A Documentation of Ethereum's PeerDAS

Benedikt Wagner [1] Arantxa Zapico [1]

[1] Ethereum Foundation Cryptography Research
{benedikt.wagner,arantxa.zapico}@ethereum.org

Abstract

Data availability sampling allows clients to verify availability of data on a peer-to-peer network provided by an untrusted source. This is achieved without downloading the full data by sampling random positions of the encoded data. The long-term vision of the Ethereum community includes a comprehensive data availability protocol using polynomial commitments and tensor codes. As the next step towards this vision, an intermediate solution called PeerDAS is about to integrated, to bridge the way to the full protocol. With PeerDAS soon becoming an integral part of Ethereum's consensus layer, understanding its security guarantees is essential.

This document aims to describe the cryptography used in PeerDAS in a manner accessible to the cryptographic community, encouraging innovation and improvements, and to explicitly state the security guarantees of PeerDAS.

Contents

1 Introduction 2
1.1 History and Ethereum's Roadmap . . . 2
1.2 Outline of this Document . . . 3

2 Preliminaries 3
2.1 Reed-Solomon Codes and Roots of Unity . . . 3
2.2 Groups and Assumptions . . . 4
2.3 The Algebraic Group Model . . . 5
2.4 KZG Commitments and Multiproofs over Cosets . . . 6
2.5 Erasure Code Commitments and Data Availability Sampling . . . 6

3 The Erasure Code Commitment used in PeerDAS 8
3.1 Warm-Up: Single Row Commitment Scheme . . . 8
3.2 The Full Commitment Scheme . . . 11

4 Optimizations 14
4.1 Computing Commitment and All Openings . . . 14
4.2 Optimizations for Verification . . . 15
4.3 Efficient Reconstruction of Data . . . 17

Disclaimer. Purpose of this work is to document the PeerDAS update for Ethereum in a way that is accessible to the cryptographic community. Our analysis concerns commit 54093964c95f of the Ethereum consensus specification [Eth24a], and analysis does not necessarily carry over to future versions.

Acknowledgments. We thank Justin Traglia, Gottfried Herold, George Kadianakis, Dmitry Khovratovich, and all members of Ethereum Foundation Cryptography Research for providing feedback on earlier drafts of this document.

1 Introduction

Data availability sampling [ASBK21, HASW23] enables clients to verify the availability of data (e.g., a list of transactions) on a peer-to-peer network. To this end, data is redundantly encoded and stored across different nodes on the network. Clients are only assumed to have downloaded a succinct commitment to the data. To check availability, clients query random positions of the encoding from these nodes and verify them against the commitment. If a sufficient number of these random samples succeed, clients can conclude that the data is available.

To formalize the security guarantees provided by data availability sampling (DAS), Hall-Andersen, Simkin, and Wagner [HASW23] recently defined it as a cryptographic primitive and analyzed several constructions. In essence, a data availability solution must satisfy two key properties: (1) *soundness*, meaning that if enough clients accept, then some data can be reconstructed from their transcripts, ensuring availability, and (2) *consistency*, ensuring that clients with the same commitments will always reconstruct the same data.

The concept of data availability sampling is central to Ethereum's roadmap [D'A23]. One of the solutions analyzed in [HASW23] is expected to be used in the future. At the time of writing, an intermediate solution called PeerDAS is about to be integrated into the Ethereum protocol. PeerDAS aims to pave the way for the full data availability solution and will soon become an essential part of Ethereum's protocol. Therefore, it is crucial to formally understand its security guarantees.

This document focuses on PeerDAS as described in Ethereum's consensus specifications [Eth24a, Eth24b]. Our intention is two-fold: first, we aim to provide a description of the cryptography used in PeerDAS that is accessible to the cryptographic community, potentially leading to new ideas and improvements that can be incorporated in the future. Second, we want to explicitly state the security and efficiency guarantees of PeerDAS. In terms of security, this document justifies the following claim.

Theorem 1 (Main Theorem, Informal). *Assuming plausible cryptographic hardness assumptions,* PeerDAS *is a secure data availability sampling scheme in the algebraic group model, according to the definition in [HASW23].*

1.1 History and Ethereum's Roadmap

We use this section to provide some context explaining how PeerDAS relates to the roadmap of Ethereum. For more details, we refer the reader to [D'A23]. Motivated by increasing scalability of Ethereum using rollups, Ethereum's long term vision is to use data availability sampling. The solution that is currently envisioned is based on KZG commitments and a two-dimensional tensor code of the Reed-Solomon code. PeerDAS is an intermediate solution that is used to introduce the concept of sampling and most of the cryptographic building blocks that are needed for this full solution.

The Scaling Problem. Modern blockchains, including Ethereum, face significant scalability challenges. At the heart of this issue is keeping balance between making space expensive enough to limit workload on nodes and providing sufficient capacity for transactions. The current approach involves setting a gas limit, which controls the amount of computational work that can be included in each block. On the downside, this limits the capacity for transactions. One can not simply increase the gas limit per block, as this would also require nodes to perform more work to verify the blockchain.

Rollups to the Rescue. One promising solution to this scalability problem is the implementation of rollups, which are a type of Layer 2 (L2) scaling solution. Rollups operate under the control of an Ethereum mainnet contract (called Layer 1, or L1) and are designed to handle large volumes of transactions off-chain, thereby reducing the computational burden on the Ethereum mainnet. In a nutshell, a rollup functions as follows: a sequencer collects a batch of L2 transactions and submits this batch to the rollup contract on L1. Importantly, nodes on L1 do not execute and verify all transactions, but could, for example, verify a succinct proof that the transactions are correct. In this way, the block can execute much more transactions without increasing the work for nodes on L1. At the same time, the L2 transactions inherit security and consensus from Ethereum L1. That being said, it is clear that Ethereum mainnet (L1) primarily stores data, and the availability of this data on L1 is essential for maintaining the liveness of the rollup and enabling nodes to synchronize the rollup's state accurately.

Proto-Danksharding and EIP 4844. To prepare for the use of data availability sampling, EIP 4844 [BFL$^+$22] has introduced a blob-carrying transaction type. Essentially, transactions can now contain so-called *blobs* of data. More precisely, a blob consists of d_r field elements, and there are separate fees for blobs and execution. Notably, there is no data availability sampling yet: all nodes ensure data availability by downloading the entire blob. This also means that blobs are limited in size, but it is a first step towards the full solution.

PeerDAS. The next step after EIP 4844 is PeerDAS, which is what we study in this document. In PeerDAS, data is first split into blobs as in EIP 4844. Then, these blobs are individually extended using a Reed-Solomon code: each blob defines a polynomial of some degree $d_r - 1$, and this polynomial is evaluated at $2d_r$ points to obtain an *extended blob*. These extended blobs are then arranged in a matrix, where each extended blob is one row, and clients download KZG commitments [KZG10a] for each extended blob. In addition to this encoding, PeerDAS also introduces *sampling*: to check availability, each client now downloads random *columns* of this matrix and verifies their KZG opening proofs with respect to the commitments.

Tensor Codes and the Future of DAS. The long-term vision for data availability sampling in Ethereum is a two-dimensional variant of PeerDAS. Concretely, after the matrix is obtained from horizontally extending each blob as in PeerDAS, this matrix is also extended *vertically*. This can again be done using a Reed-Solomon code: if we assume that there are d_c blobs, then each column of the PeerDAS matrix contains d_c field elements, and so each column can be interpreted as a degree $d_c - 1$ polynomial and evaluated at $2d_c$ points. As a result, the matrix contains $4d_cd_r$ evaluations of a bivariate polynomial over variables X, Y of individual degree at most $d_r - 1$ in X and $d_c - 1$ in Y. From a coding theory perspective, this means that the data is encoded using the tensor code of two Reed-Solomon codes, see [HASW23]. The second change compared to PeerDAS would be that clients sample single cells instead of entire columns. We note that this scheme has been analyzed formally in [HASW23]. Compared to PeerDAS, its main advantage is that individual rows or columns can easily be reconstructed, and that sampling individual cells is more efficient in terms of bandwidth [NB23].

1.2 Outline of this Document

In Section 2, we introduce our notation and recall the necessary background. This includes Reed-Solomon codes and KZG commitments, as well as the formal framework for data availability sampling by Hall-Andersen, Simkin, and Wagner [HASW23]. Then, in Section 3, we define and analyze the cryptographic core of PeerDAS. The final section is Section 4, where we discuss how algorithms in PeerDAS are implemented efficiently.

2 Preliminaries

For a finite set Z, we write $z \xleftarrow{\$} Z$ to say that z is sampled uniformly at random from Z. For an algorithm A, we denote the running time of A by $\mathbf{T}(\mathsf{A})$. The notation $y := \mathsf{A}(x)$ implicitly means that A is deterministic and that A is run on input x and the output is y. For a probabilistic A, we instead write $y := \mathsf{A}(x; \rho)$ if we want to make the random coins ρ explicit and $y \leftarrow \mathsf{A}(x)$ if not. We use the notation $y \in \mathsf{A}(x)$ to indicate that y is a possible output of A on input x. Throughout this document, we use standard cryptographic notions such as a security parameter λ, PPT (probabilistic polynomial-time) algorithms, and negligible functions. We denote the set of the first r natural numbers by $[r] = \{1, \dots, r\} \subseteq \mathbb{N}$. For strings $x \in \Sigma^{\ell_1}$ and $y \in \Sigma^{\ell_2}$ over the same alphabet, we write $x \parallel y \in \Sigma^{\ell_1+\ell_2}$ to denote their concatenation.

2.1 Reed-Solomon Codes and Roots of Unity

In this section, we call relevant mathematical background. This includes Reed-Solomon codes, basic facts about roots of unity and Lagrange polynomials, and the reverse bit ordering.

Erasure Codes. An erasure code is specified by an encoding function $\mathcal{C} \colon \Gamma^k \to \Lambda^n$ that maps messages over an alphabet Γ to codewords over an alphabet Λ. The crucial property of such an erasure code is that given any $t < n$ symbols of a codeword, one can reconstruct the message. Sometimes, we also refer to the image of the function as the erasure code, which is standard in the coding theory literature. That is, we treat the erasure code as a set $\mathcal{C} \subseteq \Lambda^n$ given by $\mathcal{C} = \mathcal{C}(\Gamma^k)$. For this document, we will only focus on (variants of) one particular erasure code, as discussed next.

Reed-Solomon Codes. PeerDAS uses (a variant of) the Reed-Solomon code, so we recall this code here. The Reed-Solomon code is defined over a finite field $\mathbb{F}$, which in our context typically is $\mathbb{F} = \mathbb{Z}_q$ for some large prime q. It is also parameterized by an integer $d \in \mathbb{N}$ and a set $\mathcal{L} \subseteq \mathbb{F}$. With these parameters at hand, the Reed-Solomon code (viewed as a set, see above) contains all codewords of the form $(f(x))_{x\in\mathcal{L}}$, where $f \in \mathbb{F}[X]$ is a polynomial of degree strictly less than d. We sometimes interpret codewords as vectors in $\mathbb{F}^{|\mathcal{L}|}$, assuming there is an implicit ordering on $\mathcal{L}$. Alternatively, we can interpret codewords as maps $f\colon \mathcal{L} \to \mathbb{F}$. We denote this code, i.e., the set of all such codewords, by $\mathcal{RS}[d, \mathcal{L}, \mathbb{F}]$. Often it is also useful to view the Reed-Solomon code by its encoding function taking d field elements specifying a polynomial f as input, and returning its evaluations over $\mathcal{L}$, i.e., $\mathcal{RS}[d, \mathcal{L}, \mathbb{F}]\colon \mathbb{F}^d \to \mathbb{F}^{|\mathcal{L}|}$. The Reed-Solomon code is a so-called MDS code: any d of the evaluations allow to reconstruct the polynomial via interpolation.

Lagrange Polynomials and Roots of Unity. Given some finite field $\mathbb{F}$, let ω be a primitive mth root of unity, i.e., $\omega^m = 1$ and $\omega^r \neq 1$ for all $1 \leq r < m$. We denote by $\mathbb{H} = \{\omega^1, \ldots, \omega^m\}$ the group generated by ω, that has cardinality m. As we will use $\mathbb{H}$ as the evaluation domain of a Reed-Solomon code, we assume that $\mathbb{H}$ is implicitly ordered. The most natural ordering would be $\omega^1, \ldots, \omega^m$, but we will see a more useful ordering below. The ith Lagrange basis polynomial associated to $\mathbb{H}$ is denoted by $\lambda_i(X)$. The vanishing polynomial of $\mathbb{H}$ will be denoted by $z_{\mathbb{H}}(X)$. These polynomials are:

$$z_{\mathbb{H}}(X) = X^m - 1, \quad \text{and} \quad \lambda_i(X) = \frac{\omega^i}{m}\frac{(X^m - 1)}{(X - \omega^i)} \quad \text{for all } i \in [m].$$

Cosets and Reverse Bit Ordering. Assume that m is a power of two, that is, $m = 2^t$ for some t. For each $j \in [t]$, there exists a subset $\mathbb{H}_j$ of $\mathbb{H}$ of size 2^{t-j} that is a subgroup. In particular, we have $\mathbb{H}_j = \{\omega^i \in \mathbb{H} \mid i \equiv 0 \mod 2^j\}$. Note that $\nu = \omega^{2^j}$ is a 2^{t-j}th root of unity and $\mathbb{H}_j = \{\nu, \nu^2, \ldots, \nu^{2^{t-j}}\}$. Also, for any s with $0 \leq s < 2^j$, the set $\omega^s\mathbb{H}_j = \{\omega^i \in \mathbb{H} \mid i \equiv s \mod 2^j\}$ is a *coset* of $\mathbb{H}_j$, that we will denote by $\mathbb{H}_{j,s}$. We are interested in these cosets because, as it is the case for subgroups, their vanishing and Lagrange polynomials have sparse representations. We denote by $\{\lambda_i^{j,s}(X)\}_{i=1}^r$ and $z_{j,s}(X)$ these Lagrange and vanishing polynomials and set $r = 2^{t-j}$. Then, we have

$$z_{j,s}(X) = X^r - \omega^{sr}, \quad \text{and} \quad \lambda_i^{s,r}(X) = \frac{\omega^i}{r\omega^{s(r-1)}}\frac{X^r - \omega^{sr}}{X - \omega^{si}} \quad \text{for all } i \in [r].$$

For this reason, we order the elements in $\mathbb{H}$ following a *reverse bit ordering* [KDF22]. That is, for a set of roots of unity of size 2^t, $\mathbb{H} = \{1, \omega, \omega^2, \ldots, \omega^{2^t-1}\}$, we consider the binary representation of all the indices from 0 to $2^t - 1$ and take the ordering given by reversing each binary representation. For instance, if $m = 2^3$, then $\mathbb{H} = \{1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, , \omega^6, \omega^7\}$ with indices $\{000, 001, 010, 011, 100, 101, 110, 111\}$. Then, the reverse bit ordering is $\{1, \omega^4, \omega^2, \omega^6, \omega, \omega^5, \omega^3, \omega^7\}$, where the re-ordering is given by set $\{000, 100, 010, 110, 001, 101, 011, 111\}$. We will denote by $\tilde{\omega}_i$ the ith element in this ordering. In general, we notice that for any $j \in [t]$, if we split the elements in this ordering into consecutive chunks of size 2^{t-j}, then these chunks exactly correspond to the cosets $\mathbb{H}_{j,s}$. We show an example in Figure 1.

2.2 Groups and Assumptions

A bilinear group $\mathcal{G}$ is a tuple $\mathcal{G} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q with generators $\mathcal{P}_1, \mathcal{P}_2$, respectively, and $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate bilinear map, i.e., the point $\mathcal{P}_T = e(\mathcal{P}_1, \mathcal{P}_2)$ generates $\mathbb{G}_T$. We also assume that there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ (this is sometimes called the type-3 pairing setting [GPS08]). Elements in these groups are denoted implicitly as $[a]_\gamma = a\mathcal{P}_\gamma$, where $\gamma \in \{1, 2, T\}$. With this notation, we have $e([a]_1, [b]_2) = [ab]_T$. While in practice a fixed $\mathcal{G}$ is used, security proofs asymptotically only make sense if we consider a family of groups. Therefore, we assume that there is some algorithm PGGen that takes as input the security parameter 1^λ and outputs the description of $\mathcal{G}$. The security of KZG commitments with batch opening [KZG10b], as well as our security proof for PeerDAS, relies on the d-BSDH assumption as defined next. This assumption is an instance of the Uber assumption for rational fractions and flexible targets as studied by Bauer, Fuchsbauer and Loss [BFL20]. To gain confidence in this class of assumptions, they show that it is implied by a suitable q-type variant of the discrete logarithm assumption.

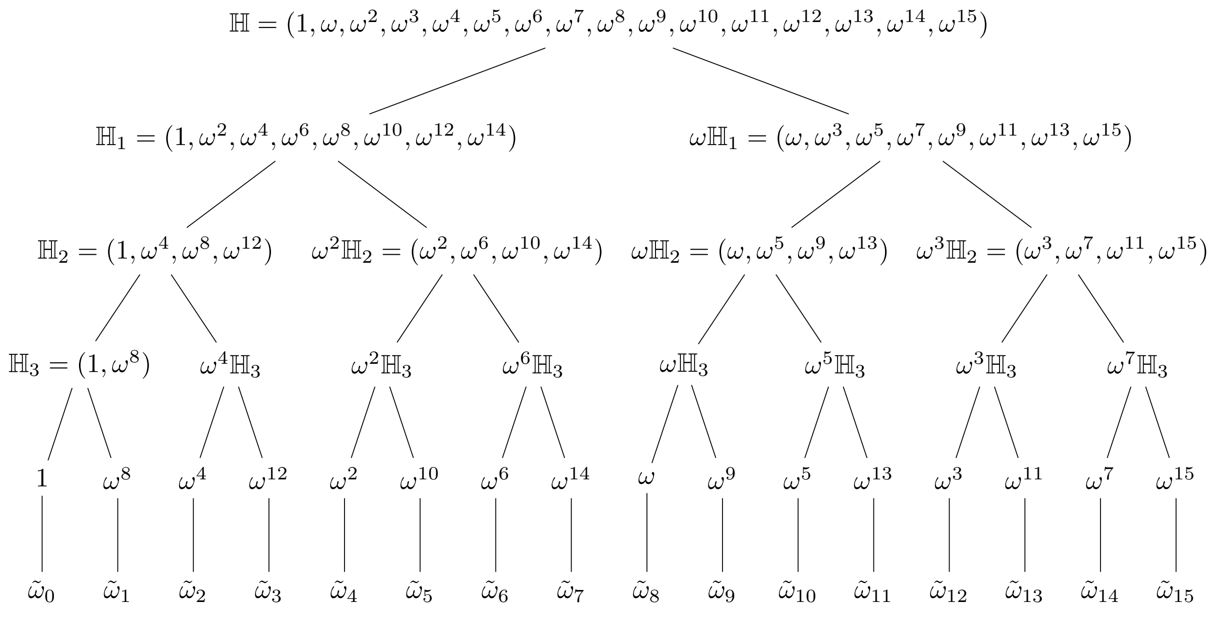

$\mathbb{H} = (1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, \omega^6, \omega^7, \omega^8, \omega^9, \omega^{10}, \omega^{11}, \omega^{12}, \omega^{13}, \omega^{14}, \omega^{15})$
$\mathbb{H}_1 = (1, \omega^2, \omega^4, \omega^6, \omega^8, \omega^{10}, \omega^{12}, \omega^{14})$
$\omega\mathbb{H}_1 = (\omega, \omega^3, \omega^5, \omega^7, \omega^9, \omega^{11}, \omega^{13}, \omega^{15})$
$\mathbb{H}_2 = (1, \omega^4, \omega^8, \omega^{12})$
$\omega^2\mathbb{H}_2 = (\omega^2, \omega^6, \omega^{10}, \omega^{14})$
$\omega\mathbb{H}_2 = (\omega, \omega^5, \omega^9, \omega^{13})$
$\omega^3\mathbb{H}_2 = (\omega^3, \omega^7, \omega^{11}, \omega^{15})$
$\mathbb{H}_3 = (1, \omega^8)$
$\omega^4\mathbb{H}_3$
$\omega^2\mathbb{H}_3$
$\omega^6\mathbb{H}_3$
$\omega\mathbb{H}_3$
$\omega^5\mathbb{H}_3$
$\omega^3\mathbb{H}_3$
$\omega^7\mathbb{H}_3$
1 ω^8 ω^4 ω^{12} ω^2 ω^{10} ω^6 ω^{14} ω ω^9 ω^5 ω^{13} ω^3 ω^{11} ω^7 ω^{15}
$\tilde{\omega}_0$ $\tilde{\omega}_1$ $\tilde{\omega}_2$ $\tilde{\omega}_3$ $\tilde{\omega}_4$ $\tilde{\omega}_5$ $\tilde{\omega}_6$ $\tilde{\omega}_7$ $\tilde{\omega}_8$ $\tilde{\omega}_9$ $\tilde{\omega}_{10}$ $\tilde{\omega}_{11}$ $\tilde{\omega}_{12}$ $\tilde{\omega}_{13}$ $\tilde{\omega}_{14}$ $\tilde{\omega}_{15}$

Figure 1: The tree of subgroups and cosets for the group of mth roots of unity where $m = 2^4$. The figure visualizes the reverse bit ordering.

Definition 1 (d-BSDH Assumption)**.** We say that the d-BSDH assumption holds relative to PGGen, if for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{d\text{-}\mathsf{BSDH}}_{\mathcal{A},\mathsf{PGGen}}(\lambda) := \Pr\left[\, c \neq \tau \wedge W = [1/(\tau - c)]_T \;\middle|\; \begin{array}{l} \mathcal{G} \leftarrow \mathsf{PGGen}(1^\lambda),\ \tau \xleftarrow{\$} \mathbb{Z}_q, \\ (c, W) \leftarrow \mathcal{A}(\mathcal{G}, d, ([\tau^i]_1)_{i=0}^d, ([\tau^i]_2)_{i=0}^d) \end{array} \right].$$

2.3 The Algebraic Group Model

For parts of our security proof (concretely, for code-binding), we will rely on the algebraic group model introduced by Fuchsbauer, Kiltz, and Loss [FKL18], which is widely used to analyze cryptographic protocols. When proving security in the algebraic group model, we assume that the adversary is a so-called *algebraic algorithm*. Informally, an algebraic algorithm derives new group elements only by applying the group operation to received group elements, and it can therefore explain each group element that it outputs by explaining how it combined the received group elements. This is formalized as follows: suppose an adversary $\mathcal{A}$ outputs a group element $[z]_\gamma \in \mathbb{G}_\gamma$ for $\gamma \in \{1, 2, T\}$. Let $[x_1]_\gamma, \ldots, [x_m]_\gamma \in \mathbb{G}_\gamma$ be the list of all group elements that $\mathcal{A}$ received so far, either as input or via oracles. Then, $\mathcal{A}$ must also output coefficients $z_1, \ldots, z_m \in \mathbb{Z}_q$ such that $z = \sum_{i=1}^m z_i x_i$.

Remark 1 (Extensions of the Algebraic Group Model). Lipmaa [Lip22], and later Lipmaa, Parisella, and Siim [LPS23] have proposed refined versions of the algebraic group model, which give adversaries access to obliviously sampled group elements. This is motivated by the observation that in practice, one can sample a random group element $[z]_\gamma$ obliviously, i.e., without learning $z \in \mathbb{Z}_q$. To formally model this, they give the adversary access to an oracle that outputs obliviously sampled group elements, for which neither the adversary nor the reduction knows the discrete logarithms. They also note that one can prove false knowledge assumptions in the algebraic group model without their extension, which means that results in algebraic group model have to be used with care. Still, we opted not to use their model in this document to keep the proofs as readable as possible. Instead, we avoid using the algebraic group model whenever possible, and only use it in the proof of code-binding in Lemma 3. We view studying code-binding in the extended algebraic group models [Lip22, LPS23] or even without the algebraic group model as an interesting problem for future work. The results in [LPS23] about extractability of KZG seem promising in this regard.

2.4 KZG Commitments and Multiproofs over Cosets

The polynomial commitment scheme introduced by Kate, Zaverucha and Goldberg [KZG10a] for a degree $d \in \mathbb{N}$ allows to commit succinctly to a polynomial $f \in \mathbb{Z}_q[X]$ of degree d and later compute succinct opening proofs that convince a verifier that $f(x) = y$ for evaluation points $x \in \mathbb{Z}_q$. In PeerDAS, we need to open multiple elements at once, i.e., we make use of so-called KZG multiproofs. We first define the setup and commitment algorithms of the KZG scheme. The setup algorithm samples a random secret τ and outputs its powers over the groups. The commitment algorithm then evaluates f on τ over $\mathbb{G}_1$:

- $\mathsf{KZG.Setup}(1^\lambda, 1^d) \to \mathsf{ck}$:
  1. Generate $\mathcal{G} \leftarrow \mathsf{PGGen}(1^\lambda)$, where $\mathcal{G} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathcal{P}_1, \mathcal{P}_2)$.
  2. Sample $\tau \overset{\$}{\leftarrow} \mathbb{Z}_q$ and set commitment key $\mathsf{ck} := \left(\mathcal{G}, d, ([\tau^i]_1)_{i=0}^d, ([\tau^i]_2)_{i=0}^d\right)$.
- $\mathsf{KZG.Com}(\mathsf{ck}, f) \to \mathsf{com}$ for $f \in \mathbb{Z}_q[X]$ with $\deg f \leq d$:
  1. Write $f(X) = \sum_{i=0}^d f_i X^i$.
  2. Set $\mathsf{com} = [f(\tau)]_1 = \sum_{i=0}^d f_i [\tau^i]_1$.

We now turn to multiproofs and their verification. For concreteness, we focus on opening polynomials over cosets of a subgroup of the group of roots of unity $\mathbb{H}$ where we assume $\mathbb{H} \geq d$. The algorithms take as input the size of $\mathbb{H}$, denoted by $m = 2^t$, the size of the subgroup, denoted by $r = 2^{t-j}$, and an index $\hat{s}$ with $0 \leq \hat{s} < 2^j$ identifying the coset:

- $\mathsf{KZG.MultiOpen}(\mathsf{ck}, f, m, r, \hat{s}) \to \pi$ for $f \in \mathbb{Z}_q[X]$ with $\deg f \leq d$, $m = 2^t$, $r = 2^{t-j}$, and $0 \leq \hat{s} < 2^j$:
  1. Set $\hat{m}_i := f(\tilde{\omega}_{\hat{s}r+i})$ for each $i \in [r]$.
  2. Let s be such that $0 \leq s < 2^j$ and $\omega^s \mathbb{H}_j = \{\tilde{\omega}_{\hat{s}r}, \ldots, \tilde{\omega}_{\hat{s}r+(r-1)}\}$.
  3. Set $I(X) := \sum_{i=1}^r \hat{m}_i \lambda_i^{j,s}(X) \in \mathbb{Z}_q[X]$.
  4. Let $Q(X) := (f(X) - I(X))/z_{j,s}(X)$ and observe that $Q \in \mathbb{Z}_q[X]$ and $\deg Q \leq d$.
  5. Write $Q(X) = \sum_{i=0}^d Q_i X^i$ and set $\pi := [Q(\tau)]_1 = \sum_{i=0}^d Q_i [\tau^i]_1$.
- $\mathsf{KZG.MultiVer}(\mathsf{ck}, \mathsf{com}, m, r, \hat{s}, \hat{m}, \pi) \to b$ for $\hat{m} \in \mathbb{Z}_q^l$ and $m = 2^t$, $r = 2^{t-j}$, and $0 \leq \hat{s} < 2^j$:
  1. Let s be such that $0 \leq s < 2^j$ and $\omega^s \mathbb{H}_j = \{\tilde{\omega}_{\hat{s}r}, \ldots, \tilde{\omega}_{\hat{s}r+(r-1)}\}$.
  2. Set $I(X) := \sum_{i=1}^r \hat{m}_i \lambda_i^{j,s}(X) \in \mathbb{Z}_q[X]$.
  3. If $e\left(\mathsf{com} - [I(\tau)]_1, [1]_2\right) = e\left(\pi, [z_{j,s}(\tau)]_2\right)$, return $b := 1$. Otherwise, return $b := 0$.

This scheme is correct, i.e., honestly committing and opening makes the verification algorithm accept.

2.5 Erasure Code Commitments and Data Availability Sampling

Hall-Andersen, Simkin, and Wagner [HASW23] have introduced a cryptographic framework to analyze data availability sampling schemes. In a nutshell, their framework consists of three steps:

1. *Erasure Codes.* Given a string of data, an erasure code is used to encode the data. For example, one could use the Reed-Solomon code.
2. *Erasure Code Commitment.* The cryptographic core is formalized as a special commitment scheme that commits to codewords of an erasure code. For example, a polynomial commitment scheme may be used to commit to codewords of a Reed-Solomon code.
3. *Compilation to Data Availability Sampling.* The erasure code commitment scheme is compiled to a data availability sampling scheme. If the commitment scheme satisfies suitable security notions, then the data availability sampling scheme is secure.

We follow this approach to justify the security of PeerDAS. Concretely, we will specify an erasure code and an associated erasure code commitment scheme that models what happens in PeerDAS. We will show the security properties of this commitment scheme, which then implies the security of PeerDAS using [HASW23]. Let us now recall the definition of erasure code commitments from [HASW23].

Definition 2 (Erasure Code Commitment Scheme)**.** Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. An erasure code commitment scheme for $\mathcal{C}$ is a tuple $\mathsf{CC} = (\mathsf{CC.Setup}, \mathsf{CC.Com}, \mathsf{CC.Open}, \mathsf{CC.Ver})$ of PPT algorithms, with the following syntax:

- $\mathsf{CC.Setup}(1^\lambda) \to \mathsf{ck}$: takes as input the security parameter and outputs a commitment key ck.
- $\mathsf{CC.Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$: takes as input a commitment key ck and a string $m \in \Gamma^k$, and outputs a commitment com and a state St.
- $\mathsf{CC.Open}(\mathsf{ck}, St, j) \to \pi$: takes as input a commitment key ck, a state St, and an index $j \in [n]$, and outputs an opening π.
- $\mathsf{CC.Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi) \to b$: is deterministic, takes as input a commitment key ck, a commitment com, and index $j \in [n]$, a symbol $\hat{m}_j \in \Lambda$, and an opening π, and outputs a bit $b \in \{0, 1\}$.

Further, we require that the following completeness property holds: For every $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$, every $m \in \Gamma^k$, and every $j \in [n]$, we have

$$\Pr\left[\mathsf{CC.Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{com}, St) \leftarrow \mathsf{CC.Com}(\mathsf{ck}, m), \\ \hat{m} := \mathcal{C}(m),\ \pi \leftarrow \mathsf{CC.Open}(\mathsf{ck}, St, j) \end{array}\right] = 1.$$

Having defined the syntax and completeness of erasure code commitments, we now turn to the security properties that are needed for the compiler from [HASW23] to apply:

- *Position-Binding.* One cannot open the commitment at a position to two different codeword symbols.
- *Code-Binding.* Any set of openings output by an adversary is consistent with some codeword.

We now give the formal definitions taken from [HASW23].

Definition 3 (Position-Binding of CC)**.** Let CC be an erasure code commitment scheme for an erasure code $\mathcal{C}$. We say that CC is position-binding if for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) := \Pr\left[\begin{array}{rl} & \hat{m}_j \neq \hat{m}'_j \\ \wedge & \mathsf{CC.Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi) = 1 \\ \wedge & \mathsf{CC.Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}'_j, \pi') = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{CC.Setup}(1^\lambda), \\ (\mathsf{com}, j, \hat{m}_j, \pi, \hat{m}'_j, \pi') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right].$$

Definition 4 (Code-Binding of CC)**.** Let CC be an erasure code commitment scheme for an erasure code $\mathcal{C}$. We say that CC is code-binding if for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) := \Pr\left[\begin{array}{r} \neg\left(\exists c \in \mathcal{C}(\Gamma^k) : \forall j \in J : c_j = \hat{m}_j\right) \\ \wedge\, \forall j \in J :\ \mathsf{CC.Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi_j) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{CC.Setup}(1^\lambda), \\ (\mathsf{com}, J, (\hat{m}_j, \pi_i)_{j \in J}) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right].$$

Erasure code commitments will be the central cryptographic object we study in this document. Still, we want to informally explain how an erasure code commitment scheme can be used in data availability sampling, and state the main properties of the compiler in [HASW23]. For an extensive formal treatment, we refer the reader to [HASW23].

So, let us assume Alice holds some string of data, denoted by data, that should be posted on the peer-to-peer network, and that some clients want to check if data is available using data availability sampling. To do so, Alice will use the code $\mathcal{C}$ to encode data into a codeword $\widehat{\mathsf{data}}$. For instance, if $\mathcal{C}$ is the Reed-Solomon code, Alice will interpret her data as coefficients of a polynomial of suitable degree and then compute the codeword as the evaluations of this polynomial. Alice then uses the erasure code commitment scheme. Namely, she computes $(\mathsf{com}, St) \leftarrow \mathsf{Com}(\mathsf{ck}, \mathsf{data})$. The commitment com is downloaded by the clients. We emphasize that it is not part of data availability sampling how to agree on

com. On the peer-to-peer network, Alice stores the codeword $\widehat{\mathsf{data}}$. She also attaches to every position i its corresponding opening $\pi_i \leftarrow \mathsf{Open}(\mathsf{ck}, St, i)$.

Now, suppose that clients want to check availability of the data. To do so, the clients sample random positions i of the codeword, and try to download $\widehat{\mathsf{data}}_i$ and π_i from the network. If this succeeds for enough samples, i.e., they always get back the symbols and the opening verifies with respect to com, then they accept. In terms of security, Hall-Andersen, Simkin, and Wagner [HASW23] show that (1) from accepting samples of sufficiently many[1] clients, one can reconstruct the data, meaning that it is indeed available, and (2) no matter where the accepting samples come from, one will always extract the same data. We informally summarize the sketched transformation in the following lemma. For the formal statement, see [HASW23], Section 6.2.

Lemma 1 (Informal)**.** *Let $\mathcal{C}$ be an erasure code and let* CC *be an erasure code commitment for $\mathcal{C}$. Assume that* CC *is position-binding and code-binding. From that, one can construct a secure data availability sampling scheme.*

3 The Erasure Code Commitment used in PeerDAS

As outlined in Section 2.5, the cryptographic core of PeerDAS is an erasure code and a corresponding erasure code commitment scheme. In this section, we explain and formally specify this code and commitment scheme. From a bird's eye view, PeerDAS splits the data into chunks of equal size, which are called *blobs*. Each such blob is then extended individually using a Reed-Solomon code, and a KZG commitment for that blob is computed. These *extended blobs* are then assembled into a matrix, where each row is one extended blob. This matrix is the codeword of the erasure code. In what follows, we take a two-step approach to explain this in more detail and analyze the security of this construction. Concretely, in Section 3.1, we focus on what happens to a single blob, i.e., we look at one row of the final codeword. Then, in Section 3.2, we define and analyze the full code and commitment scheme.

3.1 Warm-Up: Single Row Commitment Scheme

As already explained, data in PeerDAS is treated as a sequence of *blobs*, and each blob is encoded and committed to individually, before the resulting extended blobs are combined into a matrix. Here, we focus on encoding and committing to a single blob.

Erasure Code. Let us explain the erasure code used for a single row, which we denote by $\mathcal{RS}_{\mathsf{row}}$. Intuitively, it is a Reed-Solomon code, where contiguous segments of D evaluations are interpreted as a single symbol. We call these segments *cells* (see Remark 2). To be more precise, let $k, n \in \mathbb{N}$ with $n > k$ be parameters defining the number of cells of a blob (i.e., message) and extended blob (i.e., codeword), respectively. We assume that D, k, and n are all powers of two. Consider the Reed-Solomon code $\mathcal{RS}[kD, \mathcal{L}, \mathbb{Z}_q]$, where $\mathcal{L} = \mathbb{H}$ is the group of roots of unity such that $|\mathbb{H}| = nD$. By grouping together segments of D symbols, we obtain codewords in $\mathcal{RS}_{\mathsf{row}}$. More formally, let $\iota \colon \left(\mathbb{Z}_q^D\right)^n \to \mathbb{Z}_q^{nD}$ be the natural bijection given by concatenation, i.e., $\iota \colon (c_1, \ldots, c_n) \mapsto c_1 \parallel \cdots \parallel c_n$. Then, we define

$$\mathcal{RS}_{\mathsf{row}} := \left\{ c \in \left(\mathbb{Z}_q^D\right)^n \;\middle|\; \iota(c) \in \mathcal{RS}[kD, \mathcal{L}, \mathbb{Z}_q] \right\} \subset \left(\mathbb{Z}_q^D\right)^n .$$

In other words, we have $\mathcal{RS}_{\mathsf{row}} = \iota^{-1}(\mathcal{RS}[kD, \mathcal{L}, \mathbb{Z}_q])$. We visualize the code $\mathcal{RS}_{\mathsf{row}}$ in Figure 2.

Remark 2 (The Role of Symbols). It is natural to wonder why we explicitly say that the symbols of the codeword are in $\mathbb{Z}_q^D$. Of course, we could also say that the codeword contains Dn symbols that are in $\mathbb{Z}_q$. However, recall that an erasure code commitment allows one to open individual *symbols* of the codeword. This has the following implication: if we define the codeword to consist of n symbols in $\mathbb{Z}_q^D$, i.e., of n cells, then we need to open cells. In the other case, we would have to open individual evaluations. Taking this to the level of data availability sampling, defining symbols as cells instead of individual evaluations would mean that clients never download individual evaluations but always entire cells. We will see in the full commitment construction in Section 3.2 that symbols will consist of columns containing multiple cells, and so clients in PeerDAS download entire columns of cells.

[1]Here, what *sufficiently many* means depends on the properties of the erasure code. We refer to Section 6.2, Section 10, and Appendix J of [HASW23] for detailed discussion. Roughly, using a Reed-Solomon code of rate 1/2 as in PeerDAS, the clients in total need to make at least $\Omega(k + \lambda)$ many samples, where k is the dimension of the Reed-Solomon code and λ is the security parameter.

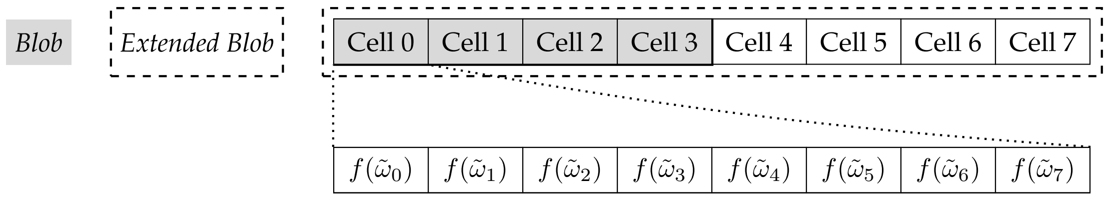


Figure 2: Visualization of the encoding in a single row as described and analyzed in Section 3.1. The figure shows a codeword of the code $\mathcal{RS}_{\mathsf{row}}$ with parameters $D = 8$, $k = 4$, and $n = 8$. This codeword is also called an *extended blob*, and it contains as its first half the data, which is called the *blob*. Intuitively, the codeword is a codeword of a Reed-Solomon code where contiguous segments of evaluations are interpreted as symbols called *cells*.

Remark 3 (The Role of the Reverse Bit Ordering). As explained in Section 2.1, we assume that elements in $\mathbb{H}$ are ordered following the reverse bit ordering. As a consequence, we know that each cell (i.e., symbol of the codeword) corresponds to the evaluations of the polynomial over a coset of a subgroup of $\mathbb{H}$. This subgroup has order D.

Erasure Code Commitment Scheme. Below, we define an erasure code commitment scheme $\mathsf{CC}_{\mathsf{row}}$ that models how to commit to an extended blob. The algorithms make use of the KZG polynomial commitment scheme as defined in Section 2. Intuitively, a blob defines a polynomial of degree less than Dk. The erasure code commitment is simply the KZG polynomial commitment for this polynomial. Symbols of the extended blob, i.e., cells, are opened using KZG multiproofs. We now give the formal definition:

- $\mathsf{CC}_{\mathsf{row}}.\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$
  1. Sample $\mathsf{ck} \leftarrow \mathsf{KZG}.\mathsf{Setup}(1^\lambda, 1^d)$ for $d = kD - 1$.
- $\mathsf{CC}_{\mathsf{row}}.\mathsf{Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$ for $m \in \mathbb{Z}_q^{Dk}$
  1. Let $f \in \mathbb{Z}_q[X]$ with $\deg f < Dk$ such that $f(\tilde{\omega}_i) = m_i$ for all $i \in [Dk]$.
  2. Set $\mathsf{com} := \mathsf{KZG}.\mathsf{Com}(\mathsf{ck}, f)$ and $St := f$.
- $\mathsf{CC}_{\mathsf{row}}.\mathsf{Open}(\mathsf{ck}, St, j) \to \pi$ for $j \in [n]$
  1. Compute $\pi := \mathsf{KZG}.\mathsf{MultiOpen}(\mathsf{ck}, f, Dk, D, j - 1)$.
- $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}, \pi) \to b$ for $i \in [n]$ and $\hat{m} \in \mathbb{Z}_q^D$
  1. Set $b := \mathsf{KZG}.\mathsf{MultiVer}(\mathsf{ck}, \mathsf{com}, Dk, D, j - 1, \hat{m}, \pi)$.

Correctness of this scheme follows directly from the correctness of KZG.

Security Analysis. Here, we show that $\mathsf{CC}_{\mathsf{row}}$ is position-binding and code-binding. We will use these results when we prove position-binding and code-binding of the full commitment construction in Section 3.2. Note that the proof of Lemma 3 is the only proof where we use the algebraic group model.

Lemma 2 (Position-Binding of $\mathsf{CC}_{\mathsf{row}}$). *Let $d = kD - 1$. If the d-*BSDH *assumption holds relative to* PGGen*, then* $\mathsf{CC}_{\mathsf{row}}$ *is position-binding. Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}_{\mathsf{row}}}(\lambda) \leq \mathsf{Adv}^{d\text{-}\mathsf{BSDH}}_{\mathcal{B},\mathsf{PGGen}}(\lambda).$$

Proof. We consider a PPT adversary $\mathcal{A}$ against position-binding of $\mathsf{CC}_{\mathsf{row}}$ as in the statement. Our goal is to construct a reduction $\mathcal{B}$ that breaks the d-BSDH assumption if $\mathcal{A}$ breaks position-binding. To recall, in the position-binding game, $\mathcal{A}$ takes as input the commitment key $\mathsf{ck} \leftarrow \mathsf{KZG}.\mathsf{Setup}(1^\lambda, 1^d)$ for $d = kD - 1$, which has the form $\mathsf{ck} = \left(\mathcal{G}, d, ([\tau^i]_1)_{i=0}^d, ([\tau^i]_2)_{i=0}^d\right)$. When completing its computation, $\mathcal{A}$ then outputs a commitment $\mathsf{com} \in \mathbb{G}_1$, a position $j \in [n]$, and two symbol-opening pairs $(\hat{m}, \pi) \in \mathbb{Z}_q^D \times \mathbb{G}_1$ and $(\hat{m}', \pi') \in \mathbb{Z}_q^D \times \mathbb{G}_1$. Recall that $\mathcal{A}$ wins the position-binding game if $\hat{m} \neq \hat{m}'$, $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}, \pi) = 1$, and $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}', \pi') = 1$. By definition of $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}$ and $\mathsf{KZG}.\mathsf{MultiVer}$, this means that

$$e\left(\mathsf{com} - [I(\tau)]_1, [1]_2\right) = e\left(\pi, [z(\tau)]_2\right), \quad e\left(\mathsf{com} - [I'(\tau)]_1, [1]_2\right) = e\left(\pi', [z(\tau)]_2\right),$$

where I (resp. I') is the interpolation polynomial for $\hat{m}$ (resp. $\hat{m}'$) over the coset indexed by j, and z is the vanishing polynomial of this coset. Let[2] $\psi \in \mathbb{Z}_q$ (resp. $\psi' \in \mathbb{Z}_q$) be such that $\pi = [\psi]_1$ (resp. $\pi' = [\psi']_1$). Let $\zeta \in \mathbb{Z}_q$ be such that $\mathsf{com} = [\zeta]_1$. Then, because e is non-degenerate, we get

$$\psi \cdot z(\tau) + I(\tau) = \zeta = \psi' \cdot z(\tau) + I'(\tau). \tag{1}$$

There is at least one $i \in [D]$ for which $\hat{m}_i \neq \hat{m}'_i$. This is because $\hat{m} \neq \hat{m}'$. Fix the minimal such i. Write $z(X) = (X - \omega)\tilde{z}(X) \in \mathbb{Z}_q[X]$, where $\omega \in \mathbb{Z}_q$ is the root of unity associated with this i, i.e., $I(\omega) = \hat{m}_i$. With this z' at hand, define the polynomials φ, φ' via

$$\varphi(X) = \frac{I(X) - I(\omega)}{X - \omega} = \frac{I(X) - \hat{m}_i}{X - \omega}, \quad \varphi'(X) = \frac{I'(X) - I'(\omega)}{X - \omega} = \frac{I'(X) - \hat{m}'_i}{X - \omega}.$$

Now, from Equation (1), we get

$$\psi \cdot (\tau - \omega) \cdot \tilde{z}(\tau) + \varphi(\tau) \cdot (\tau - \omega) + \hat{m}_i = \psi' \cdot (\tau - \omega) \cdot \tilde{z}(\tau) + \varphi'(\tau) \cdot (\tau - \omega) + \hat{m}'_i.$$

Isolating $(\tau - \omega)$, rearranging, and taking to $\mathbb{G}_T$ yields

$$\left[\frac{1}{\tau - \omega}\right]_T = \frac{1}{\hat{m}_i - \hat{m}'_i} \cdot \left(e(\pi - \pi', [\tilde{z}(\tau)]_2) + e([\varphi(\tau)]_1, [1]_2) - e([\varphi'(\tau)]_1, [1]_2)\right). \tag{2}$$

Note that our reduction $\mathcal{B}$ can compute the right hand side efficiently, given $\mathcal{A}$'s output. With this observation, we can now describe the reduction $\mathcal{B}$:

1. Reduction $\mathcal{B}$ takes as input $\left(\mathcal{G}, d, ([\tau^i]_1)_{i=0}^d, ([\tau^i]_2)_{i=0}^d\right)$. It defines $\mathsf{ck} := \left(\mathcal{G}, d, ([\tau^i]_1)_{i=0}^d, ([\tau^i]_2)_{i=0}^d\right)$.
2. The reduction runs $\mathcal{A}$ on input ck and obtains a commitment $\mathsf{com} \in \mathbb{G}_1$, a position $j \in [n]$, and two symbol-opening pairs $(\hat{m}, \pi) \in \mathbb{Z}_q^D \times \mathbb{G}_1$ and $(\hat{m}', \pi') \in \mathbb{Z}_q^D \times \mathbb{G}_1$ from $\mathcal{A}$.
3. The reduction computes $[1/(\tau - \omega)]_T$ using $\mathcal{A}$'s output via Equation (2). Note that the reduction can compute $[\tilde{z}(\tau)]_2$ via the $[\tau^i]_2$'s that it received as input. The reduction then outputs $(\omega, [1/(\tau - \omega)]_T)$.

It is clear that ck is distributed as in the position-binding game, which means that $\mathcal{B}$ perfectly simulates the game for $\mathcal{A}$. It is also clear that $\mathcal{B}$'s running time is about that of $\mathcal{A}$. Finally, the observation above implies that $\mathcal{B}$ breaks Q-BSDH whenever $\mathcal{A}$ breaks position-binding, which concludes the proof. $\square$

Lemma 3 (Code-Binding of $\mathsf{CC}_{\mathsf{row}}$). *Let $d = kD - 1$. If the d-BSDH assumption holds relative to PGGen, then $\mathsf{CC}_{\mathsf{row}}$ is code-binding in the algebraic group model. Concretely, for any algebraic PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A}, \mathsf{CC}_{\mathsf{row}}}(\lambda) \leq \mathsf{Adv}^{d\text{-}\mathsf{BSDH}}_{\mathcal{B}, \mathsf{PGGen}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an algebraic PPT adversary against code-binding of $\mathsf{CC}_{\mathsf{row}}$ as in the statement. Our strategy will be to construct a reduction $\mathcal{R}$ that simulates the code-binding game for $\mathcal{A}$ and breaks position-binding of $\mathsf{CC}_{\mathsf{row}}$ if $\mathcal{A}$ breaks code-binding. We first make an observation and then give the reduction: Recall that in the code-binding game, $\mathcal{A}$ takes as input the commitment key $\mathsf{ck} \leftarrow \mathsf{KZG}.\mathsf{Setup}(1^\lambda, 1^d)$ for $d = kD - 1$, and it outputs a commitment $\mathsf{com} \in \mathbb{G}_1$ and a set of positions $J \subseteq [n]$ and pairs $(\hat{m}_j, \pi_j) \in \mathbb{Z}_q^D \times \mathbb{G}_1$ for each $j \in J$. We denote the ith coordinate of $\hat{m}_j$ by $\hat{m}_{j,i} \in \mathbb{Z}_q$ for $i \in [D]$. By definition, $\mathcal{A}$ breaks code-binding if all openings verify, i.e., $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi_j) = 1$ for all $j \in J$, and there is no polynomial $f \in \mathbb{Z}_q[X]$ of degree at most d that is consistent with these openings, i.e., no polynomial f

[2]Note: we do not use the algebraic group model here. Our reduction never uses ψ or ψ' over $\mathbb{Z}_q$.

with $f(\tilde{\omega}_{j\cdot D+i}) = \hat{m}_{j,i}$ for all $j \in J$ and $i \in [D]$ and $\deg f \leq d$. In particular, if $\mathcal{A}$ breaks code-binding, we must have $|J| > k$, as otherwise, there is always such a polynomial f. As $\mathcal{A}$ is algebraic, in particular it has to output (coefficients of) a polynomial f of degree at most d such that $\mathsf{com} = [f(\tau)]_1$. Because f is of degree at most d, we know that there is at least one $(j, i) \in J \times [D]$ such that $f(\tilde{\omega}_{j\cdot D+i}) \neq \hat{m}_{j,i}$. We can now use this observation to break position-binding for position j: one of the openings is the one provided by $\mathcal{A}$, and the other one can be computed honestly by the reduction which knows f. Concretely, our reduction $\mathcal{R}$ works as follows:

1. Reduction $\mathcal{R}$ takes as input the commitment key ck from the position-binding game. It uses this commitment key to simulate the code-binding game for $\mathcal{A}$.

2. Namely, it runs $\mathcal{A}$ on input ck and obtains a commitment $\mathsf{com} \in \mathbb{G}_1$, a set of positions $J \subseteq [n]$, and pairs $(\hat{m}_j, \pi_j) \in \mathbb{Z}_q^D \times \mathbb{G}_1$ for each $j \in J$ from $\mathcal{A}$. Because $\mathcal{A}$ is algebraic, it also outputs (coefficients of) a polynomial f of degree at most d such that $\mathsf{com} = [f(\tau)]_1$.

3. The reduction now identifies the first pair $(j, i) \in J \times [D]$ such that $f(\tilde{\omega}_{j\cdot D+i}) \neq \hat{m}_{j,i}$. If $\mathcal{A}$ breaks code-binding, we know that this exists. If it does not exist, the reduction aborts.

4. The reduction computes $\pi'_j := \mathsf{KZG.MultiOpen}(\mathsf{ck}, f, Dk, D, j-1)$ and defines $\hat{m}'_j \in \mathbb{Z}_q^D$ by setting $\hat{m}'_{j,i} := f(\tilde{\omega}_{j\cdot D+i})$ for all $i \in [D]$.

5. The reduction outputs $(\mathsf{com}, j, \hat{m}_j, \pi_j, \hat{m}'_j, \pi'_j)$ to the position-binding game.

Clearly, $\mathcal{R}$ perfectly simulates the code-binding game for $\mathcal{A}$, and its running time is dominated by the running time of $\mathcal{A}$. From the observation above, we know that $\mathcal{R}$ breaks position-binding whenever $\mathcal{A}$ breaks code-binding. Using Lemma 2, we obtain the reduction $\mathcal{B}$ as in the statement. $\square$

Remark 4 (Algebraic Group Model). The proof of Lemma 3 relies on the algebraic group model. Concretely, our reduction exploits the fact that when the adversary outputs a commitment $\mathsf{com} \in \mathbb{G}_1$, it also has to output an algebraic representation of com, given by a polynomial f such that $\mathsf{com} = [f(\tau)]_1$. That is, thanks to the algebraic group model, the reduction can know such a polynomial f and later use it to break position-binding (which is proven without the algebraic group model in Lemma 2). In practice, an adversary may just sample a random group element for com, for which it does not know any discrete logarithm or algebraic representation. Such an adversary is not captured by our proof, as it is not algebraic. However, we stress that such an adversary would not automatically break code-binding: it would have to compute valid openings, and intuitively this requires to know an algebraic representation of the commitment. We leave making this intuition formal as an interesting problem for future work.

3.2 The Full Commitment Scheme

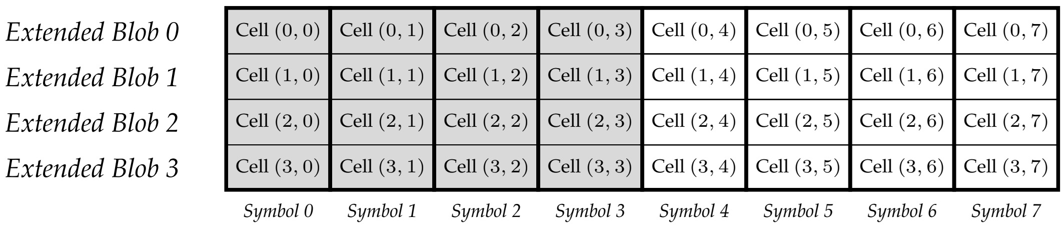
Equipped with the erasure code and commitment for a single blob, we now define the full erasure code and erasure code commitment used in PeerDAS. We will show that its security follows from the security of the commitment scheme we have defined in Section 3.1. Intuitively, we split the data into multiple blobs and apply the erasure code and commitment from Section 3.1 to each blob individually. Then, we think of the extended blobs as being the rows of a matrix, and open the resulting codeword column-wise, see Figure 3.

Erasure Code. Consider the code $\mathcal{RS}_{\mathsf{row}}$ introduced in Section 3.1 and let $\ell \in \mathbb{N}$ be a parameter. Intuitively, ℓ corresponds to the number of rows in the matrix, or equivalently, the number of blobs into which we split the data. We define our new code $\mathcal{RS}_{\mathsf{full}}$ by describing how data $m \in \mathbb{Z}_q^{Dk\ell}$ is encoded:

1. The data is naturally split into ℓ blobs $m_1, \ldots, m_\ell$, where $m_i \in \mathbb{Z}_q^{Dk}$ for all $i \in [\ell]$.

2. Each blob is individually encoded using the code $\mathcal{RS}_{\mathsf{row}}$. As a result, we obtain extended blobs $c_1, \ldots, c_\ell$, where $c_i \in \left(\mathbb{Z}_q^D\right)^n$ for all $i \in [\ell]$.

3. The jth symbol of the codeword is $(c_{1,j}, \ldots, c_{\ell,j}) \in \left(\mathbb{Z}_q^D\right)^\ell$, where $c_{i,j} \in \mathbb{Z}_q^D$ denotes the jth cell of c_i.



Extended Blob 0	Cell $(0,0)$	Cell $(0,1)$	Cell $(0,2)$	Cell $(0,3)$	Cell $(0,4)$	Cell $(0,5)$	Cell $(0,6)$	Cell $(0,7)$
Extended Blob 1	Cell $(1,0)$	Cell $(1,1)$	Cell $(1,2)$	Cell $(1,3)$	Cell $(1,4)$	Cell $(1,5)$	Cell $(1,6)$	Cell $(1,7)$
Extended Blob 2	Cell $(2,0)$	Cell $(2,1)$	Cell $(2,2)$	Cell $(2,3)$	Cell $(2,4)$	Cell $(2,5)$	Cell $(2,6)$	Cell $(2,7)$
Extended Blob 3	Cell $(3,0)$	Cell $(3,1)$	Cell $(3,2)$	Cell $(3,3)$	Cell $(3,4)$	Cell $(3,5)$	Cell $(3,6)$	Cell $(3,7)$
	Symbol 0	*Symbol 1*	*Symbol 2*	*Symbol 3*	*Symbol 4*	*Symbol 5*	*Symbol 6*	*Symbol 7*

Figure 3: Visualization of the encoding of data in PeerDAS. The figure shows a codeword of the code $\mathcal{RS}_{\mathsf{full}}$. Data is split into multiple chunks, which are called *blobs*. Each blob (a row of shaded cells) is then individually extended as shown in Figure 2. The resulting *extended blobs* are treated as the rows of a matrix, and the columns are the symbols of the resulting codeword.

As a result, we have

$$\mathcal{RS}_{\mathsf{full}} \subset \left(\left(\mathbb{Z}_q^D\right)^\ell\right)^n.$$

That is, a codeword consists of n many symbols, where each symbol consists of ℓ cells, each containing D many field elements. We visualize a codeword of $\mathcal{RS}_{\mathsf{full}}$ in Figure 3.

Remark 5 (Interleaved Code). For readers familiar with coding theory or with the notation in [HASW23], the code $\mathcal{RS}_{\mathsf{full}}$ is the *interleaved code* of code $\mathcal{RS}_{\mathsf{row}}$. Using their notation: $\mathcal{RS}_{\mathsf{full}} = \mathcal{RS}_{\mathsf{row}}{}^{\equiv \ell}$.

Erasure Code Commitment Scheme. Next, we specify the erasure code commitment scheme $\mathsf{CC}_{\mathsf{full}}$ used in PeerDAS. We do so by invoking the erasure code commitment scheme $\mathsf{CC}_{\mathsf{row}}$ that we have defined in Section 3.1:

- $\mathsf{CC}_{\mathsf{full}}.\mathsf{Setup} = \mathsf{CC}_{\mathsf{row}}.\mathsf{Setup}$
- $\mathsf{CC}_{\mathsf{full}}.\mathsf{Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$ for $m \in \mathbb{Z}_q^{Dk\ell}$
  1. Parse $m = m_1 \parallel \cdots \parallel m_\ell$ with $m_i \in \mathbb{Z}_q^{Dk}$ for all $i \in [\ell]$.
  2. For each $i \in [\ell]$, compute $(\mathsf{com}_i, St_i) := \mathsf{CC}_{\mathsf{row}}.\mathsf{Com}(\mathsf{ck}, m_i)$.
  3. Set $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_\ell)$ and $St := (St_1, \ldots, St_\ell)$.
- $\mathsf{CC}_{\mathsf{full}}.\mathsf{Open}(\mathsf{ck}, St, j) \to \pi$ for $j \in [n]$
  1. Parse $St = (St_1, \ldots, St_\ell)$.
  2. For each $i \in [\ell]$, compute $\pi_i := \mathsf{CC}_{\mathsf{row}}.\mathsf{Open}(\mathsf{ck}, St_i, j)$.
  3. Set $\pi := (\pi_1, \ldots, \pi_\ell)$.
- $\mathsf{CC}_{\mathsf{full}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}, \pi) \to b$ for $j \in n$ and $\hat{m} \in \left(\mathbb{Z}_q^D\right)^\ell$
  1. Parse $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_\ell)$, $\pi = (\pi_1, \ldots, \pi_\ell)$, and $\hat{m} = (\hat{m}_1, \ldots, \hat{m}_\ell)$.
  2. If $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, j, \hat{m}_i, \pi_i) = 1$ for all $i \in [\ell]$, set $b := 1$. Otherwise, set $b := 0$.

Correctness of $\mathsf{CC}_{\mathsf{full}}$ follows directly from correctness of $\mathsf{CC}_{\mathsf{row}}$.

Security Analysis. We now show position-binding and code-binding of $\mathsf{CC}_{\mathsf{full}}$. For that, we will use our results on position-binding and code-binding of $\mathsf{CC}_{\mathsf{row}}$, see Lemmata 2 and 3.

Lemma 4 (Position-Binding of $\mathsf{CC}_{\mathsf{full}}$)**.** *If $\mathsf{CC}_{\mathsf{row}}$ is position-binding, then $\mathsf{CC}_{\mathsf{full}}$ is position-binding. Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}_{\mathsf{full}}}(\lambda) \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B},\mathsf{CC}_{\mathsf{row}}}(\lambda).$$

Proof. Intuitively, breaking position-binding of $\mathsf{CC}_{\mathsf{full}}$ means breaking position-binding of $\mathsf{CC}_{\mathsf{row}}$ for at least one of the rows. More formally, consider an adversary $\mathcal{A}$ breaking position-binding of $\mathsf{CC}_{\mathsf{full}}$. This means that on input $\mathsf{ck} \leftarrow \mathsf{CC}_{\mathsf{full}}.\mathsf{Setup}(1^\lambda)$, $\mathcal{A}$ outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \dots, \mathsf{com}_\ell)$, an index $j \in [n]$, and two symbols with corresponding opening proofs. Namely, it outputs a symbol $\hat{m} = (\hat{m}_1, \dots, \hat{m}_\ell) \in \left(\mathbb{Z}_q^D\right)^\ell$ with an opening proof $\pi = (\pi_1, \dots, \pi_\ell)$ and a symbol $\hat{m}' = (\hat{m}'_1, \dots, \hat{m}'_\ell) \in \left(\mathbb{Z}_q^D\right)^\ell$ with an opening proof $\pi' = (\pi'_1, \dots, \pi'_\ell)$. Assuming that $\mathcal{A}$ breaks position-binding, we know that (1) $\hat{m} \neq \hat{m}'$, and (2) that both opening proofs verify with respect to the commitment com. Let $i \in [\ell]$ be the minimal index such that $\hat{m}_i \neq \hat{m}'_i$. This index exists because of (1). Because of (2) and the definition of $\mathsf{CC}_{\mathsf{full}}.\mathsf{Ver}$, we know that $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, j, \hat{m}_i, \pi_i) = 1$ and $\mathsf{CC}_{\mathsf{row}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_i, j, \hat{m}'_i, \pi'_i) = 1$. This suggests the following reduction $\mathcal{B}$ running in the position-binding game of $\mathsf{CC}_{\mathsf{row}}$:

1. Get as input a commitment key ck and run $\mathcal{A}$ on input ck.
2. When $\mathcal{A}$ outputs a commitment com, an index j, and $\hat{m}, \pi, \hat{m}', \pi'$ as above, abort if $\hat{m} = \hat{m}'$.
3. Otherwise, let $i \in [\ell]$ be the minimal index such that $\hat{m}_i \neq \hat{m}'_i$ as above.
4. Output $(\mathsf{com}_i, j, \hat{m}_i, \pi_i, \hat{m}'_i, \pi'_i)$.

As $\mathsf{CC}_{\mathsf{full}}.\mathsf{Setup} = \mathsf{CC}_{\mathsf{row}}.\mathsf{Setup}$, the reduction perfectly simulates the position-binding game of $\mathsf{CC}_{\mathsf{full}}$ for $\mathcal{A}$. By the discussion above, if $\mathcal{A}$ breaks position-binding of $\mathsf{CC}_{\mathsf{full}}$, then $\mathcal{B}$ breaks position-binding of $\mathsf{CC}_{\mathsf{row}}$. Finally, the running time of $\mathcal{B}$ is dominated by that of $\mathcal{A}$, finishing the proof of the claim. □

Lemma 5 (Code-Binding of $\mathsf{CC}_{\mathsf{full}}$)**.** *If $\mathsf{CC}_{\mathsf{row}}$ is code-binding, then $\mathsf{CC}_{\mathsf{full}}$ is code-binding. Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}_{\mathsf{full}}}(\lambda) \leq \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B},\mathsf{CC}_{\mathsf{row}}}(\lambda).$$

Proof. Similar to the proof of Lemma 4, the proof easily follows from the observation that to break code-binding of $\mathcal{RS}_{\mathsf{full}}$, there needs to be at least one row for which the adversary already breaks code-binding of $\mathsf{CC}_{\mathsf{row}}$. We now make this more formal. Let $\mathcal{A}$ be an adversary against code-binding of $\mathcal{RS}_{\mathsf{full}}$ as in the lemma. By definition of the code-binding game, $\mathcal{A}$ gets as input $\mathsf{ck} \leftarrow \mathsf{CC}_{\mathsf{full}}.\mathsf{Setup}(1^\lambda)$. It then outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \dots, \mathsf{com}_\ell)$, a set of positions $J \subseteq [n]$, and symbols with opening proofs for these positions. Concretely, it outputs $(\hat{m}_j, \pi_j)_{j \in J}$, where $\hat{m}_j = (\hat{m}_{1,j}, \dots, \hat{m}_{\ell,j}) \in \left(\mathbb{Z}_q^D\right)^\ell$ and $\pi_j = (\pi_{1,j}, \dots, \pi_{\ell,j})$ for every $j \in J$. Conditioned on $\mathcal{A}$ breaking code-binding, we know from the definition that the following two properties hold:

- *No Consistent Codeword.* For any $c \in \mathcal{RS}_{\mathsf{full}}$, we denote by $c_{i,j} \in \mathbb{Z}_q^D$ the cell in the ith row and jth column. Namely, we have $c = (c_1, \dots, c_n) \in \left(\left(\mathbb{Z}_q^D\right)^\ell\right)^n$ and $c_j = (c_{1,j}, \dots, c_{\ell,j}) \in \left(\mathbb{Z}_q^D\right)^\ell$ for each $j \in [n]$. With this notation, there is no $c \in \mathcal{RS}_{\mathsf{full}}$ such that $c_{i,j} = \hat{m}_{i,j}$ for all $(i,j) \in [\ell] \times J$. By definition of $\mathcal{RS}_{\mathsf{full}}$, this implies that there is at least one $i \in [\ell]$ such that there is no $c_i = (c_{i,1}, \dots, c_{i,n}) \in \mathcal{RS}_{\mathsf{row}}$ with $c_{i,j} = \hat{m}_{i,j}$ for all $j \in J$. Let $i^* \in [\ell]$ denote the minimal such i.
- *Openings Verify.* For every $j \in J$, we have $\mathsf{CC}_{\mathsf{full}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \pi_j)$. By definition of $\mathsf{CC}_{\mathsf{full}}.\mathsf{Ver}$, this means that in particular, $\mathsf{CC}_{\mathsf{full}}.\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_{i^*}, j, \hat{m}_{i^*,j}, \pi_{i^*,j})$ for every $j \in J$.

We now use this observation to design a reduction $\mathcal{B}$ running in the code-binding game of $\mathsf{CC}_{\mathsf{row}}$:

1. Get as input a commitment key ck and run $\mathcal{A}$ on input ck.
2. When $\mathcal{A}$, terminates, it outputs a commitment com, a set $J \subseteq [n]$, and $(\hat{m}_j, \pi_j)_{j \in J}$ as above.
3. If i^* as above does not exist, then abort. Otherwise, output $(\mathsf{com}_{i^*}, J, (\hat{m}_{i^*,j}, \pi_{i^*,j})_{j \in J})$.

As $\mathsf{CC}_{\mathsf{full}}.\mathsf{Setup} = \mathsf{CC}_{\mathsf{row}}.\mathsf{Setup}$, $\mathcal{B}$ perfectly simulates the code-binding game of $\mathsf{CC}_{\mathsf{full}}$ for $\mathcal{A}$. Note that the check in Step 3 can be performed efficiently by $\mathcal{B}$, e.g., by interpolating a polynomial from the first Dk points and checking the remaining points on that row for consistency. Hence, reduction $\mathcal{B}$ has about the same running time as $\mathcal{A}$. Using our observation above, we also get that if $\mathcal{A}$ breaks code-binding of $\mathsf{CC}_{\mathsf{full}}$, then $\mathcal{B}$ does not abort in Step 3, and hence breaks code-binding of $\mathsf{CC}_{\mathsf{row}}$. □

We can now combine Lemmata 3 and 5 to obtain code-binding of $\mathsf{CC}_{\mathsf{full}}$ and Lemmata 2 and 4 to obtain position-binding of $\mathsf{CC}_{\mathsf{full}}$. With that, we our main result, stated next.

Corollary 1 (Security of $\mathsf{CC}_{\mathsf{full}}$)**.** *If the d-BSDH assumption holds relative to PGGen, for d as in Lemma 4, then $\mathsf{CC}_{\mathsf{full}}$ is position-binding, and code-binding in the algebraic group model. Concretely, for any PPT algorithm $\mathcal{A}$ and any algebraic PPT algorithm $\mathcal{A}'$, there are PPT algorithms $\mathcal{B}$ and $\mathcal{B}'$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$, $\mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A}')$, and*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}_{\mathsf{full}}}(\lambda) \leq \mathsf{Adv}^{d\text{-}\mathsf{BSDH}}_{\mathcal{B},\mathsf{PGGen}}(\lambda) \text{ and } \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A}',\mathsf{CC}_{\mathsf{row}}}(\lambda) \leq \mathsf{Adv}^{d\text{-}\mathsf{BSDH}}_{\mathcal{B}',\mathsf{PGGen}}(\lambda).$$

Concrete Instantiation. Finally, we specify how parameters are set in PeerDAS:

- *Curve and Field.* The pairing-friendly curve BLS12-381 is used. This means that elements in the field $\mathbb{Z}_q$ are of size 32 bytes.
- *Size of a Cell.* A cell consists of $D = 64$ evaluations.
- *Size of a Blob.* A blob consists of 4096 field elements, or equivalently, of $k = 64$ cells. This means a blob represents 2^{17} bytes of data.
- *Size of an Extended Blob.* An extended blob consists of 8192 field elements or of $n = 2k = 128$ many cells. This means an extended blob has size 2^{18} bytes.
- *Number of Blobs.* The concrete setting of ℓ depends on networking conditions and is still subject to ongoing research.

4 Optimizations

In the final section of this document, we discuss common optimizations that are used when implementing PeerDAS. We discuss their efficiency as well as their implications for security. Concretely, we discuss three optimizations: (1) how to compute the commitment and all openings efficiently in one shot, (2) how to verify multiple proofs at once efficiently using batch verification, and (3) how to efficiently reconstruct data and opening proofs if a large enough subset of openings is given. We emphasize that only (2) impacts security, as security only depends on how proofs are verified. To show that security is preserved when using (2), we prove a simple lemma showing that batch verification is *as good as* individual verification.

4.1 Computing Commitment and All Openings

In PeerDAS, the party computing the extended blob from the data also needs to compute all KZG opening proofs. It turns out that computing all opening proofs at once can be done more efficiently than computing each opening proof individually. This has been observed in 2020 by Feist and Khovratovich [FK20], who introduced a technique to pre-compute all KZG opening proofs of a polynomial of degree d over an evaluation domain of d roots of unity[3] in only $\Theta(d \log d)$ steps and storing d elements in $\mathbb{G}_1$ during the computation. Naively, computing every opening proof individually would require $\Theta(d^2)$ steps.

In [FK20, Section 3], Feist and Khovratovich also introduced an extended algorithm to precompute KZG *multiproofs* for cosets. As openings in PeerDAS are KZG multiproofs, this is what is used for PeerDAS. Recall that $\mathbb{H}$ is the group of roots of unity of size Dn. Also, recall that a cell consists of D evaluations of a polynomial over a coset of a subgroup of size D. The polynomial has degree less than Dk, and there are n such cells in an extended blob, containing all evaluations over $\mathbb{H}$ in reverse bit ordering. By using Feist and Khovratovich's algorithm, one can compute all KZG multiproofs for an extended blob in $\Theta(\log(Dk-1)Dk)$ steps.

[3]This has been further formalized in [FK23].

4.2 Optimizations for Verification

When using the KZG-based commitment scheme we have presented for data availability sampling, clients need to verify multiple openings. Instead of verifying each opening individually, it has been shown that one can efficiently *batch* these verification checks [KDF22]. In this way, one can verify a set of openings using an equation that only involves two pairing evaluations, which significantly reduces the work load for clients. In the following, we first derive and motivate this batched verification equation. Then, we prove a lemma stating that using it preserves security: if the batched verification equation holds, then all openings individually verify as well.

The Setting: Verifying Multiple Openings. We consider the matrix explained in Section 3.2, Figure 3. Now, suppose Alice holds the commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_\ell)$, and she is given L cells[4] $\hat{m}_k \in \mathbb{Z}_q^D$ and openings $\pi_k \in \mathbb{G}_1$ for each $k \in \{0, \ldots, L-1\}$. Namely, opening π_k is a KZG multiproof for the cell $\hat{m}_k$ in row[5] i_k and column j_k with respect to commitment com_{i_k}. Alice's goal is to verify all of these openings efficiently, i.e., she only wants to accept if *all* of the openings are valid. To recall, verifying an opening π_k individually, Alice would check the equation

$$e\left(\pi_k, [Z_k(\tau)]_2\right) = e\left(\mathsf{com}_{i_k} - [I_k(\tau)]_1, [1]_2\right), \tag{3}$$

where $I_k(X)$ (computed from $\hat{m}_k$) and $Z_k(X)$ are the interpolation polynomial and vanishing polynomial of the cell, respectively. Naively, this costs evaluating $2L$ pairings.

The Batched Verification Equation. To derive a more efficient way of checking Equation (3) for each k, we first recall the structure of the vanishing polynomials. Namely, recall that a row in Figure 3 corresponds to evaluations of a polynomial over a group of roots of unity of size $D \cdot n$ with generator ω, and each cell is associated with a coset of a subgroup of size D. The vanishing polynomial for this coset has the form $Z_k(X) = X^D - \omega^{s_k D}$, where s_k is determined by the column index $j_k \in [n]$ and can be thought of as identifying the coset. Using this definition of Z_k and rewriting Equation (3), we get

$$e\left(\pi_k, [\tau^D]_2\right) - e\left(\pi_k, [\omega^{s_k D}]\right) = e\left(\mathsf{com}_{i_k} - [I_k(\tau)]_1, [1]_2\right).$$

Rearranging, and noting that ω is public, we get

$$e\left(\pi_k, [\tau^D]_2\right) = e\left(\mathsf{com}_{i_k} - [I_k(\tau)]_1 + \omega^{s_k D}\pi_k, [1]_2\right).$$

We observe that now on both sides of the equation, one of the operands for the pairing is independent of k. In particular, this means that if Equation (3) holds for every k, then we know that for every coefficient $r \in \mathbb{Z}_q$, we have

$$\sum_{k=0}^{L-1} r^k \cdot e\left(\pi_k, [\tau^D]_2\right) = \sum_{k=0}^{L-1} r^k \cdot e\left(\mathsf{com}_{i_k} - [I_k(\tau)]_1 + \omega^{s_k D}\pi_k, [1]_2\right). \tag{4}$$

This is not the final batched verification equation, but it is already mathematically equivalent to it.

Remark 6. To make this point clear, we now only know that if the individual equations hold, then Equation (4) holds for a random r. We will see in our analysis that it follows from Schwartz-Zippel that the converse direction also holds with overwhelming probability. This means that checking Equation (4) is as good as verifying the proofs individually.

Coming back to Equation (4), we use the bilinearity of the pairing, and get

$$e\left(\sum_{k=0}^{L-1} r^k \pi_k, [\tau^D]_2\right) = e\left(\sum_{k=0}^{L-1} r^k \mathsf{com}_{i_k} - \left[\sum_{k=0}^{L-1} r^k I_k(\tau)\right]_1 + \sum_{k=0}^{L-1} (r^k \omega^{s_k D})\pi_k, [1]_2\right). \tag{5}$$

[4]Recall that a cell consists of D field elements, see Section 3.1.

[5]In the commitment in Section 3.2, we always open entire columns, but the batched verification works for every subset of cells of the matrix.

This is the *batched verification equation*[6], which allows us to verify the L openings using just *two pairings*. Notably, the random coefficient r is derived using a random oracle $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_q$ to make verification deterministic. That is, Alice would compute

$$r := \mathsf{H}\left(L, (\mathsf{com}_i)_{i=1}^{\ell}, (i_k, j_k, \hat{m}_k, \pi_k)_{k=0}^{L-1}\right).$$

Security Implications. Mathematically, the batched verification equation does not imply that all individual verification equations hold. That is, if Alice is unlucky, the random coefficient r could be a *bad* coefficient for which even non-verifying openings make the batched verification equation accept. We show that this happens with negligible probability. In other words, no efficient adversary can find an input for which the batched verification equation holds but the individual openings do not all verify. This justifies that using the batched verification equation is secure.

Lemma 6 (Batched Verification Preserves Security)**.** *Let* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_q$ *be a random oracle. Consider any PPT algorithm* $\mathcal{A}$ *in the following experiment:*

1. *Generate a commitment key* $\mathsf{ck} \leftarrow \mathsf{KZG.Setup}(1^\lambda, 1^d)$ *for* $d = kD - 1$ *as in Section 3.1.*
2. *Run* $\mathcal{A}$ *with access to* H *on input* ck*.*
3. *Obtain from* $\mathcal{A}$ *an integer* L*, a commitment* $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_\ell)$*, and cells with indices and openings* $(i_k, j_k, \hat{m}_k, \pi_k)_{k=0}^{L-1}$*.*
4. *Output* 1 *if and only if Equation (5) holds but there is a* $k \in \{0, \ldots, L-1\}$ *such that Equation (3) does not hold for* k*.*

Then, if Q_{H} *denotes the number of queries* $\mathcal{A}$ *makes to* H *and* L_{max} *denotes the maximum* L *submitted to* H*, the probability that the game outputs is at most* $Q_{\mathsf{H}}(L_{max} - 1)/q$*.*

Proof. We define the event $\mathsf{BadQuery}$ if there is a random oracle query $\mathsf{H}\left(L, (\mathsf{com}_i)_{i=1}^{\ell}, (i_k, j_k, \hat{m}_k, \pi_k)_{k=1}^{L}\right)$ returning a field element r such that with these inputs and this coefficient Equation (5) holds but there is a $k \in \{0, \ldots, L-1\}$ such that Equation (3) does not hold for k. Clearly, if the game outputs 1, then $\mathsf{BadQuery}$ must occur. Hence, we can focus on bounding the probability of $\mathsf{BadQuery}$. We now consider a fixed query and later do a union bound over all Q_{H} queries. For this fixed query, we now consider Equation (5) over the field $\mathbb{Z}_q$. Namely, we let $c_i \in \mathbb{Z}_q$ be such that $[c_i]_1 = \mathsf{com}_i$ for all $i \in [\ell]$ and $Q_k \in \mathbb{Z}_q$ be such that $[Q_k] = \pi_k$ for all $k \in \{0, \ldots, L-1\}$. Then, if Equation (5) holds, we get

$$\left(\sum_{k=0}^{L-1} r^k Q_k\right) \cdot \tau^D = \sum_{k=0}^{L-1} r^k c_{i_k} - \sum_{k=0}^{L-1} r^k I_k(\tau) + \sum_{k=0}^{L-1} r^k \omega^{s_k D} Q_k.$$

This is a polynomial equation in the variable r. More precisely, define the polynomial $\Psi \in \mathbb{Z}_q[X]$ of degree $L-1$ with

$$\Psi(X) = \sum_{k=0}^{L-1} (\tau^D Q_k + I_k(\tau) - c_{i_k} - \omega^{s_k D} Q_k) \cdot X^k = (Q_k Z_k(\tau) - (c_{i_k} - I_k(\tau))) \cdot X^k.$$

Then we know that $\Psi(r) = 0$ if Equation (5) holds. Note that if there is a k such that Equation (3) does not hold for k, then Ψ is not the zero polynomial. Also, Ψ is fully determined by the inputs to the random oracle query and the system parameters. Therefore, the probability that $\Psi(r) = 0$ is at most $(L-1)/q$ by the Schwartz-Zippel lemma. In combination, this means that the probability that $\mathsf{BadQuery}$ occurs for this fixed query is at most $(L_{\mathsf{max}} - 1)/q$, concluding the proof via a union bound over all queries. □

[6]There are further optimizations possible to compute the terms $\sum_{k=0}^{L-1} r^k \mathsf{com}_{i_k}$ and $[\sum_{k=0}^{L-1} r^k I_k(\tau)]_1$. For example, to compute $\sum_{k=0}^{L-1} r^k \mathsf{com}_{i_k}$ using less group operations, one can first add up all coefficients for a commitment over the field, and then use a single multi-scalar multiplication.

4.3 Efficient Reconstruction of Data

As long as at least half of the cells in an extended blob (see Figure 2) are given, one can recover the entire extended blob, and thus the original data, using polynomial interpolation. Here, we explain how this can be done efficiently.

Setting. Let $\hat{m}_1, \ldots, \hat{m}_n \in \mathbb{Z}_q^D$ be the cells of an extended blob (see Section 3.1). We consider the scenario where some cells $(\hat{m}_j)_{j \in J}$, $J \subset [n]$ are not available. Given the properties of erasure codes, we can recover them as soon as $|J| \leq n - k = k$ by computing the unique polynomial $P(X) \in \mathbb{Z}_q[X]$ of degree at most $Dk - 1$ that interpolates the available cells $(\hat{m}_j)_{j \notin J}$. In this section, we describe an optimized way of doing so, which is used in PeerDAS, see [But18].

Algorithm. Letting $\mathbb{H}$ denote the group of roots of unity of size nD, which is used as the evaluation domain for the extended blob, the algorithm works as follows:

1. Compute $Z(X)$ with $\deg(Z) \leq |J|$, the vanishing polynomial of all roots of unity in $\mathbb{H}$ whose evaluations are missing, in evaluation form.
2. Define the vector $\mathbf{e} \in \left(\mathbb{Z}_q^D\right)^n$ such that $\mathbf{e}_j = \mathbf{0}$ if $j \in J$ (i.e., if the cell is missing) and $\mathbf{e}_j = \hat{m}_j$ otherwise, where $\mathbf{e}_j \in \mathbb{Z}_q^D$.
3. Compute the polynomial $E(X)$ with $\deg(E) \leq nD - 1$, which interpolates vector $\mathbf{e}$ over $\mathbb{H}$.
4. Compute the evaluations of $E(X)Z(X)$ over $\mathbb{H}$. This can be done efficiently because $E(X)$ and $Z(X)$ are given in evaluation form.

   Claim. We have $P(x)Z(x) = E(x)Z(x)$ for all $x \in \mathbb{H}$.

   Indeed, for all points in $\mathbb{H}$, for which evaluations are missing, both sides of the equation are zero. For the ones that are available, P agrees with $\mathbf{e}$ by definition.
5. Compute $P(X)Z(X)$ in coefficient form. To do so, note that $\deg(PZ) \leq nD - 1$, and so we can recover $P(X)Z(X)$ from the evaluations of $E(X)Z(X)$ in $\mathbb{H}$.
6. Divide $P(X)Z(X)$ by $Z(X)$. This last step can be done inefficiently in the coefficient form, or efficiently the following way:

   (a) Choose some β such that $\beta\mathbb{H}$ is a coset of $\mathbb{H}$[7].

   (b) Evaluate $P(X)Z(X)$ over $\beta\mathbb{H}$.

   (c) Evaluate $Z(X)$ over $\beta\mathbb{H}$.

   (d) Divide $P(X)Z(X)$ by $Z(X)$ in evaluation form over $\beta\mathbb{H}$.

   (e) Interpolate $P(X)$ over $\beta\mathbb{H}$.

Finally, note that all evaluations and Interpolations can be done efficiently using (inverse) FFTs over groups of roots of unity or cosets thereof.

[7] β must be an element in a group of roots of unity $\hat{\mathbb{H}}$ such that $\mathbb{H} \subset \hat{\mathbb{H}}$ and $\beta \notin \mathbb{H}$.

References

[ASBK21] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2021. (Cited on page 2.)

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Heidelberg, August 2020. (Cited on page 4.)

[BFL$^+$22] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. EIP-4844: Shard Blob Transactions. https://eips.ethereum.org/EIPS/eip-4844, 2022. Accessed: 2024-07-10. (Cited on page 2.)

[But18] Vitalik Buterin. Reed-Solomon erasure code recovery in n*log$\hat{2}$(n) time with FFTs. https://ethresear.ch/t/reed-solomon-erasure-code-recovery-in-n-log-2-n-time-with-ffts/3039, 2018. Accessed: 2024-06-27. (Cited on page 17.)

[D'A23] Francesco D'Amato. From 4844 to Danksharding: a path to scaling Ethereum DA. https://ethresear.ch/t/from-4844-to-danksharding-a-path-to-scaling-ethereum-da/18046, 2023. Accessed: 2024-06-27. (Cited on page 2.)

[Eth24a] Ethereum. Ethereum Consensus Specs - Commit 54093964c95f. https://github.com/ethereum/consensus-specs/commit/54093964c95fbd2e48be5de672e3baae8531a964, 2024. Accessed: 2024-08-09. (Cited on page 1, 2.)

[Eth24b] Ethereum. Ethereum Consensus Specs - EIP 7594. https://github.com/ethereum/consensus-specs/tree/dev/specs/_features/eip7594, 2024. Accessed: 2024-06-24. (Cited on page 2.)

[FK20] Dankrad Feist and Dmitry Khovratovich. Fast amortized Kate proofs. https://github.com/khovratovich/Kate/blob/master/Kate_amortized.pdf, 2020. Accessed: 2024-06-27, Commit f4e5472. (Cited on page 14.)

[FK23] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023. https://eprint.iacr.org/2023/033. (Cited on page 14.)

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. (Cited on page 5.)

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discret. Appl. Math.*, 156(16):3113–3121, 2008. (Cited on page 4.)

[HASW23] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. Cryptology ePrint Archive, Paper 2023/1079, 2023. https://eprint.iacr.org/2023/1079. (Cited on page 2, 3, 6, 7, 8, 12.)

[KDF22] George Kadianakis, Ansgar Dietrichs, and Dankrad Feist. A Universal Verification Equation for Data Availability Sampling. https://ethresear.ch/t/a-universal-verification-equation-for-data-availability-sampling/13240, 2022. Accessed: 2024-06-24. (Cited on page 4, 15.)

[KZG10a] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010. (Cited on page 3, 6.)

[KZG10b] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Polynomial Commitments. https://cacr.uwaterloo.ca/techreports/2010/cacr2010-10.pdf, 2010. Accessed: 2024-07-10. (Cited on page 4.)

[Lip22] Helger Lipmaa. A unified framework for non-universal SNARKs. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 553–583. Springer, Heidelberg, March 2022. (Cited on page 5.)

[LPS23] Helger Lipmaa, Roberto Parisella, and Janno Siim. Algebraic group model with oblivious sampling. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 363–392. Springer, Heidelberg, November / December 2023. (Cited on page 5.)

[NB23] Valeria Nikolaenko and Dan Boneh. Data availability sampling and danksharding: An overview and a proposal for improvements. https://a16zcrypto.com/posts/article/an-overview-of-danksharding-and-a-proposal-for-improvement-of-das/, 2023. Accessed: 2024-08-14. (Cited on page 3.)