Polybase: A Decentralised Query, Index and Storage Protocol

Calum Moore and Sidhant Gandhi

Polybase Labs

Draft 2.0, October 17, 2022

Abstract

This paper describes Polybase, a horizontally scalable decentralised query, index and storage protocol for structured data. We describe the protocol's query interface, network topology, economic incentives and security mechanisms, that provide blockchain level guarantees at scale.

1. Introduction

Decentralised applications (dApps) allow their users to have complete access to their data, enabling them to use and directly benefit from it, whilst providing tight privacy controls over its access.

This is compelling for both companies and consumers who have long lamented sharing their data with centrally controlled web2-era services built on top of data monopolies. These monopolies exploit the data for their own gain, limiting economic opportunity and self-determination for many.

With additional regulation protecting consumers and with a realisation that data is a critical business asset, companies too are attempting to limit the proliferation of their own data and that of their users. However, companies are forced to choose between two undesirable options: hosted services which lack privacy and limit data access, or self hosting that results in poorly

maintained applications but with increased access and privacy. Web3 dApps enable companies to benefit from enhanced access and privacy without the burden of self hosting.

However, development of dApps is being stifled by a lack of required infrastructure. There is currently no performant structured data query, index and storage protocol. In absence of this essential infrastructure, developers have attempted to use existing blockchains for this purpose. However, this leads to a poor user experience, as blockchains suffer from slow transactions per second (tps), high latency (the time it takes for a block to be considered final and irreversible), and expensive storage. These limitations are an inherent result of blockchain architecture that requires every node to process and store all of the data, effectively limiting blockchains to vertical scaling (i.e. equivalent to running on a database on a single node).

The key to overcoming these performance characteristics and improving scalability is to store and process data off-chain, enabling the system to be scaled horizontally. This can be done securely and performantly through the use of recursive zk-SNARKs, which provide a fast path to verify cryptographic proofs that the processing and state changes have been applied correctly, without each node having to manually re-process them. This approach is often referred to as a zk-rollup, however instead of rolling up financial transactions or smart contract state, the Polybase rollup is for structured indexed data that is fast to query.

2. Protocol Design

Requirements

In order to achieve the outlined goals, the protocol must meet the following requirements:

1. ***Fast confirmation*** - minimal time between a client submitting a write and the write being irreversibly confirmed
2. ***Predictable storage and writes*** - databases should scale with demand, each additional node should increase the overall capacity and performance of the protocol
3. ***Always available*** - data should be always available and highly redundant

4. ***Decentralised*** *(autonomous and tamper-resistant)* - database definitions once deployed are strictly enforced
5. ***ACID*** - strong and/or timeline consistent state
6. ***Incentive alignment*** - strong economic incentives and disincentives to ensure correct behaviour on the network

Architecture

Polybase is a state zk-rollup protocol, that provides native support for modular data storage and indexing. Reference implementations for modules will be provided.

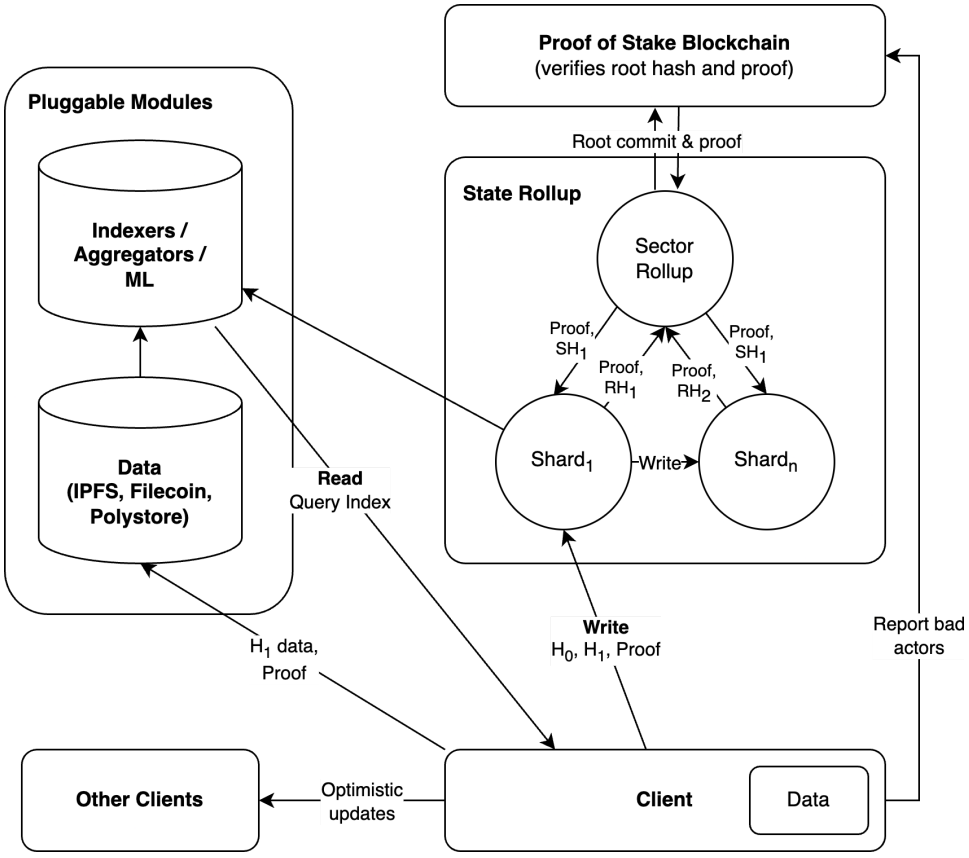


Figure 1: Polybase decentralised architecture

Roles

The Polybase protocol has a number of roles that can be performed. A node can perform one or more of the following roles.

- ***Blockchain*** - validates root zk-STARK proofs for rollups and state transitions, manages governance, payments and disputes
- ***Sector*** - responsible for rolling up hashes across a given number of shards, it verifies the rollup proofs from shards, other other sub-sectors
- ***Shard*** - responsible for validating client proofs, and updating shard root hash, and providing this to the parent sector node for inclusion in root state
- ***Indexer (pluggable)*** - creates index or aggregation of data making it accessible in a fast to query format
- ***Storage provider (pluggable)*** - stores encrypted data
- ***Challenger*** - any node can become a challenger by disputing transactions on the network and staking the required dispute stake
- ***Client*** - a peer on the network that queries data and creates proofs for writes, clients optimistically share updates with each other
- ***Peer*** - P2P is the primary communication method, all nodes in the network perform this role

Blockchain

The blockchain has the following responsibilities:

1. ***Validating root proof*** - used to verify that network participants are following the protocol rules
2. ***Staking*** - holding stakes from network participants to ensure required behaviour
3. ***Dispute resolution*** - when a submitted proof is invalid a node in the network can report it for a reward, the invalid party's stake is slashed

Polybase does not store database state on the root blockchain. Instead it inherits the security of the blockchain using zk-STARKS that verify off-chain activity using on-chain proof verification. All proofs are combined into a root proof which is verified by the blockchain.

As these proofs are posted and verified on the blockchain, any node on the network can dispute a transaction. Disputes can then be resolved automatically using the cryptographic proofs, with slashing penalties applied to the infringing party.

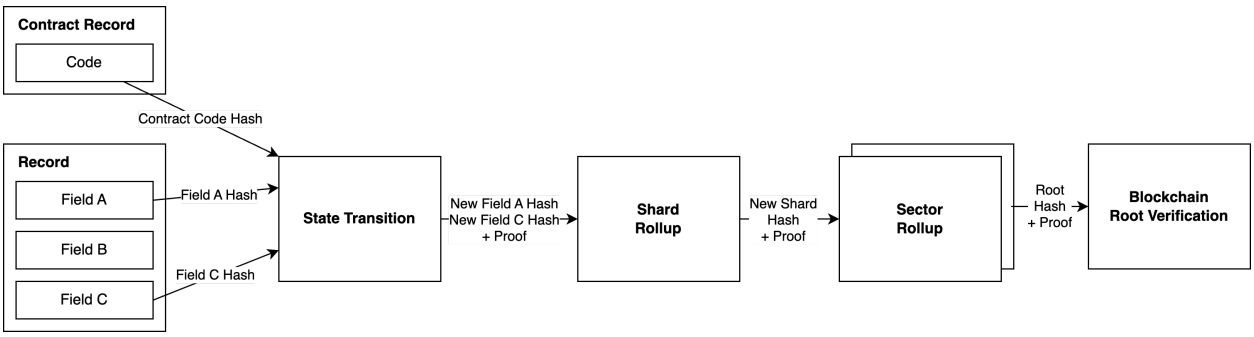
The root state hash may be posted to multiple blockchains (e.g. Ethereum, Near, etc) to allow for simple verification of proofs for use in zk-bridges.

State Rollup

The state rollup is a permissionless key value store and consists of the following parts:

- State transition
- Shard rollup
- Sector rollup
- Blockchain root verification

All state on Polybase is rolled up from individual field hashes to a global root hash stored and verified on one or more blockchains.



Each record on Polybase has a globally unique key which is a combination of the namespace and an `id` field determined by the contract's code. Keys are hashed, improving distribution of load

across the protocol, enhancing privacy, and protecting against attacks on specific datasets (as any given dataset will be spread evenly across the system).

State Transition

State transition is performed on the client. The client proves the transition they are making is valid, and provides the new hashes and proofs to the allocated shard group.

The state transition proof must prove:

- Field hashes represent the field values
- New hashes are valid based on existing field hashes and contract code hash

The proof does not include the latest root state hash of the network, otherwise the transaction would need to be resubmitted every time the root state hash changes (~ every 1 second) if the transaction had not been included at that point. Instead, the current field hash for each field being updated is used as an input to the proof, and the shard rollup is responsible for ensuring that each of those field hashes are still valid (i.e. they are included in the root state hash and haven't been updated since the proof was created).

Shard Rollup

In Polybase, each record key is assigned to a given shard group (based on a given key range). Keys are hashed, so keys with a similar namespace/prefix may not be stored within the same shard.

For each rollup a shard group leader is responsible for combining all state transition requests from clients, verifying they are valid, and creating a new shard hash which combines all changes into a new shard state hash for their key range. It then passes this hash and the corresponding proof to its parent sector.

The shard rollup must prove:

- The key prefix is allocated to this shard
- That given the existing unchanged hashes, combined with the new hashes, that a new given shard state hash is valid
- That each hash is included in the shard state hash
- Each transaction is valid:
    - Verify the client validation proof
    - Prove the field hashes (used as public inputs to the validation proof) are valid based on the last root state hash (i.e. they are included in the last root state hash rollup)

Write ordering

In distributed systems it is common to use either a logical Lamport clock [Lam78] or distributed real time clocks, such as TrueTime [Bre17]. These time based systems are designed to determine the ordering of events in a trusted system.

As Polybase operates in an untrusted environment, the protocol uses the blockchain as the timing and ordering mechanism. As the blockchain provides strong finality we can claim:

> *if T2 starts to commit after T1 finishes committing, then the timestamp for T2 is greater than the timestamp for T1*

Each rollup cycle verified by the root blockchain is considered a Polybase time interval.

Conflicts

If multiple updates are received in parallel, the shard must verify if there is a conflict between the updates. If the updates have changes to the same fields within the record, then a conflict exists. The shard will then select a deterministic random valid set of non-conflicting updates, and reject all others. Rejected requests require the client to recreate the state transition proof, and resubmit the transaction to the shard.

The deterministic random sort is derived from a randomness beacon, such as drand. This prevents network adversaries from ensuring their transactions are always processed first, by modifying their update (e.g. a nonce field) until they produce a hash with a low sort value.

Sector Rollup

The sector rollup is designed to bundle transactions across a number of shards. There may be more than one level of sector rollups to ensure continued throughput performance.

The sector rollup must prove (a subset of the shard rollup):

- The key prefix is allocated to this sector
- That given the existing unchanged hashes, combined with the new hashes, that a new given shard sector hash is valid
- That each hash is included in the shard state hash

Sector Layers

In order to ensure the required performance characteristics, each part of the rollup tree is generated by different nodes, so the work of creating proofs is shared across many nodes. Additional sector layers can be added depending on the size of the rollup.

The number of transactions per second of the rollup is based on the number of sector layers in the tree, the number of nodes and the *base tps (transactions per second)* per node. As each layer must be processed serially, each layer results in a reduction according to: *base tps / number of layers.*

The following provides an example of theoretical transactions per second based on a specified number of layers and nodes, with a *base tps* of 2,000.

Nodes	**Layers**	**TPS**

Nodes	Layers	TPS
1	1	2,000
1001 (1000 + 1)	2	1,000,000
10,016 (10,000 + 15 + 1)	3	6,600,000
100,151 (100,000 + 150 + 1)	3	66,000,000

Shard & Sector Groups

Shards and sectors hold a fixed size range of keys. Nodes are incentivised to join and work on given shards and sector groups. A configurable sized group of nodes is allocated to all shards and sector ranges, in order to ensure redundancy.

Allocation

Nodes can monitor a built-in Polybase contract to identify new opportunities to serve shards and sectors in exchange for tokens. They then send a “shard/sector allocation” write request, to request they be allocated that shard/sector. If multiple nodes request the same shard allocation at the same time (i.e. within the same rollup), then the normal conflict rules will apply.

In a similar way, shards/sectors may deallocate themselves with a “shard/sector deallocation” request, but nodes must wait for a given cool down period (such as 24 hours), to ensure that a new shard can be allocated in time. If a node stops serving the shard/sector before the cool down ends, then the stake of the indexer is slashed.

When a shard reaches a given capacity (such as 80%), it sends a “shard/sector reallocation” write request, that it intends to change its shard/sector range (the upper and/or lower bound of the range it is storing) to ensure that the size of data stored remains within an upper limit. A deterministic algorithm, that splits shards at a middle point, is used during shard splitting to ensure all nodes agree without network communication. Indexers can only split their own allocated shard range. Indexers must wait for the new shard to be allocated before they can stop accepting write requests for their old shard range.

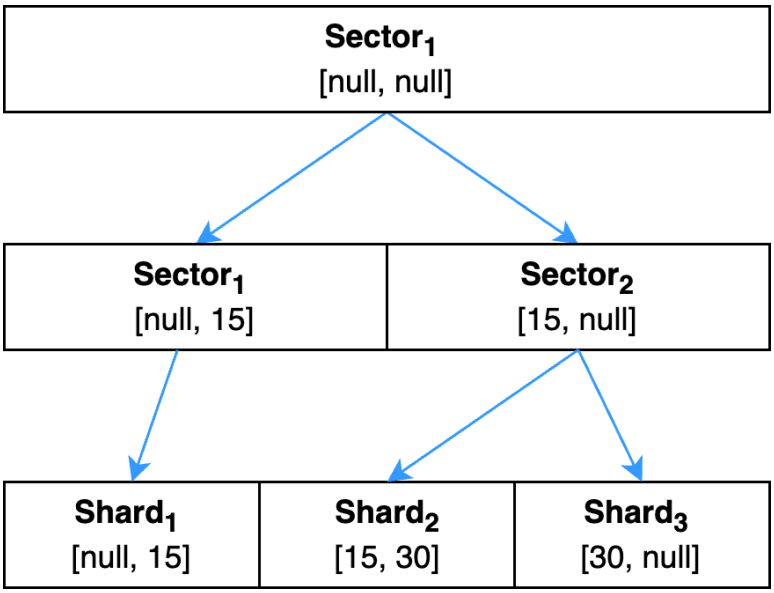


Figure 4: Shard splitting process

Blockchain Root Verification

For each rollup, the root proof is verified and the state is updated on the blockchain. This can then be used to verify any hash in rollup state is valid.

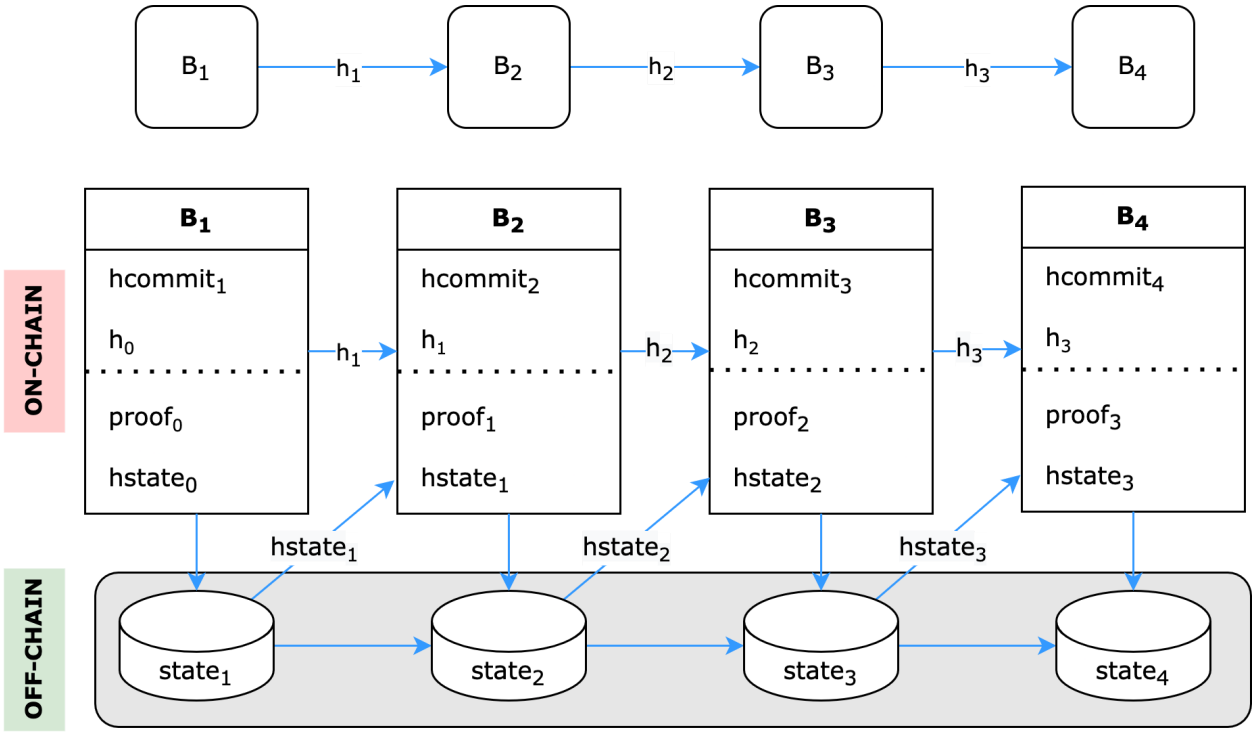


Figure 2: Validating off-chain processing and state using proofs

Data Storage Providers

The underlying data represented by the hashes stored and maintained by the state rollup protocol can be stored in any data storage provider that complies with the protocol rules. This could include local storage on the clients device, a decentralised storage network such as Filecoin/Arweave, or even a centralised storage provider such as S3. Polybase will also offer its own data storage provider service.

Polybase is designed to work across different data use cases, therefore the application creator should be responsible for accessing the trade offs between different storage providers - such as security, durability and performance.

Indexers & Aggregators

Any service or protocol can be an indexer and aggregator for the Polybase protocol, so long as they comply with the protocol rules. Indexers take the data from the data storage provider and provide fast to query views on top of the data. This allows for a wide range of indexing, search, aggregation and machine learning use cases. As the root state proofs can be verified, indexers and aggregators can build on top of these proofs, to provide proof that the aggregation was performed correctly.

Proof Algorithms

There has been rapid improvements in zk-SNARK/zk-STARK algorithms in recent years. Groth-16 [Gro16] has been traditionally the benchmark, which is used in large scale decentralised protocols such as Zcash [HBHW22]. However, this algorithm relies on a trusted setup procedure which makes updating the protocol difficult, and prevents the use of user generated functions (i.e. smart contracts), so it will not be used in Polybase.

Proofs used on Polybase either use SRS or STARK based proofs, and so they can be updated. Different proof algorithms have different trade-offs, therefore a number of different proof systems will be used in Polybase:

1. zk-VM based proofs (such as Miden VM that utilises STARKS [EYIM18]), for generating poofs for state transitions based on arbitrary user provided computation
2. Circuit based proofs (such as Plonky2 [GWC22]), for aggregating proofs
3. Rollup specific proofs (such as Caulk [ZBKMNS22]), for the shard/sector rollups

Historically, proof performance has been a major limiting factor for proof based rollups. However, by using application specific proofs for different parts of the rollup process, and utilising the latest proof algorithms, we can dramatically increase the overall performance.

In addition, further benchmarks using GPU accelerated hardware has shown 6x improvement in performance [Wan21], and further enhancements on the PlonK algorithm.

P2P

All network participants communicate over a P2P system (built on the modular p2plib library).

In the IPFS P2P protocol [Ben14], each request for content results in a walk of the DHT and a new connection before any transfer can begin (as each piece of content is equally likely to be on any given node in the network). This results in high latency when fetching content, which would be unacceptable within the requirements of the Polybase protocol.

Instead Polybase's P2P protocol optimises for stable connections to nodes who are interested in similar content, creating a sub-network of indexers and clients for each sub-section of data. Each node can be part of one or more of these sub-networks.

All P2P connections are run using *Multiplex*, allowing different Polybase sub-protocols (blockchain, indexers, DHT) to reuse the existing connections between nodes.

Availability

The CAP theorem, first proposed by Brewer in 1999 [Bre12], posits that you can only have two of the three desirable properties of Consistency, Availability, and Partition tolerance. This leads to three kinds of systems- CA, CP, and AP- based on what letter you leave out. Many systems have zero or one of these properties.

For distributed systems, it is widely agreed that partitions are inevitable, although not necessarily common [BK14]. Once you accept that partitions are inevitable, any distributed system must be prepared to forfeit either consistency (AP) or availability (CP). There are two important caveats to the theorem. Firstly, you only need to forfeit something during an actual partition. Secondly, the actual theorem is about 100% availability, whereas we propose that at a sufficiently high availability > 99.99999% (1 failure in 10^5 or less), the availability guarantee results in a real world Availability property.

Therefore as similarly argued by Brewer when describing Spanner [Bre17], in absolute terms the Polybase protocol is CP, but in real terms it provides all three CAP properties.

Polybase provides very high availability guarantees by leveraging its highly decentralised and distributed network of participants, reducing single point of failure characteristics provided by traditional cloud infrastructure.

Contract definition

Database creators can specify a data model contract for a database.

These can define:

- Validation rules
- Access controls
- Indexes

The data model schema is based on GraphQL schema model, with directives providing additional instructions for validation, access control and indexes. The following provides an example of the schema.

```
contract Account {
  # These rules would only apply when calling .set()
  name: string @regex(/^Cool.*/)
  age: number @min(10)
  balance: number @readonly @min(0)
  publicKey: string @creator # pk of creator

  # Ignores all above rules, so anything needed must be reimplemented,
  # but can still use any imported helper methods
  function transfer (a Account, b Account, amount: number) {
    # auth is in global scope of fn (based on signature when calling fn)
    if (a.publicKey == auth.publicKey) throw error('invalid user')

    # can edit both records, b/c both from the same collection
    a.balance -= amount
```

```
    b.balance += amount

    # the validation rules can be used in predefined fns
    # min has to be reimplemented/declared b/c rules are seperate
    a.balance.min(10)
  }

  # function set (a Account) { ... } // autogenerated, not defined by user
}
```

Encryption

Where `@read` validation rules are defined in the validation rule set, the resulting dataset records and indexes will be encrypted. To ensure that granular record level permissions can be applied to the dataset, each record will be encrypted with a different encryption key. Each `@read` access permission results in a new index which maps a record to its encryption key. The index is also encrypted, and each user with `@read` access has the index's encryption key.

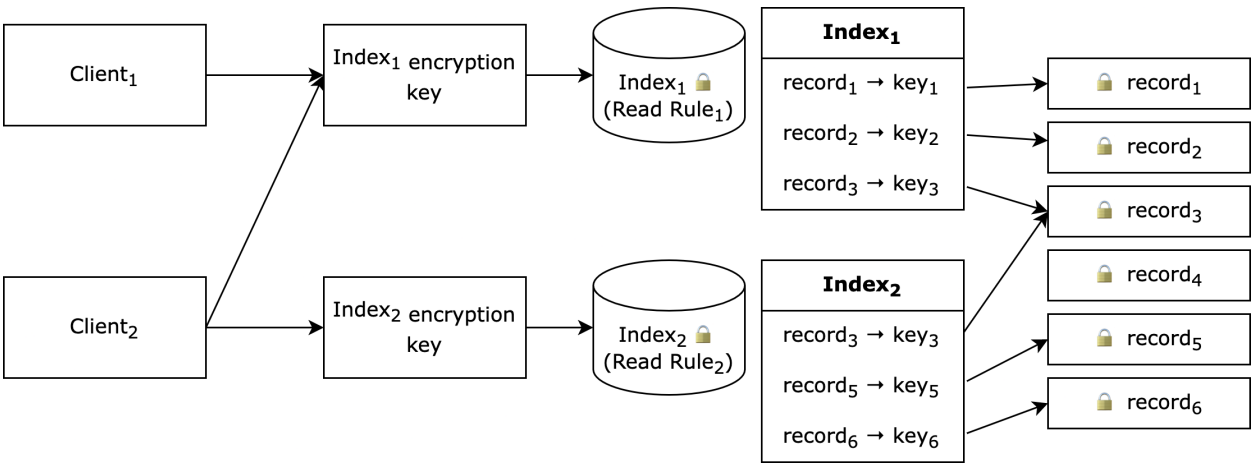


Figure 6: Encryption / decryption of indexes and related records

3. Economics

The Polybase protocol relies heavily on economic incentives and disincentives to ensure correct behaviour on the network.

Token usage

The use of the Polybase token is described throughout the paper, but the following provides a complete list of token usage:

- Indexers stake token to be eligible to perform work
- Indexers receive token for committing to store indexed data
- Indexers receive token for responding to queries
- Indexers contribute to and receive tokens from the rebate pool
- Indexers receive token for rollup of writes
- Indexers receive token for rollup of state
- Challenger stakes token to create a dispute challenge
- Challenger receive token if their challenge is valid (although less than is slashed from the infringing party)
- Challenger lose their dispute challenge stake if unsuccessful
- Indexers are slashed if they lose a dispute challenge
- Indexers are slashed for failing to adhere to a commitment
- Indexers use payment channels when working with other indexers to respond to queries

Payment

The Polybase token will be used for payment on Polybase, however it is not required that users of the protocol need to hold the native Polybase token. Instead a decentralised payment protocol can be used to transparently pay the protocol using the token of their choice. The payment protocol would convert the token in real time for each required payment.

If users do stake the Polybase token, they can use interest earned from staking to pay for usage fees on the protocol.

Fees

To ensure stable fees for users, the protocol will use an Oracle that will provide a stabilisation value that will counter balance any increase or decrease in value of the Polybase token compared to a stablecoin, ensuring currency fluctuations do not influence the fee.

A base fee multiplier is used to allow for dynamic fee changes. Fee changes will be determined by network utilisation. The protocol has a target utilisation rate (such as 80%), when utilisation falls below this rate the base fee multiplier is reduced, and when above the base fee multiplier is increased.

Worker token

Polybase's token economics are based on the utility worker token model [Sam18], that ensures token value is directly correlated to usage and value generated on the protocol.

In order to perform work on the Polybase protocol nodes must stake tokens. The work allocated and the reward received is directly proportional to the amount of tokens staked. As performing work on the protocol is designed to be profitable, the more work you are able to perform, the more profit you are able to obtain. Thus, tokens can be easily valued using the DCF (discounted cash flow) model for a perpetuity:

$$\text{PV of Perpetuity} = \frac{D}{r}$$
$$\text{PV} = \text{Present Value}$$
$$\text{D} = \text{Coupon per period}$$
$$\text{r} = \text{discount rate}$$

Rebate Pool

A portion of all query and write rewards (such as 30%) will be withheld from an indexer and instead allocated to a rebate pool. This rebate pool redistributes rewards based on the amount staked by an indexer compared to the amount they have contributed to the pool using the Cobbs-Douglas algorithm. This algorithm incentivises indexers to perform work roughly equal to their stake, as they will receive 100% rebate from the pool if they do [BWWZ19].

The Cobbs-Douglas algorithm can be represented for our purposes as:

$$pool * fee^{\alpha} * stake^{\alpha-1}$$

Where *pool* is the total rebate pool size (i.e. all contributed fees), *fee* is the ratio of fees contributed to the pool, and *stake* is the ratio of tokens staked (compared to total staked).

Payment Channels

Payment channels are used to enable efficient financial payments between protocol participants, without the need to settle every transaction immediately on-chain. Once a channel is open, network participants can send signed payments to each other which they later submit to the blockchain for payment. In the Polybase protocol, payment channels are primarily used for payment exchanges between indexers.

4. Conclusion

In this paper, we presented the requirements and design for a decentralised query, index and storage protocol for structured data that can scale horizontally. We presented the underlying advances in zk-SNARK/zk-STARK algorithms that enable secure off-chain processing and storage, as well as the economic incentives that drive correct behaviour. For the first time, the Polybase protocol provides the necessary infrastructure to enable the next generation of dApps, which are particularly compelling for enterprises. These new enterprise focused dApps will provide the basis for wide scale adoption of decentralised applications within enterprises.

5. References

[BB19] Baylina, J., & Bell, M. (2019). 4-bit Window Pedersen Hash On The Baby Jubjub Elliptic Curve.

[Ben14] Benet, J. (2014). IPFS - Content Addressed, Versioned, P2P File System.

[BGH19] Bowe, S., Grigg, J., & Hopwood, D. (2019). Halo: Recursive Proof Composition without a Trusted Setup. *IACR Cryptol. ePrint Arch., 2019*, 1021.

[BK14] Bailis, P., & Kingsbury, K. (2014). The Network is Reliable. *Commun. ACM, 57* (9), 48–55. doi:10.1145/2643130

[BM18] Baumgart, I., & Mies, S. (2008). S/Kademlia: A practicable approach towards secure key-based routing. *2*, 1–8. doi:10.1109/ICPADS.2007.4447808

[Bre12] Brewer, E. (2012). CAP twelve years later: How the ‘rules’ have changed. *Computer, 45* (2), 23–29. doi:10.1109/MC.2012.37

[Bre17] Brewer, E. (2017). Spanner, TrueTime and the CAP Theorem.

[BWWZ19] Bandeali, A., Warren, W., Wu, W.,& Zeitz P. (2019). Protocol Fees and Liquidity Incentives in the 0x Protocol.

[EYIM18] Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity.

[Gro16] Groth, J. (2016). On the Size of Pairing-Based Non-interactive Arguments. Στο M. Fischlin & J.-S. Coron (Επιμ.), *Advances in Cryptology -- EUROCRYPT 2016* (σσ. 305–326). Berlin, Heidelberg: Springer Berlin Heidelberg.

[HBHW22] Hopwood, D., Bowe, S., Hornby, T., & Wilcox, N. (2022). Zcash Protocol Specification. https://zips.z.cash/protocol/protocol.pdf

[Lam78] Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM, 21(7), 558–565. doi:10.1145/359545.359563

[MM02] Maymounkov, P., & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information system Based on the XOR Metric. Στο P. Druschel, F. Kaashoek, & A. Rowstron (Επιμ.), *Peer-to-Peer Systems* (σσ. 53–65). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Sam18] Samani, K. (2018). New Models for Utility Tokens. Multicoin Capital. https://multicoin.capital/2018/02/13/new-models-utility-tokens/

[Wal19] Walton-Pocock, T. (2019). Why Hashes Dominate in SNARKs [Blog post]. Retrieved from: https://medium.com/aztec-protocol/why-hashes-dominate-in-snarks-b20a555f074c

[Wal20] Walton-Pocock, T. (2020). PLONK Benchmarks II — ~5x faster than Groth16 on Pedersen Hashes [Blog post]. Retrieved from: https://medium.com/aztec-protocol/plonk-benchmarks-ii-5x-faster-than-groth16-on-pedersen-hashes-ea5285353db0

[Wan21] Wang, H. (2021). Plonk with GPU acceleration. Retrieved from: https://medium.com/@wanghs.thu/plonk-with-gpu-acceleration-411c77c90a22

[Won22] Wong, D. (2022). Kimchi: The latest update to Mina’s proof system [Blog post]. Retrieved from: https://minaprotocol.com/blog/kimchi-the-latest-update-to-minas-proof-system

[ZBKMNS22] Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., & Simkin, M. (2022). Caulk: Lookup Arguments in Sublinear Time