A Brief Introduction to Commitment Schemes in Decentralized Data Storage

Dr. Chloe I. Avery

December 18, 2024

Contents

1 What is a Commitment Scheme? **1**

2 Vector Commitments in Decentralized Data Storage **2**

3 KZG Commitments **3**

4 Further Reading **3**

Abstract

This piece gives a brief introduction to commitment schemes, including vector commitments and polynomial commitments, and discusses how they might be used in a decentralized data storage system.

1 What is a Commitment Scheme?

A **commitment scheme** is a protocol that is often used in cryptography which allows one party (the committer) to commit to a value (a message) while keeping it hidden from others, with the ability to reveal the committed value later to the other party (the verifier). Commitment schemes are characterized by two properties:

- **Binding**: A commitment scheme is binding if the committer is unable to change the value of the commitment once it has been made. There are two types of binding that a scheme may require:
  - **Computationally Binding**: A commitment scheme is computationally binding if it is computationally infeasible to find another value that has the same commitment.
  - **Perfectly Binding**: A commitment scheme is perfectly binding if there is no other value with the same commitment.
- **Hiding**: A commitment scheme is hiding if the commitment reveals no information about the value being committed to. There are two types of hiding that a scheme may require:
  - **Computationally Hiding**: A commitment scheme is computationally hiding if, given the commitment, it is computationally infeasible to find the value that was committed to.
  - **Perfectly Hiding**: A commitment scheme is perfectly hiding if the commitment gives no information at all about the value being committed to.

Commitment schemes have many applications, including zero-knowledge proofs, secure multi-party computation, digital signatures, and decentralized data storage.

A simple example of a commitment scheme is this: Alice writes a message on a piece of paper and locks it in a box. Alice gives the box to Bob, but retains the key. Later, to open the commitment, Alice can unlock the box and show the message to Bob.

Now, suppose instead that your data is a vector $[a_0, \ldots, a_{n-1}]$. Sure, you could make a commitment to this data as you otherwise would (by, for example, appending the data or by committing to each a_i individually). However, what if you wanted to make a commitment to the entire vector and then at the reveal stage be able to reveal one element of the vector (a_i for some i) without revealing the rest of the vector. This idea is called a **vector commitment**. I like to think of these as a row of locked boxes and a ring of keys, one key belonging to each box. A commitment can be made by handing the verifier the row of boxes, and the contents of each box can be revealed by the committer handing the verifier the corresponding key.

A great example of vector commitments is Merkle Trees, which I explain in my other piece, [2].

2 Vector Commitments in Decentralized Data Storage

In Orchid's decentralized data storage system, the setup is this: Clients erasure code[1] their data, thus breaking it into pieces called erasure blocks, and give each erasure block to a different Provider. We call this collection of Providers responsible for this data a *cohort*. Each Provider in the cohort regularly posts commitments on chain, monitoring the commitments of the others in their cohort. Providers get paid to participate in the repair of any lost data (for example, sending parts of their data to a new Provider if a Provider has failed to make a commitment) as well as for storing data.

We use these commitments as proof that the Provider is still holding the erasure block in question. We care mostly about the binding feature of commitment schemes, however, a corollary of the hiding feature in some schemes is that the commitment is much smaller than the data being committed to. This is particularly useful in systems like ours where commitments are posted on-chain.

In our system, rather than committing to the entire piece of the data, Providers can commit to a smaller subset of the piece, chosen by a random beacon. The subset being chosen randomly provides assurances that the Provider in question likely (with high probability) has the entire erasure-coded piece of data because the Provider is making these commitments repeatedly. That is, it is unlikely that if the Provider was not storing some or all of the data that they would be able to repeatedly make correct commitments.

However, Providers in the system will likely belong to multiple cohorts, therefore having to post multiple commitments on chain during each epoch. Vector commitments allow providers to bundle commitments together, committing to one polynomial for all of their data. Many vector commitment schemes (such as Merkle Tree commitments) have the nice property that the commitment size does not change with the number of elements in the vector, making it efficient for a Provider to make a commitment to data from multiple Clients at once.

In our system, Clients are able to go offline for a period of time, and are assured by the incentive design and self-repair mechanisms that their data will be there when they get back. When a Client is online, the Provider can request payment from the Client by opening their outstanding commitments and sending this proof that they made correct commitments to the Client directly. However, if a Client is offline and a Provider wants to receive payment for their outstanding commitments, the Provider must post these proofs (the opening/verification of the commitment) on-chain. Posting anything on-chain can get expensive, and unfortunately, for many well-known vector commitment schemes (such as Merkle Tree commitments), the size of the reveal scales with the size of the vector being committed to. Therefore, doing payment settlement for even just one of a Provider's Clients could be costly, as it depends on the number of Clients that the Provider has. KZG commitments, which we discuss in Section 3, have the nice property that the proof size is independent of the vector size, therefore dramatically reducing costs.

[1] Erasure coding is a useful technique for efficiently breaking data into overlapping pieces such that any large enough subset of the pieces can be used to reconstruct the data.

3 KZG Commitments

Polynomial commitment schemes are a type of commitment scheme that allows one to make a commitment to a polynomial specifically.

We can format our data as a vector of length n, $[a_0, \ldots, a_{n-1}]$. Then, there are two main ways of creating a polynomial that contains the data.

The a_i can be the coefficients of the polynomial:

$$P(x) = \sum_{i=1}^{n-1} a_i x^i = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

Or, the polynomial can interpolate the points (i, a_i). That is, we can construct a polynomial which passes through each of those points. To do this, for each i, we first construct a polynomial $Q_i(x)$ which evaluates to 0 at each $j \neq i$ for $j \in \{0, \ldots, n-1\}$ and evaluates to 1 at i:

$$Q_i(x) = \prod_{\substack{0 \leq j \leq n-1 \\ j \neq i}} \frac{x - j}{i - j}$$

Once we've done this, we can scale and sum these polynomials to get our polynomial P such that $P(i) = a_i$ for each $i \in \{0, \ldots, n-1\}$:

$$P(x) = \sum_{i=0}^{n-1} a_i Q_i(x)$$

KZG commitments, first introduced by and named for Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg in [4], are a particularly nice kind of polynomial commitment because they allow the committer to reveal an evaluation of the polynomial which is verifiable without revealing the entire polynomial. When the polynomial being committed to is one which interpolates points coming from a vector, the KZG commitment is actually a vector commitment. That is, given a vector $[a_0, \ldots, a_{n-1}]$, we know we can construct a polynomial P such that $P(i) = a_i$ for each $i \in \{0, \ldots, n-1\}$ and construct a KZG commitment to P. Then, as a KZG commitment, the committer can reveal an evaluation of P (specifically, for some i that $P(i) = a_i$) without revealing the rest of P (and thus not revealing a_j for $j \neq i$).

4 Further Reading

References [3] and [5] give particularly nice explanations of how KZG commitments work, and [5] also teaches the mathematical background necessary to understand them. Reference [6] discusses the EIP-4844 reference implementation of KZG commitments as used in Ethereum and discusses further how Orchid uses KZG commitments in decentralized data storage, from a coding perspective. Finally, reference [1] gives more background on Orchid's decentralized data storage project and [2] discusses precisely how KZG commitments are used in Orchid storage, beyond what we describe in Section 2.

References

[1] Chloe Avery and Justin Sheek. Orchid Storage: A New Open Source Initiative For Decentralized, Incentivized Data Storage. `https://www.orchid.com/storage-litepaper-latest.pdf`.

[2] Chloe Avery and Justin Sheek. Storage Auditing Using Merkle Trees and KZG Commitments. `https://www.orchid.com/storage-auditing-latest.pdf`.

[3] Dankrad Feist. KZG Polynomial Commitments. `https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html`.

[4] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. `https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf`.

[5] Luksgrin. A quick insight on Algebra and KZG Commitments. https://github.com/luksgrin/opensense-algebra-and-kzg/blob/main/algebra_and_kzg.ipynb.

[6] Patrick Niemeyer. Based Blobs - Using Polynomial Commitments in Your Projects with the EIP-4844 KZG Reference Libs. https://pat.net/ckzg/.