Avail: The Data Availability Blockchain

Polygon

ABSTRACT

We study the scalability issue in present blockchain architecture and propose a design which decouples the data hosting, execution and verification. We discuss few suitable primitives for the construction of a data hosting layer that guarantees data availability. We sketch out the system goals, design, security and implementation details of Polygon Avail which is our availability focussed blockchain solution.

1 PRESENT DAPP ARCHITECTURE

In present day Ethereum-like ecosystems, there are mainly three types of peers - validator nodes, full nodes and light clients. A block is appended to the blockchain by a validator node which collects transactions from the mempool, executes them, generates the block before propagating it across the network. The block contains a small block header containing digest and metadata related to the transactions included in the block. The full nodes across the network receive this block and verify its correctness by re-executing the transactions. The light clients only fetch the block header and fetch transaction details from neighbouring full nodes on an as-needed basis. The metadata inside the block header enables the light client to verify the authenticity of the received transactional details.

While this architecture is extremely secure and has been widely adopted, it has some serious practical limitations [9]. Because every transaction is executed by all the full nodes in the system, it becomes a bottleneck resulting in limited throughput. With wide adoption of such DApps, the number of transactions grow rapidly resulting in increased cost of including a transaction into a block. To counter these issues, layer 2 solutions have been proposed.

Layer 2 scaling solutions work by creating a chain anchored to the main chain like Ethereum. In such an architecture, DApps reside on the L2 chain and only periodic checkpoints are recorded on the main chain. The throughput in such a system depends on the transaction processing rate of the L2 chain whereas any dispute resolution can be performed on the main chain. The benefit of such a decoupled execution and dispute resolution is that it gives much higher throughput while retaining the security properties. In practice, multiple such L2 chains may function in parallel resulting in improved processing power of the system, while retaining the security of the main chain.

2 PLASMA, OPTIMISTIC ROLLUP AND ZK ROLLUP

A major part of a successful layer 2 solution is the execution and verification framework. Plasma based sidechain approach can process thousands of transactions and submit only a single hash as checkpoint to the main chain. However, it faces an important issue. In case of a dispute on the sidechain, Plasma lacks an efficient mechanisms for users to exit the sidechain. Also, Plasma lacks the ability to process arbitrary instructions although they are supported by a quasi Turing-complete main chain like Ethereum.

Optimistic Rollups like Arbitrum and Optimism use the optimistic execution paradigm where a manager/operator executes the sidechain transactions and submits an assertion on the main chain. In case of dispute, other participants can challenge the assertion on chain within a fixed timeout and then the main chain performs the dispute resolution. While this scales up the transaction processing rate, it has a few downsides like delayed finality for non-fungible assets among others.

Zero-knowledge based rollups like Polygon Hermez and Starkware take the ZK based approach to off-chain execution. The operators execute the transactions and submit a ZK proof to the main chain. The main chain can quickly verify the proof and on-chain participants can be assured that only valid state transitions were performed.

Another class of rollup is Validium which is a hybrid between Plasma and ZK based rollups. They make use of Zero-knowledge based proofs of execution and at the same time keep the transactional data off-chain. Although this improves the throughput of the system, they suffer similar problems to that of Plasma. User funds can be frozen or seized and other crypto-economic attacks are also possible on such a system. [5] provides a detailed discussion on the approach and possible attacks.

While there is still debate inside the community about which of these approaches is the best, we envision multiple such roll-ups operating simultaneously together forming the execution layer of our design.

3 THE DATA AVAILABILITY PROBLEM

While the off-chain execution based architecture improves throughput, it is still limited by the amount of data that the main chain like Ethereum can handle. This is because although the execution is off-chain, the verification or dispute resolution is strictly on-chain. The transactional data is submitted as calldata on Ethereum to ensure that the data is available for future reconstruction. This is extremely important. In case of optimistic rollups, the operator may submit invalid transactions and then suppress parts of the block to the world. This way, the other full nodes in the system will not be able to verify whether the submitted assertion is correct. Due to lack of data, they will not be able to produce any fraud proof/challenge to show that the assertion is indeed invalid. In case of Zero-knowledge based rollups, the ZKP soundness ensures that accepted transactions are valid. However, even in the presence of such guarantees, not revealing the data backing a transaction can

have serious side effects. It may lead to other validators not being able to calculate the current state of the system. [3] contains a detailed discussion on the subject.

We recognize that to achieve higher throughput, we not only need to put execution off-chain but also need to have a scalable data hosting layer that guarantees data availability. In later sections we discuss a design of such a decentralized Data Availability layer.

4 SYSTEM OVERVIEW

4.1 Decoupling the components

On a high level, a successful blockchain design needs to address the following components:

- **Data Hosting and Ordering:** This component would receive transactional data and order it without any execution. It would then store the data and ensure complete data availability in a decentralized manner. We call this component the *DA layer*.
- **Execution:** The execution component should take ordered transactions from the *DA layer* and execute them. It should create a checkpoint/assertion/proof and submit it to the *DR layer*. We call this the *Exec layer*.
- **Verification/Dispute Resolution:** This component represents the main chain to which the system is anchored. The security of the design is dependent on the robustness and security properties of this component along with the *DA layer*. The checkpoint or assertion or proof submitted by *Exec layer* is processed by this layer to guarantee that only valid state transitions are accepted in the system (provided that the data is available). We refer to this component as the *DR layer*.

In this work, we are proposing Avail, which acts as the *DA layer*. We envision the multiple roll-up initiatives or legacy execution layers to form the *Exec layer*. The *DR layer* can be any secure settlement layer which supports the execution verification.

4.2 Participants and System Goals

4.2.1 Types of Nodes. We consider the following types of nodes for Avail:

- **Full Nodes:** The full nodes download the blocks and validate their correctness but do not participate in consensus. They store the blockchain but are not incentivized to participate or remain honest.
- **Validator Nodes:** The validator nodes take part in block generation and decide on transaction inclusion and ordering. These nodes are incentivized to participate in consensus and host the blockchain. Essentially, they are also full nodes with stake in the system.
- **Light Clients:** These are clients with resource constraints who have access to only the block header and query transactional data from other full nodes on an as-needed basis. They want to have high confidence that the block is available, and when querying data they want a proof that the data belongs to the block.

4.2.2 System Goals. Our design for Avail should provide the following guarantees:

(1) The decentralized data availability blockchain keeps producing a canonical chain of blocks even in the presence of an adversary that controls $< \frac{1}{3}$ of the validator nodes in the network.
(2) The honest participants of the system with access to the canonical chain of block headers will not accept a block whose underlying data is unavailable even if a very powerful adversary is controlling all the other nodes in the system. Under the assumption that there are enough honest participants, the data should remain available for a limited amount of time.

We want to guarantee achieving both these system goals. We want to emphasize that both the goals have different adversarial assumptions. Goal 1 ensures that the blockchain system continues to function in a decentralized manner as long as a super-majority of validator nodes remain honest. Goal 2 ensures that an outside participant like an application running a light node, who has access to the canonical chain of block headers, need not trust any full node to have the guarantee that the underlying data for a particular block is available. This is an extremely strong assumption which eliminates any trust assumption to detect data hiding attempt. In the next sections we discuss our construction ideas and argue that they achieve these goals.

5 DESIGNING THE DATA AVAILABILITY LAYER

5.1 Primitives

5.1.1 Why is redundancy important in ensuring Data Availability? Suppose we have a block B divided into data chunks $D_1, \ldots, D_n$. The block producer wants to suppress one chunk. Without loss of generality, let us assume that the first chunk is hidden by the block producer. Clients can query one chunk at random to get guarantees that data is indeed available. It can repeat this process many times so as to get sufficient confidence that the data is accessible. Hence, for each query, the block producer needs to be lucky enough that D_1 is not queried.

However, with redundancy included by schemes like erasure coding, suppose the n chunks are encoded into $2n$ chunks. Erasure coding ensures that any n out of the $2n$ chunks are sufficient to recreate the data. This makes the hiding task harder for the block producer. To hide one particular chunk, it needs to make at least $n + 1$ chunks unavailable. As a client, querying constant number of times gives a very high confidence that the data is indeed available. Hence, redundancy plays a vital role in the data availability. An erasure coding based design along with the related tradeoffs are discussed in [1].

5.1.2 Fraud Proofs. In the previous section although we discussed the advantage of redundancy in data, we omitted the case when the erasure coded chunks are misconstructed by the block producer. In these cases, even though most chunks are indeed available, the entire block data might not be accessible. Hence, fraud proofs are constructed by other full nodes in the system and propagated to

the light nodes. The light nodes verify the fraud proof and get convinced that the received block header is that of an erroneous block. An interesting property of fraud proofs is that they function even under a minority honest assumption. This is because having a single honest full node as a neighbour is sufficient for an honest light node to be guaranteed that it receives the fraud proof.

An important factor to consider during design of *DA layer* is the size of fraud proofs. In simple erasure coded chunks, to proof that the encoding is incorrect, the entire original data block needs to be propagated to the light client. Hence, the fraud proof size is at least linear to the size of the block. Although we do not discuss the higher dimensional erasure coding based design here, we refer to [1] for a detailed analysis of fraud proof sizes.

5.1.3 Commitment Size. In a one dimensional erasure coding based design, once a block producer selects data chunks $D_1, \ldots, D_n$, it encodes the chunks to generate $D_1, \ldots, D_{2n}$ (assuming coding rate to be 0.5). It then constructs a Merkle tree over the chunks and keeps the root of it in the block header which acts as the commitment. A light client fetching D_i gets a Merkle membership proof along with the data chunk so that it can quickly verify that a legitimate chunk has been supplied. The commitment (root of Merkle tree) size is an important factor to consider in any design. With large commitments, block headers become larger resulting in increased network traffic. While higher dimensional erasure coding based schemes achieve shorter fraud proof size, it comes at the cost of larger commitments.

5.2 Coded Merkle Tree based Design

We briefly discuss the Coded Merkle Tree (CMT) based approach [10] proposed by Yu *et al.* The novelty of this approach is that it gives constant sized commitments with logarithmic sized fraud proofs.

The design uses systematic erasure codes at each layer of the Merkle tree. In particular, at the base layer, it takes k data chunks and extends it to n data chunks in such a manner that the first k out of n chunks are the original chunks and the rest are parity symbols. Each chunk in the next layer of the tree is made from hashes of q chunks of the previous layer. Similar systematic erasure code is applied on that layer as well and we keep building successive levels of the tree until we have t hashes which we keep as the commitment inside the block header.

When a light node samples a base layer chunk, the chunk along with a Merkle membership proof is given in such a manner that not only availability of base layer is ensured but also the higher layer availability is guaranteed. A light client sampling s base layer chunks the Merkle membership proof will automatically sample s intermediate layer chunks with high probability.

For a full client, decoding the tree and generation of fraud proof is also extremely efficient. With access to the root layer of the CMT present in the header and some coded chunks of the previous layer, the hash-aware peeling decoder decodes the previous layer getting access to all the hashes of the preceding layer. The decoder continues until all the chunks in the base layer is decoded or it finds evidence of an incorrect coding. In case of the former, the full node gets access to the entire data and in case of the latter, it generates the fraud proof and broadcasts it.

A detailed description of the construction along with performance analysis is present in [10].

5.3 Kate Commitment based Design

In this section we first discuss polynomial commitments proposed by Kate *et al.* [6]. Then we go on to discuss a *DA layer* design based on Kate commitments as proposed in [4].

Given a polynomial $\phi(x) \in \mathbb{Z}_p[x]$ over a bilinear pairing group, Kate *et al.* proposed a scheme to have a commitment to the polynomial using a single group element. Moreover, the scheme supports opening of a commitment at a point i to get $\phi(i)$ using constant sized witness that allows a verifier to confirm that $\phi(x)$ was indeed evaluated at i to get $\phi(i)$. The commitment scheme is both computationally hiding and binding. The commitment scheme is additively homomorphic and supports a single witness for a batch of openings on multiple points of the same polynomial.

Given such a scheme, the block producer breaks the block data into chunks such that each chunk is an element of the field. It arranges the chunks into n rows and m columns such that it forms an $n \times m$ matrix D. It uses the evaluation form to construct a polynomial from each row to obtain $\phi_1(x), \ldots, \phi_n(x)$. It then commits each polynomial to get $C_1, \ldots, C_n$ respectively. For redundancy, it extends $C_1, \ldots, C_n$ to $C_1, \ldots, C_{2n}$. It puts $C_1, \ldots, C_{2n}$ inside the block header and broadcasts it. Figure 1 shows the data arrangement and the corresponding commitments.
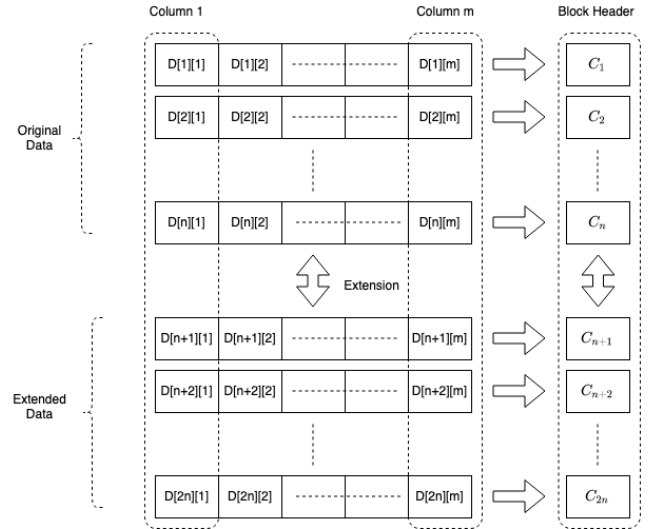
Figure 1: Data Arrangement in Kate Commitment based *DA layer*

The light clients querying a data block will sample some chunk $D[i][j]$. Along with the data, the light client gets a witness $w[i][j]$ and it can immediately verify the validity using the Kate Commitment scheme discussed above. If it queries multiple chunks of the same row, the batch commitment scheme helps have a single witness for all the sampled points. Hence the membership proofs are extremely efficient.

For full nodes, the system can have two types of full nodes: 1. classical full nodes having entire block, 2. Column full nodes which

keep only a single column of the data. For classical ones, it takes the entire matrix D, extends each column to $2n$ points and getting an extended matrix D'.

It then verifies for each row of D' whether the commitment to i^{th} row is C_i for $1 \leq i \leq 2n$.

For the column full nodes, they can fetch and keep only a column of the matrix D. They would extend each column to check whether they belong to the extended set of commitments. This is possible because of the homomorphic nature of the commitments and witnesses. In particular, having $D[1][j], \ldots, D[n][j]$ and $w[1][j], \ldots, w[n][j]$, the node can extend both to $2n$ points and immediately verify whether the extended openings are valid. If the number of such column full node is $O(m)$, and each column full node ensures at least one column is available, then collaboratively the entire data is available within the column full nodes.

As the block size grows, so does the number of commitments (keeping number of columns fixed, the number of rows grow). Hence the commitment size grows linear to that of the block size. However, we do not need any fraud proofs in such a system. For each light client, the communication and computational overhead is constant. For each full node, the computational overhead is $O(m * nlog(n))$ as it needs to perform $O(m)$ FFTs, $O(1)$ for each column, along with some pairing checks for verification. However, a column node has to only perform $O(nlog(n))$ work because it works on a single column of data.

6 ANALYSIS

6.1 Attack Vectors

Any *DA layer* needs to counter the following attacks on data availability:

(1) A super-majority of the validators in the *DA layer* wants to change the ordering of an already finalized block.
(2) A super-majority of the validators create a wrong block header, i.e., the commitment of the data present in the header is wrong.
(3) A super-majority of the validators want to hide at least one chunk of the block. For a light node, this would mean not being able to detect hiding of data with a non-negligible probability. For a full node, this would mean not being able to reconstruct the data.

We take the case of the super-majority of validators attacking the *DA layer* because we want to minimize honesty assumption required for our design.

Under the assumption that finality is reached, the reordering attack (attack 1) is hard to mount. This is because, to successfully mount this attack the attacker would need to break the non-equivocation property of the underlying consensus. In particular, the finality layer ensures that if equivocation occurs, the dishonest parties can be identified and their stake slashed.

In the subsequent sections we show how Attack 2 and Attack 3 is countered in each of the two design choices - CMT and Kate Commitments.

6.2 Coded Merkle Trees

In the CMT based approach, the block proposer(s) may attempt to construct wrong header in two ways:

- The erasure coding in one or more layers is wrong.
- The Merkle tree construction is wrong.

In both cases, the fraud proof captures such an attack. A full node, with access to the data, can build the incorrect-coding proof and broadcast it for the light clients. [10] contains a detailed description of such a fraud proof.

The random sampling from base layer by a light client also samples higher layers. As each layer is erasure coded, hence, probability of hiding any chunk from any of the layers of the Merkle tree is extremely low for sufficiently many samplings. A full client who wants to reconstruct the Merkle tree needs access to the block header which contains the root hashes. Along with that, it needs $(1-\alpha)n$ samples of the each layer, where n is the total chunks in the layer and α is the minimum fraction of coded symbols a malicious block producer needs to make unavailable to prevent full decoding. With these information it can decode the entire tree. The entire decoding strategy along with the hash-aware peeling decoder construction is given in [10].

Hence, the CMT approach can mitigate both attack 2 and 3. We however note that the mitigation assumes a single honest full node, which is not as strong as Goal 2 defined in Section 4.2.2.

6.3 Kate Commitments

In the Kate commitment based approach, attack 2 amounts to a wrong commitment by the block producer. Without loss of generality, suppose C_1 is wrong. This would mean that at least one out of $D[1][1]$ to $D[1][n]$ does not belong to C_1. Again, let us assume $D[1][1] \notin C_1$. This would mean that at least one of this is true: $D[n+1][1] \notin C_{n+1}, \ldots, D[2n][1] \notin C_{2n}$. This is because $C_{n+1}, \ldots, C_{2n}$ are extended from $C_1, \ldots, C_n$ and $D[n+1][1], \ldots, D[2n][1]$ are extended from $D[1][1], \ldots, D[n][1]$. Hence, such an attack is caught by a light client with overwhelming probability.

In case of attack 3, a light client querying constant samples can achieve arbitrarily high confidence that the data is indeed available. This is due to the redundancy in the data as discussed previously. For a full node, it needs to download the at least n chunks from each column so that it can reconstruct the entire extended data. It checks the commitments to know whether the downloaded data is correct. A column full node needs to perform similar operations but only for one column of data. Even if the supermajority wants to suppress data after some time, the light client P2P layer contains the data with redundancy. This enables new clients to recover block data without reliance on the full nodes or validators.

Hence, this approach mitigates all discussed attacks and achieves the goals set in Section 4.2.2.

7 IMPLEMENTATION

In this section, we discuss the implementation details of Avail. Among the two techniques discussed above, we use the Kate commitment based approach in order to avoid fraud proofs and provide strong data availability guarantees even in presence of powerful adversaries.

7.1 Consensus

We need our validators to reach consensus on the next block which contains the ordered set of transactions along with redundant erasure coded data. Although there exists many possible blockchain consensus protocols, we opted for the Proof-of-Stake (PoS) family of consensus algorithms. We wanted a PoS consensus that supports high number of validators, provable finality and a robust security framework. In particular, we chose the BABE/GRANDPA hybrid consensus used by Polkadot [7]. It uses two separate protocols for block production and finality.

The block production is done using Blind Assignment for Blockchain Extension protocol (BABE) [2]. The block producers produce blocks based on a Verifiable Random Function (VRF). The protocol does not assume access to a central clock. In case no validator is chosen as the block producer for a particular slot, or if the chosen producer(s) go down, there is a secondary block producer who can step in. BABE ensures liveness which guarantees that transactions submitted to honest players will eventually be inserted into the chain.

GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) [8] is a finality gadget ensuring provable finality. Without the finality gadget, the users can only have probabilistic finality, like in classical blockchain systems. GRANDPA guarantees that blocks reach quicker finality and a finalized block can never be reverted.

7.2 Full Nodes

To implement our design in a standalone blockchain, we use the Substrate framework [1]. Substrate provides great flexibility to implement custom runtime logic with inherent BABE/GRANDPA consensus support. Our design needed major changes in the Substrate codebase and hence we worked on a separate codebase [2]. The major changes that we made are:

- The contents of the block are arranged into the data matrix, extended and polynomial commitment generation is performed during the block building process.
- The header structure is changed to contain the commitments as part of the header.
- Block sizes are made dynamic, according to the number of transactions on the mempool. However, the maximum block size allowed is 4 MB, including the data redundancy.
- Additional RPC methods to support data availability queries are introduced. These can be used by light clients to gain confidence about availability and inclusion.

We have modified the Polkadot Apps [3] to build our block explorer [4] for our test network.

7.3 Light Client

In our present design, light clients are application nodes that participate in the system without investing resources to host a full node. Keeping this in mind, we have developed a light client [5]

which keeps track of the chain header and performs availability queries to gain confidence on the availability of interested blocks. They also help keeping the data available by hosting it within the light client P2P network. The client can gain as high confidence as deemed suitable for the specific application. We envisage the applications using our chain to host a light client.

7.4 Ethereum Data Availability Guarantees

A *DR layer* layer like Ethereum, when validating proofs or resolving dispute, need assurance that the underlying data is indeed available (as discussed in Sec 2). To enable this in an organic manner, we want to guarantee data availability to a smart contract on Ethereum (or any main chain which supports this logic). This allows the checkpoint/assertion/proof accepting smart contract to communicate with this light client contract and directly get DA assurance.

We discuss three main approaches on how to provide the on-chain DA guarantee. We also briefly mention their advantages and limitations.

7.4.1 Data Availability Oracle. A widely accepted approach for offloading expensive tasks from smart contracts is the oracle based approach. When on-chain DA guarantee for a particular block is needed, multiple oracle nodes can be incentivized to submit their respective confidence factors. The on-chain contract can aggregate the metrics to decide whether it has sufficient guarantee that the data is available. Although well-accepted within the community, this approach has to trust the oracle nodes resulting in reduced security guarantees. The amount of trust can be limited by proper incentivization and decentralization.

Proof of Concept Implementation: To emphasize the viability of the protocol, we have implemented an on-chain light client using Chainlink Oracle Network, based on the second design discussed above. Whenever the on-chain contract needs availability guarantee for a particular block, it requests the on-chain Chainlink Oracle which uses events to communicate with multiple off-chain oracle nodes. These off-chain nodes use their own light clients or hosted instances to gain confidence and communicate the same back to the original contract through the Chainlink Oracle. The incentivization is handled using the LINK token.

7.4.2 Bridges. A well-known approach for cross-chain message passing is using bridges. Using a bridge from Avail to *DR layer*, the validators of Avail can attest to the fact that a particular piece of data is available. The downside of such approach is that it relies on the supermajority of the Avail validator set to be honest and to keep the data available. For most applications, this is a good enough assumption.

Under this model, the rollup operator submits the data on Avail and gets back a finalized block post transaction inclusion. It computes a proof of transaction inclusion within that block and posts it, along with proof of transactional validity, to the settlement rollup contract. The bridge operator(s), on the other hand, post finalized block metadata on the settlement layer. The rollup contract, upon receiving the proofs, checks the correctness of proofs against the bridge contract. The rollup transaction is accepted iff the proofs are verified to be correct.

---

[1]https://github.com/paritytech/substrate
[2]https://github.com/maticnetwork/avail
[3]https://github.com/maticnetwork/avail-apps
[4]https://testnet.polygonavail.net/
[5]https://github.com/maticnetwork/avail-light

8 ADDITIONAL MODULES

8.1 Application Specific Data Retrieval

We would want our construction to allow applications to download data which is only relevant to them. To enable this, we want the full nodes to prove to an application client that the complete set of data relevant to the application has been communicated.

To enable Application Specific Data Retrieval (ASDR), we have introduced two main fields:

- **AppID**: Every transaction has been associated with an application identifier (appID). We envision rollup or application specific chains to register as applications if they want to download parts of block with only relevant application data. Any user is free to use the default appID '0' which is used for all types of transaction, including Avail system transactions. Any transaction with nonzero appID has to be of type application data submission only.

- **appDataLookup**: Every block header contains an index field, containing the starting cell indices of application cells inside the block. This is created by the block producer when creating the block by grouping the data matrix by appID and then flattening it to infer the starting position of each group.

The changes above together ensure that an application client can download only app-specific cells without downloading other application data.

However, it is worth noting that there is no reliance on the block producer to create the index or group the data correctly. This is because, we create a deterministic algorithm that all honest application clients follow. Hence, all honest clients would retrieve the same app data for a particular appID. It can happen that clients get partial data, i.e., there exists data within the block associated with an appID which was not downloaded following the algorithm. However, the aim is to have the same view of their app to all honest app clients by providing them with same data. The polynomial commitments inside header assure that two clients with access to the same header cannot receive different sets of data.

REFERENCES

[1] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. 2019. Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities. arXiv:1809.09044 [cs.CR]

[2] Handan Kilinc Alper. 2020 (accessed December 18, 2020). *BABE*. https://research.web3.foundation/en/latest/polkadot/block-production/Babe.html

[3] Vitalik Buterin. 2018 (accessed November 24, 2020). *A note on data availability and erasure coding*. https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding

[4] Vitalik Buterin. 2020 (accessed November 24, 2020). *2D data availability with Kate commitments*. https://ethresear.ch/t/2d-data-availability-with-kate-commitments/8081

[5] Alex Gluchowski. 2020 (accessed November 24, 2020). *zkRollup vs. Validium*. https://medium.com/matter-labs/zkrollup-vs-validium-starkex-5614e38bc263

[6] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.

[7] Polkadot. 2020 (accessed December 18, 2020). *Polkadot Consensus*. https://wiki.polkadot.network/docs/en/learn-consensus

[8] Alistair Stewart and Eleftherios Kokoris-Kogia. 2020. GRANDPA: a Byzantine Finality Gadget. arXiv:2007.01560 [cs.DC]

[9] David Tse. 2020 (accessed November 24, 2020). *Stanford Engineering - EE374: Scaling Blockchains*. https://web.stanford.edu/class/archive/ee/ee374/ee374.1206/

[10] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. 2019. Coded Merkle Tree: Solving Data Availability Attacks in Blockchains. arXiv:1910.01247 [cs.CR]