma_{i=0->r-1}\ timeoutForRoundZero.2^i \\ &\equiv si_{h,v} + timeoutForRoundZero.(2^r-1) \end{aligned}$$

Let v_f be the first honest validator of instance h to start round r, v_ℓ be the last honest validator of instance h to start round r. minTimeAllHonestValidatorsAreInTheSameRound(r), representing the length of the minimum time segment where all honest validators that started instance h are in round r at the same time, is expressed as follows:

$$minTimeAllHonestValidatorsAreInTheSameRound(r) \equiv \max(si_{h,v_\ell} - si_{h,v_f} + timeoutForRoundZero \cdot 2^r, 0)$$

$$nonForcedRoundStart_{h,v_f}(r+1) - nonForcedRoundStart_{h,v_\ell}(r)$$

$$\equiv si_{h,v_f} \cdot timeoutForRoundZero \cdot 2^{r+1} - (si_{h,v_e} + timeoutForRoundZero \cdot 2^r)$$

$$\equiv si_{h,v_f} - si_{h,v_l} + timeoutForRoundZero \cdot (2^{r+1} - 2^r)$$

$$\equiv si_{h,v_f} - si_{h,v_l} + timeoutForRoundZero \cdot 2^r$$