Foundations of Data Availability Sampling

Mathias Hall-Andersen*[1] Mark Simkin [2] Benedikt Wagner† [3,4]

February 12, 2024

[1] Aarhus University
ma@cs.au.dk
[2] Ethereum Foundation
mark.simkin@ethereum.org
[3] CISPA Helmholtz Center for Information Security
benedikt.wagner@cispa.de
[4] Saarland University

Abstract

Towards building more scalable blockchains, an approach known as data availability sampling (DAS) has emerged over the past few years. Even large blockchains like Ethereum are planning to eventually deploy DAS to improve their scalability. In a nutshell, DAS allows the participants of a network to ensure the full availability of some data without any one participant downloading it entirely. Despite the significant practical interest that DAS has received, there are currently no formal definitions for this primitive, no security notions, and no security proofs for any candidate constructions. For a cryptographic primitive that may end up being widely deployed in large real-world systems, this is a rather unsatisfactory state of affairs.

In this work, we initiate a cryptographic study of data availability sampling. To this end, we define data availability sampling precisely as a clean cryptographic primitive. Then, we show how data availability sampling relates to erasure codes. We do so by defining a new type of commitment schemes which naturally generalizes vector commitments and polynomial commitments. Using our framework, we analyze existing constructions and prove them secure. In addition, we give new constructions which are based on weaker assumptions, computationally more efficient, and do not rely on a trusted setup, at the cost of slightly larger communication complexity. Finally, we evaluate the trade-offs of the different constructions.

Keywords: Data Availability Sampling, Commitments, Erasure Codes, Coupon Collector

*Funded by the Concordium Foundation.

†This work was done while the third author was doing an internship at the Ethereum Foundation.

Part I
Main Content

Table of Contents

1 Introduction

As cryptocurrencies continue to grow in popularity, their scalability is becoming more and more of an issue. While the VISA[1] payment system handles around 1700 transactions per second and claims to be able to handle up to 24000 transactions per second, the Ethereum blockchain can at most handle around 60 per second[2]. Increasing the number of transactions that a blockchain can process is not an easy task. Transactions correspond to data that needs to be stored in a replicated fashion across a large number of independent validators. Thus, increasing the number of transactions means increasing the amount of data that needs to be stored and validated. At its core, blockchains aim to be distributed systems by the people for the people, which should not require any sort of trusted centralized authorities. As such, it is of crucial importance that regular individuals with reasonable amounts of computational power and memory are able to participate in the distributed systems that form the blockchain.

The data that comprises a blockchain can be seen as a sequence of blocks, where each block is composed of a small block header and a larger block content. To enable everyone to participate, clients can either join as so called full nodes that store and verify both block header and content or as light nodes that only store the headers. Light nodes can still use the functionalities that a blockchain provides as they can verify the information they receive is consistent with the corresponding block headers. However, they can not verify whether all data associated with the block headers they store is valid. A block may, for instance, contain transactions that illegally attempt to spend the same coin twice. This would not be visible from inspecting the header alone. Therefore, light nodes rely on full nodes to inform them when an adversarial party tries to provide them with a header for malformed data. This is done via a mechanism known as fraud proofs.

Abstractly speaking, a fraud proof allows a full node to convince a light node that a block header and the corresponding block content do not form a valid block. To produce a fraud proof, the full node needs access to the malformed block's content, but the adversary may only publish a header and either partially or fully withhold the corresponding block content. While full nodes can convince light nodes that a block is malformed, they cannot convince them that a block's content is just not available on the network. For this reason, light nodes need a mechanism for determining whether the block content corresponding to some header is available or not. Naively, light nodes could attempt to download the full content in addition to the header, but this would completely defeat the whole point of being a light node in the first place. Thus, light nodes need some way of efficiently checking that block contents are *fully* available on the network without actually fully retrieving them.

Data availability sampling (DAS) schemes, first introduced by Al-Bassam et al. [ASBK21], aim to solve the problem outlined above. Informally speaking, such schemes allow a possibly malicious block proposer to encode a bit string data, such as a block's content, into a short commitment com and a codeword π. The commitment com is added to the block header and allows light nodes to verify the availability of the full encoded block content π by randomly probing it in only a few positions. If enough light nodes successfully probed π, DAS ensures that the data is indeed fully available. Note that one light node alone cannot be convinced that the data is fully available, as it only queries a small part of the encoding and thus we need to talk about sufficiently large groups of light nodes.

Unfortunately, and despite its significant practical importance, there are no proper theoretical foundations for this new primitive. Existing works [ASBK21, YSL$^+$20, SXKV21, NNT21] all discuss DAS schemes at an informal level without precise security definitions and without full proofs of security for the proposed constructions. For a cryptographic primitive that is planned to become a key component of major blockchains like Ethereum[3], this is a rather unsatisfactory state of affairs.

1.1 Our Contributions

In this work, we provide a comprehensive theoretical treatment of data availability sampling. We formally define what DAS schemes are, precisely state the security notions they should satisfy, prove existing constructions such as as the one on the Ethereum roadmap secure, present new constructions, and finally compare all constructions in terms of concrete efficiency and discuss various trade-offs.

[1] See `https://usa.visa.com/run-your-business/small-business-tools/retail.html`.

[2] A new block on the Ethereum blockchain is produced every 12 seconds, has 15000000 gas available, and a transaction costs 21000 gas, meaning that the network can process at most $15000000/(21000 \cdot 12) \approx 60$ transactions per second.

[3] See `https://ethereum.org/en/developers/docs/data-availability`.

Formal Definitions. On an intuitive level, a DAS scheme should satisfy three main properties. First, it should be *complete*, meaning that verifiers[4] holding a valid commitment com and probing a valid encoding π, should successfully conclude that the encoded data is fully available. Second, it should be *sound* in the sense that enough successful probes to the encoding should allow for recovering some data bit string. Third, a DAS scheme should provide *consistency*, requiring that for a fixed but possibly malformed commitment com, one can recover at most one unique data bit string. We stress that DAS schemes ensure the full availability of some data, but they do not provide any guarantees about the structure or the contents thereof. A more detailed discussion and the main formal definitions themselves, along with several extensions are given in Section 3.

Constructions. We present multiple constructions, some being new, some being old but previously unproven. More concretely, we provide four constructions in this work. We show how to build DAS schemes (1) generically from vector commitments and SNARKs, (2) from collision-resistant hash functions in the random oracle model, (3) from homomorphic collision-resistant hash functions in the random oracle model, and (4) from polynomial commitments. The first construction can be seen as the "trivial solution", the second and third constructions are new to this work, whereas the fourth construction is the one that is currently envisioned by Ethereum, but was previously lacking any kind of security proof.

As a building block that may be of independent interest, we introduce the notion of erasure code commitments. Roughly, these ensure that any set of openings belonging to the same commitment must be consistent with at least one codeword from the corresponding erasure code. Polynomial commitments, for example, can be seen as erasure code commitments for Reed-Solomon codes. We formalize the notion of erasure code commitments, study their properties, and explain how they are related to DAS in Section 6. We describe our DAS schemes in Sections 7 to 9.

Benchmarks. In addition to the theoretical parts of our work, we also investigate the concrete efficiency of all constructions presented in this work. We compare them with each other, but also with some "naive" approaches to DAS, with respect to metrics like commitment size, encoding size, number of probes needed per verifier, and number of probes needed for reconstruction of the data. Our experiments show that no one shoe fits all and that the choice of construction really depends on the context within which they are used. We provide a detailed analysis and do our best to elucidate the trade-off between the different constructions in Section 10.

1.2 Related Work

DAS schemes are closely related to multiple already existing cryptographic primitives. In the following, we highlight some differences that make DAS a primitive of its own.

Proofs of Retrievability. The general concept of verifiers ensuring that some encoded data is fully available via a small number of probes is not new. Proofs of retrievability (PoR) [JK07, ABC+07, SW08, DVW09, CKW13, SSP13] consider a setting, where a *trusted* client encodes some data and then stores the encoding on an untrusted server. DAS schemes and PoRs are different in multiple ways. The most important difference is that in DAS, we do not assume the encodings to be generated in an honest manner. To deal with malicious encodings, we additionally require a consistency property as outlined above. While it is conceivable that some PoR constructions do achieve some form of consistency, they would only do so with very poor parameters as they are not designed to quickly detect malicious encodings. In PoR, a single server stores the encoding and may need to perform computations on it to respond to verifiers' queries. In our setting, the verifiers only need the ability to access arbitrary symbols of the encoding. In particular, this means that our codewords can be stored in a distributed fashion in a network and need not be fully stored on a single machine. Lastly, we consider retrieving back the data from the codeword as part of the functionality that DAS provides, whereas PoR consider it part of the security definition. As such, PoR schemes may use non-blackbox techniques to extract the original data, whereas our definitions require that the data is extractable from a sufficient number of independently performed probes.

Verifiable Information Dispersal. In the verifiable information dispersal setting [Rab89, CT05, NNT21] a potentially malicious party encodes a data bit string and stores the encoding in a distributed fashion among n servers of which at most t are corrupt. Upon receiving their share of the encoding, the servers interact with each other to determine whether the encoding they jointly store is valid or not.

[4] From now on we take a step back from the concrete application and focus more on the primitive itself. We we shall use light nodes, light clients, clients, and verifiers interchangeably to denote the same entities.

This setting inherently considers a non-adaptive adversary as the encoding needs to be fixed before the servers start interacting with each other. In our setting, we do not make any assumptions about how many servers there are, how many of them are corrupt, or how the encoding is stored in the network and leave this up to the application that makes use of our DAS schemes, thus allowing for greater flexibility as the storage servers could for example change over time. Consequently, we also do not require any interaction between any servers, meaning that encoding is a non-interactive process. The security notions we formulate for DAS consider adaptive adversaries that are not bound to a specific malicious encoding, but that can instead just answer probe requests by the verifiers in an adaptive malicious fashion.

PCPP, IOPP and Proximity Testing. Intuitively, the concepts of (extractable) probabilistically checkable proofs of proximity (PCPP) [BGH+06] and its interactive generalization interactive oracle proofs of proximity (IOPP) [BCG+17] share features with our notion of erasure code commitments: namely a verifier which makes a small number of queries to the encoding. There are, however, some crucial differences. PCPPs and IOPPs require that openings are close to a valid codeword, but we require openings to be consistent with a valid codeword. Furthermore, our commitments rely on computational assumptions, whereas PCPPs and IOPPs are usually studied in the information-theoretic security setting and computational assumptions are only used to compile them into non-interactive arguments. We leave it up to future work to explore the connection between PCPP/IOPP literature [BGKS20] and erasure code commitments more closely.

Vector, Polynomial, and Functional Commitments. Our new notion of erasure code commitments is a generalization of both vector commitments [CHL+05, CFM08, LY10] and polynomial commitments [KZG10, BDFG20, CHM+20], but can (conceptually at least) be seen as a special case of functional commitments [LRY16]. Our constructions of erasure code commitments are simpler, computationally more efficient, and rely on weaker assumptions than the currently known constructions of functional commitments. In [ADVZ21], the notion of erasure coding proof systems is introduced to construct verifiable information dispersal. Although their notion shares some similarities with our notion of erasure code commitments, the presentation in [ADVZ21] is rather informal, especially when it comes to security definitions. We on the other hand provide precise security definitions and full proofs for all of our constructions.

2 Preliminaries

In this section, we fix notation and preliminaries.

Notation. The set $[L] := \{1, \dots, L\} \subseteq \mathbb{N}$ is the set of the first L natural numbers. If S is a finite set, $s \leftarrow_{\$} S$ means that s is sampled uniformly at random from S. If $\mathcal{D}$ is a distribution, $x \leftarrow \mathcal{D}$ means that x is sampled from $\mathcal{D}$. If $\mathcal{A}$ is a probabilistic algorithm, we write $s := \mathcal{A}(x; \rho)$ to indicate that $\mathcal{A}$ outputs s on input x with random coins ρ, and $s \leftarrow \mathcal{A}(x)$ means that ρ is sampled uniformly at random. The notation $s \in \mathcal{A}(x)$ means that there are random coins ρ such that $\mathcal{A}$ outputs s on input x with these coins ρ. For an algorithm $\mathcal{A}$, a string $s \in \Sigma^*$ over some alphabet Σ, and an integer $t \in \mathbb{N}$, the notation $y \leftarrow \mathcal{A}^{s,t}(x)$ indicates that $\mathcal{A}$ has t-time oracle access to s on input x and outputs y. That is, $\mathcal{A}$ can query i and obtains the ith symbol $s_i \in \Sigma$ of s, for at most t queries. Let $\mathcal{A}$ be an algorithm as above, and let $\mathcal{B}$ be a (potentially stateful) algorithm $\mathcal{B}$. We write $y \leftarrow \mathcal{A}^{\mathcal{B},t}(x)$ to indicate that the oracle queries of $\mathcal{A}$ are answered by $\mathcal{B}$. Further, we use the notation $(y_i)_{i=1}^{\ell} \leftarrow \mathsf{Interact}\left[\mathcal{A}, \mathcal{B}\right]_{t,\ell}(x)$ to indicate that ℓ independent copies of $\mathcal{A}$ get x as input, and have t-time oracle access to $\mathcal{B}$ (i.e., the oracle queries of $\mathcal{A}$ are answered by $\mathcal{B}$), and the ith copy outputs y_i for each $i \in [\ell]$. Here, $\mathcal{B}$ can schedule the oracle queries of these ℓ copies in an arbitrary concurrently interleaved way. That is, $\mathcal{B}$ has access to an oracle $\mathrm{O}_{\mathrm{nextQ}}$, that on input $i \in [\ell]$ outputs the next query of the ith copy, given that $\mathcal{B}$ already submitted the response to the previous queries of that copy. All algorithms get the security parameter λ in unary at least implicitly as input. An algorithm $\mathcal{A}$ is said to be PPT if its running time, denoted by $\mathbf{T}(\mathcal{A})$, is bounded by a polynomial in its input. An algorithm $\mathcal{A}$ is said to be EPT if its expected running time, denoted by $\mathbf{ET}(\mathcal{A})$, is bounded by a polynomial in its input. We write $\Pr_{\mathbf{G}}[E]$ or $\Pr[E \mid \mathbf{G}]$ to denote the probability that some event E occurs in the experiment $\mathbf{G}$. Also, we denote the event that an experiment $\mathbf{G}$ outputs a bit b by $\mathbf{G} \Rightarrow b$. A function f is said to be negligible in its input λ, if $f \in \lambda^{-\omega(1)}$. Throughout, negl always denotes a negligible function.

Cryptographic Building Blocks. For some constructions, we make use of common cryptographic

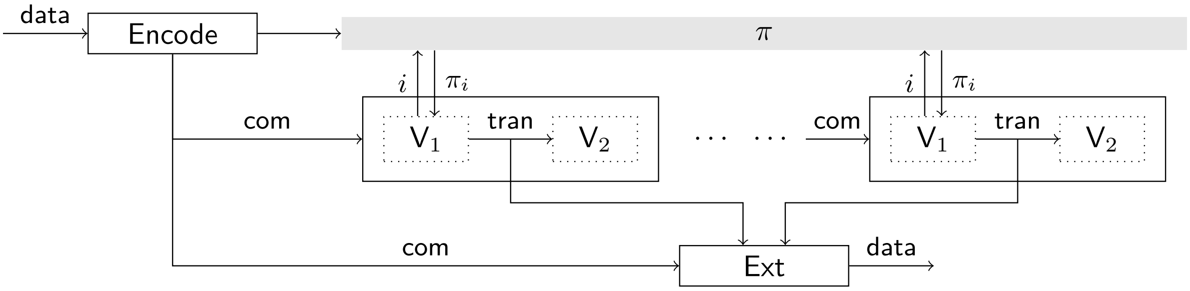

Figure 1: Overview of the syntax of a data availability sampling scheme. All algorithms get sytem parameters $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$ as input. Algorithm Encode encodes data into an encoding π. Multiple clients $(\mathsf{V}_1, \mathsf{V}_2)$ can then query this encoding. From enough transcripts, data can be reconstructed using algorithm Ext.

building blocks, including vector commitments, non-interactive arguments, and homomorphic hash functions. We recall their formal definitions in Appendix A.

3 Definition of Data Availability Sampling

This section is dedicated to presenting our definition of data availability sampling. In Section 3.1, we define a data availability sampling scheme as a cryptographic primitive. Then, in Section 3.2, we introduce extensions for this basic definition.

3.1 Basic Definition

Here, we introduce our definition of a data availability sampling scheme.

Setting. We consider a scenario in which a party, called the proposer, holds a large piece of data. The proposer wants to store this data within a network, possibly in a distributed way. For example, this data could be a block that should be published in a peer-to-peer network running a blockchain. In addition to the proposer and the network, there are parties with limited resources, which we call (light) clients, or verifiers. They can only download small headers, which hold information about the corresponding data, but are not capable of downloading the entire data itself. Still, they want to verify that the data is available within the network. To do so, light clients can issue queries to the network. Our formal definition of data availability sampling models such a scenario.

Syntax. We give a schematic overview of our syntax in Figure 1. Suppose the proposer holds a piece of data data to be distributed. In our syntax, the proposer runs an algorithm Encode with input data to obtain a commitment com and an encoding π of the data. We assume that every party downloads com. For example, we may think of com to be part of a block header. We do not explicitly model how π is being stored. As our security notions treat π as being fully controlled by the adversary, this means any way of storing π is covered. For example, we may think of π as being stored in a distributed way on nodes within a network. We model clients by two algorithms V_1 and V_2, where V_1 can probabilistically query positions of the encoding π. The resulting transcript tran, which contains all queries and responses, is then input into V_2, which deterministically outputs 0 (for reject) or 1 (for accept). We split the client into these two algorithms to talk about (accepting) transcripts explicitly. Finally, we define an algorithm Ext that extracts the original data data from enough of these transcripts. The idea is that clients share their transcripts with others and, once a party has enough transcripts, it can run Ext to get data.

Properties. Having defined the syntax of a data availability sampling scheme, we now turn to the properties these algorithms should satisfy. Clearly, we need a *completeness* definition. As usual, this definition states that everything works as expected, given that all algorithms are executed honestly. In our concrete case, this means that if some data data is encoded honestly, then all clients will accept, and Ext outputs data when run on enough transcripts.

Of course, we need to include security definitions for the case of malicious behavior. On a high level, we want light clients to verify that data is available. Informally, this means that if they accept, then data should be available. We capture this formally in our definition of *soundness*. To understand this definition, we need to make the concept of data being available more precise. We do this using the extraction algorithm Ext that we defined. Namely, we think of data as being available if Ext can extract something, and does not output $\bot$. With this in mind, soundness means that if enough clients accept, then Ext can extract something from their transcripts. Notably, we have to define this in the presence of a malicious encoding π that is fully controlled by an adversary. The adversary can thus schedule the queries that the clients issue, and it can answer these queries adaptively. This also shows why we require that enough clients accept, and not only that one client accepts. An adversary could, for example, answer the queries of one client honestly and not respond anything to any of the other clients. Clearly, there is no hope to extract anything from only one accepting transcript, as it is shorter than the data.

The definitions of completeness and soundness alone are not meaningful by themselves yet, since Ext could just output some default value when it fails to reconstruct. This is clearly not what we want. Also note that in applications of data availability, clients may want to pool together their transcripts to run Ext to recover the encoded data. In this case, it is essential that we always recover the *same* data, no matter where the transcripts come from. We capture this wish by defining *consistency*. This notion means that whenever Ext is run twice on two (possibly intersecting) sets of transcripts and the same commitment com, and it outputs $\mathsf{data}_1 \neq \bot$ and $\mathsf{data}_2 \neq \bot$, respectively, then $\mathsf{data}_1 = \mathsf{data}_2$, i.e., the extracted data is consistent. In other words, the data availability sampling scheme bootstraps consensus on the commitment com to consensus on data. Furthermore, it should be noted that for our consistency notion we let the adversary output the transcripts, which makes it very strong and flexible. We are now ready to present the complete formal definition.

Definition 1 (Data Availability Sampling Scheme)**.** A data availability sampling scheme (DAS) with data alphabet Γ, encoding alphabet Σ, data length $K \in \mathbb{N}$, encoding length $N \in \mathbb{N}$, query complexity $Q \in \mathbb{N}$, and threshold $T \in \mathbb{N}$ is a tuple $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V}, \mathsf{Ext})$ of algorithms with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$ is a PPT algorithm that takes as input the security parameter, and outputs system parameters par. All algorithms get par implicitly as input.
- $\mathsf{Encode}(\mathsf{data}) \to (\pi, \mathsf{com})$ is a deterministic polynomial time algorithm that takes as input data $\mathsf{data} \in \Gamma^K$ and outputs an encoding $\pi \in \Sigma^N$ and a commitment com.
- $\mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2)$ is a pair of algorithms, where
  - $\mathsf{V}_1^{\pi,Q}(\mathsf{com}) \to \mathsf{tran}$ is a PPT algorithm that has Q-time oracle access to an encoding $\pi \in \Sigma^N$, gets as input a commitment com, and outputs a transcript tran, containing the Q queries to π and the respective responses.
  - $\mathsf{V}_2(\mathsf{com}, \mathsf{tran}) \to b$ is a deterministic polynomial time algorithm that takes as input a transcript tran, and outputs a bit $b \in \{0,1\}$.
- $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \to \mathsf{data}/\bot$ is a deterministic polynomial time algorithm that takes as input a commitment com, a list of transcripts tran_i, and outputs data $\mathsf{data} \in \Gamma^K$ or an abort symbol $\bot$.

We require that the following properties are satisfied:

- **Completeness.** For any $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$ and any integer $\ell = \mathsf{poly}(\lambda)$ with $\ell \geq T$, and all $\mathsf{data} \in \Gamma^K$, we have

$$\Pr\left[\forall i \in [\ell] : b_i = 1 \wedge \mathsf{data}' = \mathsf{data} \;\middle|\; \begin{array}{l} (\pi, \mathsf{com}) := \mathsf{Encode}(\mathsf{data}), \\ \forall i \in [\ell] : \mathsf{tran}_i \leftarrow \mathsf{V}_1^{\pi,Q}(\mathsf{com}), \\ \quad b_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}_i), \\ \mathsf{data}' := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Soundness.** For any stateful PPT algorithm $\mathcal{A}$ and any integer $\ell = \mathsf{poly}(\lambda)$ with $\ell \geq T$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A},\ell,\mathsf{DAS}}(\lambda) := \Pr\left[\forall i \in [\ell] : b_i = 1 \wedge \mathsf{data}' = \bot \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda),\; \mathsf{com} \leftarrow \mathcal{A}(\mathsf{par}), \\ (\mathsf{tran}_i)_{i=1}^{\ell} \leftarrow \mathsf{Interact}\left[\mathsf{V}_1, \mathcal{A}\right]_{Q,\ell}(\mathsf{com}), \\ \forall i \in [\ell] : b_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}_i), \\ \mathsf{data}' := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \end{array}\right].$$

- **Consistency.** For any PPT algorithm $\mathcal{A}$ and any $\ell_1, \ell_2 = \mathsf{poly}(\lambda)$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{cons}}_{\mathcal{A},\ell_1,\ell_2,\mathsf{DAS}}(\lambda) := \Pr\left[\begin{array}{rl} & \mathsf{data}_1 \neq \bot \\ \wedge & \mathsf{data}_2 \neq \bot \\ \wedge & \mathsf{data}_1 \neq \mathsf{data}_2 \end{array} \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{com}, (\mathsf{tran}_{1,i})_{i=1}^{\ell_1}, (\mathsf{tran}_{2,i})_{i=1}^{\ell_2}) \leftarrow \mathcal{A}(\mathsf{par}), \\ \mathsf{data}_1 := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{1,1}, \ldots, \mathsf{tran}_{1,\ell_1}), \\ \mathsf{data}_2 := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{2,1}, \ldots, \mathsf{tran}_{2,\ell_2}) \end{array}\right].$$

Discussion. We want to highlight a few aspects of our definition. First, note that we require Encode to be deterministic. At a first glance, this may seem to be too restrictive, as encoding could make use complex cryptographic tools, e.g., a (succinct) non-interactive argument [BFM88, Kil92, Gro16]. However, observe that in the context of data availability sampling, we do not require any privacy properties, e.g., zero-knowledge, from these tools. If any, we require their correctness and soundness properties, which hold even if the randomnes is fixed, i.e., we make these schemes deterministic. Second, one could wonder why we do not require that re-encoding data leads to the same commitment com in our soundness definition. Here, we observe that this is not satisfied by natural constructions based on perfectly-hiding commitments in style of [Ped92, KZG10]. Namely, an adversary could run $(\pi, \mathsf{com}) := \mathsf{Encode}(\mathsf{data})$, rerandomize com (and adjust π if needed), and then behave honestly. Third, we emphasize that our definition of soundness and consistency allows the adversary to be fully adaptive. That is, the adversary can schedule the queries of clients and answer them in an adaptive way. This is much stronger than what is present in the informal security goals stated by previous works, where the adversary first decides which parts of the encoding should be available, and then clients start probing. We believe that such strong adaptive security notions are more appropriate for real-world settings, where independent verifiers asynchronously query parts of an encoding that is stored in a distributed fashion among multiple possibly malicious nodes.

Efficiency Measures. When constructing data availability sampling schemes, there are several properties we aim to optimize. It is of primary interest to minimize the computational and communication complexity of clients. In particular, we would like to minimize the the computational complexity of V, the communication complexity $\log N + \max_{s \in \Sigma} |s|$ per query[5], and the size of commitments $|\mathsf{com}|$. Additionally we would also like to minimize the encoding size $|\pi|$ and the computational complexity of Encode, i.e., the effort of the parties encoding the data and storing the encoding. Lastly, we want to minimize the number T of transcripts that are needed to reconstruct the data. The smaller the number T, the more "meaningful" is each verifier's transcript, when it comes to establishing the availability of some data. Note that minimizing T and the query complexity per client Q at the same time also minimizes the total communication complexity that is needed to reconstruct the data.

Strawman Solutions. At first sight, it may seem easy to construct DAS schemes. Let us discuss a few natural, but failing attempts. Firstly, we can easily achieve query complexity $Q = 1$ and threshold $T = 1$ by setting $\Sigma := \Gamma^K$, i.e., by considering the full data as a single symbol and letting every client download it in full. Obviously, this solution has a terrible communication complexity per client query. Alternatively, one can also make this communication complexity small by storing the whole data as part of the commitment com, which would be equally undesirable. A slightly more intelligent approach may be to store the root of a Merkle tree [Mer88], computed over the data, as the commitment and let the verifiers query random leaves in the tree. This solution has a small commitment size and a small communication complexity per query, but the required number of transcripts T for reconstructing the data with high probability would be very large (cf. Example 5). Intuitively, T being very large corresponds to each client individually not really being very much convinced about the availability of the full data. Lastly, one could try and use ideas from the proofs of retrievability literature and first encode the data with an erasure code, before computing a Merkle tree over the symbols of the code. Note however, that the

[5] One needs $\log N$ bits upload for the query (to specify which position) and $\max_{s \in \Sigma} |s|$ bits download for the response.

encoding may be done in a malicious way. For instance, the first half of the leaves could correspond to the first half of a valid codeword encoding data, whereas the second half of the leaves could correspond to the second half of a codeword encoding data' with $\mathsf{data} \neq \mathsf{data}'$, which would allow an adversary to violate the consistency requirement. To prevent this attack, one would need to pick a large value for Q, which would then render the solution inefficient.

Subset-Soundness. We imagine that clients send the transcripts of their interaction to the network. Then any node that collects enough of these transcripts should be able to reconstruct data from these transcripts, according to soundness. However, under the realistic assumption that an adversary controls parts of the network, it may adaptively drop some of the transcripts after seeing them. We extend our basic soundness definition to cover this attack scenario, and call the resulting notion *subset-soundness*. In this notion, we run the basic soundness experiment, but additionally let the adversary select a subset of the transcripts from which we try to reconstruct data. In other words, we let the adversary drop a limited number of transcripts of its choice. After defining subset-soundness, we show that it is implied by standard soundness for certain parameter ranges.

Definition 2 (Subset-Soundness)**.** Let $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2), \mathsf{Ext})$ be a data availability sampling scheme. We say that DAS satisfies (L, ℓ)-subset-soundness, if for any stateful PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{sub\text{-}sound}}_{\mathcal{A},L,\ell,\mathsf{DAS}}(\lambda) := \Pr\left[\forall j \in [\ell] : b_{i_j} = 1 \wedge \mathsf{data}' = \bot \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{com} \leftarrow \mathcal{A}(\mathsf{par}), \\ (\mathsf{tran}_i)_{i=1}^{L} \leftarrow \mathsf{Interact}\left[\mathsf{V}_1, \mathcal{A}\right]_{Q,L}(\mathsf{com}) \\ \forall i \in [L] : b_i := \mathsf{V}_2(\mathsf{tran}_i), \\ (i_j)_{j=1}^{\ell} \leftarrow \mathcal{A}(\mathsf{tran}_1, \dots, \mathsf{tran}_L), \\ \mathsf{data}' := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{i_1}, \dots, \mathsf{tran}_{i_\ell}) \end{array} \right].$$

Lemma 1. *Let* $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2), \mathsf{Ext})$ *be a data availability sampling scheme with threshold* $T \in \mathbb{N}$*, and let* $L, \ell \in \mathbb{N}$ *be such that* $\binom{L}{\ell} \leq \mathsf{poly}(\lambda)$ *and* $\ell \geq T$*. Then,* DAS *satisfies* (L, ℓ)*-subset-soundness. More specifically, for any stateful PPT algorithm* $\mathcal{A}$*, there is a stateful PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *and*

$$\mathsf{Adv}^{\mathsf{sub\text{-}sound}}_{\mathcal{A},L,\ell,\mathsf{DAS}}(\lambda) \leq \binom{L}{\ell} \cdot \mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A},\ell,\mathsf{DAS}}(\lambda).$$

Proof. The proof is trivial via a guessing argument, i.e. the reduction $\mathcal{B}$ simply guesses the subset that will be chosen by $\mathcal{A}$. □

Necessity of Assumptions. Naturally, one can ask whether it is possible to construct a (non-trivial) data availability sampling scheme satisfying our notions. Throughout this work, we will show several constructions and therefore show that it is possible. However, all constructions rely on computational assumptions or idealized models, e.g., the random oracle model. Again, one can ask whether this is necessary, or whether one can construct data availability sampling schemes without any cryptographic assumption. The result is negative: we show that any non-trivial scheme, i.e where the commitment is smaller than the data, implies a collision-resistant hash function. The hash function is induced by the mapping from data to com via algorithm Encode. We formally show this in Appendix C.1.

3.2 Extensions

Here, we informally introduce two extensions of our basic definition of data availability sampling schemes. We postpone a formal definition to Appendices C.2 and C.3.

Repairability. Ideally, a data availability sampling scheme allows to reconstruct data even if small parts of the encoding are broken or lost. In this case, it is natural to ask whether one can return from such a damaged state to a stable state by repairing the encoding. More precisely, we would like to have a way to recover an encoding from a set of transcripts with which we can continue as if it was the original encoding. Importantly, it should work with the original commitment. This enables a transparent repair on the fly without notifying every party about the change, and without updating the commitment. For example, one problem when changing the commitment is that we would have to convince every party that the new commitment commits to the same data as the old one. We define an extension of data availability with such a repairability feature by requiring the existence of an algorithm Repair. On input a commitment com and a set of transcripts, Repair outputs a new encoding $\bar{\pi}$. Informally, we expect $\bar{\pi}$ to be compatible with the commitment com, and function as the original encoding, assuming that Repair obtained enough accepting transcripts. We make this formal by introducing the notion of *repair liveness*. In this notion, we let an adversary output a com and interact with clients as in the subset-soundness notion, i.e., clients query an encoding provided adaptively by the adversary. Then, we repair from a subset of the resulting transcripts by running algorithm Repair. Finally, we expect that all clients accept when querying this repaired encoding with com as input. If this does not hold, the adversary breaks repair liveness.

Example 1 (Accountability for Trivial Repairability). If we were to naively implement repairability, we would extract the data from a sufficient number of transcripts and recompute an encoding. Note that this approach will not work in general, because there is no guarantee that the new encoding is compatible with the old commitment. Especially, an adversary might be able to compute a functioning pair of commitment and encoding different from an honestly computed pair for some data. However, when this trivial approach fails, it produces a certificate that the original commitment was computed incorrectly. Namely, by having the proposer sign the commitment, a set of transcripts from which reconstruction is possible but the re-encoding of the data does not yield the original commitment forms a publicly verifiable certificate that the original encoding and commitment was not computed honestly. This observation has possible applications in scenarios where fallback to the trivial data availability scheme is feasible. For example, in cryptocurrency applications, the full data can be posted on the chain to repair and the malicious encoders deposit can be forfeited to cover the cost of posting the full data.

Local Accessibility. A natural question to ask is whether one needs to reconstruct the entire data, even if one is only interested in small parts of it. Concretely, say a client is interested in learning the ith symbol of the encoded data. We enhance our basic definition of data availability sampling schemes with such a local accessibility feature by introducing an algorithm Access. Roughly, it recovers a specific symbol of the encoded data by querying the encoding. Namely, it gets as input a commitment com and an index $i \in [K]$, has oracle access to an encoding, and outputs a symbol d, which should be understood as being the ith symbol of the encoded data data. Crucially, we need to ensure that this new way of obtaining (parts of) the data does not introduce inconsistencies. Thus, we introduce the notion of *local access consistency*, which states that whatever Access outputs is consistent with data extracted using a set of transcripts. More precisely, for any index $i \in [K]$, we let the adversary output a commitment com and a set of transcripts. Then, we run Access on input com, i to obtain a symbol d. The queries of Access are answered by the adversary. Further, we run the extractor Ext on input com and the set of transcripts to extract data data. We require that d is the ith symbol of data, given that both are not $\perp$.

Example 2 (Trivial Local Accessibility). There is a simple way to make every data availability sampling scheme locally accessible. Namely, every data availability sampling scheme with query complexity $Q \in \mathbb{N}$ and threshold $T \in \mathbb{N}$ is locally accessible with query complexity $L = QT$, i.e., Access makes QT queries to access one symbol of the data. This is because Access can simply run T clients internally and then extract from the resulting transcripts. The clear drawback of this trivial approach is that Access has a huge query complexity. Ideally, we aim for a way that lets us access any symbol with query complexity significantly smaller than QT, e.g., only with one query.

4 Overview of Constructions

In this section, we give an overview of our constructions of data availability sampling. We first introduce a generic framework of constructing data availability sampling from erasure codes and associated commitment schemes with suitable properties (Section 6). Equipped with this framework, we then focus on constructing such commitment schemes for several erasure codes (Sections 7 to 9). Finally, we compare instantiations of these constructions in terms of efficiency (Section 10). In this overview, we explain our framework and the constructions.

4.1 From Codes and Commitments to Data Availability

We construct data availability schemes by the introducing the new notion we call *erasure code commitments*. In the following, we first explain what erasure code commitments are. Then, we explain how to turn them into data availability sampling schemes.

Erasure Code Commitments. Erasure code commitments are binding vector commitments with the additional property that any set of openings produced by a computationally bounded adversary is consistent with at least one codeword from the erasure code. We call this additional notion *code-binding*. The existing notion of polynomial commitments is a special case for the Reed-Solomon code, although we do not require extraction of the commitment which is often required in applications of polynomial commitments, nor do we require hiding. Similarly vector commitments are erasure code commitments for the trivial erasure code, mapping every message to itself. In Section 6.1, we formally define erasure code commitments. Additionally, we also define a variety of additional security notions for these commitments and study their relations. We are confident in the usefulness of this natural generalization of polynomial commitments beyond data availability schemes.

Data Availability from Erasure Code Commitments. From erasure code commitments we follow an intuitive avenue to arrive at a data availability scheme:

- *Encoding.* The encoding algorithm $\mathsf{Encode}(\mathsf{data})$ first applies the erasure code to data obtaining a codeword, and then commits to the codeword using an erasure code commitment, which forms the data availability commitment com. The resulting encoding π consists of the symbols of the codeword and their corresponding openings of com.
- *Clients.* The first part of the client $\mathsf{V}_1^{\pi,Q}(\mathsf{com})$ relies on a randomized *index sampler* which returns a set of indices in the codeword. The client V_1 then queries the provided indexes of π, the list of responses forms the tran. The second part of the client $\mathsf{V}_2(\mathsf{com}, \mathsf{tran})$ verifies all the erasure code commitment openings obtained by V_1 against com.
- *Extraction.* Given enough accepting transcripts, one can then extract the encoded data (i.e., run algorithm Ext), assuming the transcripts contain sufficiently many of the symbols of the codeword.

The details on our compiler from erasure code commitments to data availability sampling are given in Section 6.3. It is clear that the parameters of the data availability sampling scheme depend on the parameters of the erasure code. In addition, the choice of the index sampler (e.g., sampling uniformly with replacement or without replacement) influences how many transcripts we need to collect enough distinct symbols of the codeword with high probability. To capture this, we define the quality of an index sampler. We study different index samplers and their quality in Section 6.2.

4.2 Constructions of Erasure Code Commitments

When constructing data availability sampling schemes, our framework introduced above allows us to concentrate on erasure codes, erasure code commitments, and index samplers. Here, we give a brief overview of our erasure code commitments.

Generic Construction. We show that one can generically construct erasure code commitments for any erasure code from vector commitments and succinct arguments of knowledge. Namely, this is done by proving that a vector commitment contains a codeword using a succinct argument of knowledge. The code commitment consists of the vector commitment and the succinct proof, and the proof is verified by clients. While this construction is far from being practical in general, it serves as a template for other constructions. We formally define and analyze this generic construction in Section 7.

Tensor Construction (Ethereum Construction). Ethereum has proposed a data availability scheme which can be phrased as an erasure code commitment for the tensor code of two Reed-Solomon codes: The data is arranged into a square $k \times k$ matrix, then every row is encoded using a Reed-Solomon code, yielding a $k \times n$ matrix, finally every column is encoded using a Reed-Solomon code yielding a $n \times n$ matrix. A commitment is formed by committing to each column individually using a polynomial commitment (i.e., a code commitment for the Reed-Solomon code), then checking consistency of the rows by exploiting the linear homomorphism of the commitment similar to Feldman secret sharing [Fel87]. We provide a formal description and analysis of this scheme for the tensor code of arbitrary codes in Section 8.

Hash-Based Construction. We provide a new construction for interleaved linear codes from random oracles, which is partially inspired by the Ligero proximity test [AHIV17]. Encoding and committing is done as follows: The message is first interpreted as a $k \times k$ matrix $\mathbf{M} \in \mathbb{F}^{k \times k}$ over a finite field $\mathbb{F}$. Each row is encoded independently using a linear code $\mathcal{C}$, leading to a $k \times n$ matrix $\mathbf{X} \in \mathbb{F}^{k \times n}$. The columns are now treated as the symbols of the interleaved code. To commit to such a codeword, the encoder commits to each column individually by hashing it with a collision-resistant hash function, producing n hashes $h_1, \ldots, h_n$. Including these hashes in the commitment already ensures position-binding. To ensure code-binding, i.e., that openings are always consistent with the code, the hashes are fed into a random oracle, which returns a challenge vector[6] $\mathbf{r} \in \mathbb{F}^k$. The encoder then computes the linear combination $\mathbf{w} = \mathbf{r}^\top \mathbf{X}$ of the rows and includes $\mathbf{w}$ in the commitment. Note that the resulting $\mathbf{w}$ always forms a codeword in the code $\mathcal{C}$, which is checked by the clients as we will describe below. For any fixed set $\mathcal{I} \subseteq [n]$ of positions that an adversary may now open inconsistently with the code, we could in principle argue that the verification only passes with negligible probability. However, in the notion of code-binding, the adversary is allowed to freely choose this set $\mathcal{I} \subseteq [n]$. It turns out that, if the committed $\mathbf{X}$ is far from the code, then we need a too wasteful union bound over the different choices of $\mathcal{I}$. To solve this issue, we need to add a proximity test: Using another random oracle, a random set of indices $J \subseteq [n]$ is determined, and the encoder has to add the columns $\{\mathbf{X}_j\}_{j \in J}$ to the commitment. The encoding π is simply the codeword $\mathbf{X}$ of the interleaved code. To be explicit, a coordinate $\mathbf{X}_j$ of the encoding is verified by checking $\mathbf{w}_j = \mathbf{r}^\top \mathbf{X}_j$ and $h_j = \mathsf{H}(\mathbf{X}_j)$. In addition, the openings in J are checked in a similar manner and it is verified that $\mathbf{w}$ is in the code $\mathcal{C}$. An advantage of this scheme is that we can implement it over small fields for computational efficiency, and the very small opening overhead. Namely, the opening proof of a position is simply the symbol itself. On the other hand, the commitment is large, both concretely and asymptotically. We present the construction in detail in Section 9.1.

Construction from Homomorphic Hashing. The hash-based construction for interleaved codes can be optimized by relying on linearly homomorphic hashes, which improves both the concrete and asymptotic size of the commitment. The description of this construction is provided in Section 9.2.

5 Background on Coding Theory

In this section, we discuss background about coding theory. We introduce notation, definitions, and basic facts about some specific codes. Looking ahead, we will show how codes relate to data availability sampling in subsequent sections.

5.1 Codes and Distance

We will now introduce codes and their properties. Informally, a *code* allows to deterministically encode a message over some alphabet Γ into a codeword over some alphabet Λ.

Erasure Codes. An *erasure code* has the additional property that any t symbols of the codeword are sufficient to reconstruct the message, for some $t \in \mathbb{N}$. The parameter t is called the *reception efficiency* of the code. In this work, we only consider erasure codes. Before we give the formal definition, we highlight that throughout the paper, we assume that the encoding and the reconstruction algorithm are efficiently computable. To make this assumption formal, we would have to talk about families of codes. We opt for a concise and readable notation instead of doing this.

Definition 3 (Erasure Code)**.** Let $k, n, t \in \mathbb{N}$ be natural numbers and Γ, Λ be sets. A function $\mathcal{C} \colon \Gamma^k \to \Lambda^n$ is an erasure code with alphabets Γ, Λ, message length k, code length n, and reception efficiency t, if there is a deterministic algorithm $\mathsf{Reconst}$, such that for any $m \in \Gamma^k$, and any $I \subseteq [n]$ with $|I| \geq t$ we have $\mathsf{Reconst}((\hat{m}_i)_{i \in I}) = m$ for $\hat{m} := \mathcal{C}(m)$. We say that $\mathsf{Reconst}$ is the reconstruction algorithm of $\mathcal{C}$, and assume that $\mathsf{Reconst}$ outputs $\bot$ if its input is not consistent with any codeword in $\mathcal{C}$ or if it gets less than t symbols as input. For convenience, we sometimes treat an erasure code $\mathcal{C}$ as a subset $\mathcal{C} \subseteq \Lambda^n$, where we implicitly mean the image of $\mathcal{C}$, i.e., $\mathcal{C}(\Gamma^k)$. We may then write $x \in \mathcal{C}$ to indicate that x is a codeword.

Distance. In coding theory, we are often interested in the *distance* between words. Naturally, the metric we consider is the *Hamming metric.* Concretely, for two strings x, y over the same alphabet and with the same length L, we define $d\,(x, y)$ to be the number of positions $i \in [L]$ for which $x_i \neq y_i$. An important attribute of a code is its minimum distance, which we define next.

[6] Depending on the field size, some parallel repetition may be needed.

Definition 4 (Minimum Distance)**.** Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. The (absolute) minimum distance d of $\mathcal{C}$ is defined as

$$d := \min_{m_1 \neq m_2 \in \Gamma^k} d\left(\mathcal{C}(m_1), \mathcal{C}(m_2)\right).$$

Further, we introduce the notion of *column-wise distance* of matrices in $\Lambda^{\ell \times n}$ for some $\ell \in \mathbb{N}$. This is just the hamming distance when the matrices are treated as strings over Λ^ℓ, i.e., every column is interpreted as a symbol. To make this explicit when needed, we write $d_{col}\left(\mathbf{X}, \mathbf{X}'\right)$ for two such matrices $\mathbf{X}, \mathbf{X}'$. Moreover, we extend the notion of distance to sets. Concretely, for a set of strings $S \subseteq \Lambda^\ell$ over some alphabet Λ and a string $x \in \Lambda^\ell$, we define $d\left(S, x\right) = d\left(x, S\right) := \min_{s \in S} d\left(s, x\right)$. The same can be done for the column-wise distance. Finally, we highlight an important property of the minimum distance d. Namely, if $\mathcal{C}$ is an erasure code with minimum distance d and $d\left(\mathcal{C}, x\right) \leq \lfloor (d-1)/2 \rfloor$ for some string x, then there is a unique codeword $c \in \mathcal{C}$ which is closest to x. We may say that x is within *unique decoding distance* of $\mathcal{C}$.

5.2 Special Families of Codes

In this section, we introduce some families of codes that will be of interest for this work.

Systematic Encoding. We say that a code $\mathcal{C}$ has a *systematic encoding*, if the message m is contained in the codeword $\mathcal{C}(m)$. Such a systematic encoding makes it easy to retrieve (parts of) the message from the codeword. In our context, we will use this property to extend the basic functionality of data availability sampling with local accessibility. We slightly generalize the standard definition of a systematic encoding. One reason for this generalization is that messages and codewords are over different alphabets.

Definition 5 (Generalized Systematic Encoding)**.** Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code with alphabets Γ, Λ, message length k, code length n, and reception efficiency t, with reconstruction algorithm $\mathsf{Reconst}$. We say that $\mathcal{C}$ has a generalized systematic encoding, if the following hold:

- There are two deterministic polynomial time algorithms Find and Proj, such that for any $m \in \Gamma^k$ and $\hat{m} := \mathcal{C}(m)$, and for any $i \in [k]$ and $\hat{i} := \mathsf{Find}(i)$, we have $\mathsf{Proj}(i, \hat{m}_{\hat{i}}) = m_i$.
- Let $I \subseteq [n]$ be arbitrary with $|I| \geq t$, and let $(\hat{m}_i)_{i \in I} \in \Lambda^{|I|}$ be any sequence of symbols in Λ. Let $m := \mathsf{Reconst}((\hat{m}_i)_{i \in I})$, and let $I^* := \{i \in [k] \mid \mathsf{Find}(i) \in I\}$. Then, for all $i \in I^*$ and $\hat{i} := \mathsf{Find}(i)$ it should hold that $\mathsf{Proj}(i, \hat{m}_{\hat{i}}) = m_i$.

We say that Proj is the symbol projection algorithm and Find is the symbol finding algorithm of $\mathcal{C}$.

Linear Erasure Codes. If $\mathcal{C}$ is a code that is a subspace of some vector space, we call it a *linear erasure code*. Alternatively, when viewing the code as an encoding function, it corresponds to an injective homomorphism of vector spaces. We restrict ourselves to vector spaces of finite size in this work.

Definition 6 (Linear Erasure Codes)**.** Let $\mathbb{F}$ be a finite field, possibly implicitly parameterized by the security parameter. A linear erasure code over $\mathbb{F}$ is an erasure code $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$, such that $\mathcal{C}$ is an injective homomorphism from the vector spaces $\mathbb{F}^k$ to the vector space $\mathbb{F}^n$ over $\mathbb{F}$.

Let us discuss some important properties of linear erasure codes. Linear erasure codes can be specified by one of two matrices. Namely, if $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$ is a linear erasure code, then there is a *generator matrix* $\mathbf{G} \in \mathbb{F}^{n \times k}$ with full rank such that for all $\mathbf{m} \in \mathbb{F}^k$ we have $\mathcal{C}(\mathbf{m}) = \mathbf{G}\mathbf{m}$. Additionally, there is a *parity-check matrix* $\mathbf{H} \in \mathbb{F}^{(n-k) \times n}$ such that $\mathcal{C}$ is exactly the kernel of $\mathbf{H}$. We also have $\mathbf{H}\mathbf{G} = \mathbf{0}$.

MDS Codes. The well-known *singleton bound* states that for linear[7] erasure codes $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$ with minimum distance d we have $d \leq n - k + 1$. An *MDS (maximum distance separable) code* is a linear erasure code that satisfies the singleton bound with equality.

Definition 7 (MDS Code)**.** Let $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$ be a linear erasure code over field $\mathbb{F}$, and let d denote its minimum distance. Then, $\mathcal{C}$ is called an MDS code, if $d = n - k + 1$.

[7] The singleton bound can also be stated for arbitrary codes, but we do not need that in our work.

MDS codes have several interesting properties. Most importantly for us, every set of $n-k$ columns of the parity-check matrix forms an invertible matrix. Further, one can show that for any k symbols $x_{i_1}, \ldots, x_{i_k}$ there is a unique codeword $\mathbf{x} \in \mathcal{C}$ such that the i_jth symbol of $\mathbf{x}$ is x_{i_j} for all $j \in [n]$. That is, every k symbols are consistent with the code. To see this, note that the function that maps messages to the symbols of the codeword at positions $i_1, \ldots, i_k$ is injective and thus surjective.

Reed-Solomon Codes. One of the most widely used MDS codes is the *Reed-Solomon code*. Roughly, it corresponds to evaluations of polynomials. More precisely, given an (ordered) set $E = \{e_1, \ldots, e_n\} \subseteq \mathbb{F}$ of size n, the Reed-Solomon code for message length k works as follows. To encode a given message $\mathbf{m} \in \mathbb{F}^k$, interpret $\mathbf{m}$ as a degree $k-1$ polynomial f over $\mathbb{F}$. This can be done in various ways. For example, if a systematic encoding is needed, one can interpolate f such that it satisfies $f(e_i) = \mathbf{m}_i$ for all $i \in [k]$. Next, f is evaluated at all points in E, leading to the codeword $\mathbf{c} = (f(e_1), \ldots, f(e_n))^\top$. As said, Reed-Solomon codes are MDS codes, meaning that their minimum distance is $n-k+1$. Throughout this work, we will denote the Reed-Solomon code as defined above by $\mathcal{RS}[k, n, \mathbb{F}]$, where we leave the set E implicit.

Interleaved Codes. Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code with alphabets Γ, Λ, message length k, code length n, and reception efficiency t. Given $\mathcal{C}$, we construct a new code $\mathcal{C}^{\equiv \ell}\colon \Gamma^{\ell k} \to \Lambda'^n$ as follows, where $\Lambda' := \Lambda^\ell$. To encode a message $m \in \Gamma^{\ell k}$, write it as $m = (m^{(1)}, \ldots, m^{(\ell)})$, where $m^{(i)} \in \Gamma^k$ for each $i \in [\ell]$. Then, for each $i \in [\ell]$, compute $\hat{m}^{(i)} := \mathcal{C}(m^{(i)})$. Now, for each $j \in [n]$, the jth symbol of the codeword $\hat{m}$ is $\hat{m}_j := (\hat{m}_j^{(1)}, \ldots, \hat{m}_j^{(\ell)})$. It is easy to see that if $\mathcal{C}$ has reception efficiency t and minimum distance d, then $\mathcal{C}^{\equiv \ell}$ also has reception efficiency t and minimum distance d. The code $\mathcal{C}^{\equiv \ell}$ that we just constructed is sometimes called *interleaved code*. We note that sometimes the interleaved code is defined with codewords of length ℓn over alphabet Λ. For us, it will be better to treat the codeword as a string of length n over alphabet Λ^ℓ. This is also done in [CDD+16].

Linear Interleaved Codes. Starting with a linear erasure code $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$, the interleaved code $\mathcal{C}^{\equiv \ell}\colon \mathbb{F}^{\ell k} \to (\mathbb{F}^\ell)^n$ can be written in a more concise way. For that, let $\mathbf{G} \in \mathbb{F}^{n \times k}$ be the generator matrix of $\mathcal{C}$. To encode a message $\mathbf{m} \in \mathbb{F}^{\ell k}$ using $\mathcal{C}^{\equiv \ell}$, $\mathbf{m}$ is first written as a matrix $\mathbf{M} \in \mathbb{F}^{\ell \times k}$ in an arbitrary canonical way. Then, each row is encoded with $\mathbf{G}$. That is, we compute $\mathbf{X} := \mathbf{M}\mathbf{G}^\top \in \mathbb{F}^{\ell \times n}$. Finally, the columns of $\mathbf{X}$ are interpreted as the symbols of the resulting codeword.

Tensor Codes. Given two codes $\mathcal{C}_r$ and $\mathcal{C}_c$, with message lengths k_r, k_c, respectively, one can write the message as a $k_c \times k_r$ matrix. Then, one can encode the message by encoding rows with $\mathcal{C}_r$ and columns with $\mathcal{C}_r$. This is called the *tensor code*[8] of $\mathcal{C}_r$ and $\mathcal{C}_c$. More concretely, assume two linear erasure codes $\mathcal{C}_r\colon \mathbb{F}^{k_r} \to \mathbb{F}^{n_r}$ and $\mathcal{C}_c\colon \mathbb{F}^{k_c} \to \mathbb{F}^{n_c}$ over the same field $\mathbb{F}$. Let t_r, t_c denote their respective reception efficiencies, and $\mathbf{G}_r \in \mathbb{F}^{n_r \times k_r}$ and $\mathbf{G}_c \in \mathbb{F}^{n_c \times k_c}$ denote their respective generator matrices. The tensor code of $\mathcal{C}_r$ and $\mathcal{C}_c$ is $\mathcal{C}_r \otimes \mathcal{C}_c\colon \mathbb{F}^{k_r \cdot k_c} \to \mathbb{F}^{n_r \cdot n_c}$, which works as follows. To encode a message $\mathbf{m} \in \mathbb{F}^{k_r \cdot k_c}$, write $\mathbf{m}$ as a matrix $\mathbf{M} \in \mathbb{F}^{k_c \times k_r}$ in some fixed canonical way. Then, compute $\mathbf{X} := \mathbf{G}_c\mathbf{M}\mathbf{G}_r^\top \in \mathbb{F}^{n_c \times n_r}$. Finally, flatten $\mathbf{X}$ into a vector $\mathbf{x} \in \mathbb{F}^{n_c n_r}$, which is the codeword. To ease notation, for each $j' \in [n_c n_r]$, we will write $(i, j) := \mathsf{ToMatIdx}(j')$ to indicate the unique pair of indices $i \in [n_c], j \in [n_r]$ such that $\mathbf{x}_{j'} = \mathbf{X}_{i,j}$. One can show that the tensor code is also a linear code. Next, we give a bound on the reception efficiency of the tensor code. In Appendix D, we show that the reception efficiency of $\mathcal{C}_r \otimes \mathcal{C}_c$ as above is $n_c n_r - (n_c - t_c + 1)(n_r - t_r + 1) + 1$. For instance, consider a code $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^{2k}$ with reception efficiency k. Then, the reception efficiency of $\mathcal{C} \otimes \mathcal{C}$ is $3k^2 - 2k$.

6 From Codes and Commitments to Data Availability Sampling

In this section, we show how to generically construct a data availability sampling scheme from a special class of commitments for codes. Namely, we abstract existing constructions that use vector commitments, and polynomial commitments, or similar structured commitments. First, we formally define the commitments that we consider. In a second step, we introduce and analyze index samplers as the combinatorial core component of the final data availability sampling scheme. Third, we present and analyze our generic construction of data availability sampling from any such commitment and any such index sampler.

[8] Sometimes the tensor code is also called the product code.

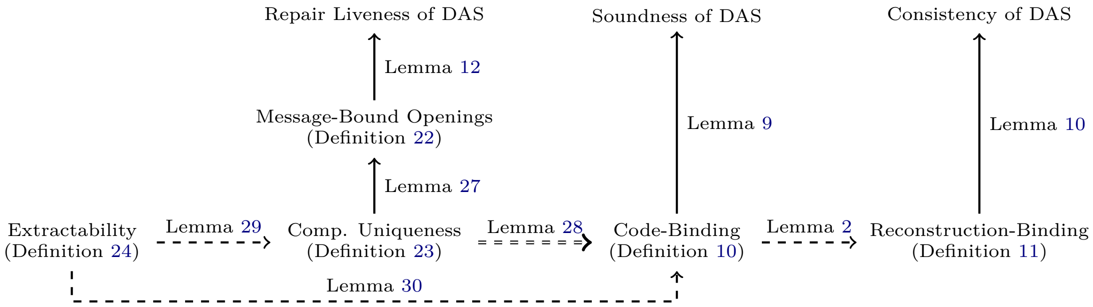

Figure 2: Overview of the different security properties we define for erasure code commitments, how they relate to each other, and how they relate to the security of the resulting data availability sampling scheme. An arrow denotes an implication. A dashed arrow denotes an implication that holds if additionally position-binding is assumed. For the implication from computational uniqueness to code-binding (double dashed), we additionally assume position-binding and that the code is an MDS code.

6.1 Erasure Code Commitments

We introduce erasure code commitments as a generalization of vector commitments and polynomial commitments. Roughly, we can use such a commitment to commit to a codeword of an erasure code. One can view a vector commitment as an erasure code commitment for the identity code, and polynomial commitment as an erasure code commitment for the Reed-Solomon code.

Syntax. The next definition introduces the syntax of erasure code commitments.

Definition 8 (Erasure Code Commitment Scheme)**.** Consider an erasure code $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ with alphabets Γ, Λ, message length k, code length n, reception efficiency t, and reconstruction algorithm $\mathsf{Reconst}$. An erasure code commitment scheme for $\mathcal{C}$ with opening alphabet Ξ is a tuple $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ of PPT algorithms, with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ takes as input the security parameter and outputs a commitment key ck.
- $\mathsf{Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$ takes as input a commitment key ck and a string $m \in \Gamma^k$, and outputs a commitment com and a state St.
- $\mathsf{Open}(\mathsf{ck}, St, i) \to \tau$ takes as input a commitment key ck, a state St, and an index $i \in [n]$, and outputs an opening $\tau \in \Xi$.
- $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau) \to b$ is deterministic, takes as input a commitment key ck, a commitment com, and index $i \in [n]$, a symbol $\hat{m}_i \in \Lambda$, and an opening $\tau \in \Xi$, 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 $i \in [n]$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau) = 1 \;\middle|\; \begin{array}{l} (\mathsf{com}, St) \leftarrow \mathsf{Com}(\mathsf{ck}, m), \\ \hat{m} := \mathcal{C}(m), \\ \tau \leftarrow \mathsf{Open}(\mathsf{ck}, St, i) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Now that we have specified the syntax of erasure code commitment schemes, we turn to the security properties they should have. We define a variety of such properties, most importantliy position-binding and code-binding. Later, we will see how these properties imply the security of the resulting data availability sampling scheme. We summarize the relations between these properties in Figure 2.

Binding Notions. The first notion we define is *position-binding*, which is analogous to the position-binding notion for vector commitments. The intuition of position-binding is that no efficient adversary can open a commitment to two different values at the same position.

Definition 9 (Position-Binding of CC)**.** Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ 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:

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

Requiring only position-binding, we could easily implement an erasure code commitment by committing to a codeword using a standard vector commitment. However, one should only be able commit to codewords. For that, we define *code-binding*. Roughly, it requires that an adversary can not open a commitment on a set of positions in a way that is inconsistent with the code.

Definition 10 (Code-Binding of CC)**.** Let $\mathsf{CC} = (\mathsf{Setup},\mathsf{Com},\mathsf{Open},\mathsf{Ver})$ 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:

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) := \Pr\left[\begin{array}{l} \neg\left(\exists c \in \mathcal{C}(\Gamma^k) : \forall i \in I : c_i = \hat{m}_i\right) \\ \wedge\ \forall i \in I :\ \mathsf{Ver}(\mathsf{ck},\mathsf{com},i,\hat{m}_i,\tau_i) = 1 \end{array}\middle|\begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{com},(\hat{m}_i,\tau_i)_{i\in I}) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right].$$

We introduce a third binding notion called *reconstruction-binding*. When we want to use erasure code commitments in the context of data availability sampling schemes, reconstruction-binding, as defined next, is a natural requirement. Namely, it will ensure that extracting from two sets of transcripts leads to consistent results. In other words, reconstruction-binding states that one can not provide two sets of openings for the same commitment, such that reconstructing from these sets leads to inconsistent messages. After giving the formal definition of reconstruction-binding, we show that it is implied by position-binding and code-binding. Later, we show that it implies the consistency property of our data availability sampling scheme.

Definition 11 (Reconstruction-Binding of CC)**.** Let $\mathsf{CC} = (\mathsf{Setup},\mathsf{Com},\mathsf{Open},\mathsf{Ver})$ be an erasure code commitment scheme for an erasure code $\mathcal{C}$ with reception efficiency t and reconstruction algorithm $\mathsf{Reconst}$. We say that CC is reconstruction-binding, if for every PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{rec\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) := \Pr\left[\begin{array}{ll} & |I| \geq t \wedge |I'| \geq t \wedge \bot \notin \{m,m'\} \\ \wedge & \forall i \in I : \mathsf{Ver}(\mathsf{ck},\mathsf{com},i,\hat{m}_i,\tau_i) = 1 \\ \wedge & \forall i \in I' : \mathsf{Ver}(\mathsf{ck},\mathsf{com},i,\hat{m}'_i,\tau'_i) = 1 \\ \wedge & m \neq m' \end{array}\middle|\begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{com},(\hat{m}_i,\tau_i)_{i\in I},(\hat{m}'_i,\tau'_i)_{i\in I'}) \\ \qquad\qquad \leftarrow \mathcal{A}(\mathsf{ck}), \\ m := \mathsf{Reconst}((\hat{m}_i)_{i\in I}), \\ m' := \mathsf{Reconst}((\hat{m}'_i)_{i\in I'}) \end{array}\right].$$

Lemma 2. *Let* $\mathsf{CC} = (\mathsf{Setup},\mathsf{Com},\mathsf{Open},\mathsf{Ver})$ *be an erasure code commitment scheme for an erasure code* $\mathcal{C}$*. If* CC *is position-binding and code-binding, then* CC *is reconstruction-binding. Precisely, for any PPT algorithm* $\mathcal{A}$*, there are PPT algorithms* $\mathcal{B}_1,\mathcal{B}_2$ *with* $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$*,* $\mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$*, and*

$$\mathsf{Adv}^{\mathsf{rec\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_1,\mathsf{CC}}(\lambda) + \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B}_2,\mathsf{CC}}(\lambda).$$

The proof of Lemma 2 is given in Appendix F.1.

Other Notions. We introduce further security notions for erasure code commitments. As indicated by Figure 2, these notions are not directly necessary if we want to construct data availability sampling schemes satisfying our basic definition in Section 3.1. However, they turn out to be useful for two reasons. First, these notions are necessary if we want to construct repairable data availability sampling schemes. Second, some of these notions are stronger than others and help us to avoid repeating parts of our analysis.

The first of these additional notions is a strong notion called *extractability*. Intuitively, a (deterministic) erasure code commitment is extractable, if there is an efficient algorithm Ext that can extract a message m from any commitment com output by an adversary, as long as the adversary provides at least one opening. When committing to the extracted message m, one obtains com. This typically requires the use of the algebraic group model. We formally define the notion of extractability and study it in Appendix E.3.

A second property we consider is called *message-bound openings*. This property turns out to be useful for repairability. Intuitively, we want to repair an encoding from a set of transcripts by first reconstructing the data, and then re-encoding this data. The challenge is that the new encoding has to be compatible with the old commitment that an adversary made up. Our notion of message-bound openings ensures this. Namely, it requires that it is hard for an adversary to come up with two commitments for the same message and enough valid openings that can not be arbitrarily "mixed-and-matched". We postpone the formal definition to Appendix E.1.

A final notion we introduce and study is *computational uniqueness*. The notion is almost as the notion of message-bound openings, but just requires the adversary to output two distinct commitments. In other words, if a scheme is computationally unique, it means that whenever an adversary can open two commitments to codewords that reconstruct to the same message, then the commitments are the same. In Appendix E.2, we give the formal definition and show that this notion is strong enough to imply both code-binding and message-bound openings.

Simple Examples. Before finishing this section, we mention simple examples of erasure code commitments. These examples also shed light on how erasure code commitments relate to other commitment schemes.

Example 3 (Vector Commitments). We can view any vector commitment [Mer88, CF13] for vectors in Γ^k as being an erasure code commitment for the code $\mathcal{C}\colon \Gamma^k \to \Gamma^k$ with $\mathbf{x} \mapsto \mathbf{x}$ for all $\mathbf{x} \in \Gamma^k$. In this case, code-binding holds trivially and position-binding is equivalent to the definition of position-binding for vector commitments.

Example 4 (Polynomial Commitments). Polynomial commitments [KZG10] are a special case of erasure code commitments for the Reed-Solomon code. Our notion of position-binding matches the definition of position-binding for polynomial commitment schemes in [KZG10]. Interestingly, [KZG10] does not define a notion matching code-binding. That is, there is no notion in [KZG10] stating that an adversary can not open a commitment to points which are not on a polynomial of appropriate degree. It is easy to see that the KZG polynomial commitment scheme [KZG10] satisfies this notion. For that, it is sufficient to observe that it is extractable in the algebraic group model [FKL18], see Appendix E.3.

6.2 Index Samplers

Our goal is to construct a data availability sampling scheme from any erasure code commitment scheme. The high level idea is that clients query and verify some positions of the encoding. Every such position contains a symbol of a codeword and its corresponding opening for the erasure code commitment. Now, a natural question is how clients sample the indices that they query. We abstract the strategy that the clients use by defining so called index samplers. An index sampler is just an algorithm that outputs Q indices in some range $[N]$. An example of an index sampler is given by sampling uniformly with replacement, i.e., the index sampler outputs Q indices sampled uniformly at random from $[N]$. Intuitively, different index samplers may lead to different guarantees for the resulting data availability sampling scheme. For example, an index sampler is a good choice if only a few clients with a few samples are needed to guarantee that at least a certain number of distinct indices (i.e., symbols of the codeword) from $[N]$ are touched, and thus data can be reconstructed. We make this intuition formal by defining the quality of an index sampler. This measure will translate to the soundness and completeness error of the resulting data availability sampling scheme.

Definition 12 (Index Sampler). An index sampler with quality $\nu\colon \mathbb{N}^4 \to \mathbb{R}$ is a PPT algorithm Sample with the following syntax and properties:

- $\mathsf{Sample}(1^Q, 1^N) \to (i_j)_{j \in [Q]}$ takes as input integers $Q, N \in \mathbb{N}$ and outputs Q indices $i_j \in [N]$.
- For any $N, \Delta \in \mathbb{N}$ with $\Delta < N$, and any $Q, \ell \in \mathbb{N}$, we have

$$\Pr_{\mathcal{G}}\left[\left| \bigcup_{l \in [\ell]} \{ i_{l,j} \mid j \in [Q] \} \right| \leq \Delta \right] \leq \nu(\Delta, N, Q, \ell),$$

  where experiment $\mathcal{G}$ is given by running $(i_{l,j})_{j \in [Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$ for each $l \in [\ell]$.

In the context of data availability sampling schemes, the encoding may be distributed over many physical nodes. Ideally, indices are sampled in a way that minimizes the number of nodes a client has to query. For that reason, we define a locality measure for index samplers. It is defined as the number of physical nodes that the index sampler touches.

Definition 13 (Locality of Index Samplers)**.** Let Sample be an index sampler, $Q, N, D \in \mathbb{N}, \epsilon \in [0,1]$ with $D \leq Q$, and $\mathcal{S}\colon [N] \to \mathbb{N}$ be a function. We say that Sample is $(Q, N, \mathcal{S}, D, \epsilon)$-local, if

$$\Pr_{\mathcal{G}}\left[|\{\mathcal{S}(i_j) \mid j \in [Q]\}| > D\right] \leq \epsilon,$$

where $\mathcal{G}$ is given by running $(i_j)_{j\in[Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$.

Of course, every index sampler has optimal locality (i.e. $\epsilon = 0, \Sigma = 1$) if the function $\mathcal{S}$ is constant, i.e., the entire encoding is stored on one physical node. A more natural function $\mathcal{S}$ would be $\mathcal{S}(x) = \lfloor (x-1)/Q \rfloor$, i.e., each node stores a contiguous part of the encoding of equal size.

Next, we discuss three examples of index samplers. Namely, we consider natural index samplers that sample all indices uniformly at random, either with replacement or without replacement. Finally, we also introduce an index sampler that is optimized in terms of locality.

Sampling With Replacement. Sampling with replacement is given via the following algorithm.

- $\mathsf{Sample}_{\mathsf{wr}}(1^Q, 1^N)$: For each $j \in [Q]$, sample $i_j \leftarrow_{\$} [N]$. Return $(i_j)_{j\in[Q]}$.

We analyze the quality of algorithm $\mathsf{Sample}_{\mathsf{wr}}$ that samples indices uniformly at random with replacement.

Lemma 3. *Algorithm* $\mathsf{Sample}_{\mathsf{wr}}$ *is an index sampler with quality* $\nu_{\mathsf{wr}} : \mathbb{N}^4 \to \mathbb{R}$*, where*

$$\nu_{\mathsf{wr}}(\Delta, N, Q, \ell) = \binom{N}{\Delta}\left(\frac{\Delta}{N}\right)^{Q\ell}.$$

In particular, $\mathsf{Sample}_{\mathsf{wr}}$ *is an index sampler with quality* $\nu'_{\mathsf{wr}} : \mathbb{N}^4 \to \mathbb{R}$*, where*

$$\nu'_{\mathsf{wr}}(\Delta, N, Q, \ell) = c^{Q\ell - (1-\log_c(e))\Delta} \textit{ for } c := \Delta/N.$$

The proof of Lemma 3 is given in Appendix F.2. In Section 6.3, we will see that the quality $\nu(\Delta, N, Q, \ell)$ of an index sampler corresponds to the advantage of an adversary against soundness of the resulting data availability sampling scheme. Namely, if our data consists of K symbols, and our encoding consists of N symbols, such that any $\Delta + 1$ are sufficient to reconstruct the data, then we need to choose Q and ℓ such that $\nu(\Delta, N, Q, \ell)$ is negligible in the security parameter λ. To get an intuition for the bound that Lemma 3 provides, let us consider two examples.

Example 5 (Trivial Encoding). Assume that we do not use any erasure code at all, or in other words, we use the identity function as an erasure code. In this case, we have $K = N$ and $\Delta = K - 1$, because we need all symbols to reconstruct the data. Using the first bound in Lemma 3, we can upper bound the advantage against soundness by

$$\binom{N}{N-1}\left(1 - \frac{1}{N}\right)^{Q\ell} = N\left(1 - \frac{1}{N}\right)^{N\frac{Q\ell}{N}} \leq 2^{\log N} e^{-\frac{Q\ell}{N}} = 2^{\log N - \log e \frac{Q\ell}{N}}.$$

This bound is negligible once we set

$$Q\ell \geq \Omega(N\lambda + N\log N) = \Omega(K\lambda + K\log K).$$

Example 6 (Using Erasure Codes). Assume that we encode the K symbols of data with an erasure code into $N = 2K$ symbols, such that any K of these are sufficient to reconstruct the data. For example, we could use a Reed-Solomon code and a polynomial commitment to realize this. We can now use the second bound in Lemma 3 with $c = \Delta/N < 1/2$. This yields an upper bound on the advantage against soundness of $2^{-Q\ell + (1-\log_{1/2} e)(K-1)} \leq 2^{-Q\ell + 3K}$. The bound is negligible once we set

$$Q\ell \geq \Omega(K + \lambda).$$

Now, let us consider the notion of subset-soundness instead. In Lemma 1, we showed that soundness implies (L,ℓ)-subset-soundness, with a security loss of at most $(Le/\ell)^{\ell}$. Assuming $L = C \cdot \ell$ for some constant $C > 1$, we get an upper bound on the advantage against (L,ℓ)-subset-soundness of $2^{\ell(\log C + \log e) - Q\ell + 3K}$. If $Q \geq \log C + \log e + 1$, this bound is also negligible once we set $Q\ell \geq \Omega(K + \lambda)$.

The two examples demonstrate that using an erasure code results in a significant improvement in terms of the number of samples we need to reconstruct the data with overwhelming probability. Additionally, the second example demonstrates a significant difference between soundness and subset-soundness. Namely, while having ℓ clients with Q queries per client is equivalent to having 1 client with ℓQ queries and to having ℓQ clients with 1 query in terms of soundness, these three settings are not equivalent in terms of subset-soundness. Especially, to get subset-soundness, we have to set Q large enough. Intuitively, this is because the number of transcripts from which the adversary can choose differs in the three settings.

Next, we want to understand the locality of algorithm $\mathsf{Sample}_{\mathsf{wr}}$. Intuitively, sampling indices uniformly at random should lead to a bad locality. The next lemma states exactly that, especially when N or $Q - D$ is large.

Lemma 4. *Let $Q, N, D \in \mathbb{N}, \epsilon \in [0,1]$ with $D \leq Q$, and $\mathcal{S}\colon [N] \to \mathbb{N}$ be a Q-to-1 function mapping onto a set of size N/Q. Then, if $\mathsf{Sample}_{\mathsf{wr}}$ is $(Q, N, \mathcal{S}, D, \epsilon)$-local, then*

$$\epsilon > 1 - e^{D} \cdot \left(\frac{D}{N/Q}\right)^{Q-D}.$$

The proof of Lemma 4 is given in Appendix F.2.

Sampling Without Replacement. Sampling without replacement is given by the following algorithm.

- $\mathsf{Sample}_{\mathsf{wor}}(1^Q, 1^N)$: For each $j \in [Q]$, sample $i_j \leftarrow\!\!\$\, [N] \setminus \{i_1, \ldots, i_{j-1}\}$. Return $(i_j)_{j \in [Q]}$.

We analyze the quality of algorithm $\mathsf{Sample}_{\mathsf{wor}}$ in the following lemma.

Lemma 5. *Algorithm $\mathsf{Sample}_{\mathsf{wor}}$ is an index sampler with quality $\nu_{\mathsf{wor}} : \mathbb{N}^4 \to \mathbb{R}$, where*

$$\nu_{\mathsf{wor}}(\Delta, N, Q, \ell) = \binom{N}{\Delta}\left(\binom{\Delta}{Q}\Big/\binom{N}{Q}\right)^{\ell}.$$

The proof of Lemma 5 is given in Appendix F.2.

Segment Sampling. We introduce a third index sampler. The idea is to partition the set $[N]$ into N/Q segments of size Q. Then, the sampler picks one of the segments at random and queries this entire segment. The advantage is minimal randomness complexity and the locality of the sampled indices. We define algorithm $\mathsf{Sample}_{\mathsf{seg}}$ as follows:

- $\mathsf{Sample}_{\mathsf{seg}}(1^Q, 1^N)$: If $N \mod Q \neq 0$, return $(i_j)_{j \in [Q]} \leftarrow \mathsf{Sample}_{\mathsf{wr}}(1^Q, 1^N)$. Otherwise, sample $\mathsf{seg} \leftarrow\!\!\$\, [N/Q]$. For each $j \in [Q]$, set $i_j := (\mathsf{seq} - 1)\,Q + j$. Return $(i_j)_{j \in [Q]}$.

Next, we analyze the quality and locality of $\mathsf{Sample}_{\mathsf{seg}}$. Intuitively, the analysis of $\mathsf{Sample}_{\mathsf{seg}}$ reduces to an analysis of $\mathsf{Sample}_{\mathsf{wr}}$ over the segments.

Lemma 6. *Assuming algorithm $\mathsf{Sample}_{\mathsf{wr}}$ is an index sampler with quality $\nu_{\mathsf{wr}} : \mathbb{N}^4 \to \mathbb{R}$, the algorithm $\mathsf{Sample}_{\mathsf{seg}}$ is an index sampler with quality $\nu_{\mathsf{seg}} : \mathbb{N}^4 \to \mathbb{R}$, where*

$$\nu_{\mathsf{seg}}(\Delta, N, Q, \ell) = \begin{cases} \nu_{\mathsf{wr}}(\Delta, N, Q, \ell) & \text{if } N \mod Q \neq 0 \\ \nu_{\mathsf{wr}}(\Delta/Q, N/Q, 1, \ell) & \text{if } N \mod Q = 0 \end{cases}.$$

In particular, $\mathsf{Sample}_{\mathsf{seg}}$ is an index sampler with quality $\nu'_{\mathsf{seg}} : \mathbb{N}^4 \to \mathbb{R}$, where

$$\nu'_{\mathsf{seg}}(\Delta, N, Q, \ell) = \begin{cases} c^{Q\ell - (1 - \log_c(e))\Delta} & \text{if } N \mod Q \neq 0 \\ c^{\ell - (1 - \log_c(e))\Delta/Q} & \text{if } N \mod Q = 0 \end{cases}$$

for $c := \Delta/N$.

The proof of Lemma 6 is given in Appendix F.2.

Lemma 7. *Let $Q, N \in \mathbb{N}$ be such that Q divides N. Consider $\mathcal{S}\colon [N] \to \mathbb{N}$ with $\mathcal{S}(x) = \lfloor (x-1)/Q \rfloor$. Then,* $\mathsf{Sample}_{\mathsf{seg}}$ *is $(Q, N, \mathcal{S}, 1, 0)$-local.*

Lemma 7 follows trivially by inspection.

Simulation. The analytical results in this section heavily rely on the use of probabilistic bounds, e.g., the union bound or the Chernoff bound. One may ask whether more precise results can be obtained by other means. To this end, we simulated the three index samplers discussed in this section and compared their quality. We present and discuss our results in Appendix J.

6.3 Construction of Data Availability Sampling Schemes

Now that we have introduced erasure code commitments and index samplers, we come to the main construction of this section. Namely, we show how to construct a data availability sampling scheme from any erasure code commitment scheme and any index sampler. If the erasure code has a generalized systematic encoding, the resulting data availability sampling scheme is locally accessible with optimal query complexity $L = 1$.

Overview. We start with an erasure code $\mathcal{C}$ with reception efficiency t and an erasure code commitment scheme CC for it. In the data availability sampling scheme, a proposer encodes the data data by first applying the code $\mathcal{C}$ to it to get a codeword $\widehat{\mathsf{data}} = \mathcal{C}(\mathsf{data})$. For consistency, the proposer commits to this codeword using CC. The resulting commitment com will be given to the clients. In addition to that, the proposer computes openings τ_i for all positions i of the codeword. Then, each symbol $\widehat{\mathsf{data}}_i$ together with its opening τ_i forms a symbol π_i of the encoding π. Clients are defined in the following way: First, they determine some set of indices $i_1, \dots, i_Q$ using an index sampler and query these indices, getting $(\widehat{\mathsf{data}}_{i_j}, \tau_{i_j})$ as responses. Then, they verify all openings with respect to com, and accept if and only if they are all valid. To extract data from a given set of transcripts, we first check that all transcripts are accepting, and that they contain at least t distinct positions of π. If this holds, then we have at least t distinct positions of the codeword $\widehat{\mathsf{data}}$ and can reconstruct the data.

Construction. Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code with alphabets Γ, Λ, message length k, code length n, and reception efficiency t, with reconstruction algorithm $\mathsf{Reconst}$. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ with opening alphabet Ξ. Further, let Sample be an index sampler with quality ν. We construct a data availability sampling scheme $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}] = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V}, \mathsf{Ext})$ with data length $K := k$, encoding length $N := n$, data alphabet Γ, encoding alphabet $\Sigma = \Lambda \times \Xi$, query complexity $Q \in \mathbb{N}$, and threshold $T \in \mathbb{N}$. We emphasize that T and Q have to be chosen appropriately and depend on n, t, and the quality ν of Sample. We refer to our analysis for a concrete bound. The construction is as follows.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$: Run $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and set $\mathsf{par} := \mathsf{ck}$.
- $\mathsf{Encode}(\mathsf{data}) \to (\pi, \mathsf{com})$:
  1. Run $(\mathsf{com}, St) := \mathsf{Com}(\mathsf{ck}, \mathsf{data}; \rho)$ for some hardcoded coins ρ.
  2. Compute $\widehat{\mathsf{data}} := \mathcal{C}(\mathsf{data})$.
  3. For each $i \in [N]$, run $\tau_i \leftarrow \mathsf{Open}(\mathsf{ck}, St, i)$, and set $\pi_i := (\widehat{\mathsf{data}}_i, \tau_i)$.
- $\mathsf{V}_1^{\pi,Q}(\mathsf{com}) \to \mathsf{tran}$: Run $(i_j)_{j\in[Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$ and query $(\widehat{\mathsf{data}}_{i_j}, \tau_{i_j}) := \pi_{i_j}$ for each $j \in [Q]$. Set $\mathsf{tran} := (i_j, \widehat{\mathsf{data}}_{i_j}, \tau_{i_j})_{j\in[Q]}$.
- $\mathsf{V}_2(\mathsf{com}, \mathsf{tran}) \to b$: If there is a $j \in [Q]$ with $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i_j, \widehat{\mathsf{data}}_{i_j}, \tau_{i_j}) = 0$, then return $b := 0$. Otherwise, return $b := 1$.
- $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \dots, \mathsf{tran}_L) \to \mathsf{data}/\bot$:
  1. Write $\mathsf{tran}_l := (i_{l,j}, \widehat{\mathsf{data}}_{l,i_{l,j}}, \tau_{l,i_{l,j}})_{j\in[Q]}$ for each $l \in [L]$.
  2. If there is an $l \in [L]$ such that $\mathsf{V}_2(\mathsf{com}, \mathsf{tran}_l) = 0$, return $\bot$.

3. Let $I \subseteq [N]$ be the set of indices $i \in [N]$ such that there is a $(l, j) \in [L] \times [Q]$ with $i_{l,j} = i$. If $|I| < t$, then return $\perp$.
4. Otherwise, for each $i \in I$, pick an arbitrary such $(l, j) \in [L] \times [Q]$ with $i_{l,j} = i$ and set $\widehat{\mathsf{data}}_i := \widehat{\mathsf{data}}_{l,i_{l,j}}$.
5. Return $\mathsf{data} := \mathsf{Reconst}((\widehat{\mathsf{data}}_i)_{i \in I})$.

Analysis. Next, we analyze the construction given above. Namely, we show completeness, soundness, and consistency. For analyzing completeness and soundness, we rely on the quality of the index sampler in combination with the reception efficiency of $\mathcal{C}$. Namely, reception efficiency tells us how many of the N indices we need to recover the data. Then, the quality of the index sampler gives a bound on the probability that we did not collect enough indices when we have a certain number of clients with a certain number of queries. This determines the threshold of the scheme, i.e., the number of clients needed to make the completeness and soundness error negligible. For soundness, we additionally need to rule out the case that we collected enough indices, but the responses of the adversary at these indices are not consistent with the code. For that, we can use code-binding of the commitment scheme. To show consistency, we rely on reconstruction-binding of the commitment scheme.

Lemma 8. *The scheme* $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ *satisfies completeness, if*

$$\nu(\Delta, N, Q, T) \leq \mathsf{negl}(\lambda), \text{ where } \Delta := t - 1.$$

Proof. Let $\ell = \mathsf{poly}(\lambda)$, $\ell \geq T$ and $\mathsf{data} \in \Gamma^K$ as in the definition of completeness. First, by the completeness of CC, we know that the ℓ copies of V_2 output 1 in the completeness experiment, except with negligible probability. Thus, it remains to bound the probability of the bad event that Ext outputs $\perp$ because not enough indices are covered, i.e. the set $I \subseteq [N]$ of indices $i \in [N]$ such that there is a $(l, j) \in [L] \times [Q]$ with $i_{l,j} = i$ has size strictly less than t. This is equivalent to saying it has size at most Δ. Clearly, if we only consider the first T instead of all $\ell \geq T$ transcripts, the size of this set can not increase. As the indices are sampled using Sample, one can easily verify that the probability of the bad event is at most the probability that

$$\left| \bigcup_{l \in [T]} \{ i_{l,j} \mid j \in [Q] \} \right| \leq \Delta,$$

where $(i_{l,j})_{j \in [Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$ for all $l \in [T]$. By definition of the quality of Sample, this is at most $\nu(\Delta, N, Q, T)$. □

Lemma 9. *Assume that* CC *is code-binding and* $\nu(\Delta, N, Q, T)$ *is negligible for* $\Delta := t - 1$. *Then, the scheme* $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ *satisfies soundness. Concretely, for any PPT algorithm* $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *such that for any* $\ell \geq T$ *we have*

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A},\ell,\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]}(\lambda) \leq \nu(\Delta, N, Q, T) + \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B},\mathsf{CC}}(\lambda).$$

Proof. Consider an adversary $\mathcal{A}$ against soundness of $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$. We first recall the soundness game and introduce some notation. First, parameters $\mathsf{par} := \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ are sampled and given to $\mathcal{A}$. Then, $\mathcal{A}$ outputs a commitment com. Then, ℓ copies of V_1 are run and their oracle queries are answered by $\mathcal{A}$. Let $\mathsf{tran}_l = (i_{l,j}, \widehat{\mathsf{data}}_{l,i_{l,j}}, \tau_{l,i_{l,j}})_{j \in [Q]}$ for $l \in [\ell]$ be the respective transcripts. The adversary $\mathcal{A}$ breaks soundness if all of these verify, i.e., for all $l \in [\ell]$ and all $j \in [Q]$ we have $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i_{l,j}, \widehat{\mathsf{data}}_{l,i_{l,j}}, \tau_{l,i_{l,j}}) = 1$, and $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell)$ outputs $\perp$. Recall that Ext outputs $\perp$ either because a transcript does not verify, or the set $I \subseteq [N]$ of covered indices is not large enough, i.e., $|I| < t$, or if algorithm $\mathsf{Reconst}$ outputs $\perp$. We analyze the game by considering these cases separately. Namely, we define the following events.

- Event $\mathsf{InvalidTrans}$: This event occurs, if $\mathcal{A}$ breaks soundness and Ext outputs $\perp$ because a transcript does not verify.

- Event $\mathsf{NotEnough}$: This event occurs, if $\mathcal{A}$ breaks soundness and Ext outputs $\perp$ because $|I| < t$.
- Event $\mathsf{Inconsistent}$: This event occurs, if $\mathcal{A}$ breaks soundness and Ext outputs $\perp$ because algorithm $\mathsf{Reconst}$ outputs $\perp$.

It is clear that

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A},\ell,\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]}(\lambda) \leq \Pr[\mathsf{InvalidTrans}] + \Pr[\mathsf{NotEnough}] + \Pr[\mathsf{Inconsistent}].$$

We bound these three terms separately. First, it is clear that event $\mathsf{InvalidTrans}$ can not occur. This is because if one transcript does not verify, $\mathcal{A}$ never wins by definition. Second, if all copies of V_2 output 1, i.e. all transcripts are accepting, we can argue exactly as in the analysis of completeness. That is, using the quality of Sample, we rule out that not enough indices are covered and Ext outputs $\perp$. We get that the probability of $\mathsf{NotEnough}$ is at most $\nu(\Delta, N, Q, T)$. Finally, we have to bound the probability of $\mathsf{Inconsistent}$. Recall that algorithm $\mathsf{Reconst}$ outputs $\perp$ if either not enough symbols are input, or if its input is not consistent with any codeword. The first case can not happen, as in this case Ext would have output $\perp$ because of $|I| < t$ and $\mathsf{Reconst}$ would not have been run. The second case easily reduces to code-binding. Namely, a reduction $\mathcal{B}$ can run $\mathcal{A}$ in the soundness game while forwarding its input ck to $\mathcal{A}$. Then, if $\mathsf{Inconsistent}$ occurs, $\mathcal{B}$ knows valid openings that are not consistent with a codeword, and can output these openings to break codebinding. We get that

$$\Pr[\mathsf{Inconsistent}] \leq \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B},\mathsf{CC}}(\lambda).$$

□

Lemma 10. *If* CC *is reconstruction-binding, then* $\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]$ *satisfies consistency. Concretely, for any PPT algorithm* $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *such that for any* $\ell_1, \ell_2 = \mathsf{poly}(\lambda)$*, we have*

$$\mathsf{Adv}^{\mathsf{cons}}_{\mathcal{A},\ell_1,\ell_2,\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]} \leq \mathsf{Adv}^{\mathsf{rec\text{-}bind}}_{\mathcal{B},\mathsf{CC}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an algorithm running in the consistency game of $\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]$. We construct a reduction $\mathcal{B}$ that simulates the consistency game for $\mathcal{A}$ and breaks reconstruction-binding of CC if $\mathcal{A}$ breaks consistency. Namely, the reduction $\mathcal{B}$ gets as input a commitment key ck, sets $\mathsf{par} := \mathsf{ck}$, and runs $\mathcal{A}$ on input par as in the consistency game. Then, $\mathcal{A}$ outputs $(\mathsf{com}, (\mathsf{tran}_{1,i})_{i=1}^{\ell_1}, (\mathsf{tran}_{2,i})_{i=1}^{\ell_2})$. We use the notation $I_j, \widehat{\mathsf{data}}_{j,i}$ for the variables I, $\widehat{\mathsf{data}}_i$ as in Ext for the jth extraction, $j \in \{1,2\}$. The reduction $\mathcal{B}$ outputs $\mathsf{com}, \left(\widehat{\mathsf{data}}_{1,i}, \tau_{1,i}\right)_{i \in I_1}, \left(\widehat{\mathsf{data}}_{2,i}, \tau_{2,i}\right)_{i \in I_2}$. It remains to argue that $\mathcal{B}$ breaks reconstruction-binding, assuming that $\mathcal{A}$ breaks consistency. For that, assume both extractions $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{1,1}, \ldots, \mathsf{tran}_{1,\ell_1})$ and $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{2,1}, \ldots, \mathsf{tran}_{2,\ell_2})$ do not output $\perp$, and they output $\mathsf{data}_1 \neq \mathsf{data}_2$. As both extractions did not output $\perp$, the transcripts must contain valid openings $\tau_{1,i}$ such that $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \widehat{\mathsf{data}}_{1,i}, \tau_{1,i}) = 1$ for all $i \in I_1$, and $\tau_{2,i}$ such that $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \widehat{\mathsf{data}}_{2,i}, \tau_{2,i}) = 1$ for all $i \in I_2$. Also, it must hold that $|I_1| \geq t$ and $|I_2| \geq t$. This is by definition of algorithm Ext. In combination with $\mathsf{data}_1 \neq \mathsf{data}_2$, this implies that $\mathcal{B}$ breaks reconstruction-binding. □

Local Accessibility. Now, assume that $\mathcal{C}$ has a generalized systematic encoding with symbol projection algorithm Proj and symbol finding algorithm Find. Then, we show that our generic construction $\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}]$ is locally accessible with optimal query complexity $L = 1$. For that, we define algorithm Access as follows.

- $\mathsf{Access}^{\pi,L}(\mathsf{com}, i) \to d/\perp$:
  1. Compute $\hat{i} := \mathsf{Find}(i)$ and query $(\widehat{\mathsf{data}}_{\hat{i}}, \tau_{\hat{i}}) := \pi_{\hat{i}}$.
  2. If $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, \hat{i}, \widehat{\mathsf{data}}_{\hat{i}}, \tau_{\hat{i}}) = 0$, return $\perp$. Otherwise, return $\mathsf{Proj}(i, \widehat{\mathsf{data}}_{\hat{i}})$.

By the first part of the definition of a generalized systematic encoding and the completeness of CC, it is easy to see that local access completeness holds. We show that local access consistency holds.

Lemma 11. *Assume that* CC *is reconstruction-binding and* $\mathcal{C}$ *has a generalized systematic encoding. Then,* $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ *with algorithm* Access *satisfies local access consistency. Concretely, for any PPT algorithm* $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ *such that for any* $i_0 \in [K]$*, and any* $\ell = \mathsf{poly}(\lambda)$*, we have*

$$\mathsf{Adv}^{\mathsf{acc\text{-}cons}}_{\mathcal{A},i_0,\ell,\mathsf{DAS},\mathsf{Access}}(\lambda) \leq \mathsf{Adv}^{\mathsf{rec\text{-}bind}}_{\mathcal{B},\mathsf{CC}}(\lambda).$$

The proof of Lemma 11 is given in Appendix F.3.

Repairability. Now, assume that CC has message-bound openings. Then, we show that our generic construction $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ is (L, ℓ)-repairable, provided that it satisfies (L, ℓ)-subset-soundness. For that, we define algorithm Repair as follows.

- $\mathsf{Repair}(\mathsf{com}, \mathsf{tran}_1, \dots, \mathsf{tran}_\ell) \to \bar{\pi}/\bot$:
  1. Run $\overline{\mathsf{data}} := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \dots, \mathsf{tran}_\ell)$. If $\overline{\mathsf{data}} := \bot$, return $\bot$.
  2. Compute $(\bar{\pi}, \overline{\mathsf{com}}) := \mathsf{Encode}(\overline{\mathsf{data}})$ and return $\bar{\pi}$.

Lemma 12. *If* CC *has message-bound openings and* $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ *satisfies* (L, ℓ)*-subset-soundness, then* $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$ *is* (L, ℓ)*-repairable. Concretely, for any PPT algorithm* $\mathcal{A}$ *there are PPT algorithms* $\mathcal{B}_1, \mathcal{B}_2$ *with* $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$ *such that*

$$\mathsf{Adv}^{\mathsf{repairlive}}_{\mathcal{A},L,\ell,\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}],\mathsf{Repair}}(\lambda) \leq \mathsf{Adv}^{\mathsf{sub\text{-}sound}}_{\mathcal{B}_1,L,\ell,\mathsf{DAS}}(\lambda) + \mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{B}_2,\mathsf{CC}}(\lambda).$$

The proof of Lemma 12 is given in Appendix F.3.

7 Commitments for Arbitrary Codes

In this section, we show how to construct an erasure code commitment scheme for any erasure code from a vector commitment and a non-interactive argument of knowledge. The idea is simple. We encode the message and commit to the encoding using a vector commitment. Then, we prove that we committed to a valid codeword. The vector commitment and the proof will form our erasure code commitment, and openings will correspond to openings of the vector commitment.

Supported Erasure Code. The scheme presented in this section works generically for an arbitrary erasure code. Throughout the section, we let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code with alphabets Γ, Λ, message length k, code length n, and reception efficiency t, with reconstruction algorithm $\mathsf{Reconst}$.

Commitment Construction. Let $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be a vector commitment scheme over alphabet Λ with length n and opening alphabet Ξ, and let $\mathsf{PS} = (\mathsf{Setup}, \mathsf{PProve}, \mathsf{PVer})$ be a non-interactive argument of knowledge for relation

$$\mathcal{R} := \left\{ (\mathsf{stmt}, \mathsf{witn}) \;\middle|\; \begin{array}{c} \mathsf{witn} = m, \mathsf{stmt} = (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC}}, \rho), \\ \exists St_{\mathsf{VC}} :\ (\mathsf{com}_{\mathsf{VC}}, St_{\mathsf{VC}}) = \mathsf{VC}.\mathsf{Com}(\mathsf{ck}_{\mathsf{VC}}, \mathcal{C}(m); \rho) \end{array} \right\}.$$

We construct an erasure code commitment scheme $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}] = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ for $\mathcal{C}$ with opening alphabet Ξ as follows.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$:
  1. Compute $\mathsf{ck}_{\mathsf{VC}} \leftarrow \mathsf{VC}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs} \leftarrow \mathsf{PS}.\mathsf{Setup}(1^\lambda)$.
  2. Sample coins ρ for algorithm $\mathsf{VC}.\mathsf{Com}$.
  3. Set and return $\mathsf{ck} := (\mathsf{ck}_{\mathsf{VC}}, \mathsf{crs}, \rho)$.
- $\mathsf{Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$:
  1. Compute $\hat{m} := \mathcal{C}(m)$.
  2. Run $(\mathsf{com}_{\mathsf{VC}}, St_{\mathsf{VC}}) := \mathsf{VC}.\mathsf{Com}(\mathsf{ck}_{\mathsf{VC}}, \hat{m}; \rho)$.

3. Compute $\pi \leftarrow \mathsf{PProve}(\mathsf{crs}, \mathsf{stmt}, \mathsf{witn})$ for $\mathsf{witn} := m$ and $\mathsf{stmt} := (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC}}, \rho)$.
4. Set and return $\mathsf{com} := (\mathsf{com}_{\mathsf{VC}}, \pi)$ and $St := St_{\mathsf{VC}}$.

- $\mathsf{Open}(\mathsf{ck}, St, i) \rightarrow \tau$: Return $\tau \leftarrow \mathsf{VC.Open}(\mathsf{ck}_{\mathsf{VC}}, St_{\mathsf{VC}}, i)$.
- $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau) \rightarrow b$
  1. Parse $\mathsf{com} = (\mathsf{com}_{\mathsf{VC}}, \pi)$
  2. If $\mathsf{PVer}(\mathsf{crs}, \mathsf{stmt}, \pi) = 0$ for $\mathsf{stmt} := (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC}}, \rho)$, then return $b := 0$.
  3. If $\mathsf{VC.Ver}(\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC}}, i, \hat{m}_i, \tau) = 0$, then return $b := 0$.
  4. Return $b := 1$.

Completeness follows directly from the completeness of VC and PS.

Security. We show that the construction $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$ above is position-binding and code-binding. In addition, we show that it has message-bound openings. To recall, the notion of message-bound openings (cf. Definition 22) implies repairability for the resulting data availability sampling scheme. In the following, let PS.Ext be the knowledge extractor of PS.

Lemma 13. *If* VC *is position-binding, then* $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$ *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*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}[\mathcal{C},\mathsf{VC},\mathsf{PS}]}(\lambda) \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B},\mathsf{VC}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an algorithm breaking position-binding of $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$. We construct an algorithm $\mathcal{B}$ breaking position-binding of VC. It gets as input a commitment key $\mathsf{ck}_{\mathsf{VC}}$ for VC. It computes $\mathsf{crs} \leftarrow \mathsf{PS.Setup}(1^\lambda)$ and samples coins ρ for algorithm VC.Com. Then, it defines $\mathsf{ck} := (\mathsf{ck}_{\mathsf{VC}}, \mathsf{crs}, \rho)$ and runs $\mathcal{A}$ on input ck. Finally, $\mathcal{A}$ outputs $(\mathsf{com} = (\mathsf{com}_{\mathsf{VC}}, \pi), i, \hat{m}, \tau, \hat{m}', \tau')$ and the reduction $\mathcal{B}$ outputs $(\mathsf{com}_{\mathsf{VC}}, i, \hat{m}, \tau, \hat{m}', \tau')$. As Ver internally runs VC.Ver, it is clear that $\mathcal{B}$ breaks position-binding of VC if $\mathcal{A}$ breaks position-binding of $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$. □

Lemma 14. *If* VC *is position-binding and* PS *satisfies knowledge soundness, then* $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$ *is code-binding. Concretely, for any PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$, $\mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A}) + \mathbf{T}(\mathsf{PS.Ext}) + \mathbf{T}(C)$, and*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}[\mathcal{C},\mathsf{VC},\mathsf{PS}]}(\lambda) \leq \mathsf{Adv}^{\mathsf{kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{PS},\mathsf{PS.Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_2,\mathsf{VC}}(\lambda).$$

We postpone a formal proof to Appendix G. The intuition is as follows. Assume an adversary breaks code-binding of the scheme. This means the adversary outputs a commitment com and some openings, such that these openings are valid, but they are not consistent with the code. In the first step, we extract a witness from the proof contained in com. This witness is a message m such that the vector commitment part of com is a commitment of $\mathcal{C}(m)$. Because the openings are not consistent with the code, we know that at least one of these openings is not consistent with the symbol of $\mathcal{C}(m)$ at this position, which allows us to break position-binding.

Lemma 15. *If* VC *is position-binding and* PS *satisfies knowledge soundness, then* $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$ *has message-bound openings. Concretely, for any PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$, $\mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{B}_3) \approx \mathbf{T}(\mathcal{A}) + 2 \cdot \mathbf{T}(\mathsf{PS.Ext}) + 2 \cdot \mathbf{T}(C)$, and*

$$\mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathsf{kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{PS},\mathsf{PS.Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_2,\mathsf{VC}}(\lambda).$$

We postpone a formal proof to Appendix G.

Instantiation and Discussion. On the positive side, the construction presented in this section is generic. That is, we can construct an erasure code commitment for arbitrary codes from it. Also, the construction serves as a high level recipe for other constructions that we will present. While these other constructions are tailored to more specific families of codes, they will also contain parts that mimic the role of the vector commitment, and parts that take the role of the proof. On the negative side the construction presented in this section is hard to instantiate efficiently. For example, if we use a hash-based vector commitment, e.g., a Merkle Tree [Mer88], then the relation for which we need a non-interactive argument is also defined a hash function, and thus it is too unstructured for an efficient argument. Additionally, computing the non-interactive argument is computationally expensive. Finally, well-known impossibility results [GW11, CGKS22] show the need of non-falsifiable assumptions when we rely on succinct non-interactive arguments.

8 Commitments for Tensor Codes

In this section, we give a construction of an erasure code commitment scheme for the tensor code of two given linear codes.

Supported Erasure Code. For our construction, we assume two linear erasure codes $\mathcal{C}_r \colon \mathbb{F}^{k_r} \to \mathbb{F}^{n_r}$ and $\mathcal{C}_c \colon \mathbb{F}^{k_c} \to \mathbb{F}^{n_c}$ over the same field $\mathbb{F}$. Let t_r, t_c denote their respective reception efficiencies, and $\mathbf{G}_r \in \mathbb{F}^{n_r \times k_r}$ and $\mathbf{G}_c \in \mathbb{F}^{n_c \times k_c}$ denote their respective generator matrices. We consider the tensor code $\mathcal{C}_r \otimes \mathcal{C}_c \colon \mathbb{F}^{k_r \cdot k_c} \to \mathbb{F}^{n_r \cdot n_c}$.

Commitment Construction. We present an erasure code commitment scheme $\mathsf{CC}^{\otimes}$ for the code $\mathcal{C}_r \otimes \mathcal{C}_c \colon \mathbb{F}^{k_r \cdot k_c} \to \mathbb{F}^{n_r \cdot n_c}$ as above. In the construction, we assume that we already have an erasure code commitment scheme CC_c for the code $\mathcal{C}_c$. Further, we have to assume that CC_c is linear and extractable, in a sense we define next.

Definition 14 (Linear Erasure Code Commitment Scheme)**.** Let $\mathcal{C} \colon \mathbb{F}^k \to \mathbb{F}^n$ be a linear erasure code, where $\mathbb{F}$ is a finite field. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$. We say that CC is linear if the following properties hold:

- Com is deterministic. We use the notation $\mathsf{com} = \widehat{\mathsf{Com}}(\mathsf{ck}, \mathbf{m})$ for $(\mathsf{com}, St) = \mathsf{Com}(\mathsf{ck}, \mathbf{m})$.
- The commitment space is a vector space over $\mathbb{F}$ with efficiently computable vector addition and scalar multiplication. We use the usual symbols $+$ and $\cdot$ to denote these operations.
- For any fixed key $\mathsf{ck} \in \mathsf{Setup}(1^\lambda)$, the function $\widehat{\mathsf{Com}}(\mathsf{ck}, \cdot)$ is a vector space homomorphism over $\mathbb{F}$ from the vector space $\mathbb{F}^k$ to the commitment space.

From now on, assume that $\mathsf{CC}_c = (\mathsf{Setup}_c, \mathsf{Com}_c, \mathsf{Open}_c, \mathsf{Ver}_c)$ is linear and extractable. The new erasure code commitment scheme $\mathsf{CC}^{\otimes} = (\mathsf{Setup}^{\otimes}, \mathsf{Com}^{\otimes}, \mathsf{Open}^{\otimes}, \mathsf{Ver}^{\otimes})$ for code $\mathcal{C}_r \otimes \mathcal{C}_c \colon \mathbb{F}^{k_r \cdot k_c} \to \mathbb{F}^{n_r \cdot n_c}$ is as follows.

- $\mathsf{Setup}^{\otimes}(1^\lambda) \to \mathsf{ck}$: Return $\mathsf{ck} \leftarrow \mathsf{Setup}_c(1^\lambda)$.
- $\mathsf{Com}^{\otimes}(\mathsf{ck}, \mathbf{m}) \to (\mathsf{com}, St)$:
  1. Write $\mathbf{m}$ as a matrix $\mathbf{M} \in \mathbb{F}^{k_c \times k_r}$ and compute $\mathbf{Y} := \mathbf{M}\mathbf{G}_r^\top \in \mathbb{F}^{k_c \times n_r}$. Let $\mathbf{Y}_j \in \mathbb{F}^{k_c}$ denote the jth column of $\mathbf{Y}$, for each $j \in [n_r]$.
  2. For each $j \in [n_r]$, compute $(\mathsf{com}_j, St_j) := \mathsf{Com}_c(\mathsf{ck}, \mathbf{Y}_j)$.
  3. Set and return $\mathsf{com} := (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$ and $St := (St_1, \ldots, St_{n_r})$.
- $\mathsf{Open}^{\otimes}(\mathsf{ck}, St, j) \to \tau$: Let $(i^*, j^*) := \mathsf{ToMatIdx}(j)$ and return $\tau \leftarrow \mathsf{Open}_c(\mathsf{ck}, St_{j^*}, i^*)$.
- $\mathsf{Ver}^{\otimes}(\mathsf{ck}, \mathsf{com}, j, \hat{m}_j, \tau) \to b$:
  1. Let $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$.
  2. Let $\mathbf{H} \in \mathbb{F}^{(n_r - k_r) \times n_r}$ be the parity-check matrix of $\mathcal{C}_r$.
  3. Sample $\mathbf{a} \leftarrow\!\!\$\, \mathbb{F}^{n_r - k_r}$ and set $\mathbf{h} := \mathbf{H}^\top \mathbf{a}$.
  4. If $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) \neq \sum_{i=1}^{n_r} \mathbf{h}_j \cdot \mathsf{com}_j$, return 0.
  5. Let $(i^*, j^*) := \mathsf{ToMatIdx}(j)$.

6. If $\mathsf{Ver}_c(\mathsf{ck}, \mathsf{com}_{j^*}, i^*, \hat{m}_j, \tau) = 0$, return 0.

Completeness follows directly from the completeness and linearity of CC_c.

Security. We first show that the scheme $\mathsf{CC}^{\otimes}$ satisfies position-binding. Second, we show that it is computationally unique (cf. Definition 23). By Lemma 27, this implies that it has message-bound openings, and thus the resulting data availability sampling scheme is repairable. Note that the tensor code is not an MDS code, and so Lemma 28, which lifts computational uniqueness to code-binding, does not apply. Thus, we show code-binding from scratch.

Lemma 16. *If CC_c is position-binding, then $\mathsf{CC}^{\otimes}$ is position-binding. Concretely, for every PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$, such that*

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

Lemma 16 is proven by giving a simple reduction. We postpone the formal details to Appendix H.

Lemma 17. *Assume that $\mathcal{C}_c$ is an MDS code. If CC_c is linear, extractable, and satisfies position-binding, then $\mathsf{CC}^{\otimes}$ is computationally unique. Concretely, for every PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}, \mathcal{B}'$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{B}'') \approx \mathbf{T}(\mathcal{A})$, such that*

$$\mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{A},\mathsf{CC}^{\otimes}}(\lambda) \leq 2\left(\mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}_c}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}_c}(\lambda) + \frac{1}{|\mathbb{F}|}\right).$$

The formal proof of Lemma 17 is given in Appendix H. We provide an intuition for the proof. To prove that the scheme is computationally unique, we prove a simpler yet stronger statement. Namely, we show that whenever an adversary outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$ and enough valid openings $\mathbf{X}_{i,j} \in \mathbb{F}, \tau_{i,j}$ for $(i,j) \in I \subseteq [n_c] \times [n_r]$ that define a message $\mathbf{M} \in \mathbb{F}^{k_c \times k_r}$ via reconstruction, then committing to $\mathbf{M}$ yields com. To prove this, we first consider every column $j \in [n_r]$ for which the adversary output an opening. In the first k_r of these columns, we leverage the extractability of CC_c to extract a preimage of the corresponding column commitment com_j. Now, we can extend these columns into a matrix $\mathbf{Y}$ with rows in $\mathcal{C}_r$. Our next step is to show that the columns of $\mathbf{Y}$ commit to the com_j. For that, we rely on the homomorphic check and the fact that multiplying by a random element in the span of the parity-check matrix of $\mathcal{C}_r$ is as good as multiplying by the entire parity-check matrix. Next, we use position-binding to argue that the $\mathbf{G}_c\mathbf{Y}$ has to be consistent with the openings $\mathbf{X}_{i,j}$ that the adversary output. Finally, we use this to argue that $\mathbf{Y} = \mathbf{M}\mathbf{G}_r^{\top}$. In combination, this implies that committing to $\mathbf{M}$ yields com, as desired.

Lemma 18. *Assume that $\mathcal{C}_c$ is an MDS code. If CC_c is linear, extractable, and satisfies code-binding and position-binding, then $\mathsf{CC}^{\otimes}$ satisfies code-binding. Concretely, for every PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}, \mathcal{B}'$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{B}'') \approx \mathbf{T}(\mathcal{A})$, such that*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}^{\otimes}}(\lambda) \leq \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B},\mathsf{CC}_c}(\lambda) + \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B}',\mathsf{Ext},\mathsf{CC}_c}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}'',\mathsf{CC}_c}(\lambda) + \frac{1}{|\mathbb{F}|}.$$

We provide an intuition for proof of Lemma 18. The formal analysis is given in Appendix H. Assume that an adversary breaks code-binding. By definition, this means that it outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$ and some openings, such that all of the openings verify, and no codeword in $\mathcal{C}_r \otimes \mathcal{C}_c$ is consistent with these openings. In particular, there is at least one row or one column for which the openings are not consistent with any codeword in $\mathcal{C}_r$ or $\mathcal{C}_c$, respectively. Consider the case that there is such a column. As com contains a commitment to that column for scheme CC_c, this means that the adversary breaks code-binding of CC_c, which we assume is not possible. In the other case, the adversary outputs openings in a row that are not consistent with $\mathcal{C}_r$. This case is more involved, because there are no commitments for rows in com. It turns out that we can handle this case in a way almost identical to the proof of Lemma 17. Roughly, we can combine the strong statement that we showed there with the assumption that $\mathcal{C}_r$ is an MDS code and binding of CC_c.

Instantiation and Discussion. As an example, we can instantiate the construction in this section using Reed-Solomon codes for both $\mathcal{C}_r$ and $\mathcal{C}_c$. In this case, we need an extractable linear polynomial commitment scheme for the construction. Here, we can use the KZG commitment scheme [KZG10]. One can easily see that KZG is extractable and linear, see Appendix E.3. An instantiation like this is used by Ethereum [Fei23]. One advantage of this construction is that the size of openings is constant, i.e., it does not depend on the data length. The main drawback in this case is that we rely on a trusted setup.

9 Commitments for Interleaved Codes

In this section, we show two constructions of erasure code commitments for linear interleaved codes. These construction are partially inspired by Ligero [AHIV17, AHIV22] and mostly make use of hash functions.

9.1 Construction from Hash Functions

In this section, we present a construction of erasure code commitments for linear interleaved codes. The main benefit of this construction is that we can purely rely on hash functions.

Supported Erasure Code. Let $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$ be a linear erasure code with generator matrix $\mathbf{G} \in \mathbb{F}^{n\times k}$ and minimum distance $d^* \in \mathbb{N}$. We construct an erasure code commitment for the interleaved code $\mathcal{C}^{\equiv k}\colon \mathbb{F}^{k^2} \to \left(\mathbb{F}^k\right)^n$. To recall, this code consists of all sets of columns of matrices that have the form $\mathbf{M}\mathbf{G}^\top$ for some $\mathbf{M} \in \mathbb{F}^{k\times k}$.

Commitment Construction. Let $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. Let $P, L \in \mathbb{N}$ be parameters, and $\mathsf{H}_1\colon \{0,1\}^* \to \mathbb{F}^{P\times k}$ be a random oracle. Also, let $\mathsf{H}_2\colon \{0,1\}^* \to \binom{[n]}{L}$ be a random oracle. We construct an erasure code commitment scheme $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ for $\mathcal{C}^{\equiv k}$. The construction is as follows, making use of subroutines VerCol and VerCom.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: Return $\mathsf{ck} := \perp$.
- $\mathsf{Com}(\mathsf{ck}, \mathbf{m}) \to (\mathsf{com}, St)$:
  1. Write $\mathbf{m}$ as a matrix $\mathbf{M} \in \mathbb{F}^{k\times k}$, and compute $\mathbf{X} := \mathbf{M}\mathbf{G}^\top \in \mathbb{F}^{k\times n}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ for $j \in [n]$ be the jth column of $\mathbf{X}$.
  2. For each $j \in [n]$, compute $h_j := \mathsf{H}(\mathbf{X}_j)$.
  3. Compute $\mathbf{R} := \mathsf{H}_1(h_1, \dots, h_n)$. We have $\mathbf{R} \in \mathbb{F}^{P\times k}$.
  4. Compute linear combinations of rows, i.e., $\mathbf{W} := \mathbf{R}\mathbf{X} \in \mathbb{F}^{P\times n}$. Observe that each row of $\mathbf{W}$ is in the code $\mathcal{C}$.
  5. Compute $J := \mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$. We have $J \subseteq [n]$ and $|J| = L$.
  6. Set $\mathsf{com} := \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$ and $St := \perp$.
- $\mathsf{Open}(\mathsf{ck}, St, j) \to \tau$: Return $\tau := \perp$.
- $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j^*, \hat{m}_{j^*} = \mathbf{X}_{j^*}, \tau = \perp) \to b$:
  1. If $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}_{j^*}) = 0$, return 0, where subroutine $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}_{j^*})$ is as follows:
     (a) Let $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$.
     (b) If $h_{j^*} \neq \mathsf{H}(\mathbf{X}_{j^*})$, return 0.
     (c) Compute $\mathbf{R} := \mathsf{H}_1(h_1, \dots, h_n)$.
     (d) Let $\mathbf{W}_{j^*}$ be the j^*th column of $\mathbf{W}$. If $\mathbf{W}_{j^*} \neq \mathbf{R}\mathbf{X}_{j^*}$, return 0. Otherwise, return 1.
  2. If $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 0$, return 0, where subroutine $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com})$ is as follows:
     (a) Let $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$.
     (b) If there is a row $\mathbf{w}^\top \in \mathbb{F}^{1\times n}$ of $\mathbf{W}$ such that $\mathbf{w} \notin \mathcal{C}$, then return 0.
     (c) If $J \neq \mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$, return 0.
     (d) Return 1, if for all $j \in J$, we have $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j, \mathbf{X}_j) = 1$. Otherwise, return 0.

3. Return 1.

Completeness can easily be checked.

Security. We show position-binding and code-binding of our construction.

Lemma 19. *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. Then, the scheme* CC *is position-binding. Concretely, for every algorithm* $\mathcal{A}$ *that makes at most* Q_H *queries to random oracle* H*, we have*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq \frac{Q_\mathsf{H}^2}{2^\lambda}.$$

Proof. If we have an adversary that breaks position-binding of CC, then it must provide two distinct preimages of one of the hash values contained in the commitment. Formally, let $\mathcal{A}$ be an algorithm in the position-binding game of CC making at most Q_H queries to random oracle H. This includes the queries that algorithm Ver issues when it checks the validity of openings in $\mathcal{A}$'s final output. The probability that there are two queries x and x' of $\mathcal{A}$ with $x \neq x'$ but $\mathsf{H}(x) = \mathsf{H}(x')$ is at most $Q_\mathsf{H}^2/2^\lambda$. Assuming this event does not occur, $\mathcal{A}$ can not break position-binding, and the claim follows. □

Lemma 20. *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda, \mathsf{H}_1\colon \{0,1\}^* \to \mathbb{F}^{P\times k}$*, and* $\mathsf{H}_2\colon \{0,1\}^* \to \binom{[n]}{L}$ *be a random oracle. Then, the scheme* CC *is code-binding. Concretely, for any* $\Delta_1, \Delta_2 \in [n]$ *with* $\Delta_1 + \Delta_2 < d^*$ *and* $\Delta_1 \leq d^*/4$*, and every algorithm* $\mathcal{A}$ *that makes at most* $Q_\mathsf{H}, Q_{\mathsf{H}_1}, Q_{\mathsf{H}_2}$ *queries to random oracles* $\mathsf{H}, \mathsf{H}_1, \mathsf{H}_2$*, respectively, we have*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq\ & \frac{\bar{Q}_\mathsf{H}\bar{Q}_{\mathsf{H}_1}n + \bar{Q}_\mathsf{H}^2}{2^\lambda} \\ & + \bar{Q}_{\mathsf{H}_1}\bar{Q}_{\mathsf{H}_2} \cdot \left(\left(\frac{\Delta_1+1}{|\mathbb{F}|}\right)^P + \left(1 - \frac{\Delta_1+1}{n}\right)^L + \left(1 - \frac{\Delta_2}{n}\right)^L + \frac{1}{|\mathbb{F}|^P} \right),\end{aligned}$$

where $\bar{Q}_\mathsf{H} := Q_\mathsf{H} + n$*,* $\bar{Q}_{\mathsf{H}_1} := Q_{\mathsf{H}_1} + Q_{\mathsf{H}_2} + 1$*,* $\bar{Q}_{\mathsf{H}_2} := Q_{\mathsf{H}_2} + 1$*.*

Code-binding is proven via a sequence of lemmas. The goal is to show Lemma 20, which states that CC satisfies code-binding. To do that, we first abstract the interactions of the adversary with the random oracles away. In the resulting game, the adversary essentially runs an interactive five round protocol with the challenger. Namely, it sends a matrix $\mathbf{X}$ and receives a random challenge matrix $\mathbf{R}$. Then, it sends a matrix $\mathbf{W}$ and receives a challenge $J \subseteq [n]$. Finally, it submits a set $J' \subseteq [n]$. The adversary wins the game if these matrices suffice to break code-binding, namely, if (1) there is no $\mathbf{X}'$ in the interleaved code $\mathcal{C}^{\equiv k}$ that is consistent with $\mathbf{X}$ on all columns in J', and (2) each row of $\mathbf{W}$ is in the code $\mathcal{C}$, and (3) for all $j \in J \cup J'$, we have $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$. The central lemma of our analysis (Lemma 35) shows that the adversary can not win this game. We split the proof of it into three main steps (Lemmata 32 to 34):

1. Lemma 32: $\mathbf{X}$ has to be close to the interleaved code for a winning adversary. Concretely, it has to be within the unique decoding distance, i.e., there is a unique $\mathbf{X}^*$ in the code that is close to $\mathbf{X}$.
2. Lemma 33: $\mathbf{RX}$ and $\mathbf{W}$ have to be sufficiently close, due to the randomness of challenge set J.
3. Lemma 34: Using the previous two statements, we get that the distance of $\mathbf{RX}^*$ and $\mathbf{W}$ is at most d^*. As both are in the code, we get that $\mathbf{RX}^* = \mathbf{W}$. Therefore, there is a column in which $\mathbf{X}^*$ and $\mathbf{X}$ differ, but $\mathbf{RX}^*$ and $\mathbf{RX}$ agree on that column. The probability of this can then be bounded, which allows us to prove the central lemma.

We give the formal analysis in Appendix I.1.

Instantiation and Discussion. The main drawback of the construction presented in this section is the following. When we use it to construct a data availability sampling scheme, a single symbol of the encoding is rather large. Concretely, it has size $\sqrt{|\mathsf{data}|/\log|\mathbb{F}|} \cdot \log|\mathbb{F}|$ bits, where $|\mathsf{data}|$ denotes the size of the encoded data in bits. Another drawback is that the scheme does not have message-bound openings, as defined in Definition 22. We can easily see this by considering an adversary that outputs (1) an honest commitment to some message and enough openings including the first symbol, and (2) an almost honest commitment to the same message, where h_1 is malformed, and enough openings not including the first symbol. On the other hand, the main advantage of the construction in this section is that it only relies on the security of hash functions and does not require expensive operations such as multiplications over cyclic groups or pairings. Especially, no trusted setup is needed, and we can instantiate the construction over a small field $\mathbb{F}$, e.g., the field with 2^{32} elements, leading to computational efficiency.

9.2 Construction from Homomorphic Hash Functions

In this section, we present a variant of our construction in Section 9.1. This variant makes use of homomorphic hash functions (see Definition 19). Compared to the construction in Section 9.1, this can reduce the size of the commitment for certain instantiations.

Supported Erasure Code. Let $\mathcal{C}\colon \mathbb{F}^k \to \mathbb{F}^n$ be a linear erasure code and let $\mathbf{G} \in \mathbb{F}^{n\times k}$ be its generator matrix. We construct an erasure code commitment scheme for the interleaved code $\mathcal{C}^{\equiv k}\colon \mathbb{F}^{k^2} \to \left(\mathbb{F}^k\right)^n$.

Commitment Construction. We make use of random oracles $\mathsf{H}_1\colon \{0,1\}^* \to \mathbb{F}^{P\times k}$ and $\mathsf{H}_2\colon \{0,1\}^* \to \mathbb{F}^{n\times L}$, where $P, L \in \mathbb{N}$ are parameters. In addition, we rely on a homomorphic hash function family $\mathsf{HF} = (\mathsf{Gen}, \mathsf{Eval})$ with domain $\mathcal{D} = \mathbb{F}^k$ (see Definition 19). Denote the key space and range of HF by $\mathcal{K}, \mathcal{R}$, respectively. Our erasure code commitment scheme $\mathsf{CC}[\mathsf{HF}] = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ for $\mathcal{C}^{\equiv k}$ is as follows.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: Return $\mathsf{ck} := \mathsf{hk} \leftarrow \mathsf{HF.Gen}(1^\lambda)$.
- $\mathsf{Com}(\mathsf{ck}, \mathbf{m}) \to (\mathsf{com}, St)$:
  1. Write $\mathbf{m}$ as a matrix $\mathbf{M} \in \mathbb{F}^{k\times k}$, and compute $\mathbf{X} := \mathbf{M}\mathbf{G}^\top \in \mathbb{F}^{k\times n}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ for $j \in [n]$ be the jth column of $\mathbf{X}$.
  2. For each $j \in [n]$, compute $h_j := \mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}_j)$.
  3. Compute $\mathbf{R} := \mathsf{H}_1(h_1, \ldots, h_n)$. We have $\mathbf{R} \in \mathbb{F}^{P\times k}$.
  4. Compute $\mathbf{W} := \mathbf{R}\mathbf{X} \in \mathbb{F}^{P\times n}$.
  5. Compute $\mathbf{S} := \mathsf{H}_2(h_1, \ldots, h_n, \mathbf{W})$. We have $\mathbf{S} \in \mathbb{F}^{n\times L}$.
  6. Compute $\mathbf{Y} := \mathbf{X}\mathbf{S} \in \mathbb{F}^{k\times L}$.
  7. Set $\mathsf{com} := \big((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\big)$ and $St := \bot$.
- $\mathsf{Open}(\mathsf{ck}, St, j) \to \tau$: Return $\tau := \bot$.
- $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j^*, \hat{m}_{j^*} = \mathbf{X}_{j^*}, \tau = \bot) \to b$:
  1. If $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}_{j^*}) = 0$, return 0, where subroutine $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}_{j^*})$ is as follows:
     (a) Let $\mathsf{com} = \big((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\big)$.
     (b) If $h_{j^*} \neq \mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}_{j^*})$ or $\mathbf{X}_{j^*} \notin \mathbb{F}^k$, return 0.
     (c) Compute $\mathbf{R} := \mathsf{H}_1(h_1, \ldots, h_n)$.
     (d) Let $\mathbf{W}_{j^*}$ be the j^*th column of $\mathbf{W}$. If $\mathbf{W}_{j^*} \neq \mathbf{R}\mathbf{X}_{j^*}$, return 0. Otherwise, return 1.
  2. If $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 0$, return 0, where subroutine $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com})$ is as follows:
     (a) Let $\mathsf{com} = \big((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\big)$.
     (b) If there is a row $\mathbf{w}^\top \in \mathbb{F}^{1\times n}$ of $\mathbf{W}$ such that $\mathbf{w} \notin \mathcal{C}$, then return 0.
     (c) Compute $\mathbf{R} := \mathsf{H}_1(h_1, \ldots, h_n)$.
     (d) Set $\mathbf{S} := \mathsf{H}_2(h_1, \ldots, h_n, \mathbf{W})$. Let $\mathbf{S}_j \in \mathbb{F}^k$ and $\mathbf{Y}_j \in \mathbb{F}^k$ for $j \in [L]$ be the jth column of $\mathbf{S}$ and $\mathbf{Y}$, respectively.
     (e) Return 1, if for each $j \in [L]$, we have $\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{Y}_j) = [h_1, \ldots h_n]\mathbf{S}_j$ and $\mathbf{R}\mathbf{Y} = \mathbf{W}\mathbf{S}$. Otherwise, return 0.
  3. Return 1.

Completeness easily follows from the homomorphism property of HF.

Security. We show position-binding and code-binding. Position-binding follows directly from the collision-resistance of HF.

Lemma 21. *Given that* HF *is a homomorphic family of hash functions, we have that* $\mathsf{CC}[\mathsf{HF}]$ *is position-binding. Concretely, for every PPT algorithm* $\mathcal{A}$*, there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$*, such that*

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{CC}[\mathsf{HF}]}(\lambda) \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathcal{B},\mathsf{HF}}(\lambda).$$

Proof. If we have an adversary that breaks position-binding of $\mathsf{CC}[\mathsf{HF}]$, then it must provide two distinct preimages of one of the hash values contained in the commitment. More formally, let $\mathcal{A}$ be a PPT algorithm in the position-binding game of $\mathsf{CC}[\mathsf{HF}]$. We construct a reduction $\mathcal{B}$ against collision-resistance of HF as follows. Reduction $\mathcal{B}$ gets input hk from the collision-resistance experiment. It defines $\mathsf{ck} := \mathsf{hk}$, and runs $\mathcal{A}$ on input ck. When $\mathcal{A}$ terminates, it outputs $\mathsf{com}, j^*, \mathbf{X}_{j^*}, \tau, \mathbf{X}'_{j^*}, \tau'$. The reduction outputs $\mathbf{X}_{j^*}$ and $\mathbf{X}'_{j^*}$ to the collision-resistance game. It is clear that $\mathcal{B}$ perfectly simulates the position-binding game for $\mathcal{A}$, and its running time is dominated by the running time of $\mathcal{A}$. Further, assume $\mathcal{A}$ breaks position-binding, i.e. $\mathbf{X}_{j^*} \neq \mathbf{X}'_{j^*}$, $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}_{j^*}, \tau) = 1$, and $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j^*, \mathbf{X}'_{j^*}, \tau') = 1$. Write $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\right)$. By definition of Ver, in particular the definition of subroutine VerCol, we know that this implies

$$\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}_{j^*}) = h_{j^*} = \mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}'_{j^*}).$$

As $\mathbf{X}_{j^*} \neq \mathbf{X}'_{j^*}$, $\mathcal{B}$ breaks collision-resistance. □

Lemma 22. *Let* HF *be a homomorphic family of hash functions. Let* $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{F}^{P\times k}$*, and* $\mathsf{H}_2 \colon \{0,1\}^* \to \mathbb{F}^{n\times L}$ *be a random oracle. Then, the scheme* $\mathsf{CC}[\mathsf{HF}]$ *is code-binding. Concretely, for any PPT algorithm* $\mathcal{A}$ *that makes at most* $Q_{\mathsf{H}_1}, Q_{\mathsf{H}_2}$ *queries to random oracles* $\mathsf{H}_1, \mathsf{H}_2$*, respectively, there is an EPT algorithm* $\mathcal{B}$ *with expected running time* $\mathbf{ET}(\mathcal{B}) \approx (1+n)\mathbf{T}(\mathcal{A})$ *and*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}[\mathsf{HF}]}(\lambda) \leq \bar{Q}_{\mathsf{H}_1}\bar{Q}_{\mathsf{H}_2} \cdot \left(\frac{n}{|\mathbb{F}|^L} + \frac{1}{|\mathbb{F}|^P} + \frac{1}{|\mathbb{F}|^L} + \mathsf{Adv}^{\mathsf{coll}}_{\mathcal{B},\mathsf{HF}}(\lambda)\right),$$

where $\bar{Q}_{\mathsf{H}_1} := Q_{\mathsf{H}_1} + Q_{\mathsf{H}_2} + 1$ *and* $\bar{Q}_{\mathsf{H}_2} := Q_{\mathsf{H}_2} + 1$.

We provide an overview of the proof strategy we use to prove Lemma 22. The formal analysis is given in Appendix I.2. To show code-binding, we first specify a security game without random oracles by abstracting random oracles away. The central lemma of our analysis (Lemma 36) shows that the adversary can not win this game. Then, we show code-binding using this central lemma, similar to what we have done for our construction based on (non-homomorphic) hash functions. In the game of our central lemma, the adversary first obtains a hash key hk and then specifies hash values $h_1, \ldots, h_n$. Then, a matrix $\mathbf{R}$ is sampled at random from $\mathbb{F}^{P\times k}$ and given to the adversary. The adversary outputs a matrix $\mathbf{W}$, and gets back a random matrix $\mathbf{S} \in \mathbb{F}^{n\times L}$. This reflects the interaction between the adversary and the random oracles. Finally, the adverary outputs $\mathbf{Y}, J', (\mathbf{X}_j)_{j\in J'}$, which reflects that the adversary outputs a commitment and some openings in the code-binding game. The adversary wins if the matrices and openings satisfy all conditions as in the code-binding game. For examples, the openings $\mathbf{X}_j$ have to satisfy $\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}_j) = h_j$. A major challenge we have to deal with when proving our central lemma is that initially we only get hash values $h_1, \ldots, h_n$ from the adversary, and not their preimages. Later, we get some of the preimages. This is in contrast to our construction based on non-homomorphic hash functions modeled as random oracles, for which we could easily extract the preimages by observing the random oracle. Thus, we need another way of extracting these preimages. Our idea is as follows. We first fix some hash key and adversarial randomness, leading to fixed hash values $h_1, \ldots, h_n$. Then, we run the rest of the experiment a number of times, i.e., we rewind the adversary. Recall that one winning condition is that a homomorphic check on the hash values, given by the condition $\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{Y}_j) = [h_1, \ldots h_n]\mathbf{S}_j$ for each $j \in [L]$. From this check, we observe that if we have enough such $\mathbf{S}$ with enough linearly independent columns, we find the preimages of $h_1, \ldots, h_n$ by solving a linear system of equations. Once we have this, we run the game a final time, rule out inconsistent openings by reducing to collision-resistance, and conclude using statistical arguments. Turning this intuition into a formal proof is surprisingly challenging, especially to make the rewinding work without subtle problems. For example, to get expected polynomial running time of our reduction, we have to ensure that the rewinding always (not only in an overwhelming fraction of cases) ends after a finite number of repetitions.

Instantiation and Discussion. The scheme presented in this section comes with many of the drawbacks and advantages of the scheme presented in Section 9.1. Namely, while a single symbol of the encoding is rather large, we avoid a trusted setup when instantiating the homomorphic hash function appropriately. In contrast to the scheme in Section 9.1, we can get a smaller commitment when using a large field. This is because we only require minimal parallel repetition (parameter L) whereas the scheme in Section 9.1 requires a large L even with a large field. The prize we pay is the use of a computationally more expensive large field and public key operations. An example instantiation of the homomorphic hash function is the function

$$\mathbb{Z}_p^k \to \mathbb{G}, \quad (x_1, \ldots, x_k) \mapsto \prod_{i=1}^{k} g_i^{x_i}$$

over a cyclic group $\mathbb{G}$ of prime order p with generators g_i. The function is collision-resistant if the DLOG assumption holds in $\mathbb{G}$. We leave investigating a lattice-based instantiation of the homomorphic hash function as future work.

10 Evaluation and Comparison

In this section, we give an overview of how the different constructions compare in terms of efficiency. As many of these constructions are written in a generic way, we can not cover all possible instantiations and parameter settings. Instead, we pick reasonable instantiations, suitable for comparison across schemes.

10.1 Setting the Stage

Before we discuss the results of our comparison, we first explain which constructions of data availability sampling we consider, which aspects we analyze, and how our results are derived.

Schemes. We consider data availability sampling schemes that follow our construction in Section 6. That is, they are constructed using an erasure code $\mathcal{C}$, an erasure code commitment CC for $\mathcal{C}$, and an index sampler. We use the index sampler $\mathsf{Sample}_{\mathsf{wr}}$ from Section 6.2, i.e. sampling with replacement, and assume that each client makes $Q = 1$ query, which has no effect for this particular sampler.

Concrete Erasure Code Commitments. All our concrete instantiations of erasure code commitments target 128-bits of computational security, and we include two "trivial" schemes as a baseline. The following schemes are compared: 1. Naive, the naive scheme, where the encoding has a single symbol, containing all the data, and the commitment is a $\mathsf{SHA\text{-}256}$ hash of the data. 2. Merkle, a trivial scheme based on Merkle Trees [Mer88] and the identity code. 3. RS, a scheme where we encode the data using a Reed-Solomon code and commit to it using the KZG [KZG10] polynomial commitment scheme. 4. Tensor, an instantiation of the tensor code construction (Section 8) using KZG as a base scheme. 5. Hash, the scheme for interleaved codes from random oracles (Section 9.1), instantiated with $\mathsf{SHA\text{-}256}$ and Reed-Solomon codes over a 32-bit field. 6. $\mathsf{HomHash}$, the scheme for interleaved codes from homomorphic hashing (Section 9.2), instantiated with Pedersen commitments over the $\mathsf{Secp256k1}$ curve, $\mathsf{SHA\text{-}256}$, and Reed-Solomon codes over the scalar field of $\mathsf{Secp256k1}$. An overview is provided in Table 1.

Qualitative Criteria. To evaluate the schemes mentioned above, we consider both qualitative aspects and efficiency aspects. In terms of qualitative aspects we are interested in the cryptographic assumptions, the idealized models that the schemes rely on and whether the schemes require a trusted setup.

Efficiency Criteria. We compare the schemes by fixing the data size. Then, we compute the encoding and commitment size and the communication complexity per query of a client. We are also interested in estimating the threshold of the schemes, i.e., the number of queries need to be made by clients, such that the probability of reconstructing is overwhelming. In our comparison, we want it to be at least $1 - 2^{-40}$ (40-bits of statistical security). We determine the threshold using the bounds in Lemma 3 and Examples 5 and 6. Once we determined the threshold, we can then also compute the overall communication complexity required to reconstruct the data. Finally, we will briefly discuss the asymptotic computational efficiency of the schemes. We leave implementing the schemes and comparing concrete running times for future work.

10.2 Results

We implemented our methodology in Python scripts given in Appendix K. Our results are presented in Tables 2 and 3 and Figure 3. We now discuss the results.

Assumptions, Models, and Setup. In terms of qualitative criteria, the schemes Hash, Naive, Merkle are the most desirable ones, as they do not rely on a trusted setup and only rely on hash functions. Scheme $\mathsf{HomHash}$ is also a good choice as it avoids trusted setup. Depending on the instantiation of the

Name	Code $\mathcal{C}$	Commitment CC	Parameters/Comments
Naive	-	Hash	All data in one encoding symbol
Merkle	Identity	Merkle Tree	Size of Leaf: 2^{10} bit
RS	$\mathcal{RS}[k,n,\mathbb{F}]$	KZG [KZG10]	$n = 4k,\ \mathbb{F} = \mathbb{Z}_p$
Tensor	$\mathcal{RS}[k,n,\mathbb{F}]^{\otimes}$	Section 8	$n = 2k,\ \mathbb{F} = \mathbb{Z}_p$
Hash	$\mathcal{RS}[k,n,\mathbb{F}]^{\equiv k}$	Section 9.1	$n = 4k,\ \lvert\mathbb{F}\rvert = 2^{32},\ P = 8,\ L = 64$
HomHash	$\mathcal{RS}[k,n,\mathbb{F}]^{\equiv k}$	Section 9.2	Pedersen Hash, $n = 4k, \mathbb{F} = \mathbb{Z}_p, P = L = 2$

Table 1: Overview of the different instantiations of erasure code commitments that we compare in Section 10. For each scheme, parameter k is picked such that the input domain fits the data length. The notation $\mathcal{RS}[k,n,\mathbb{F}]^{\otimes}$ is a short notation for $\mathcal{RS}[k,n,\mathbb{F}] \otimes \mathcal{RS}[k,n,\mathbb{F}]$.

Scheme	Assumption	Idealized Model	Trusted Setup
Naive	Hash	-	✗
Merkle	Hash	-	✗
RS	q-Type	AGM	✓
Tensor	q-Type	AGM	✓
Hash	-	ROM	✗
HomHash	DLOG	ROM	✗

Table 2: Qualitative comparison of different data availability sampling schemes. The details of the schemes are given in Table 1. We compare the cryptographic assumptions and idealized models that these schemes use, and whether they rely on a trusted setup or not.

homomorphic hash function, it only relies on mild cryptographic assumptions, e.g., DLOG. Schemes RS and Tensor require trusted setup and stronger assumptions.

Encoding Size. In terms of encoding size, schemes RS and Tensor have a slightly larger encoding than Hash and HomHash, which comes from the KZG [KZG10] openings that have to be stored in addition to the codeword. It is natural that Hash and HomHash have (almost) the same encoding size, as they encode data using the same code with no explicit opening, the field size does not affect the size of the encoding significantly – the minimal discrepancy comes from rounding.

Commitment Size. In terms of commitment size, schemes Naive, Merkle, RS, and Tensor perform best. The commitment for Naive, Merkle is a single hash value. For RS, the commitment is a single group element over a group of size p, namely, a single KZG [KZG10] commitment. Especially, the commitment size for these three schemes Naive, Merkle, and RS is constant, i.e., independent of the size of the data. For Tensor, $\Theta(\sqrt{|\mathsf{data}|/\log p})$ such KZG commitments are needed. The schemes Hash and HomHash perform worse in terms of commitment size. Especially, Hash has a larger commitment. This is because due to the small field size, we require large repetition factor L which shows up in the commitment size. Concretely, the commitment contains L random colums of the codeword, which are of size $k = \sqrt{|\mathsf{data}|/32}$ field elements. On the other hand, for HomHash, we had to choose a large field to implement the homomorphic hash function, leading to small repetition factors and thus a smaller commitment size than for Hash.

Communication per Query. In terms of communication complexity per query, scheme Naive disqualifies, as expected. Optimal with respect to this measure are RS and Tensor, for which the communication complexity per query is constant, i.e., independent of the data size. This is because both return a single KZG [KZG10] opening and a single field element. Schemes Hash and HomHash perform worse in terms of communication complexity per query, which is due to the use of the interleaved code, which has symbols of size $f \cdot \sqrt{|\mathsf{data}|/f}$, where f is the number of bits needed to represent one field element. If we compare these two schemes, we see the inverse of what we saw for the commitment size. Namely, Hash performs better. This can be explained by the different field sizes. Namely, f does not cancel out in the symbol size $f \cdot \sqrt{|\mathsf{data}|/f} = \sqrt{|\mathsf{data}|} \cdot \sqrt{f}$. The ratio between $\sqrt{256}$ and $\sqrt{32}$ matches the gap that we see in Table 3 and Figure 3.

Total Communication. Multiplying the communication per query with the number of samples required to reconstruct the data with high probability, we obtain the total communication cost. We see that Merkle disqualifies due to a huge number of samples, which follows Lemma 3 and Examples 5 and 6. Further, we see that RS and Tensor perform worse than Hash and HomHash. This is because Hash and HomHash use an interleaved code, leading to a smaller number of symbols and therefore to a smaller number of required samples. One could expect that the large communication per query of Hash and HomHash outweighs this, but our results show that this is not the case. We can explain this by comparing with scheme Naive, which has only one symbol. Of course, this scheme achieves the optimal total communication of exactly $|\mathsf{data}|$. We can think of Hash and HomHash as being between this naive scheme and schemes like RS and Tensor. Namely, they have a small number of large symbols. We thus expect that the total communication gets worse if we increase the number of symbols and decrease their size.

Computational Efficiency. Clients are computationally lightweight in all schemes. For example, in KZG-based constructions (RS and Tensor), each sample is verified using two pairings. For encoding, the computational complexity for all schemes depends on the encoding complexity for the underlying code. For the interleaved constructions (Hash and HomHash), we can assume that the code has encoding time of $\Theta(k \log k)$ using FFT techniques. Then, encoding for the interleaved code takes time $\Theta(\sqrt{k} \cdot (\sqrt{k} \log \sqrt{k})) = \Theta(k \log k)$. A similar complexity can be achieved for Tensor if KZG opening proofs are computed efficiently using recent techniques [FK23].

Conclusion. Clearly, the schemes Naive and Merkle are far from being usable in practice due to huge communication costs per query or in total, respectively. They should only be understood as a baseline. If we are interested in using schemes that do not rely on trusted setup and use minimal assumptions, the schemes Hash and HomHash are desirable. If we compare these two, Hash performs better in terms of communication complexity per query, but worse in terms of commitment size. Additionally, Hash avoids computationally expensive public key operations and instead only needs hash operations and arithmetic over small fields. On the other hand, if the communication effort per client is our primary goal, schemes RS and Tensor are the best choice, as the commitment size is minimal and the communication per query is constant.

	Scheme	$|\mathsf{com}|$ [KB]	$|\pi|$ [MB]	Query [KB]	Samples	Total [MB]
$|\mathsf{data}| = 1$ MB	Naive	0.03	1.00	1000.00	1	1.00
	Merkle	0.03	4.25	0.55	286655	156.40
	RS	0.05	8.00	0.10	35881	3.52
	Tensor	6.96	8.07	0.10	160115	15.70
	Hash	256.00	4.00	2.00	879	1.76
	HomHash	80.00	4.01	5.67	323	1.83
$|\mathsf{data}| = 32$ MB	Naive	0.03	32.00	32000.00	1	32.00
	Merkle	0.03	176.00	0.71	10038776	7089.80
	RS	0.05	256.00	0.10	1147584	113.23
	Tensor	39.22	256.32	0.10	4626776	456.52
	Hash	1448.45	128.05	11.32	4888	55.32
	HomHash	452.00	128.00	32.00	1740	55.68

Table 3: Efficiency comparison of different data availability sampling schemes. Details of the schemes are given in Table 1. For given size of data, we compare the size of commitments com, encodings π, and communication complexity per query. Column "Samples" shows the total number of samples that clients need to query such that data can be reconstructed with probability at least $1 - 2^{-40}$, and the final column denotes the total communication cost for this process.

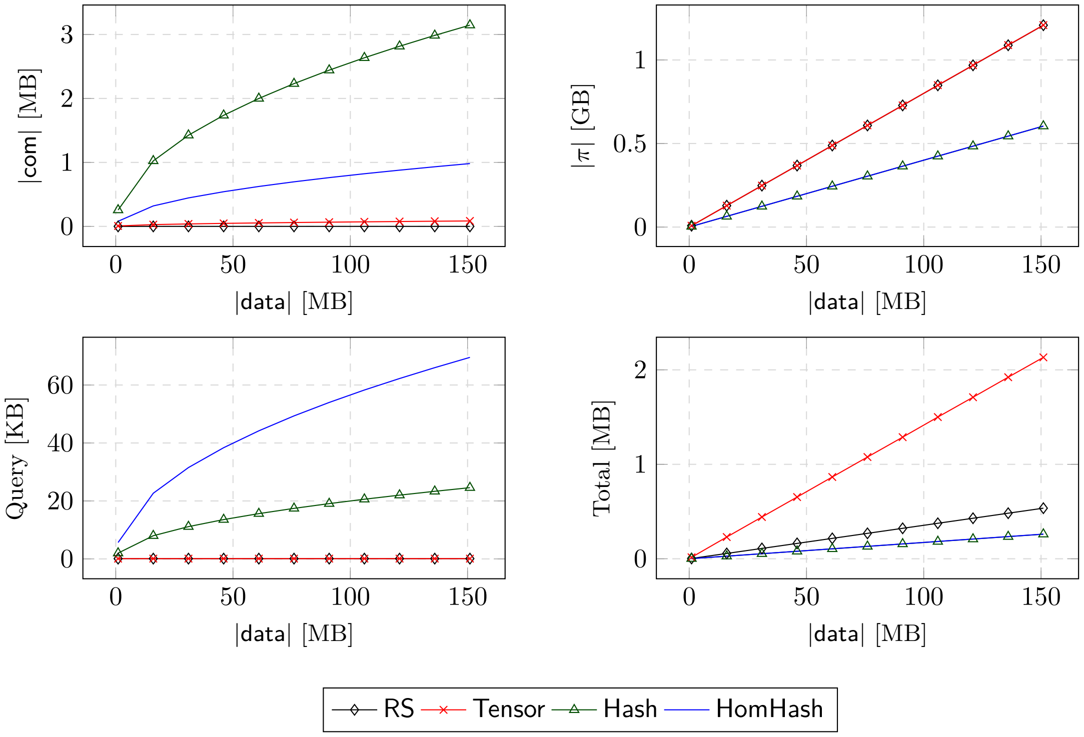

Figure 3: Efficiency of data availability sampling schemes. The details of the schemes are given in Table 1. We compare the size of commitments, the size of the encoding, the communication complexity per query, and the total communication complexity when increasing the data size. Schemes Naive and Merkle are omitted.

References

[ABC$^+$07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 598–609. ACM Press, October 2007. (Cited on page 4.)

[ADVZ21] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. Cryptology ePrint Archive, Report 2021/1500, 2021. https://eprint.iacr.org/2021/1500. (Cited on page 5.)

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. (Cited on page 11, 27.)

[AHIV22] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. Cryptology ePrint Archive, Paper 2022/1608, 2022. https://eprint.iacr.org/2022/1608. (Cited on page 27, 55.)

[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 3.)

[BCG$^+$17] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Interactive oracle proofs with constant rate and query complexity. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl, July 2017. (Cited on page 5.)

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081. (Cited on page 5.)

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. (Cited on page 8.)

[BGH$^+$06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006. (Cited on page 5.)

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 5:1–5:32. LIPIcs, January 2020. (Cited on page 5.)

[CDD$^+$16] Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, Heidelberg, August 2016. (Cited on page 14.)

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013. (Cited on page 17.)

[CFM08] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 433–450. Springer, Heidelberg, April 2008. (Cited on page 5.)

[CGKS22] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. Cryptology ePrint Archive, Report 2022/638, 2022. https://eprint.iacr.org/2022/638. (Cited on page 25.)

[CHL$^+$05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 422–439. Springer, Heidelberg, May 2005. (Cited on page 5.)

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. (Cited on page 5.)

[CKW13] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 279–295. Springer, Heidelberg, May 2013. (Cited on page 4.)

[CT05] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In Pierre Fraigniaud, editor, *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 503–504. Springer, 2005. (Cited on page 4.)

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 109–127. Springer, Heidelberg, March 2009. (Cited on page 4.)

[Fei23] Dankrad Feist. Data availability encoding. https://notes.ethereum.org/ReasmW86SuKqC2FaX83T1g, 2023. Accessed: 2023-05-08. (Cited on page 27.)

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, October 1987. (Cited on page 11.)

[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 33.)

[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 17.)

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. (Cited on page 8.)

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. (Cited on page 25.)

[JK07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 584–597. ACM Press, October 2007. (Cited on page 4.)

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. (Cited on page 8.)

[KZG10] 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 5, 8, 17, 27, 31, 32, 45.)

[LRY16] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016. (Cited on page 5.)

[LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010. (Cited on page 5.)

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. (Cited on page 8, 17, 25, 31.)

[NNT21] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. Cryptology ePrint Archive, Report 2021/1544, 2021. `https://eprint.iacr.org/2021/1544`. (Cited on page 3, 4.)

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. (Cited on page 8.)

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989. (Cited on page 4.)

[SSP13] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 325–336. ACM Press, November 2013. (Cited on page 4.)

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, Heidelberg, December 2008. (Cited on page 4.)

[SXKV21] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. ACeD: Scalable data availability oracle. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 299–318. Springer, Heidelberg, March 2021. (Cited on page 3.)

[YSL$^+$20] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 114–134. Springer, Heidelberg, February 2020. (Cited on page 3.)

Part II
Appendix

Table of Contents

A Definition of Cryptographic Building Blocks

Definition 15 (Vector Commitment Scheme)**.** A vector commitment scheme over alphabet Σ with length ℓ and opening alphabet Ξ is a tuple $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ of PPT algorithms, with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ takes as input the security parameter, and outputs a commitment key ck.
- $\mathsf{Com}(\mathsf{ck}, m) \to (\mathsf{com}, St)$ takes as input a commitment key ck and a string $m \in \Sigma^\ell$, and outputs a commitment com and a state St.
- $\mathsf{Open}(\mathsf{ck}, St, i) \to \tau$ takes as input a commitment key ck, a state St, and an index $i \in [\ell]$, and outputs an opening $\tau \in \Xi$.
- $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, m_i, \tau) \to b$ is deterministic, takes as input a commitment key ck, a commitment com, and index $i \in [\ell]$, a symbol $m_i \in \Sigma$, and an opening $\tau \in \Xi$, 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 $m \in \Sigma^\ell$, and every $i \in [\ell]$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, m_i, \tau) = 1 \;\middle|\; \begin{array}{l} (\mathsf{com}, St) \leftarrow \mathsf{Com}(\mathsf{ck}, m), \\ \tau \leftarrow \mathsf{Open}(\mathsf{ck}, St, i) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 16 (Position-Binding of VC)**.** Let $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be a vector commitment scheme over alphabet Σ with length ℓ. We say that VC is position-binding, if for every PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{A},\mathsf{VC}}(\lambda) := \Pr\left[\begin{array}{r} m \neq m' \\ \wedge\ \mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, m, \tau) = 1 \\ \wedge\ \mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, m', \tau') = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{com}, i, m, \tau, m', \tau') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right].$$

Definition 17 (**NP**-Relation)**.** Let $\mathcal{R} = (\mathcal{R}_\lambda)_\lambda$ be a family of binary relations $\mathcal{R}_\lambda \subseteq \{0,1\}^* \times \{0,1\}^*$. We define the language of yes-instances $\mathcal{L}_\lambda$ via

$$\mathcal{L}_\lambda := \left\{\mathsf{stmt} \in \{0,1\}^* \;\middle|\; \exists\ \mathsf{witn} \in \{0,1\}^* : (\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}_\lambda\right\}.$$

We say that $\mathcal{R}$ is an **NP**-relation, if the following properties hold:

- There exists a polynomial poly, such that for any $\mathsf{stmt} \in \mathcal{L}_\lambda$, we have $|\mathsf{stmt}| \leq \mathsf{poly}(\lambda)$.
- Membership in $\mathcal{R}_\lambda$ is efficiently decidable, i.e. there exists a deterministic polynomial time algorithm that decides $\mathcal{R}_\lambda$.
- There is a polynomial poly' such that for all $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}_\lambda$ we have $|\mathsf{witn}| \leq \mathsf{poly}'(|\mathsf{stmt}|)$.

Definition 18 (Non-Interactive Argument of Knowledge)**.** Let $\mathcal{R}$ be an **NP**-relation. A non-interactive argument of knowledge for $\mathcal{R}$ is a tuple $\mathsf{PS} = (\mathsf{Setup}, \mathsf{PProve}, \mathsf{PVer})$ of PPT algorithms with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$ takes as input the security parameter, and outputs a common reference string crs.
- $\mathsf{PProve}(\mathsf{crs}, \mathsf{stmt}, \mathsf{witn}) \to \pi$ takes as input a common reference string crs, a statement stmt, and a witness witn, and outputs a proof π.
- $\mathsf{PVer}(\mathsf{crs}, \mathsf{stmt}, \pi) \to b$ is deterministic, takes as input a common reference string crs, a statement stmt, a proof π, and outputs a bit $b \in \{0,1\}$.

We require that the following properties hold:

- **Completeness.** For all $\mathsf{crs} \in \mathsf{Setup}(1^\lambda)$, and all $(\mathsf{stmt}, \mathsf{witn}) \in \mathcal{R}_\lambda$, we have

  $$\Pr\left[\mathsf{PVer}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1 \mid \pi \leftarrow \mathsf{PProve}(\mathsf{crs}, \mathsf{stmt}, \mathsf{witn})\right] = 1.$$

- **Knowledge Soundness.** There is a PPT algorithm Ext, such that for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:

  $$\mathsf{Adv}^{\mathsf{kn\text{-}sound}}_{\mathcal{A},\mathsf{PS},\mathsf{Ext}}(\lambda) := \Pr\left[(\mathsf{stmt}, \mathsf{witn}) \notin \mathcal{R}_\lambda \wedge \mathsf{PVer}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1 \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{witn} \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{stmt}, \pi). \end{array}\right].$$

  We say that Ext is the knowledge extractor of PS.

Definition 19 (Homomorphic Hash Function)**.** Let $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda, \mathcal{D} = \{\mathcal{D}_\lambda\}_\lambda, \mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ be families of sets, such that for each λ, $\mathcal{D}_\lambda$ and $\mathcal{R}_\lambda$ are abelian groups. We denote both group operations additively. A homomorphic hash function family with key space $\mathcal{K}$, domain $\mathcal{D}$, and range $\mathcal{R}$ is a pair $\mathsf{HF} = (\mathsf{Gen}, \mathsf{Eval})$ of PPT algorithms, with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$ takes as input the security parameter, and outputs a hash key $\mathsf{hk} \in \mathcal{K}_\lambda$.

- $\mathsf{Eval}(\mathsf{hk}, x) \to y$ is deterministic, takes as input a hash key $\mathsf{hk} \in \mathcal{K}_\lambda$, and an element $x \in \mathcal{D}_\lambda$, and outputs an element $y \in \mathcal{R}_\lambda$.

Further, we require that the following properties holds:

- **Homomorphism.** For any $\mathsf{hk} \in \mathsf{Gen}(1^\lambda)$, and all $x, x' \in \mathcal{D}_\lambda$, we have

$$\mathsf{Eval}(\mathsf{hk}, x + x') = \mathsf{Eval}(\mathsf{hk}, x) + \mathsf{Eval}(\mathsf{hk}, x').$$

- **Collision-Resistance.** For any EPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{coll}}_{\mathcal{A},\mathsf{HF}}(\lambda) := \Pr\left[\begin{array}{c} x \neq x' \\ \wedge\ \mathsf{Eval}(\mathsf{hk}, x) = \mathsf{Eval}(\mathsf{hk}, x') \end{array} \middle| \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ (x, x') \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right].$$

For simplicity, we omit the subscript λ and write $\mathcal{K}, \mathcal{D}, \mathcal{R}$ instead of $\mathcal{K}_\lambda, \mathcal{D}_\lambda, \mathcal{R}_\lambda$, if λ is clear from the context.

B Some Useful Bounds

Lemma 23 (Chernoff Bound). *Let $X_1, \ldots, X_t$ be independent random variables with values in $\{0, 1\}$. Let $\delta \geq 0$. Then, we have*

$$\Pr\left[\sum_{i=1}^{t} X_i \leq (1 - \delta)\mu\right] \leq \exp(-\delta^2\mu/2), \ \textit{for } \mu := \mathbb{E}\left[\sum_{i=1}^{t} X_i\right].$$

Lemma 24. *Let $N, D, L \in \mathbb{N}$ with $D, L \leq N$, $L \leq N - \Delta$. Then, we have*

$$\binom{N-D}{L} \Big/ \binom{N}{L} \leq \left(1 - \frac{D}{N}\right)^L.$$

Proof. We have

$$\binom{N-D}{L} \Big/ \binom{N}{L} = \frac{(N-D)! \cdot L! \cdot (N-L)!}{L! \cdot (N-D-L)! \cdot N!} = \prod_{i=0}^{L-1} \frac{N-D-i}{N-i}$$
$$= \prod_{j=N-L+1}^{N} \frac{j-D}{j} = \prod_{j=N-L+1}^{N} 1 - \frac{D}{j} \leq \prod_{j=N-L+1}^{N} 1 - \frac{D}{N} = \left(1 - \frac{D}{N}\right)^L.$$

□

C Omitted Details from Section 3

C.1 Omitted Details from Section 3.1

The following lemma shows that data availability sampling, in particular the consistency property, implies a collision-resistant hash function induced by the mapping from data to com via algorithm Encode, given that the commitment com is smaller than the data data.

Lemma 25. *Let $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2), \mathsf{Ext})$ be a data availability sampling scheme with threshold $T \in \mathbb{N}$. For any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}) + 2\mathbf{T}(\mathsf{Encode}) + 2T\mathbf{T}(\mathsf{V}_1)$ and*

$$\Pr\left[\begin{array}{c} \mathsf{data}_1 \neq \mathsf{data}_2 \\ \wedge\ \ \mathsf{com}_1 = \mathsf{com}_2 \end{array} \middle| \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{data}_1, \mathsf{data}_2) \leftarrow \mathcal{A}(\mathsf{par}), \\ (\pi_1, \mathsf{com}_1) := \mathsf{Encode}(\mathsf{data}_1), \\ (\pi_2, \mathsf{com}_2) := \mathsf{Encode}(\mathsf{data}_2). \end{array}\right] \leq \mathsf{Adv}^{\mathsf{cons}}_{\mathcal{B},T,T,\mathsf{DAS}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be a PPT algorithm that on input par outputs $(\mathsf{data}_1, \mathsf{data}_2)$ such that there are π_1, π_2 with $(\pi_1, \mathsf{com}) = \mathsf{Encode}(\mathsf{data}_1)$ and $(\pi_2, \mathsf{com}) = \mathsf{Encode}(\mathsf{data}_2)$. Then, we construct an algorithm $\mathcal{B}$ against consistency of DAS as follows:

- When $\mathcal{B}$ gets as input par, it runs $(\mathsf{data}_1, \mathsf{data}_2) \leftarrow \mathcal{A}(\mathsf{par})$.
- Then, it computes $(\pi_1, \mathsf{com}_1) := \mathsf{Encode}(\mathsf{data}_1)$ and $(\pi_2, \mathsf{com}_2) := \mathsf{Encode}(\mathsf{data}_2)$. If $\mathsf{data}_1 = \mathsf{data}_2$ or $\mathsf{com}_1 \neq \mathsf{com}_2$, $\mathcal{B}$ aborts. Otherwise, it sets $\mathsf{com} := \mathsf{com}_1 = \mathsf{com}_2$.
- Next, $\mathcal{B}$ runs $\mathsf{tran}_{j,i} \leftarrow \mathsf{V}_1^{\pi_j,Q}(\mathsf{com})$ for all $i \in [T]$ and $j \in \{1, 2\}$
- Finally, $\mathcal{B}$ outputs $(\mathsf{com}, (\mathsf{tran}_{1,i})_{i=1}^{T}, (\mathsf{tran}_{2,i})_{i=1}^{T})$.

We claim that, except with negligible probability, $\mathcal{B}$ breaks consistency. Namely, by completeness of DAS, with overwhelming probability the following event holds for both $j \in \{1, 2\}$:

$$\mathsf{data}_k = \mathsf{Ext}(\mathsf{tran}_{j,1}, \ldots, \mathsf{tran}_{j,\ell_j}).$$

As $\mathsf{data}_1 \neq \mathsf{data}_2$, $\mathcal{B}$ breaks consistency. □

C.2 Extension: Repairability

Definition 20 (Repairable DAS)**.** Let $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2), \mathsf{Ext})$ be a data availability sampling scheme with encoding alphabet Σ, data length $K \in \mathbb{N}$, and encoding length $N \in \mathbb{N}$. We say that DAS is (L, ℓ)-repairable, if there is a deterministic polynomial time algorithm Repair, with the following syntax and properties:

- $\mathsf{Repair}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \to \bar{\pi}/\bot$ takes as input a commitment com, a list of transcripts tran_i, and outputs an encoding $\bar{\pi} \in \Sigma^N$ or an abort symbol $\bot$.
- **Repair Liveness.** Let $\mathcal{A}$ be a stateful algorithm and consider the following experiment $\mathcal{G}$:
  1. Run $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathsf{com} \leftarrow \mathcal{A}(\mathsf{par})$.
  2. Run $(\mathsf{tran}_i)_{i=1}^{L} \leftarrow \mathsf{Interact}\left[\mathsf{V}_1, \mathcal{A}\right]_{Q,L}(\mathsf{com})$ and $b_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}_i)$ for all $i \in [L]$.
  3. Run $(i_j)_{j=1}^{\ell} \leftarrow \mathcal{A}(\mathsf{tran}_1, \ldots, \mathsf{tran}_L)$.
  4. Run $\bar{\pi} \leftarrow \mathsf{Repair}(\mathsf{com}, \mathsf{tran}_{i_1}, \ldots, \mathsf{tran}_{i_\ell})$.
  5. For all $i \in [L]$, run $\mathsf{tran}'_i \leftarrow \mathsf{V}_1^{\bar{\pi},Q}(\mathsf{com})$ and $b'_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}'_i)$.

  Then, we require that for any stateful PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{repairlive}}_{\mathcal{A},L,\ell,\mathsf{DAS},\mathsf{Repair}}(\lambda) := \Pr_{\mathcal{G}}\left[\forall j \in [\ell] : b_{i_j} = 1 \wedge \exists i \in [L] :\ b'_i = 0\right].$$

On Soundness and Consistency. One may wonder why we do not define any consistency or soundness property for a scenario where clients interact with a repaired codeword. We claim that this is not needed, as our consistency and soundness notions for data availability sampling schemes are robust enough to cover such scenarios. The intuition is that whatever scenario could happen including algorithm Repair and violate soundness or consistency, could be simulated by an adversary in the soundness or consistency game, respectively.

C.3 Extension: Local Accessibility

Definition 21 (Locally Accessible DAS)**.** Let $\mathsf{DAS} = (\mathsf{Setup}, \mathsf{Encode}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2), \mathsf{Ext})$ be a data availability sampling scheme with data alphabet Γ, encoding alphabet Σ, data length $K \in \mathbb{N}$, and encoding length $N \in \mathbb{N}$. We say that DAS is locally accessible with query complexity L, if there is a PPT algorithm Access, with the following syntax and properties:

- $\mathsf{Access}^{\pi,L}(\mathsf{com}, i) \to d/\bot$ takes as input a commitment com, and an index $i \in [K]$, gets L-time oracle access to an encoding $\pi \in \Sigma^N$, and outputs a symbol $d \in \Gamma$ or an abort symbol $\bot$.

- **Local Access Completeness.** For any $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, any $i \in [K]$, and all $\mathsf{data} \in \Gamma^K$, we have

$$\Pr\left[d = \mathsf{data}_i \;\middle|\; \begin{array}{l} (\pi, \mathsf{com}) := \mathsf{Encode}(\mathsf{data}), \\ d \leftarrow \mathsf{Access}^{\pi,L}(\mathsf{com}, i) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Local Access Consistency.** For any stateful PPT algorithm $\mathcal{A}$, any index $i \in [K]$, and any integer $\ell = \mathsf{poly}(\lambda)$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{acc\text{-}cons}}_{\mathcal{A},i,\ell,\mathsf{DAS},\mathsf{Access}}(\lambda) := \Pr\left[\mathsf{data} \neq \bot \wedge d \neq \bot \wedge d \neq \mathsf{data}_i \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{com} \leftarrow \mathcal{A}(\mathsf{par}), \\ d \leftarrow \mathsf{Access}^{\mathcal{A},L}(\mathsf{com}, i), \\ (\mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \leftarrow \mathcal{A}(\mathsf{par}), \\ \mathsf{data} := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \ldots, \mathsf{tran}_\ell) \end{array} \right].$$

D Omitted Details from Section 5

Lemma 26. *Let $\mathcal{C}_r \colon \mathbb{F}^{k_r} \to \mathbb{F}^{n_r}$ and $\mathcal{C}_c \colon \mathbb{F}^{k_c} \to \mathbb{F}^{n_c}$ be linear erasure codes with reception efficiencies t_r, t_c, respectively. Then, $\mathcal{C}_r \otimes \mathcal{C}_c \colon \mathbb{F}^{k_r \cdot k_c} \to \mathbb{F}^{n_r \cdot n_c}$ is an erasure code with reception efficiency*

$$t = n_c n_r - (n_c - t_c + 1)(n_r - t_r + 1) + 1.$$

Proof. We want to reconstruct data $\mathbf{M} \in \mathbb{F}^{k_c \times k_r}$ given a set of symbols of $\mathbf{X} = \mathbf{G}_c \mathbf{M} \mathbf{G}_r^\top \in \mathbb{F}^{n_c \times n_r}$. Let $\mathcal{X} \subseteq [n_c] \times [n_r]$ be the set of indices of these symbols in $\mathbf{X}$, i.e., for each $(i,j) \in \mathcal{X}$, we know $\mathbf{X}_{i,j} \in \mathbb{F}$. We say that a row $i \in [n_c]$ (resp. column $j \in [n_r]$) is saturated if we have at least t_r (resp. t_c) symbols, i.e., $|\mathcal{X} \cap \{i\} \times [n_r]| \geq t_r$ (resp. $\mathcal{X} \cap [n_c] \times \{j\} \geq t_c$). Clearly, if a row (resp. column) is saturated, we can reconstruct the entire row (resp. column) using reconstruction of the codes $\mathcal{C}_r$ (resp. $\mathcal{C}_c$). Now, assume there is no way in which we can reconstruct $\mathbf{M}$. If at least t_c rows are saturated, we can reconstruct the entire matrix, contradicting our assumption. Thus, assume that at most $t_c - 1$ rows are saturated. Each saturated row has at most n_r symbols in $\mathcal{X}$. There are $n_c - (t_c - 1)$ remaining rows, all of which are not saturated. Each of those has at most $t_r - 1$ symbols in $\mathcal{X}$. Thus, we have at most $(t_c - 1)n_r + (n_c - t_c + 1)(t_r - 1)$ symbols in $\mathcal{X}$. In summary, if we can not reconstruct $\mathbf{M}$, then $\mathcal{X}$ has size at most $(t_c - 1)n_r + (n_c - t_c + 1)(t_r - 1)$, which can be simplified to $n_c n_r - (n_c - t_c + 1)(n_r - t_r + 1) + 1$. □

E Additional Notions for Erasure Code Commitments

In this section, we define additional notions for erasure code commitment schemes that are helpful in some cases.

E.1 Message-Bound Openings

We formally define the notion of message-bound openings for erasure code commitment schemes. To recall, this notion is used when proving repairability of the resulting data availability sampling scheme, see Section 6.3.

Definition 22 (Message-Bound Openings). Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for an erasure code $\mathcal{C}$ with reception efficiency t and reconstruction algorithm $\mathsf{Reconst}$. We say that CC has message-bound openings, if for every PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{A},\mathsf{CC}}(\lambda) :=$$

$$\Pr\left[\begin{array}{ll} & |I_0| \geq t \wedge |I_1| \geq t \wedge \bot \notin \{m_0, m_1\} \\ \wedge & m_0 = m_1 \\ \wedge & \forall i \in I_0 : \mathsf{Ver}(\mathsf{ck}, \mathsf{com}_0, i, \hat{m}_{0,i}, \tau_{0,i}) = 1 \\ \wedge & \forall i \in I_1 : \mathsf{Ver}(\mathsf{ck}, \mathsf{com}_1, i, \hat{m}_{1,i}, \tau_{1,i}) = 1 \\ \wedge & \exists i \in I_1 : \mathsf{Ver}(\mathsf{ck}, \mathsf{com}_0, i, \hat{m}_{1,i}, \tau_{1,i}) = 0 \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ \begin{pmatrix} (\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0}) \\ (\mathsf{com}_1, (\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1}) \end{pmatrix} \leftarrow \mathcal{A}(\mathsf{ck}), \\ m_0 := \mathsf{Reconst}((\hat{m}_{0,i})_{i \in I_0}), \\ m_1 := \mathsf{Reconst}((\hat{m}_{1,i})_{i \in I_1}) \end{array} \right].$$

E.2 Computational Uniqueness

We define the notion of computational uniqueness for erasure code commitments and study its implications.

Definition 23 (Computational Uniqueness)**.** Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for an erasure code $\mathcal{C}$ with reception efficiency t and reconstruction algorithm $\mathsf{Reconst}$. We say that CC is computationally unique, if for every PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{A},\mathsf{CC}}(\lambda) :=$$

$$\Pr\left[\begin{array}{ll|l} & |I_0| \geq t \wedge |I_1| \geq t \wedge \bot \notin \{m_0, m_1\} \wedge m_0 = m_1 & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ \wedge & \forall i \in I_0 : \mathsf{Ver}(\mathsf{ck}, \mathsf{com}_0, i, \hat{m}_{0,i}, \tau_{0,i}) = 1 & \begin{pmatrix}(\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0}) \\ (\mathsf{com}_1, (\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1})\end{pmatrix} \leftarrow \mathcal{A}(\mathsf{ck}), \\ \wedge & \forall i \in I_1 : \mathsf{Ver}(\mathsf{ck}, \mathsf{com}_1, i, \hat{m}_{1,i}, \tau_{1,i}) = 1 & m_0 := \mathsf{Reconst}((\hat{m}_{0,i})_{i \in I_0}), \\ \wedge & \mathsf{com}_0 \neq \mathsf{com}_1 & m_1 := \mathsf{Reconst}((\hat{m}_{1,i})_{i \in I_1}) \end{array}\right].$$

We show that computational uniqueness implies both message-bound openings and code-binding. Remark that the converse direction is not true. Message-bound openings do not imply computational uniqueness.

Lemma 27. *Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ such that CC is computationally unique. Then, CC has message-bound openings. 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*

$$\mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq \mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{B},\mathsf{CC}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary against the message-bound openings property of CC. We construct an adversary $\mathcal{B}$ against computational uniqueness as follows:

1. $\mathcal{B}$ gets as input a commitment key ck. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input ck to get commitments and openings $(\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0})$ and $(\mathsf{com}_1, (\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1})$.
2. $\mathcal{B}$ outputs $(\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0})$ and $(\mathsf{com}_1, (\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1})$.

Note that if $\mathsf{com}_0 = \mathsf{com}_1$, the adversary $\mathcal{A}$ trivially loses the message-bound opening game. Thus, if $\mathcal{A}$ wins in the message-bound openings game, $\mathcal{B}$ wins the computational uniqueness game. □

Lemma 28. *Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an MDS code. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ such that CC is computationally unique and satisfies position-binding. Then, CC satisfies code-binding. Concretely, for any PPT algorithm $\mathcal{A}$ there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A}), \mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$, and*

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_1,\mathsf{CC}}(\lambda) + \mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{B}_2,\mathsf{CC}}(\lambda).$$

Proof. We first recall the code-binding game for an adversary $\mathcal{A}$ as in the statement. The adversary $\mathcal{A}$ first gets a freshly sampled commitment key $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$. Then, it outputs a commitment and a few openings. Denote them by $(\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0})$, where $I_0 \subseteq [n]$ is the set of positions for which the adversary opens the commitment. Then, $\mathcal{A}$ breaks code-binding, if all openings verify, i.e., $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_0, i, \hat{m}_{0,i}, \tau_{0,i}) = 1$ for all $i \in I_0$, and there is no codeword in $\mathcal{C}$ that is consistent with these openings $\hat{m}_{0,i}$. Our proof is as follows. We first oberve that $|I_0| > k$ has to hold, as $\mathcal{C}$ is an MDS code. Next, let $R \subset I_0$ be the set of the first k of the openings. Further, let $m = \mathsf{Reconst}((\hat{m}_{0,i})_{i \in R})$, $\hat{m}_1 = \mathcal{C}(m)$, and $(\mathsf{com}_1, St) \leftarrow \mathsf{Com}(\mathsf{ck}, m)$. We know that $m \neq \bot$ as $\mathcal{C}$ is an MDS code. In other words, $\hat{m}_1 \in \mathcal{C}$ is the unique codeword consistent with the openings in R. Now, we can consider two cases. In the first case, $\mathsf{com}_1 = \mathsf{com}_0$. In this case, we break position-binding. This is because there has to be at least one $i^* \in I_0$ with $\hat{m}_{0,i^*} \neq \hat{m}_{1,i^*}$, as otherwise the openings output by $\mathcal{A}$ would be consistent with the codeword $\hat{m}_1$. A reduction can just compute an opening for $\hat{m}_{1,i^*}$ honestly and use it in combination with $\hat{m}_{0,i^*}, \tau_{0,i^*}$ to break position-binding. In the second case, $\mathsf{com}_1 \neq \mathsf{com}_0$. Here, we break computational uniqueness. Namely, a reduction can output com_0 with all openings output by the adversary in R, and output com_1 with enough honestly computed openings. We omit a more formal exposition of these two reductions. □

E.3 Extractability

We define the notion of extractability for erasure code commitment schemes, and study its implications. We start with the formal definition.

Definition 24 (Extractable CC)**.** Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ such that Com is deterministic, and use the notation $\mathsf{com} = \widehat{\mathsf{Com}}(\mathsf{ck}, m)$ for $(\mathsf{com}, St) = \mathsf{Com}(\mathsf{ck}, m)$. We say that CC is extractable, if there is a PPT algorithm Ext, such that for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{extr}}_{\mathcal{A},\mathsf{Ext},\mathsf{CC}}(\lambda) := \Pr\left[\begin{array}{rl|l} & \mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}, \tau) = 1 & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda), \\ \wedge & \widehat{\mathsf{Com}}(\mathsf{ck}, m) \neq \mathsf{com} & (\mathsf{com}, i, \hat{m}, \tau) \leftarrow \mathcal{A}(\mathsf{ck}), \\ & & m \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}, i, \hat{m}, \tau) \end{array}\right].$$

Next, we show that extractability is a strong notion, in a sense that, in combination with position-binding, it implies code-binding and computational uniqueness.

Lemma 29. *Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ such that Com is deterministic. Further, assume that CC is position-binding and extractable. Then, CC is computationally unique. Concretely, for any PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}, \mathcal{B}'$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}), \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A})$, and*

$$\mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an algorithm that breaks computational uniqueness of CC. That is, it gets as input a commitment key ck and outputs commitments com_0 and com_1 as well as two sets of openings $(\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0}$ and $(\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1}$. It breaks computational uniqueness if $\mathsf{com}_0 \neq \mathsf{com}_1$, all openings verify, i.e., for all $b \in \{0,1\}$ and all $i \in I_b$ we have $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}_b, i, \hat{m}_{b,i}, \tau_{b,i}) = 1$, and both sets of openings reconstruct to the same message, i.e., $|I_0| \geq t$, $|I_1| \geq t$, and for $m_0 := \mathsf{Reconst}((\hat{m}_{0,i})_{i \in I_0})$ and $m_1 := \mathsf{Reconst}((\hat{m}_{1,i})_{i \in I_1})$ we have $m_0 = m_1$ and both are not $\bot$. To prove that $\mathcal{A}$ can not win, our strategy is to extract two messages that commit to com_0 and com_1, respectively. For that, we use the extractability of CC. Then, we argue that they have to be the same. More precisely, we run $m_0^* \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}_0, i_0, \hat{m}_{0,i_0}, \tau_{0,i_0})$ for some arbitrary, say the first, $i_0 \in I_0$, and $m_1^* \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}_1, i_1, \hat{m}_{1,i_1}, \tau_{1,i_1})$ for some arbitrary, say the first, $i_1 \in I_1$, where Ext is the extractor that exists by extractability of CC. Extractability tells us that $\widehat{\mathsf{Com}}(\mathsf{ck}, m_0^*) = \mathsf{com}_0$ and $\widehat{\mathsf{Com}}(\mathsf{ck}, m_1^*) = \mathsf{com}_1$, except with probability $2 \cdot \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}}(\lambda)$ for some reduction $\mathcal{B}$. Thus, as soon as we can show that $m_0^* = m_1^*$, we know that $\mathcal{A}$ can not break computational uniqueness. To show this, we define $\tilde{m}_0 := \mathcal{C}(m_0)$ and $\tilde{m}_1 := \mathcal{C}(m_1)$. Using a reduction $\mathcal{B}'$ to position-binding, we can argue that $\hat{m}_{b,i} = \tilde{m}_{b,i}$ for both $b \in \{0,1\}$ and each $i \in I_b$, except with probability $\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}}(\lambda)$. Thus, we have

$$m_0^* = \mathsf{Reconst}((\tilde{m}_{0,i})_{i \in I_0}) = \mathsf{Reconst}((\hat{m}_{0,i})_{i \in I_0}) = m_0.$$

Analogously, we can show that $m_1^* = m_1$. As $m_0 = m_1$, we can conclude that $m_0^* = m_1^*$. □

Lemma 30. *Let $\mathcal{C}\colon \Gamma^k \to \Lambda^n$ be an erasure code. Let $\mathsf{CC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ be an erasure code commitment scheme for $\mathcal{C}$ such that Com is deterministic. Further, assume that CC is position-binding and extractable. Then, CC is code-binding. Concretely, for any PPT algorithm $\mathcal{A}$, there are PPT algorithms $\mathcal{B}, \mathcal{B}'$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}), \mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A})$, and*

$$\mathsf{Adv}^{\mathsf{c\text{-}uniq}}_{\mathcal{A},\mathsf{CC}}(\lambda) \leq \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}}(\lambda).$$

Proof. We only sketch the proof, as it is very similar to the proof of Lemma 29. Let $\mathcal{A}$ be an adversary against code-binding of CC. That is, $\mathcal{A}$ gets as input a commitment key $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and outputs a commitment and a few openings. We denote them by $(\mathsf{com}, (\hat{m}_i, \tau_i)_{i \in I})$, where $I \subseteq [n]$ is the set of positions for which the adversary opens the commitment. The adversary $\mathcal{A}$ breaks code-binding if all openings verify, and there is no codeword that is consistent with the openings. Now, our goal is to break position-binding with a reduction $\mathcal{B}'$. For that, $\mathcal{B}'$ first runs the extractor, namely, it runs $m^* \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}, i_0, \hat{m}_{i_0}, \tau_{i_0})$ for some arbitrary, say the first, $i_0 \in I$. Except with probability $\mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}}(\lambda)$ for some reduction $\mathcal{B}$, we have that $\widehat{\mathsf{Com}}(\mathsf{ck}, m^*) = \mathsf{com}$. Then, setting $\hat{m}^* := \mathcal{C}(m^*)$, reduction $\mathcal{B}'$ can compute valid openings τ_i^* for $\hat{m}^*$ with respect to com for any position $i \in [n]$. As no codeword is consistent with $\mathcal{A}$'s openings $\hat{m}_i$, we know that there is at least one $i^* \in I$ for which $\hat{m}^*_{i^*} \neq \hat{m}_{i^*}$. Reduction $\mathcal{B}'$ can thus output $\mathsf{com}, i^*, \hat{m}^*_{i^*}, \tau^*_{i^*}, \hat{m}_{i^*}, \tau_{i^*}$ to break position-binding of CC. □

Lemma 31 (Informal)**.** *The KZG polynomial commitment scheme [KZG10] is extractable in the algebraic group model.*

Proof Sketch. Suppose $\mathcal{A}$ is an algebraic algorithm running in the extractability game. It gets as input a commitment key $g, g^s, \ldots, g^{s^{k-1}}$ for some degree bound $k-1$. Then, it outputs a commitment com, elements $x, y \in \mathbb{Z}_p$, and an opening τ. As both com and τ are group elements and $\mathcal{A}$ is algebraic, it also outputs coefficients α_i and γ_i such that $\mathsf{com} = \prod_{i=0}^{k-1} \left(g^{s^i}\right)^{\alpha_i}$ and $\tau = \prod_{i=0}^{k-1} \left(g^{s^i}\right)^{\gamma_i}$. The extractor can now just output the polynomial defined by coefficients α_i. □

F Omitted Details from Section 6

F.1 Omitted Details from Section 6.1

Proof of Lemma 2. Let $\mathcal{A}$ be an adversary against reconstruction-binding of CC. That is, $\mathcal{A}$ outputs $(\mathsf{com}, (\hat{m}_i, \tau_i)_{i \in I}, (\hat{m}'_i, \tau'_i)_{i \in I'})$ on input $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$. We distinguish two cases by defining the following events:

- Event Win: This event occurs if $\mathcal{A}$ breaks reconstruction-binding, i.e. for $m := \mathsf{Reconst}((\hat{m}_i)_{i \in I})$ and $m' := \mathsf{Reconst}((\hat{m}'_i)_{i \in I'})$ we have $|I| \geq t$,$|I'| \geq t$, $m \neq m'$, $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau_i) = 1$ for all $i \in I$ and $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}'_i, \tau'_i) = 1$ for all $i \in I'$.
- Event $\mathsf{BreakPosBind}$: This event occurs if there is an index $i^* \in I \cap I'$ such that $\hat{m}_{i^*} \neq \hat{m}'_{i^*}$.

Clearly, we can write

$$\mathsf{Adv}^{\mathsf{rec\text{-}bind}}_{\mathcal{A},k,\mathsf{CC}}(\lambda) = \Pr[\mathsf{Win}] = \Pr[\mathsf{Win} \land \mathsf{BreakPosBind}] + \Pr[\mathsf{Win} \land \neg\mathsf{BreakPosBind}].$$

We bound these terms individually. First, consider the event $\mathsf{Win} \land \mathsf{BreakPosBind}$. Note that if this event occurs, then especially τ_{i^*} and τ'_{i^*} are valid openings for $\hat{m}_{i^*} \neq \hat{m}'_{i^*}$, respectively. Therefore, we can easily bound the probability of this event using a reduction $\mathcal{B}_1$ that breaks position-binding of CC as follows: On input ck, $\mathcal{B}_1$ runs $\mathcal{A}(\mathsf{ck})$ and gets $(\mathsf{com}, (\hat{m}_i, \tau_i)_{i \in I}, (\hat{m}'_i, \tau'_i)_{i \in I'})$. Then, $\mathcal{B}_1$ checks if event $\mathsf{Win} \land \mathsf{BreakPosBind}$ occurs, which can be done efficiently. If it does, $\mathcal{B}_1$ outputs $(\mathsf{com}, i^*, \hat{m}_{i^*}, \tau_{i^*}, \hat{m}'_{i^*}, \tau'_{i^*})$, where i^* is the index in the definition of $\mathsf{BreakPosBind}$. It is easy to see that $\mathcal{B}_1$ breaks position-binding if event $\mathsf{Win} \land \mathsf{BreakPosBind}$ occurs, and the running time of $\mathcal{B}_1$ is dominated by running $\mathcal{A}$.

Next, we bound the probability of $\mathsf{Win} \land \neg\mathsf{BreakPosBind}$. Assume that this event occurs. Then we know that for all $i \in I \cap I'$ we have $\hat{m}_i = \hat{m}'_i$, i.e. $\hat{m}$ and $\hat{m}'$ are consistent on $I \cap I'$. We can define

$$\hat{m}^*_i := \begin{cases} \hat{m}_i, & \text{if } i \in I \setminus I', \\ \hat{m}_i = \hat{m}'_i, & \text{if } i \in I \cap I', \\ \hat{m}'_i, & \text{if } i \in I' \setminus I \end{cases} \text{ for all } i \in I \cup I'.$$

Note that for all $\hat{m}^*_i, i \in I \cup I'$, we have valid openings τ_i, as Win occurs. We claim that there is no $m^* \in \Gamma^k$, such that the codeword $c = \mathcal{C}(m^*)$ is consistent with $(\hat{m}^*_i)_{i \in I \cup I'}$. Once this is established, the reduction $\mathcal{B}_2$ breaking code-binding by outputting $(\hat{m}^*_i)_{i \in I \cup I'}$ is clear. Assume towards contradiction that such an $m^* \in \Gamma^k$ exists. Then by completeness of algorithm $\mathsf{Reconst}$, and because both $(\hat{m}_i)_{i \in I}$ and $(\hat{m}'_i)_{i \in I'}$ are a subsequence of $c = \mathcal{C}(m^*)$, we have $\mathsf{Reconst}_k((\hat{m}_i)_{i \in I}) = m^* = \mathsf{Reconst}((\hat{m}'_i)_{i \in I'})$. A contradiction. □

F.2 Omitted Details from Section 6.2

Proof of Lemma 3. We want to analyze the quality of algorithm $\mathsf{Sample}_{\mathsf{wr}}$ that samples indices uniformly at random with replacement. Recall that for that, we have to upper bound the probability that ℓ invocations of $\mathsf{Sample}_{\mathsf{wr}}(1^Q, 1^N)$ jointly sample at most Δ distinct indices in $[N]$. To this end, consider the experiment $(i_{l,j})_{j\in[Q]} \leftarrow \mathsf{Sample}_{\mathsf{wr}}(1^Q, 1^N)$ for each $l \in [\ell]$ as in the definition of index samplers. For each subset $I \subseteq [N]$ with $|I| \leq \Delta$, let E_I be the event that the sampled indices $i_{l,j}$ are all in I. Then, it is clear that

$$\Pr_{\mathcal{G}}\left[\,\left|\bigcup_{l\in[\ell]}\{i_{l,j} \mid j\in[Q]\}\right| \leq \Delta\,\right] \leq \sum_{I\subseteq[N],\ |I|\leq\Delta} \Pr\left[E_I\right].$$

Now, we fix one such subset I. The probability of E_I is at most

$$\left(\frac{|I|}{N}\right)^{Q\ell} \leq \left(\frac{\Delta}{N}\right)^{Q\ell},$$

because all $Q\ell$ indices are sampled independently. As there are $\binom{N}{\Delta}$ such subsets, the first part of the lemma follows. To obtain the simpler bound, we use the fact

$$\forall n \in \mathbb{N}:\ \forall k \in [n]:\quad \binom{n}{k} < \left(\frac{n\cdot e}{k}\right)^k.$$

Then, we have

$$\binom{N}{\Delta}\left(\frac{\Delta}{N}\right)^{Q\ell} < \frac{N^{\Delta}\cdot e^{\Delta}}{\Delta^{\Delta}}\cdot\frac{\Delta^{Q\ell}}{N^{Q\ell}} = e^{\Delta}\cdot N^{\Delta-Q\ell}\cdot\Delta^{Q\ell-\Delta}.$$

Now, we use $c := \Delta/N$, and conclude with

$$e^{\Delta}\cdot N^{\Delta-Q\ell}\cdot\Delta^{Q\ell-\Delta} = e^{\Delta}\cdot N^{\Delta-Q\ell}\cdot(cN)^{Q\ell-\Delta} \leq e^{\Delta}\cdot c^{Q\ell-\Delta} \leq c^{\log_c(e)\Delta+Q\ell-\Delta} \leq c^{Q\ell-(1-\log_c(e))\Delta}.$$

□

Proof of Lemma 4. We analyze the locality of sampling uniformly with replacement. For that, consider the experiment $(i_j)_{j\in[Q]} \leftarrow \mathsf{Sample}_{\mathsf{wr}}(1^Q, 1^N)$ and define the set $\Gamma := \{\mathcal{S}(i_j) \mid j \in [Q]\}$. Then, we need to upper bound the probability that Γ is of size at most D. For that, fix any subset $I \subset \mathbb{N}$ of size D. Using a union bound, we can as well upper bound the probability of $\Gamma \subseteq I$. As all indices i_j are sampled independently from $[N]$, and $\mathcal{S}$ is a Q-to-1 mapping onto a set of size N/Q, the probability that $I \subset \mathbb{N}$ is $(D/(N/Q))^Q$. In combination, we have

$$\begin{aligned}\Pr\left[|\Gamma| \leq D\right] &= \binom{N/Q}{D}\cdot\Pr\left[\Gamma\subseteq I\right] = \binom{N/Q}{D}\cdot\left(\frac{D}{N/Q}\right)^Q\\ &< \left(\frac{e\cdot N/Q}{D}\right)^D\cdot\left(\frac{D}{N/Q}\right)^Q = e^D\cdot\left(\frac{D}{N/Q}\right)^{Q-D},\end{aligned}$$

where we used the fact

$$\forall n \in \mathbb{N}:\ \forall k \in [n]:\quad \binom{n}{k} < \left(\frac{n\cdot e}{k}\right)^k.$$

□

Proof of Lemma 5. We want to analyze the quality of sampling uniformly without replacement. Recall that for that, we have to upper bound the probability that ℓ invocations of $\mathsf{Sample}_{\mathsf{wor}}(1^Q, 1^N)$ jointly sample at most Δ distinct indices in $[N]$. Consider the experiment $(i_{l,j})_{j\in[Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$ for each $l \in [\ell]$ as in the definition of index samplers. For each subset $I \subseteq [N]$ with $|I| = \Delta$, let E_I be the event that the sampled indices $i_{l,j}$ are all in I. Then, we have

$$\Pr\left[\,\left|\bigcup_{l\in[\ell]}\{i_{l,j} \mid j\in[Q]\}\right| \leq \Delta\,\right] \leq \sum_{I\subseteq[N],\ |I|\leq\Delta} \Pr\left[E_I\right].$$

Now, fix one such subset I. The probability of E_I is at most

$$\left(\binom{\Delta}{Q}\Big/\binom{N}{Q}\right)^{\ell}.$$

This is because each of the ℓ invocations of the sampler samples uniformly from $\binom{[N]}{Q}$, and all invocations are independent. As there are $\binom{N}{\Delta}$ such subsets I, the claim follows. □

Proof of Lemma 6. It is clear that the claim holds for $N \mod Q \neq 0$. Thus, assume that N is a multiple of Q and set $N' := N/Q$. Now, observe that ℓ copies of $\mathsf{Sample}_{\mathsf{seg}}$ output at most Δ distinct indices, if and only of they sample at most $\Delta' := \Delta/Q$ distinct segments $\mathsf{seg}_1, \dots, \mathsf{seg}_\ell \in [N']$. We can view the sampling of $\mathsf{seg}_1, \dots, \mathsf{seg}_\ell$ as ℓ independent executions of $\mathsf{Sample}_{\mathsf{wr}}(1^1, 1^{N'})$, which shows the claim. □

F.3 Omitted Details from Section 6.3

Proof of Lemma 11. Let $\mathcal{A}$ be an algorithm against local access consistency, and let $i_0 \in [K]$. We first recall the local access consistency game. In this game, $\mathcal{A}$ first gets parameters $\mathsf{par} = \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ as input. Then, it outputs a commitment com. Next, algorithm Access is run on input com, i_0 and with oracle access to $\mathcal{A}$. The algorithm outputs d after making exactly one query to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs transcripts $(\mathsf{tran}_1, \dots, \mathsf{tran}_\ell)$ and $\mathsf{data} := \mathsf{Ext}(\mathsf{com}, \mathsf{tran}_1, \dots, \mathsf{tran}_\ell)$ is run. The adversary $\mathcal{A}$ breaks local access consistency, if $\mathsf{data} \neq \bot$, $d \neq \bot$, and $d \neq \mathsf{data}_{i_0}$. Intuitively, this means that the outputs of Access and Ext are not consistent. Now, let us introduce some notation. As in algorithm Ext, write $\mathsf{tran}_l := (i_{l,j}, \widehat{\mathsf{data}}_{l,i_{l,j}}, \tau_{l,i_{l,j}})_{j\in[Q]}$ for each $l \in [L]$, and define the set $I \subseteq [N]$ of indices $i \in [N]$ such that there is a $(l,j) \in [L] \times [Q]$ with $i_{l,j} = i$. Further, let $(\widehat{\mathsf{data}}'_{\hat{i}_0}, \tau'_{\hat{i}_0})$ be the result of the query that Access made. Assuming $\mathsf{data} \neq \bot$, we know that $|I| \geq t$, by definition of Ext. Define the index $i_1 := \hat{i}_0$ if $\hat{i}_0 \in I$ and $i_1 \in I$ arbitrary if $\hat{i}_0 \notin I$. Then, define the set $I' := (I \setminus \{i_1\}) \cup \{\hat{i}_0\}$. Clearly, we have $|I'| = |I| \geq t(K)$. For each $i \in I$, define $\widehat{\mathsf{data}}_i$ exactly as in algorithm Ext. Further, for each $i \in I$ define τ_i to be the corresponding opening such that $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \widehat{\mathsf{data}}_i, \tau_i) = 1$. For each $i \in I \setminus \{i_1\}$, set $\widehat{\mathsf{data}}'_i := \widehat{\mathsf{data}}_i$ and $\tau'_i := \tau_i$. Now, we claim that $(\mathsf{com}, (\widehat{\mathsf{data}}_i, \tau_i)_{i\in I}, (\widehat{\mathsf{data}}'_i, \tau'_i)_{i\in I'})$ is an output with which a reduction $\mathcal{B}$ can break reconstruction-binding of CC. Clearly, a reduction can compute this output. Further, we $|I| \geq t$ and $|I'| \geq t$, as already observed. As $\mathsf{data} \neq \bot$ and $d \neq \bot$, we know that all openings are valid, i.e. $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau_i) = 1$ and $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}'_i, \tau'_i) = 1$ for all $i \in I, i' \in I'$. Finally, we know that the i_0th symbol of $m := \mathsf{Reconst}((\hat{m}_i)_{i\in I})$ and the i_0th symbol of $m' := \mathsf{Reconst}((\hat{m}'_i)_{i\in I'})$ are distinct. This is because the i_0th symbol of m' is d by the second property generalized systematic encoding, and the i_0th symbol of m is data_{i_0} by definition of Ext. □

Proof of Lemma 12. Let $\mathcal{A}$ be an adversary against the (L,ℓ)-repairability of $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$. We first recall the repair liveness game. In this game, parameters $\mathsf{par} := \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ are generated and $\mathcal{A}$ is run on input par. Then, $\mathcal{A}$ outputs a commitment com. After that, L copies of $\mathsf{V}_1(\mathsf{com})$ are run, where their oracle queries are answered by $\mathcal{A}$. Let $\mathsf{tran}_1, \dots, \mathsf{tran}_L$ denote the resulting transcripts that they output, and $b_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}_i)$ for all $i \in [L]$. Then, $\mathcal{A}$ gets to pick a subset $\{i_1, \dots, i_\ell\} \subseteq [L]$ of ℓ of these transcripts and algorithm Repair is run on input $\mathsf{com}, \mathsf{tran}_{i_1}, \dots, \mathsf{tran}_{i_\ell}$. It outputs a new encoding $\bar{\pi}$ or $\bot$. If it does not output $\bot$, all L clients are run again, i.e., $\mathsf{tran}'_i \leftarrow \mathsf{V}_1^{\bar{\pi},Q}(\mathsf{com})$ and $b'_i := \mathsf{V}_2(\mathsf{com}, \mathsf{tran}'_i)$ for all $i \in [L]$. The adversary $\mathcal{A}$ breaks repair liveness, if for all $j \in [L]$, we have $b_{i_j} = 1$, i.e., all selected clients accepted before the repairing took place, but there is some $i \in [L]$ with $b'_i = 0$. The latter includes the case where Repair output $\bot$. We will now distinguish two cases, captured by the following two events.

- Event $\mathsf{RepairBot}$: This event occurs, if Repair outputs $\bot$ and the adversary breaks repair liveness.
- Event $\mathsf{RepairSucc}$: This event occurs, if Repair does not output $\bot$, i.e., it outputs an encoding $\bar{\pi}$, and the adversary breaks repair liveness.

Clearly, we have

$$\mathsf{Adv}^{\mathsf{repairlive}}_{\mathcal{A},L,\ell,\mathsf{DAS}[\mathsf{CC},\mathsf{Sample}],\mathsf{Repair}}(\lambda) \leq \Pr\left[\mathsf{RepairBot}\right] + \mathsf{RepairSucc}.$$

We will bound the probability of both events separately. Let us start with event $\mathsf{RepairBot}$. Recall that algorithm Repair internally runs $\mathsf{Ext}(\mathsf{com}, \mathsf{tran}_{i_1}, \dots, \mathsf{tran}_{i_\ell})$, and only outputs $\bot$ if Ext does. Therefore, if event $\mathsf{RepairBot}$ occurs, the adversary first received parameters, then it output a commitment, and then it found ℓ accepting out of L transcripts, such that these do not suffice to reconstruct the data. Intuitively, this means that the adversary breaks subset-soundness of $\mathsf{DAS}[\mathsf{CC}, \mathsf{Sample}]$. Indeed, one can make this intuition formal. We only sketch a reduction $\mathcal{B}_1$ here. A reduction that runs in the subset-soundness game gets as input par and forwards them to $\mathcal{A}$. Then, it gets a commitment com and outputs it to the subset-soundness game. It simulates the interaction with L copies of V_1 by forwarding between $\mathcal{A}$ and the subset-soundness game. Finally, it forwards $\mathcal{A}$'s selection of indices $i_1, \dots, i_\ell$ to the subset-soundness game. One can easily see that this reduction breaks (L, ℓ)-subset-soundness if event $\mathsf{RepairBot}$ occurs. We have

$$\Pr\left[\mathsf{RepairBot}\right] \leq \mathsf{Adv}^{\mathsf{sub\text{-}sound}}_{\mathcal{B}_1, L, \ell, \mathsf{DAS}}(\lambda).$$

Next, we want to bound the probability of event $\mathsf{RepairSucc}$. Intuitively, if this event occurs, then Ext run within Repair was able to reconstruct data $\overline{\mathsf{data}}$, and thus $(\bar{\pi}, \overline{\mathsf{com}}) = \mathsf{Encode}(\overline{\mathsf{data}})$, but the new encoding $\bar{\pi}$ does not verify with respect to the initial commitment com. If we recall the structure of an encoding, i.e., each symbol consists of an opening for the erasure code commitment com, then we see that the adversary must intuitively break message-bound openings in this case. More precisely, this works as follows. Because $\widehat{\mathsf{data}}$ was extracted by Ext, we know by definition of Ext that the transcripts $\mathsf{tran}_{i_1}, \dots, \mathsf{tran}_{i_\ell}$ contain at least t valid symbols and openings $\widehat{\mathsf{data}}_i, \tau_i$ for commitment com. Due to completeness, the new encoding $\bar{\pi}$ contains at least t valid openings $\bar{\tau}_i$ for $\widehat{\overline{\mathsf{data}}}_i$ and commitment $\overline{\mathsf{com}}$. Here $\widehat{\overline{\mathsf{data}}} := \mathcal{C}(\overline{\mathsf{data}})$ as computed in Encode by algorithm Repair. Assuming adversary breaks repair liveness, we know that one of the clients after the repairing rejects, i.e., $b'_i = 0$ for some $i \in [L]$. By definition of V_2, this means that at least one of the new valid openings, say the jth, contained in the new encoding $\bar{\pi}$ does not work with the old commitment com. More precisely, letting $(\widehat{\overline{\mathsf{data}}}_j, \bar{\tau}_j)$ be the jth symbol of $\bar{\pi}$, we know that $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, j, \widehat{\overline{\mathsf{data}}}_j, \bar{\tau}_j) = 0$. This leads to a reduction $\mathcal{B}_2$ that breaks the message-bound openings property of CC. The reduction gets as input the commitment key ck and runs $\mathcal{A}$ in the repair liveness game with $\mathsf{par} := \mathsf{ck}$. If event $\mathsf{RepairSucc}$ occurs, the reduction outputs $\mathsf{com}, (\widehat{\mathsf{data}}_j, \tau_j)_j$ and $\overline{\mathsf{com}}, (\widehat{\overline{\mathsf{data}}}_j, \bar{\tau}_i)_j$. We have

$$\Pr\left[\mathsf{RepairSucc}\right] \leq \mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{B}_2, \mathsf{CC}}(\lambda).$$

□

G Omitted Details from Section 7

Proof of Lemma 14. We prove the statement via a sequence of games. Let $\mathcal{A}$ be an algorithm in the code-binding game of $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$.

Game $\mathbf{G}_0$: We define $\mathbf{G}_0$ to be the code-binding game of $\mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]$. That is, adversary $\mathcal{A}$ gets as input $\mathsf{ck} = (\mathsf{ck}_{\mathsf{VC}}, \mathsf{crs}, \rho)$ generated as in Setup and outputs $(\mathsf{com}, (\hat{m}_i, \tau_i)_{i \in I})$ to break code-biding. The game outputs 1 if all τ_i are valid openings for $\hat{m}_i$,, i.e. $\mathsf{Ver}(\mathsf{ck}, \mathsf{com}, i, \hat{m}_i, \tau_i) = 1$ for all $i \in I$, and there is no m such that the $\hat{m}_i$ are compatible with $\mathcal{C}(m)$. By definition, we have

$$\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] = \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A}, \mathsf{CC}[\mathcal{C}, \mathsf{VC}, \mathsf{PS}]}(\lambda).$$

Game $\mathbf{G}_1$: This game is defined as $\mathbf{G}_0$, with an additional check at the end. Namely, after $\mathcal{A}$ outputs $(\mathsf{com}, (\hat{m}_i, \tau_i)_{i \in I})$, the game parses $\mathsf{com} = (\mathsf{com}_{\mathsf{VC}}, \pi)$ and sets $\mathsf{stmt} := (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC}}, \rho)$. Then, it runs $\mathsf{witn} \leftarrow \mathsf{PS.Ext}(\mathsf{crs}, \mathsf{stmt}, \pi)$. It returns 0 if we have $(\mathsf{stmt}, \mathsf{witn}) \notin \mathcal{R}$ and $\mathsf{PVer}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1$. Otherwise, it returns whatever $\mathbf{G}_0$ would have returned. It is clear that the difference between $\mathbf{G}_0$ and $\mathbf{G}_1$ can be bounded using a straight-forward reduction $\mathcal{B}_1$ that breaks knowledge soundness of PS. We have

$$\left|\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_1 \Rightarrow 1\right]\right| \leq \mathsf{Adv}^{\mathsf{kn\text{-}sound}}_{\mathcal{B}_1, \mathsf{PS}, \mathsf{PS.Ext}}(\lambda).$$

Finally, we bound the probability that $\mathbf{G}_1$ outputs 1 using a reduction $\mathcal{B}_2$ that breaks position-binding of VC. The intuition is as follows: If $\mathbf{G}_1$ outputs 1, we extracted witness $\mathsf{witn} = m$ such that for $\hat{m}^* := \mathcal{C}(m)$ we have $(\mathsf{com}_{\mathsf{VC}}, St_{\mathsf{VC}}) = \mathsf{VC.Com}(\mathsf{ck}_{\mathsf{VC}}, \hat{m}^*; \rho)$ for some state St_{VC}, due to the definition of relation $\mathcal{R}$. If $\mathbf{G}_1$ outputs 1, then in particular $\mathcal{A}$ breaks code-binding. Thus, there must be some index i such that the returned $\hat{m}_i$ is different from $\hat{m}_i^*$. By completeness of VC, we can use St_{VC} to compute a valid opening τ_i for $\hat{m}_i^*$ for commitment $\mathsf{com}_{\mathsf{VC}}$. Now, we have valid openings for $\mathsf{com}_{\mathsf{VC}}$ for two different symbols $\hat{m}_i \neq \hat{m}_i^*$ at position i, i.e. we break position-binding. It is trivial to turn this intuition into a formal reduction $\mathcal{B}_2$, which gets as input $\mathsf{ck}_{\mathsf{VC}}$, simulates $\mathbf{G}_1$ for $\mathcal{A}$, and outputs $\hat{m}_i, \hat{m}_i^*$ along with their respective openings. We have

$$\Pr[\mathbf{G}_1 \Rightarrow 1] \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_2,\mathsf{VC}}(\lambda).$$

□

Proof of Lemma 15. We prove the statement via a sequence of games.

Game $\mathbf{G}_0$: Game $\mathbf{G}_0$ is the message-bound openings game. Recall that in this game, $\mathcal{A}$ outputs $\mathsf{com}_0, (\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0}$ and $\mathsf{com}_1, (\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1}$ on input $\mathsf{ck} = (\mathsf{ck}_{\mathsf{VC}}, \mathsf{crs}, \rho)$. The game $\mathbf{G}_0$ outputs 1, i.e., $\mathcal{A}$ breaks the message-bound openings property of CC, if both sets of openings $(\hat{m}_{0,i}, \tau_{0,i})_{i \in I_0}$ and $(\hat{m}_{1,i}, \tau_{1,i})_{i \in I_1}$ allow to reconstruct the same message, the openings verify with respect to their respective commitments, but the openings in I_1 do not all verify with respect to com_0. Recall that com_0 and com_1 have the form $\mathsf{com}_0 = (\mathsf{com}_{\mathsf{VC},0}, \pi_0)$ and $\mathsf{com}_1 = (\mathsf{com}_{\mathsf{VC},1}, \pi_1)$, respectively. It will be our goal to show that the vector commitments $\mathsf{com}_{\mathsf{VC},0}, \mathsf{com}_{\mathsf{VC},1}$ are the same. It is clear from the construction that this implies that $\mathcal{A}$ can not win. We have

$$\mathsf{Adv}^{\mathsf{mb\text{-}open}}_{\mathcal{A},\mathsf{CC}}(\lambda) = \Pr[\mathbf{G}_0 \Rightarrow 1].$$

Game $\mathbf{G}_1$: In $\mathbf{G}_1$, we use the knowledge extractor $\mathsf{PS.Ext}$ of PS to extract witnesses from the proofs π_0 and π_1 contained in com_0 and com_1. Namely, the game runs $m_0 \leftarrow \mathsf{PS.Ext}(\mathsf{crs}, (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC},0}, \rho), \pi_0)$ and $m_1 \leftarrow \mathsf{PS.Ext}(\mathsf{crs}, (\mathsf{ck}_{\mathsf{VC}}, \mathsf{com}_{\mathsf{VC},1}, \rho), \pi_1)$. The game outputs 0 if the first component of $\mathsf{VC.Com}(\mathsf{ck}_{\mathsf{VC}}, \mathcal{C}(m_0); \rho)$ is not $\mathsf{com}_{\mathsf{VC},0}$ or the first component of $\mathsf{VC.Com}(\mathsf{ck}_{\mathsf{VC}}, \mathcal{C}(m_1); \rho)$ is not $\mathsf{com}_{\mathsf{VC},1}$. Otherwise, $\mathbf{G}_1$ behaves as $\mathbf{G}_0$. Clearly, the difference between games $\mathbf{G}_0$ and $\mathbf{G}_1$ can bounded using the knowledge soundness of PS, i.e., we have a reduction $\mathcal{B}_1$ with

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq 2 \cdot \mathsf{Adv}^{\mathsf{kn\text{-}sound}}_{\mathcal{B}_1,\mathsf{PS},\mathsf{PS.Ext}}(\lambda).$$

Game $\mathbf{G}_2$: Game $\mathbf{G}_2$ is as game $\mathbf{G}_1$, but with an additional modification of the winning condition. Namely, if there is a $b \in \{0,1\}$, and an $i \in I_b$ such that $\hat{m}_{b,i} \neq \mathcal{C}(m_b)_i$, then the game outputs 0. Otherwise, it behaves as $\mathbf{G}_1$. Here, recall that $\hat{m}_{b,i}$ is part of the opening that $\mathcal{A}$ outputs, and m_b is the message that the game extracts from π_b as described in $\mathbf{G}_1$. Note that for $\hat{m}_{b,i}$, $\mathcal{A}$ also outputs an opening $\tau_{b,i}$ that verifies with respect to $\mathsf{com}_{\mathsf{VC},b}$. Also, a reduction can obtain a valid opening for $C(m_b)_i$ using ρ. Thus, we can easily construct a reduction $\mathcal{B}_2$ that breaks position-binding of VC if $\mathcal{A}$ can distinguish $\mathbf{G}_1$ and $\mathbf{G}_2$. We have

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}_2,\mathsf{VC}}(\lambda).$$

We can now argue that the probability that $\mathbf{G}_2$ outputs 1 is zero. For that, observe that

$$m_0 = \mathsf{Reconst}((\hat{m}_{0,i})_{i \in I_0}) = \mathsf{Reconst}((\hat{m}_{1,i})_{i \in I_1}) = m_1,$$

where the second equality follows from the winning condition of message-bound openings, and the first and last equality follow from correctness of $\mathsf{Reconst}$. From that, we get that the vector commitments $\mathsf{com}_{\mathsf{VC},0}$ and $\mathsf{com}_{\mathsf{VC},1}$ contained in com_0 and com_1 are the same. One can observe that in this case, $\mathcal{A}$ can never win, i.e.,

$$\Pr[\mathbf{G}_2 \Rightarrow 1] = 0.$$

□

H Omitted Details from Section 8

Proof of Lemma 16. Let $\mathcal{A}$ be an adversary against position-binding of $\mathsf{CC}^{\otimes}$. We give a reduction $\mathcal{B}$ that runs $\mathcal{A}$ internally, and breaks position-binding of CC_c if $\mathcal{A}$ breaks position-binding of $\mathsf{CC}^{\otimes}$. Namely, $\mathcal{B}$ gets as input a commitment key ck and runs $\mathcal{A}$ on input ck. Then, $\mathcal{A}$ terminates with output $(\mathsf{com}, j, \hat{m}, \tau, \hat{m}', \tau')$. Finally, $\mathcal{B}$ writes $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$, sets $(i^*, j^*) := \mathsf{ToMatIdx}(j)$, and outputs

$(\mathsf{com}_{j^*}, i^*, \hat{m}, \tau, \hat{m}', \tau')$. Clearly, $\mathcal{B}$ perfectly simulates the position-binding game for $\mathcal{A}$, and its running time is dominated by the running time of $\mathcal{A}$. Assuming that $\mathcal{A}$ breaks position-binding, we know that $\hat{m} \neq \hat{m}'$, and by definition of $\mathsf{Ver}^{\otimes}$, we have $\mathsf{Ver}^{\otimes}(\mathsf{ck}, \mathsf{com}_{j^*}, i^*, \hat{m}, \tau) = 1$ and $\mathsf{Ver}^{\otimes}(\mathsf{ck}, \mathsf{com}_{j^*}, i^*, \hat{m}', \tau') = 1$. This means that $\mathcal{B}$ breaks position-binding of CC_c. □

Proof of Lemma 17. We prove computational uniqueness by showing a simpler yet stronger statement. Namely, let $\mathcal{A}$ be a PPT algorithm. Assume that $\mathcal{A}$ gets as input a commitment key $\mathsf{ck} \leftarrow \mathsf{Setup}_c(1^\lambda)$ and outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$, and some openings. We denote these openings by $\mathbf{X}_{i,j} \in \mathbb{F}, \tau_{i,j}$ for $(i,j) \in I \subseteq [n_c] \times [n_r]$, where I denotes the set of indices for which $\mathcal{A}$ opens the commitment. Further, assume the following three conditions hold:

- The size of I is at least the reception efficiency t of $\mathcal{C}_r \otimes \mathcal{C}_c$.
- The reconstruction algorithm of $\mathcal{C}_r \otimes \mathcal{C}_c$ does not output $\perp$. The output is $\mathbf{m} \in \mathbb{F}^{k_c k_r}$, which defines a matrix $\mathbf{M} \in \mathbb{F}^{k_c \times k_r}$.
- All openings $\mathbf{X}_{i,j} \in \mathbb{F}, \tau_{i,j}$ are valid according to $\mathsf{Ver}^{\otimes}$.

In this case, we have (except with some negligible probability δ) that for all $j \in [n_r]$ we have $\widehat{\mathsf{Com}}_c(\mathsf{ck}, (\mathbf{M}\mathbf{G}_r^\top)_j) = \mathsf{com}_j$, where $(\mathbf{M}\mathbf{G}_r^\top)_j$ denotes the jth column of $\mathbf{M}\mathbf{G}_r$. It can easily be observed that this statement implies computational uniqueness and the advantage against computational uniqueness is bounded by 2δ. The rest of the proof is dedicated to showing this statement. We do so by providing a sequence of games.

Game $\mathbf{G}_0$: We start with $\mathbf{G}_0$, which models the setting above. Namely, in game $\mathbf{G}_0$, the game first samples $\mathsf{ck} \leftarrow \mathsf{Setup}_c(1^\lambda)$. Then, it runs $\mathcal{A}$ on input ck. As a result $\mathcal{A}$ outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$ and openings $\mathbf{X}_{i,j} \in \mathbb{F}, \tau_{i,j}$ for $(i,j) \in I \subseteq [n_c] \times [n_r]$ as above. The game outputs 1 if the three conditions from above hold, but there is a $j \in [n_r]$ such that $\widehat{\mathsf{Com}}_c(\mathsf{ck}, (\mathbf{M}\mathbf{G}_r^\top)_j) \neq \mathsf{com}_j$. Our goal is to upper bound

$$\delta := \Pr[\mathbf{G}_0 \Rightarrow 1].$$

Before we continue with the next game, we introduce the set $I^* \subseteq [n_r]$, which is defined as

$$I^* := \{j \in [n_r] \mid \exists i \in [n_r] : \ (i,j) \in I\}.$$

Intuitively, I^* corresponds the set of columns in which the adversary opened any index. It is easy to see that I^* contains at least k_r elements if $\mathbf{G}_0$ outputs 1.

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we additionally run the extractor Ext of the commitment scheme CC_c a few times. Precisely, after obtaining the output from $\mathcal{A}$, the game does the following for each $j \in I^*$: It first tries to extract a preimage of the commitment com_j via $\mathbf{Y}_j \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}_j, i, \mathbf{X}_{i,j}, \tau_{i,j})$, where $i \in [n_r]$ is the first index such that $(i,j) \in I$. Here, we have $\mathbf{Y}_j \in \mathbb{F}^{k_c}$. Then, the game outputs 0 and terminates if for $(\mathsf{com}, St) = \mathsf{Com}_c(\mathsf{ck}, \mathbf{Y}_j)$ we have $\mathsf{com} \neq \mathsf{com}_j$. Finally, if the game did not yet terminate after having done this for all $j \in I^*$, it returns whatever $\mathbf{G}_0$ would return. Clearly, games $\mathbf{G}_0$ and $\mathbf{G}_1$ only differ if extraction fails, i.e., $\mathcal{A}$ manages to output an opening $\mathbf{X}_{i,j}, \tau_{i,j}$ as above which verifies with respect to commitment com_j, but for which Ext does not output a correct preimage $\mathbf{Y}_j$. A straight-forward reduction $\mathcal{B}$ against extractability of CC_c shows

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}_c}(\lambda).$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we introduce a bad event and let the game abort if it occurs. To define the bad event, we first recall that during verification of $\mathcal{A}$'s output, vectors $\mathbf{a} \in \mathbb{F}^{n_r - k_r}$ are sampled uniformly, and the equation $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) \neq \sum_{i=1}^{n_r} \mathbf{h}_j \cdot \mathsf{com}_j$ is checked, where $\mathbf{h} = \mathbf{H}^\top \mathbf{a}$. Written differently, it is checked that $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) = \mathsf{com}\ \mathbf{H}^\top \mathbf{a}$.

- Event LinCol: This event occurs, if $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) = \mathsf{com}\ \mathbf{H}^\top \mathbf{a}$, but there is a column of $\mathsf{com}\ \mathbf{H}^\top$ which is not equal to $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0})$, where $\mathbf{a} \in \mathbb{F}^{n_r - k_r}$ is sampled uniformly during verification (see algorithm $\mathsf{Ver}^{\otimes}$).

We can easily bound the probability of LinCol. For that, observe that if a column of $\mathsf{com}\ \mathbf{H}^\top$ is not equal to $\widehat{\mathsf{Com}}_c(\mathsf{ck},\mathbf{0})$, then the map $\mathbf{a} \mapsto \mathsf{com}\ \mathbf{H}^\top\mathbf{a}$ is a non-zero homomorphism from $\mathbb{F}^{n_r-k_r}$ to the commitment space. As $\mathbf{a}$ is sampled uniformly and independent of everything else, the probability that it ends up being in the kernel of this map is at most $1/|\mathbb{F}|$. We have

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \Pr[\mathsf{LinCol}] \leq \frac{1}{|\mathbb{F}|}.$$

Next, we introduce some notation. Namely, we define the set $\mathcal{H} \subseteq [n_r]$ to be the first k_r indices in I^*. Further, we define the set $\mathcal{W} := [n_r] \setminus \mathcal{H}$ of the remaining indices. Having defined the sets $\mathcal{H}$ and $\mathcal{W}$, we now define certain matrices and vectors:

- Consider the parity-check matrix $\mathbf{H} \in \mathbb{F}^{(n_r-k_r)\times n_r}$ of $\mathcal{C}_r$. We split $\mathbf{H}$ into two matrices $\mathbf{H}_\mathcal{H} \in \mathbb{F}^{(n_r-k_r)\times k_r}$ and $\mathbf{H}_\mathcal{W} \in \mathbb{F}^{(n_r-k_r)\times(n_r-k_r)}$. This is done in the following way: The matrix $\mathbf{H}_\mathcal{H}$ contains all columns with indices in $\mathcal{H}$, and the matrix $\mathbf{H}_\mathcal{W}$ contains all columns with indices in $\mathcal{W}$. Both are ordered in the canonical way. Observe that because of our assumption that $\mathcal{C}_r$ is an MDS code, we know that $\mathbf{H}_\mathcal{W}$ and $\mathbf{H}_\mathcal{W}^\top$ are invertible.

- We partition the commitments $\mathsf{com}_j, j \in [n_r]$ in the same way into $\mathsf{com}_\mathcal{H} = (\mathsf{com}_j)_{j\in\mathcal{H}}$ and $\mathsf{com}_\mathcal{W} = (\mathsf{com}_j)_{j\in\mathcal{W}}$.

- Recall from $\mathbf{G}_1$ that the game extracts vectors $\mathbf{Y}_j \in \mathbb{F}^{k_c}$ for every $j \in I^*$. In particular, it extracts $\mathbf{Y}_j$ for every $j \in \mathcal{H} \subseteq I^*$. We arrange these $\mathbf{Y}_j$ for $j \in \mathcal{H}$ into a matrix $\mathbf{Y}_\mathcal{H} \in \mathbb{F}^{k_c\times k_r}$. Further, we define the matrix
$$\mathbf{Y}_\mathcal{W} := -\mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top \left(\mathbf{H}_\mathcal{W}^\top\right)^{-1}.$$
Also, we define the matrix $\mathbf{Y} \in \mathbb{F}^{k_c\times n_r}$ by merging $\mathbf{Y}_\mathcal{H}$ and $\mathbf{Y}_\mathcal{W}$ in the natural way, i.e., the columns in $\mathcal{H}$ of $\mathbf{Y}$ are filled by $\mathbf{Y}_\mathcal{H}$ and the columns in $\mathcal{W}$ are filled by $\mathbf{Y}_\mathcal{W}$, both by respecting the natural order.

- We encode the matrix $\mathbf{Y}$ that we just defined using the code $\mathcal{C}_c$. That is, we define a matrix $\hat{\mathbf{X}} := \mathbf{G}_c\mathbf{Y} \in \mathbb{F}^{n_c\times n_r}$.

The intuition is as follows: The matrix $\mathbf{Y}_\mathcal{W}$ completes the extracted $\mathbf{Y}_\mathcal{H}$ into a matrix with rows in the code. The matrix is consistent with the commitments and openings output by $\mathcal{A}$, as we will show. We continue by making this intuition formal in the following claims.

Claim 1. Consider the notations and assumptions from the proof of Lemma 17. Every row of $\mathbf{Y}$ and every row of $\hat{\mathbf{X}}$ is in the code $\mathcal{C}_r$.

We prove Claim 1. To do so, it is sufficient to show that $\mathbf{Y}\mathbf{H}^\top = \mathbf{0}$. Observe that

$$\begin{aligned}\mathbf{Y}\mathbf{H}^\top &= \mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top + \mathbf{Y}_\mathcal{W}\mathbf{H}_\mathcal{W}^\top\\ &= \mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top - \mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top\left(\mathbf{H}_\mathcal{W}^\top\right)^{-1}\mathbf{H}_\mathcal{W}^\top\\ &= \mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top - \mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top = \mathbf{0}.\end{aligned}$$

Claim 2. Consider the notations and assumptions from the proof of Lemma 17. Let $j \in [n_r]$ be arbitrary. Then for the jth column $\mathbf{Y}_j$ of $\mathbf{Y}$, we have that $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{Y}_j) = \mathsf{com}_j$.

To prove Claim 2, we first observe that by $\mathbf{G}_1$ and the definition of $\mathbf{Y}_\mathcal{H}$, the claim holds for all $j \in \mathcal{H}$. Thus, it remains to prove the claim for all $j \in \mathcal{W}$. For that, we first recall from $\mathbf{G}_2$, that every column of $\mathsf{com}\ \mathbf{H}^\top$ is equal to $\widehat{\mathsf{Com}}_c(\mathsf{ck},\mathbf{0})$. Using the homomorphic properties of $\widehat{\mathsf{Com}}_c(\mathsf{ck},\cdot)$, this implies that

$$\mathsf{com}_\mathcal{W}\ \mathbf{H}_\mathcal{W}^\top = -\mathsf{com}_\mathcal{H}\ \mathbf{H}_\mathcal{H}^\top.$$

Now, multiplying both sides with $\left(\mathbf{H}_\mathcal{W}^\top\right)^{-1}$, we have

$$\mathsf{com}_\mathcal{W} = -\mathsf{com}_\mathcal{H}\ \mathbf{H}_\mathcal{H}^\top\left(\mathbf{H}_\mathcal{W}^\top\right)^{-1}.$$

If we now look at one specific column $j \in \mathcal{W}$ of this equation, we have

$$\begin{aligned}\mathsf{com}_j &= -\mathsf{com}_{\mathcal{H}}\ \mathbf{H}_{\mathcal{H}}^{\top}\left(\mathbf{H}_{\mathcal{W}}^{\top}\right)_j^{-1}\\ &= \widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{Y}_{\mathcal{H}}\mathbf{H}_{\mathcal{H}}^{\top}\left(\mathbf{H}_{\mathcal{W}}^{\top}\right)_j^{-1}) = \widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{Y}_j),\end{aligned}$$

as desired.

Claim 3. Consider the notations and assumptions from the proof of Lemma 17. Let $(i, j) \in I$ be arbitrary. Then, we have $\mathbf{X}_{i,j} = \hat{\mathbf{X}}_{i,j}$, except with probability $\mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}_c}(\lambda)$, for a reduction $\mathcal{B}'$ with $\mathbf{T}(\mathcal{B}') \approx \mathbf{T}(\mathcal{A})$.

To see that Claim 3 holds, observe that a reduction simulating $\mathbf{G}_2$ knows a preimage $\mathbf{Y}_j$ for all commitments com_j output by the adversary (cf. Claim 2). Thus, in case $\mathbf{X}_{i,j} \neq \hat{\mathbf{X}}_{i,j}$ holds for some $(i, j) \in I$, a reduction can output com_j, the opening $\mathbf{X}_{i,j}, \tau_{i,j}$, and an opening for $\hat{\mathbf{X}}_{i,j}$ to break position-binding of CC_c. The reduction can compute the latter opening as it knows $\mathbf{Y}_j$.

Finally, we show how to use the three claims to argue that (except with the probability bounded by reduction $\mathcal{B}'$ in Claim 3) $\mathbf{G}_2$ does not output 1. This is done as follows. From Claim 1, we know that $\hat{\mathbf{X}} = \mathbf{G}_c\tilde{\mathbf{M}}\mathbf{G}_r^{\top}$ for some $\tilde{\mathbf{M}} \in \mathbb{F}^{k_c \times n_r}$. As $|I| \geq t$, we thus know that $\tilde{\mathbf{M}}$ is defined by the output by the reconstruction algorithm on input $(\hat{\mathbf{X}}_{i,j})_{(i,j)\in I}$. As $\mathbf{X}_{i,j} = \hat{\mathbf{X}}_{i,j}$ for all $(i, j) \in I$, we know that this input is the same as $(\mathbf{X}_{i,j})_{(i,j)\in I}$. An initial assumption (see $\mathbf{G}_0$) was that reconstructing from $(\mathbf{X}_{i,j})_{(i,j)\in I}$ yields $\mathbf{M}$. Therefore, we have $\tilde{\mathbf{M}} = \mathbf{M}$. Thus, we showed that $\mathbf{G}_c\mathbf{Y} = \hat{\mathbf{X}} = \mathbf{G}_c\mathbf{M}\mathbf{G}_r^{\top}$. As $\mathbf{G}_c$ induces an injective mapping, we have $\mathbf{Y} = \mathbf{M}\mathbf{G}_r^{\top}$. Thus, by Claim 2, we have that the jth column of $\mathbf{M}\mathbf{G}_r^{\top}$ commits to com_j for all $j \in [n_r]$, which is what we wanted to show. In summary, we showed that

$$\delta \leq \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B},\mathsf{Ext},\mathsf{CC}_c}(\lambda) + \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}',\mathsf{CC}_c}(\lambda) + \frac{1}{|\mathbb{F}|}.$$

□

Proof of Lemma 18. We want to prove that $\mathsf{CC}^{\otimes}$ is code-binding. For that, we consider two cases. Namely, either the adversary outputs openings such that in at least one column the openings are not consistent with any codeword, or it does the same for at least one row. Formally, consider the code-binding game of $\mathsf{CC}^{\otimes}$. In this game, first a key $\mathsf{ck} \leftarrow \mathsf{Setup}_c(1^{\lambda})$ is generated. Then, the adversary $\mathcal{A}$ gets this key and outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$, and some openings. We denote these openings by $\mathbf{X}_{i,j} \in \mathbb{F}, \tau_{i,j}$ for $(i, j) \in I \subseteq [n_c] \times [n_r]$, where I denotes the set of indices for which $\mathcal{A}$ opens the commitment. In terms of notation, we define $I_r(i)$ to be the set of indices in row i that are contained in I, and $I_c(j)$ to be the set of indices in column j that are contained in I. More formally, we set

$$I_r(i) := \{j \in [n_r] \mid (i, j) \in I\},\ I_c(j) := \{i \in [n_c] \mid (i, j) \in I\},$$

for each $i \in [n_c]$ and each $j \in [n_r]$ The adversary $\mathcal{A}$ breaks code-binding, if all openings verify, and there is no codeword that is consistent with these openings. Now, we define two events.

- Event $\mathsf{BreakCol}$: This event occurs, if all openings verify, and there is a column $j \in [n_r]$, such that no codeword of code $\mathcal{C}_c$ is consistent with the openings $\mathbf{X}_{i,j}$ for $i \in I_c(j)$.
- Event $\mathsf{BreakRow}$: This event occurs, if all openings verify, and there is a row $i \in [n_c]$, such that no codeword of code $\mathcal{C}_r$ is consistent with the openings $\mathbf{X}_{i,j}$ for $j \in I_r(i)$.

If $\mathcal{A}$ breaks code-binding, at least one of these two events must occur. Therefore, we have

$$\mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}^{\otimes}}(\lambda) \leq \Pr[\mathsf{BreakCol}] + \Pr[\mathsf{BreakRow} \wedge \neg\mathsf{BreakCol}].$$

Note that each column $j \in [n_r]$ is associated to a commitment com_j output by the adversary. Thus, if event $\mathsf{BreakCol}$ occurs, a reduction $\mathcal{B}$ can break code-binding of CC_c. The reduction is trivial and we omit it here. We have

$$\Pr[\mathsf{BreakCol}] \leq \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{B},\mathsf{CC}_c}(\lambda).$$

For the rest of the proof, we focus on bounding the probability of event $\mathsf{BreakRow} \wedge \neg\mathsf{BreakCol}$. That is, we need to argue that the adversary can not output openings of a row such that no codeword (in the code $\mathcal{C}_r$) is consistent with the openings. We prove this via a sequence of games. This is almost identical to the proof of Lemma 17, and we encourage the reader to read the proof of Lemma 17 first.

Game $\mathbf{G}_0$: Game $\mathbf{G}_0$ is exactly the code-binding game as above, with the modification that it outputs 1 if and only if event $\mathsf{BreakRow} \wedge \neg\mathsf{BreakCol}$ occurs. If it occurs, let $i^* \in [n_c]$ be the first row that triggers event $\mathsf{BreakRow}$. That is, let i^* be the first row for which no codeword of $\mathcal{C}_r$ is consistent with the openings $\mathbf{X}_{i^*,j}$ for $j \in I_r(i^*)$. By definition, we have

$$\Pr[\mathsf{BreakRow} \wedge \neg\mathsf{BreakCol}] = \Pr[\mathbf{G}_0 \Rightarrow 1].$$

The rest of the proof will not use the openings other than $\mathbf{X}_{i^*,j}$ for $j \in I_r(i^*)$.

Game $\mathbf{G}_1$: Game $\mathbf{G}_1$ is as $\mathbf{G}_0$. In addition, $\mathbf{G}_1$ runs the extractor Ext of the column commitment scheme CC_c a few times. Namely, when obtaining the output from $\mathcal{A}$, the game does the following for each $j \in I_r(i^*)$: It first runs $\mathbf{Y}_j \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}_j, i^*, \mathbf{X}_{i^*,j}, \tau_{i^*,j})$ to extract a preimage of com_j. We have $\mathbf{Y}_j \in \mathbb{F}^{k_c}$. The game outputs 0 and terminates if for $(\mathsf{com}, St) = \mathsf{Com}_c(\mathsf{ck}, \mathbf{Y}_j)$ we have $\mathsf{com} \neq \mathsf{com}_j$. Finally, if the game did not yet terminate after having done this for all $j \in I_r(i^*)$, it continues as $\mathbf{G}_0$ would do. The difference between $\mathbf{G}_0$ and $\mathbf{G}_1$ can easily be bounded using reduction $\mathcal{B}'$ against extractability of CC_c. We have

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{extr}}_{\mathcal{B}',\mathsf{Ext},\mathsf{CC}_c}(\lambda).$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, with an additional bad event on which the game aborts. Namely, recall that during verification of $\mathcal{A}$'s output, vectors $\mathbf{a} \in \mathbb{F}^{n_r-k_r}$ are sampled uniformly, and the game checks the equation $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) \neq \sum_{i=1}^{n_r} \mathbf{h}_j \cdot \mathsf{com}_j$, where $\mathbf{h} = \mathbf{H}^\top \mathbf{a}$. That is, it is checked that $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) = \mathsf{com}\ \mathbf{H}^\top \mathbf{a}$. We define event LinCol exactly as in $\mathbf{G}_2$ of the proof of Lemma 17.

- Event LinCol: This event occurs, if $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0}) = \mathsf{com}\ \mathbf{H}^\top \mathbf{a}$, but there is a column of $\mathsf{com}\ \mathbf{H}^\top$ which is not equal to $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{0})$, where $\mathbf{a} \in \mathbb{F}^{n_r-k_r}$ is sampled uniformly during verification (see algorithm $\mathsf{Ver}^{\otimes}$).

The probability of LinCol is at most $1/|\mathbb{F}|$, which can be seen as in the proof of Lemma 17. We have

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \Pr[\mathsf{LinCol}] \leq \frac{1}{|\mathbb{F}|}.$$

Next, we introduce some notation, which is similar to the notation in the proof of Lemma 17. The set $\mathcal{H} \subseteq [n_r]$ is defined to be the first k_r indices in $I_r(i^*)$. The set $\mathcal{W}$ is defined as $\mathcal{W} := [n_r] \setminus \mathcal{H}$. Recall that if the game outputs 1, the adversary output openings for all indices $j \in \mathcal{H}$ in row i^*. Additionally, it may have output openings for some indices in $\mathcal{W}$. We will later see that there is at least one index in $\mathcal{W}$ for which the adversary provided an opening. We now define certain matrices and vectors as in the proof of Lemma 17:

- Let $\mathbf{H} \in \mathbb{F}^{(n_r-k_r)\times n_r}$ be the parity-check matrix of $\mathcal{C}_r$. We split $\mathbf{H}$ into two matrices $\mathbf{H}_\mathcal{H} \in \mathbb{F}^{(n_r-k_r)\times k_r}$ and $\mathbf{H}_\mathcal{W} \in \mathbb{F}^{(n_r-k_r)\times(n_r-k_r)}$. This is done as follows: The matrix $\mathbf{H}_\mathcal{H}$ contains all columns with indices in $\mathcal{H}$, and the matrix $\mathbf{H}_\mathcal{W}$ contains all columns with indices in $\mathcal{W}$. Both are ordered in the canonical way. As $\mathcal{C}_r$ is an MDS code, we know that $\mathbf{H}_\mathcal{W}$ and $\mathbf{H}_\mathcal{W}^\top$ are invertible.
- We partition the commitments com_j, $j \in [n_r]$ in the same way into $\mathsf{com}_\mathcal{H} = (\mathsf{com}_j)_{j\in\mathcal{H}}$ and $\mathsf{com}_\mathcal{W} = (\mathsf{com}_j)_{j\in\mathcal{W}}$.
- Recall that the game extracts vectors $\mathbf{Y}_j \in \mathbb{F}^{k_c}$ for every $j \in I_r(i^*)$ (see $\mathbf{G}_1$). Especially, it extracts $\mathbf{Y}_j$ for every $j \in \mathcal{H} \subseteq I_r(i^*)$. We arrange these $\mathbf{Y}_j$ for $j \in \mathcal{H}$ into a matrix $\mathbf{Y}_\mathcal{H} \in \mathbb{F}^{k_c\times k_r}$. We define

  $$\mathbf{Y}_\mathcal{W} := -\mathbf{Y}_\mathcal{H}\mathbf{H}_\mathcal{H}^\top \left(\mathbf{H}_\mathcal{W}^\top\right)^{-1}.$$

  We define the matrix $\mathbf{Y} \in \mathbb{F}^{k_c\times n_r}$ by merging $\mathbf{Y}_\mathcal{H}$ and $\mathbf{Y}_\mathcal{W}$ in the natural way, i.e., the columns in $\mathcal{H}$ of $\mathbf{Y}$ are filled by $\mathbf{Y}_\mathcal{H}$ and the columns in $\mathcal{W}$ are filled by $\mathbf{Y}_\mathcal{W}$, both by respecting the natural order.
- We define a matrix $\hat{\mathbf{X}} := \mathbf{G}_c\mathbf{Y} \in \mathbb{F}^{n_c\times n_r}$.

The intuition is as follows: The matrix $\mathbf{Y}_{\mathcal{W}}$ completes the extracted $\mathbf{Y}_{\mathcal{H}}$ into a matrix with rows in the code, which is consistent with the commitments output by $\mathcal{A}$. As we assume that $\mathcal{A}$ breaks code-binding, we know that this completed matrix has to be different from the output of $\mathcal{A}$, which will allow us to break binding of CC_c. To make this intuition formal, we will show three claims.

Claim 4. Consider the notations and assumptions from the proof of Lemma 18. Every row of $\mathbf{Y}$ and every row of $\hat{\mathbf{X}}$ is in the code $\mathcal{C}_r$.

The proof of Claim 4 is identical to the proof of Claim 1.

Claim 5. Consider the notations and assumptions from the proof of Lemma 18. Let $j \in [n_r]$ be arbitrary. Then for the jth column $\mathbf{Y}_j$ of $\mathbf{Y}$, we have that $\widehat{\mathsf{Com}}_c(\mathsf{ck}, \mathbf{Y}_j) = \mathsf{com}_j$.

The proof of Claim 5 is identical to the proof of Claim 2.

Claim 6. Consider the notations and assumptions from the proof of Lemma 18. There is at least one $j^* \in \mathcal{W}$ such that $\mathcal{A}$ output an opening of index (i^*, j^*), i.e., $j^* \in I_r(i^*)$, and for this j^*, the opening $\mathbf{X}_{i^*,j^*} \in \mathbb{F}$ output by $\mathcal{A}$ is different from the element $\hat{\mathbf{X}}_{i^*,j^*}$.

To prove Claim 6, observe that if no $j^* \in \mathcal{W}$ is opened or all openings in $\mathcal{W}$ are consistent with $\hat{\mathbf{X}}$, then the opened indices in row i^* are consistent with the i^*th row of $\hat{\mathbf{X}}$. However, by Claim 4, the i^*th row of $\hat{\mathbf{X}}$ is in $\mathcal{C}_r$. This contradicts the definition of i^*.

Now that we made these observations, we can bound the probability that $\mathbf{G}_2$ outputs 1 using a reduction $\mathcal{B}''$ that breaks position-binding of CC_c. The reduction can be summarized as follows: It gets as input the commitment key ck and forwards it to $\mathcal{A}$. Once $\mathcal{A}$ outputs a commitment $\mathsf{com} = (\mathsf{com}_1, \ldots, \mathsf{com}_{n_r})$ and openings, $\mathcal{B}''$ does all the steps as in $\mathbf{G}_2$. If $\mathbf{G}_2$ outputs 1, $\mathcal{B}''$ knows the row i^*. It computes the matrix $\mathbf{Y}$ as defined above. Then, $\mathcal{B}''$ finds the index j^* as in Claim 6. Now, note that $\mathcal{B}''$ can break position-binding of CC_c by outputting com_{j^*}, the opening that $\mathcal{A}$ output, and an opening for $\hat{\mathbf{X}}_{i^*,j^*}$. Note that $\mathcal{B}''$ can compute this opening, because it knows the commitment preimage $\mathbf{Y}_{j^*}$ of com_{j^*} (see Claim 5). We have

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \leq \mathsf{Adv}^{\mathsf{pos\text{-}bind}}_{\mathcal{B}'',\mathsf{CC}_c}(\lambda).$$

□

I Omitted Details from Section 9

I.1 Omitted Details from Section 9.1

Lemma 32. *Let $\Delta \in [n]$. Let $\mathcal{A}$ be any stateful algorithm, and consider the following experiment $\mathcal{G}$:*

1. *Run $\mathcal{A}$ to obtain $\mathbf{X} \in \mathbb{F}^{k \times n}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ for $j \in [n]$ be the jth column of $\mathbf{X}$.*
2. *Sample a matrix $\mathbf{R} \leftarrow_{\$} \mathbb{F}^{P \times k}$.*
3. *Run $\mathcal{A}$ on input $\mathbf{R}$, and get a matrix $\mathbf{W} \in \mathbb{F}^{P \times n}$ from $\mathcal{A}$. Let $\mathbf{W}_j \in \mathbb{F}^P$ be the jth column of $\mathbf{W}$, for each $j \in [n]$.*
4. *Sample $J \leftarrow_{\$} \binom{[n]}{L}$ and set $\mathsf{Win} := 0$. If the following three conditions hold, set $\mathsf{Win} := 1$:*
   - *(a) For each row $\mathbf{w}^\top \in \mathbb{F}^{1 \times n}$ of $\mathbf{W}$, we have $\mathbf{w} \in \mathcal{C}$.*
   - *(b) For all $j \in J$, we have $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$.*
   - *(c) We have $d_{col}\left(\mathcal{C}^{\equiv k}, \mathbf{X}\right) > \Delta$.*

Then, for any $\mathcal{A}$ and any Δ as above that satisfies $\Delta < d^*/4$, we have

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \leq \left(\frac{\Delta + 1}{|\mathbb{F}|}\right)^P + \left(1 - \frac{\Delta + 1}{n}\right)^L.$$

Proof. The proof follows the arguments in [AHIV22], Theorem B.1. Consider game $\mathcal{G}$ specified in the lemma, and let the variables $\mathbf{X}, \mathbf{R}, \mathbf{W}$ be as in the game. We define the following event in game $\mathcal{G}$:

- Event $\mathsf{CloseRX}$: This event occurs, if there is a $\mathbf{Y} \in \mathcal{C}^{\equiv P}$ such that $d_{col}(\mathbf{Y}, \mathbf{RX}) \leq \Delta$.

Note that $\mathsf{CloseRX}$ implies that for each row $\mathbf{y}^\top$ of $\mathbf{Y}$ and the corresponding row $\mathbf{r}^\top\mathbf{X}$ of $\mathbf{RX}$, we have $d\left(\mathbf{y}^\top, \mathbf{r}^\top\mathbf{X}\right) \leq \Delta$. Now, we can apply Lemma 4.2 in [AHIV22] to each column and get

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1 \wedge \mathsf{CloseRX}] \leq \left(\frac{\Delta + 1}{|\mathbb{F}|}\right)^P.$$

Further, we can write

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \leq \Pr_{\mathcal{G}}[\mathsf{Win} = 1 \wedge \neg\mathsf{CloseRX}] + \Pr_{\mathcal{G}}[\mathsf{Win} = 1 \wedge \mathsf{CloseRX}].$$

Thus, it remains to bound the probability that $\mathsf{CloseRX}$ does not occur and $\mathsf{Win} = 1$. If $\mathsf{CloseRX}$ does not occur and $\mathsf{Win} = 1$, we know that $\mathbf{W} \in \mathcal{C}^{\equiv P}$, and for each $\mathbf{Y} \in \mathcal{C}^{\equiv P}$ we have $d_{col}(\mathbf{Y}, \mathbf{RX}) > \Delta$. Thus, we have $d_{col}(\mathbf{W}, \mathbf{RX}) > \Delta$, meaning that there are at most $n - \Delta - 1$ columns on which $\mathbf{W}$ and $\mathbf{RX}$ agree. Denote the set of these columns by $J^* \subseteq [n], |J^*| \leq n - \Delta - 1$. The probability that $\mathsf{CloseRX}$ does not occur and $\mathsf{Win} = 1$ can now be bounded by the probability that $J \subseteq J^*$, which is at most

$$\frac{\binom{|J^*|}{L}}{\binom{n}{L}} \leq \frac{\binom{n-\Delta-1}{L}}{\binom{n}{L}} \leq \left(1 - \frac{\Delta + 1}{n}\right)^L,$$

where we used Lemma 24. □

Lemma 33. *Let $\Delta \in [n]$. Let $\mathcal{A}$ be any stateful algorithm, and consider the following experiment $\mathcal{G}$:*

1. *Run $\mathcal{A}$ to obtain matrices $\mathbf{X} \in \mathbb{F}^{k \times n}$, $\mathbf{W} \in \mathbb{F}^{P \times n}$, and $\mathbf{R} \in \mathbb{F}^{P \times k}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ (resp. $\mathbf{W}_j \in \mathbb{F}^P$) for $j \in [n]$ be the jth column of $\mathbf{X}$ (resp. $\mathbf{W}$).*
2. *Sample $J \leftarrow\!\!\$ \binom{[n]}{L}$ and set $\mathsf{Win} := 0$. If the following two conditions hold, set $\mathsf{Win} := 1$:*
   (a) For all $j \in J$, we have $\mathbf{W}_j = \mathbf{RX}_j$.
   (b) We have $d_{col}(\mathbf{RX}, \mathbf{W}) > \Delta$.

Then, for any $\mathcal{A}$ and any Δ as above, we have

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \leq \left(1 - \frac{\Delta}{n}\right)^L.$$

Proof. Consider an algorithm $\mathcal{A}$ running in the game specified by the lemma. Clearly, if $n - \Delta < L$, the probability that $\mathsf{Win} = 1$ is zero. So, assume that $L \leq n - \Delta$, and let J^* be the set of columns $j \in [n]$ in which $\mathbf{RX}$ and $\mathbf{W}$ differ. Note that J^* is fixed before J is sampled. If the second winning condition holds, we know that $|J^*| > \Delta$. If the first winning condition holds, we know that $J \subseteq [n] \setminus J^*$. As J is sampled uniformly at random from the size L subsets of $[n]$, we have can upper bound the probability of $J \subseteq [n] \setminus J$ by

$$\frac{\binom{n-|J^*|}{L}}{\binom{n}{L}} \leq \frac{\binom{n-\Delta}{L}}{\binom{n}{L}} \leq \left(1 - \frac{\Delta}{n}\right)^L,$$

where we used Lemma 24. □

Lemma 34. *Let $\Delta_1, \Delta_2 \in [n]$. Let $\mathcal{A}$ be any stateful algorithm, and consider the following experiment $\mathcal{G}$:*

1. *Run $\mathcal{A}$ to obtain $\mathbf{X} \in \mathbb{F}^{k \times n}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ for $j \in [n]$ be the jth column of $\mathbf{X}$.*
2. *Sample a matrix $\mathbf{R} \leftarrow\!\!\$ \mathbb{F}^{P \times k}$.*

3. *Run $\mathcal{A}$ on input $\mathbf{R}$, and get a matrix $\mathbf{W} \in \mathbb{F}^{P\times n}$ and a set $J \subseteq [n]$ from $\mathcal{A}$. Let $\mathbf{W}_j \in \mathbb{F}^P$ be the jth column of $\mathbf{W}$, for each $j \in [n]$.*

4. *Set* Win := 0*. If the following four conditions hold, set* Win := 1*:*

   (a) *There is a $\mathbf{X}^* \in \mathcal{C}^{\equiv k}$, such that $d_{col}(\mathbf{X}^*, \mathbf{X}) \le \Delta_1$.*

   (b) *There is no $\mathbf{X}' \in \mathcal{C}^{\equiv k}$, such that for each $j \in J$, the jth column of $\mathbf{X}'$ is equal to $\mathbf{X}_j$.*

   (c) *For each row $\mathbf{w}^\top \in \mathbb{F}^{1\times n}$ of $\mathbf{W}$, we have $\mathbf{w} \in \mathcal{C}$.*

   (d) *For all $j \in J$, we have $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$, and we have $d_{col}(\mathbf{R}\mathbf{X}, \mathbf{W}) \le \Delta_2$.*

Then, for any $\mathcal{A}$ and any Δ_1, Δ_2 as above that satisfy $\Delta_1 + \Delta_2 < d^*$ and $\Delta_1 \le \lfloor (d^* - 1)/2 \rfloor$, we have

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \le \frac{1}{|\mathbb{F}|^P}.$$

Proof. Let $\mathcal{A}$ be an algorithm in the game specified in the lemma. Consider the event that $\mathcal{A}$ wins, i.e. Win = 1. If this event occurs, we note that due to the assumption $\Delta_1 \le \lfloor (d^* - 1)/2 \rfloor$, we know that $\mathbf{X}^*$ from the first winning condition is uniquely determined by $\mathbf{X}$. Because $\mathbf{X}^* \in \mathcal{C}^{\equiv k}$, the second winning condition implies that there is at least one column $j^* \in J$ such that the j^*th column of $\mathbf{X}^*$, denoted $\mathbf{X}^*_{j^*}$ is not equal to $\mathbf{X}_{j^*}$. By the fourth winning condition, we have $\mathbf{W}_{j^*} = \mathbf{R}\mathbf{X}_{j^*}$. Further, we have

$$d_{col}(\mathbf{R}\mathbf{X}^*, \mathbf{W}) \le d_{col}(\mathbf{R}\mathbf{X}^*, \mathbf{R}\mathbf{X}) + d_{col}(\mathbf{R}\mathbf{X}, \mathbf{W}) \le \Delta_1 + \Delta_2 < d^*.$$

Because $d_{col}(\mathbf{R}\mathbf{X}^*, \mathbf{W}) < d^*$ and $\mathbf{R}\mathbf{X}^* \in \mathcal{C}^{\equiv P}$ and $\mathbf{W} \in \mathcal{C}^{\equiv P}$, we have $\mathbf{R}\mathbf{X}^* = \mathbf{W}$. Thus, we have

$$\mathbf{R}\mathbf{X}_{j^*} = \mathbf{W}_{j^*} = \mathbf{R}\mathbf{X}^*_{j^*}.$$

In summary, we showed the probability that Win $= 1$ can be upper bounded by the probability of $\mathbf{R}\mathbf{X}_{j^*} = \mathbf{R}\mathbf{X}^*_{j^*}$, where $\mathbf{X}_{j^*}, \mathbf{X}^*_{j^*}$ are fixed arbitrarily such that $\mathbf{X}_{j^*} \ne \mathbf{X}^*_{j^*}$, and $\mathbf{R} \in \mathbb{F}^{P\times k}$ is sampled uniformly. Each row of $\mathbf{R}$ is sampled independently, and thus we have

$$\Pr_{\mathbf{R}}\left[\mathbf{R}\mathbf{X}_{j^*} = \mathbf{R}\mathbf{X}^*_{j^*}\right] \le \left(\frac{1}{|\mathbb{F}|}\right)^P.$$

□

Lemma 35. *Let $\mathcal{A}$ be any stateful algorithm, and consider the following experiment $\mathcal{G}$:*

1. *Run $\mathcal{A}$ to obtain $\mathbf{X} \in \mathbb{F}^{k\times n}$. Let $\mathbf{X}_j \in \mathbb{F}^k$ for $j \in [n]$ be the jth column of $\mathbf{X}$.*

2. *Sample a matrix $\mathbf{R} \leftarrow\!\!\$\, \mathbb{F}^{P\times k}$.*

3. *Run $\mathcal{A}$ on input $\mathbf{R}$, and get a matrix $\mathbf{W} \in \mathbb{F}^{P\times n}$. Let $\mathbf{W}_j \in \mathbb{F}^P$ be the jth column of $\mathbf{W}$, for each $j \in [n]$.*

4. *Sample a set $J \leftarrow\!\!\$ \binom{[n]}{L}$.*

5. *Run $\mathcal{A}$ on input J, and obtain an output J' from $\mathcal{A}$.*

6. *Set* Win := 0*. If the following four conditions hold, set* Win := 1*:*

   (a) *There is no $\mathbf{X}' \in \mathcal{C}^{\equiv k}$, such that for each $j \in J'$, the jth column of $\mathbf{X}'$ is equal to $\mathbf{X}_j$.*

   (b) *For each row $\mathbf{w}^\top \in \mathbb{F}^{1\times n}$ of $\mathbf{W}$, we have $\mathbf{w} \in \mathcal{C}$.*

   (c) *For all $j \in J$, we have $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$.*

   (d) *For all $j \in J'$, we have $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$.*

Then, for any $\mathcal{A}$ as above, and any $\Delta_1, \Delta_2 \in [n]$ with $\Delta_1 + \Delta_2 < d^*$ and $\Delta_1 \le d^*/4$, we have

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \le \left(\frac{\Delta_1 + 1}{|\mathbb{F}|}\right)^P + \left(1 - \frac{\Delta_1 + 1}{n}\right)^L + \left(1 - \frac{\Delta_2}{n}\right)^L + \frac{1}{|\mathbb{F}|^P}.$$

Proof. We prove the statement via a sequence of games, using Lemmata 32 to 34.
Game $\mathbf{G}_0$: This game is as game $\mathcal{G}$ from the lemma, and it outputs 1 if and only if $\mathsf{Win} = 1$. We have

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] = \Pr[\mathbf{G}_0 \Rightarrow 1].$$

Game $\mathbf{G}_1$: In this game, we change the winning condition of the game. Namely, the game additionally checks if $d_{col}\left(\mathcal{C}^{\equiv k}, \mathbf{X}\right) \leq \Delta_1$. If $d_{col}\left(\mathcal{C}^{\equiv k}, \mathbf{X}\right) > \Delta_1$, the game outputs 0. If all previous winning conditions hold, and $d_{col}\left(\mathcal{C}^{\equiv k}, \mathbf{X}\right) \leq \Delta_1$, the game outputs 1. It is clear that games $\mathbf{G}_0$ and $\mathbf{G}_1$ only differ if $d_{col}\left(\mathcal{C}^{\equiv k}, \mathbf{X}\right) > \Delta_1$. A simple reduction that runs in the game in Lemma 32 shows that

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \left(\frac{\Delta_1 + 1}{|\mathbb{F}|}\right)^P + \left(1 - \frac{\Delta_1 + 1}{n}\right)^L.$$

Game $\mathbf{G}_2$: In this game, we change the winning condition of the game again. Namely, as an additional check, the game checks if $d_{col}(\mathbf{RX}, \mathbf{W}) > \Delta_2$. If this holds, it outputs 0. Otherwise, it behaves as $\mathbf{G}_1$. We can easily bound the difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ using a reduction that runs in the game in Lemma 33, and get

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \left(1 - \frac{\Delta_2}{n}\right)^L.$$

Finally, we can easily bound the probability that $\mathbf{G}_2$ outputs 1 using Lemma 34. We get

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \leq \frac{1}{|\mathbb{F}|^P}.$$

□

Proof of Lemma 20. We first make some simple changes using a sequence of games. Then, we prove the statement via a reduction that runs in the game specified in Lemma 35.

Game $\mathbf{G}_0$: Let $\mathcal{A}$ be an algorithm as in the lemma, running in the code-binding game of CC. We refer to this game as $\mathbf{G}_0$. That is, $\mathcal{A}$ is run with input $\mathsf{ck} := \bot$ and access to random oracles $\mathsf{H}, \mathsf{H}_1, \mathsf{H}_2$. It makes at most $Q_{\mathsf{H}}, Q_{\mathsf{H}_1}, Q_{\mathsf{H}_2}$ queries to random oracles $\mathsf{H}, \mathsf{H}_1, \mathsf{H}_2$. Then, $\mathcal{A}$ outputs a commitment $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$ and symbols $\mathbf{X}'_j \in \mathbb{F}^k$ for all j in some set $J' \subseteq [n]$. Technically, $\mathcal{A}$ also outputs openings $\tau_j = \bot$ for all $j \in J'$. The game outputs 1, if there is no $\hat{\mathbf{X}} \in \mathcal{C}^{\equiv k}$ such that $\hat{\mathbf{X}}$ is consistent with $(\mathbf{X}'_j)_{j\in J'}$, and all openings verify, i.e. $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 1$ and for all $j \in J'$ it holds that $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j, \mathbf{X}'_j) = 1$. Without loss of generality, we assume that $\mathcal{A}$ never queries the same input to the same random oracle twice, and that $\mathcal{A}$ made all queries that algorithm Ver makes to check $\mathcal{A}$'s final output. Also, we assume that whenever $\mathcal{A}$ makes a query $\mathsf{H}_2(h_1, \ldots, h_n, \mathbf{W})$, it queried $\mathsf{H}_1(h_1, \ldots, h_n)$ before. These assumptions can be achieved by wrapping an additional algorithm around $\mathcal{A}$, which increases $Q_{\mathsf{H}}, Q_{\mathsf{H}_1}, Q_{\mathsf{H}_2}$ to $\bar{Q}_{\mathsf{H}} := Q_{\mathsf{H}} + n$, $\bar{Q}_{\mathsf{H}_1} := Q_{\mathsf{H}_1} + Q_{\mathsf{H}_2} + 1$, $\bar{Q}_{\mathsf{H}_2} := Q_{\mathsf{H}_2} + 1$, respectively. We have

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}}(\lambda).$$

Game $\mathbf{G}_1$: This game is defined as $\mathbf{G}_0$, but we introduce two bad events $\mathsf{HashPre}$ and $\mathsf{HashColl}$ and let the game abort if this event occurs. The events are defined as follows.

- Event $\mathsf{HashPre}$: This event occurs, if $\mathcal{A}$ ever makes a query $\mathsf{H}_1(h_1, \ldots, h_n)$, and later makes a query $\mathsf{H}(x)$ for some input $x \in \{0,1\}^*$ such that $\mathsf{H}(x) = h_j$ for some $j \in [n]$. Phrased differently, this event occurs, if $\mathcal{A}$ makes a query $\mathsf{H}(x)$ that evaluates to h_j, and h_j has been input to H_1 before in a query $\mathsf{H}_1(h_1, \ldots, h_n)$.
- Event $\mathsf{HashColl}$: This event occurs, if $\mathcal{A}$ ever makes two different query $\mathsf{H}(x), \mathsf{H}(x')$ for $x \neq x' \in \{0,1\}^*$ such that $\mathsf{H}(x) = \mathsf{H}(x')$.

Using a union bound over all pairs of queries to H, we can bound the probability of $\mathsf{HashColl}$ by $\bar{Q}_{\mathsf{H}}^2/2^\lambda$. To bound the probability of event $\mathsf{HashPre}$, note that for a fixed query to H, a fixed query to H_1, and a fixed index $j \in [n]$, the probability that $\mathsf{HashPre}$ occurs for these queries and this index is $2^{-\lambda}$. Thus, a union bound leads to

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{HashPre}] + \Pr[\mathsf{HashColl}] \leq \frac{\bar{Q}_{\mathsf{H}}\bar{Q}_{\mathsf{H}_1} n}{2^\lambda} + \frac{\bar{Q}_{\mathsf{H}}^2}{2^\lambda}.$$

Game $\mathbf{G}_2$: In this game, we guess the random oracle queries that are used for $\mathcal{A}$'s final output. More precisely, the game is as $\mathbf{G}_1$, but if first samples two indices $i_1 \leftarrow\!\!\$\, [\bar{Q}_{\mathsf{H}_1}]$ and $i_2 \leftarrow\!\!\$\, [\bar{Q}_{\mathsf{H}_2}]$. Then, it runs $\mathbf{G}_1$ as it is. If the i_1th query to H_1 occurs after the i_2th query to H_2, the game aborts. Also, let the i_1th query to H_1 be $\mathsf{H}_1(h_1, \dots, h_n)$ and the i_2th query to H_2 be $\mathsf{H}_2(h'_1, \dots, h'_n, \mathbf{W})$. If $(h_1, \dots, h_n) \neq (h'_1, \dots, h'_n)$, the game also aborts. Consider the final output $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$ of $\mathcal{A}$. If the i_1th query to H_1 was $\mathsf{H}_1(h_1, \dots, h_n)$ and the i_2th query to H_2 was $\mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$, the game continues as $\mathbf{G}_1$ does. Otherwise, it aborts. If $\mathbf{G}_1$ outputs 1, there has to be some indices i_1^*, i_2^* that correspond to the hash queries of $\mathcal{A}$'s final output, and such that i_1^*th query to H_1 occurs before the i_2^*th query to H_2. Therefore, $\mathbf{G}_2$ outputs 1 if and only if $i_1 = i_1^*$ and $i_2 = i_2^*$ and $\mathbf{G}_1$ outputs 1. Note that $\mathcal{A}$'s view is independent of the indices i_1, i_2 until a potential abort occurs. Thus, we have

$$\Pr[\mathbf{G}_1 \Rightarrow 1] \leq \bar{Q}_{\mathsf{H}_1}\bar{Q}_{\mathsf{H}_2} \cdot \Pr[\mathbf{G}_2 \Rightarrow 1].$$

Now, we can easily bound the probability that $\mathbf{G}_2$ outputs 1 using a reduction $\mathcal{B}$ that runs in the game specified in Lemma 35. The reduction is as follows.

1. Reduction $\mathcal{B}$ simulates $\mathbf{G}_2$ for $\mathcal{A}$, including all aborts specified before.
2. Let the i_1th query to H_1 be $\mathsf{H}_1(h_1, \dots, h_n)$ and the i_2th query to H_2 be $\mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$. We know that the i_1th query to H_1 occurs first, as otherwise $\mathbf{G}_2$ and the reduction would abort.
   (a) When the i_1th query to H_1 happens, $\mathcal{B}$ extracts a matrix $\hat{\mathbf{X}}$ as follows: For each $j \in [n]$, reduction $\mathcal{B}$ checks if there is a previous random oracle query of the form $\mathsf{H}(\hat{\mathbf{X}}_j) = h_j$, where $\hat{\mathbf{X}}_j \in \mathbb{F}^k$. As we ruled out event $\mathsf{HashColl}$, there can be at most one such query. If such a query is found, it sets the jth column of $\hat{\mathbf{X}}$ to be $\hat{\mathbf{X}}_j$. Otherwise, it sets the jth column of $\hat{\mathbf{X}}$ to be $\mathbf{0}$. Then, the reduction outputs $\hat{\mathbf{X}}$ to the game, and gets as input a matrix $\mathbf{R}$. The reduction sets $\mathsf{H}_1(h_1, \dots, h_n) := \mathbf{R}$, and continues the execution of $\mathcal{A}$.
   (b) When the i_2th query to H_2 happens, the reduction outputs $\mathbf{W}$ to the game, and gets as input a set J. It sets $\mathsf{H}_2(h_1, \dots, h_n, \mathbf{W}) := J$, and continues $\mathcal{A}$'s execution.
3. When $\mathcal{A}$ terminates with final output $\mathsf{com} = \left((h_j)_{j\in[n]}, \mathbf{W}, (\mathbf{X}_j)_{j\in J}\right)$ and $\mathbf{X}'_j \in \mathbb{F}^k$ for all j in some set $J' \subseteq [n]$, the reduction first does all checks as in $\mathbf{G}_2$. Note that if all checks pass, we know that all $\mathbf{X}_j$ and all $\mathbf{X}'_j$ are consistent with $\hat{\mathbf{X}}$ (cf. events $\mathsf{HashColl}$ and $\mathsf{HashPre}$). The reduction now outputs J' and terminates.

It is clear that the reduction perfectly simulates $\mathbf{G}_2$ for $\mathcal{A}$. Also, one can observe that if $\mathbf{G}_2$ outputs 1, then all winning conditions in the game specified in Lemma 35 hold. Thus, using Lemma 35, we have

$$\Pr[\mathbf{G}_2 \Rightarrow 1] \leq \left(\frac{\Delta_1 + 1}{|\mathbb{F}|}\right)^P + \left(1 - \frac{\Delta_1 + 1}{n}\right)^L + \left(1 - \frac{\Delta_2}{n}\right)^L + \frac{1}{|\mathbb{F}|^P}.$$

□

I.2 Omitted Details from Section 9.2

Lemma 36. *Let $\mathcal{A}$ be any stateful algorithm, and consider the following experiment $\mathcal{G}$:*

1. *Generate* $\mathsf{hk} \leftarrow \mathsf{HF.Gen}(1^\lambda)$.
2. *Run $\mathcal{A}$ on input* hk *to obtain $h_1, \dots, h_n \in \mathcal{R}$.*
3. *Sample a matrix $\mathbf{R} \leftarrow\!\!\$\, \mathbb{F}^{P\times k}$.*
4. *Run $\mathcal{A}$ on input $\mathbf{R}$, and get a matrix $\mathbf{W} \in \mathbb{F}^{P\times n}$. Let $\mathbf{W}_j \in \mathbb{F}^P$ be the jth column of $\mathbf{W}$, for each $j \in [n]$.*

5. *Sample a matrix* $\mathbf{S} \leftarrow\!\!\$\, \mathbb{F}^{n \times L}$.

6. *Run* $\mathcal{A}$ *on input* $\mathbf{S}$*, and obtain an output* $\mathbf{Y}, J', (\mathbf{X}_j)_{j \in J'}$ *from* $\mathcal{A}$.

7. *Set* $\mathsf{Win} := 0$. *If the following four conditions hold, set* $\mathsf{Win} := 1$*:*

   (a) *There is no* $\mathbf{X}' \in \mathcal{C}^{\equiv k}$*, such that for each* $j \in J'$*, the* j*th column of* $\mathbf{X}'$ *is equal to* $\mathbf{X}_j$.

   (b) *For all* $j \in J'$*, we have* $\mathbf{W}_j = \mathbf{R}\mathbf{X}_j$ *and* $\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{X}_j) = h_j$.

   (c) *For each row* $\mathbf{w}^\top \in \mathbb{F}^{1 \times n}$ *of* $\mathbf{W}$*, we have* $\mathbf{w} \in \mathcal{C}$.

   (d) *For each* $j \in [L]$*, we have* $\mathsf{HF.Eval}(\mathsf{hk}, \mathbf{Y}_j) = [h_1, \ldots h_n]\mathbf{S}_j$ *and* $\mathbf{R}\mathbf{Y} = \mathbf{W}\mathbf{S}$.

Then, for any PPT algorithm $\mathcal{A}$ *in the above game, there is an EPT algorithm* $\mathcal{B}$ *with expected running time* $\mathbf{ET}(\mathcal{B}) \approx (1+n)\mathbf{T}(\mathcal{A})$ *we have*

$$\Pr_{\mathcal{G}}[\mathsf{Win} = 1] \leq \frac{n}{|\mathbb{F}|^L} + \frac{1}{|\mathbb{F}|^P} + \frac{1}{|\mathbb{F}|^L} + \mathsf{Adv}^{\mathsf{coll}}_{\mathcal{B},\mathsf{HF}}(\lambda).$$

Proof. Our proof strategy is as follows. We first sample a random a hash key hk and adversarial randomness, and fix it. Then, we run game $\mathcal{G}$ with this fixed key and randomness multiple times with independent challenges, until we can extract preimages of all hash values $h_1, \ldots, h_n$. We run $\mathcal{G}'$ a final time, rule out inconsistencies by reducing to collision-resistance, and use statistical arguments to finish the proof.

We will now proceed more formally. Let $\mathcal{A}$ be a PPT algorithm in the game $\mathcal{G}$ specified in the lemma. We define $\varepsilon_0 := \Pr_{\mathcal{G}}[\mathsf{Win} = 1]$. We want to bound this probability ε_0. Assume that $\mathcal{A}$ makes use of $\ell = \mathsf{poly}(\lambda)$ random coins. By making states and randomness explicit, we can write $\mathcal{A}$ as a triple of PPT algorithms $(\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, with the following syntax:

- $\mathcal{A}_0(\mathsf{hk}; \rho) \to (St_0, h_1, \ldots, h_n)$ takes as input the key hk and random coins $\rho \in \{0,1\}^\ell$. It outputs a state St_0 and values $h_1, \ldots, h_n \in \mathcal{R}$.

- $\mathcal{A}_1(St_0, \mathbf{R}) \to (St_1, \mathbf{W})$ is deterministic, takes as input St_0, a matrix $\mathbf{R}$, and outputs a state St_1 and a matrix $\mathbf{W}$.

- $\mathcal{A}_2(St_1, \mathbf{S}) \to (\mathbf{Y}, J', (\mathbf{X}_j)_{j \in J'})$ is deterministic, takes as input St_1 and a matrix $\mathbf{S}$, and outputs $\mathbf{Y}, J', (\mathbf{X}_j)_{j \in J'}$.

Note that assuming that $\mathcal{A}$ gets all its random coins in the beginning is without loss of generality, as $\mathcal{A}_0$ can just pass these coins to $\mathcal{A}_1$ and $\mathcal{A}_2$ via its state. We introduce another notation. Namely, we denote the game $\mathcal{G}$ with fixed hash key hk and fixed adversarial random coins ρ by $\mathcal{G}(\mathsf{hk}, \rho)$. Also, we define $\varepsilon_{\mathsf{hk},\rho} := \Pr_{\mathcal{G}(\mathsf{hk},\rho)}[\mathsf{Win} = 1]$ for any hk, ρ.

Game $\mathcal{G}'$: We define a new game $\mathcal{G}'$. In this game, we run the adversary multiple times with the same hk and ρ. Formally, we define $\mathcal{G}'$ as follows.

1. Generate $\mathsf{hk} \leftarrow \mathsf{HF.Gen}(1^\lambda)$ and sample $\rho \leftarrow\!\!\$\, \{0,1\}^\ell$.

2. Run $\mathcal{G}(\mathsf{hk}, \rho)$ and denote all variables x involved in this game run by x^0. For example, variables $\mathsf{Win}, \mathbf{S}$ in this game run are denoted by $\mathsf{Win}^{(0)}, \mathbf{S}^{(0)}$, respectively. If $\mathsf{Win}^{(0)} = 0$, abort.

3. Initialize an empty list $\mathcal{S} := \emptyset$, an empty map $\mathsf{SY}[\cdot]$, and a counter $q := 1$.

4. While $|\mathcal{S}| < n$, repeat the following:

   (a) Run $\mathcal{G}(\mathsf{hk}, \rho)$. Denote all variables x involved in this game run by $x^{(q)}$. For example, variables $\mathsf{Win}, \mathbf{S}$ in this game run are denoted by $\mathsf{Win}^{(q)}, \mathbf{S}^{(q)}$, respectively.

   (b) If $\mathsf{Win}^{(q)} = 1$, then insert $\mathbf{S}^{(q)}$ into $\mathcal{S}$. Further, set $\mathsf{SY}[\mathbf{S}] := \mathbf{Y}$.

   (c) Set $q := q + 1$.

5. Set $q^* := q$.

We will now analyze this game. Namely, we shall show two things. First, we establish a relation between the probability of $\mathsf{Win} = 1$ in $\mathcal{G}$ and $\mathsf{Win}^{(0)} = 1$ in $\mathcal{G}'$. Second, we argue that the game runs in expected polynomial time.

Claim 7. Consider the notations and assumptions from the proof of Lemma 36. We have

$$\Pr_{\mathcal{G}'}\left[\mathsf{Win}^{(0)} = 1\right] = \Pr_{\mathcal{G}}\left[\mathsf{Win} = 1\right] = \varepsilon_0.$$

We prove Claim 7. Namely, first observe that if hk, ρ is fixed in $\mathcal{G}'$, then Step 2 is clearly independent of the rest of the game. Therefore, we have

$$\Pr_{\mathcal{G}'}\left[\mathsf{Win}^{(0)} = 1 \;\middle|\; (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] = \Pr_{\mathcal{G}(\bar{\mathsf{hk}}, \bar{\rho})}\left[\mathsf{Win} = 1\right] = \Pr_{\mathcal{G}}\left[\mathsf{Win} = 1 \mid (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right]$$

for each $\bar{\mathsf{hk}}, \bar{\rho}$. Now, we can use the law of total probability to finish the proof of the claim, i.e.

$$\begin{aligned}\Pr_{\mathcal{G}'}\left[\mathsf{Win}^{(0)} = 1\right] &= \sum_{\bar{\mathsf{hk}}, \bar{\rho}} \Pr_{\mathcal{G}'}\left[\mathsf{Win}^{(0)} = 1 \;\middle|\; (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \cdot \Pr_{\mathsf{hk}, \rho}\left[(\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \\ &= \sum_{\bar{\mathsf{hk}}, \bar{\rho}} \Pr_{\mathcal{G}}\left[\mathsf{Win} = 1 \mid (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \cdot \Pr_{\mathsf{hk}, \rho}\left[(\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \\ &= \Pr_{\mathcal{G}}\left[\mathsf{Win} = 1\right].\end{aligned}$$

Claim 8. Consider the notations and assumptions from the proof of Lemma 36. The expected running time of $\mathcal{G}'$ is at most $1 + n$ times the running time of $\mathcal{G}$.

We prove Claim 8. We show the bound on the running time for any fixed hk, ρ, which implies that it holds for random hk, ρ. Denote the random variable modeling the running time of $\mathcal{G}'$ by T' and the running time of $\mathcal{G}$ by T. Consider the case where $\varepsilon_{\mathsf{hk}, \rho} = 0$. Then, game $\mathcal{G}'$ always stops in Step 2. Thus, we can assume that $\varepsilon_{\mathsf{hk}, \rho} > 0$ from now on. We shall first argue that the expected number of iterations q^* of the loop in Step 4 is bounded by $n/\varepsilon_{\mathsf{hk}, \rho}$. Then, we can conclude using the law of total expectation and linearity of expectation. Namely,

$$\begin{aligned}\mathbb{E}\left[T'\right] &= \Pr\left[\mathsf{Win}^{(0)} = 0\right]\mathbb{E}\left[T' \mid \mathsf{Win}^{(0)} = 0\right] + \Pr\left[\mathsf{Win}^{(0)} = 1\right]\mathbb{E}\left[T' \mid \mathsf{Win}^{(0)} = 1\right] \cdot \\ &= (1 - \varepsilon_{\mathsf{hk}, \rho}) \cdot T + \varepsilon_{\mathsf{hk}, \rho} \cdot \left(T + \mathbb{E}\left[q^* \mid \mathsf{Win}^{(0)} = 1\right] \cdot T\right) = (1 + \varepsilon_{\mathsf{hk}, \rho} \cdot \mathbb{E}\left[q^*\right]) \cdot T = (1 + n) \cdot T,\end{aligned}$$

where we used that for fixed hk, ρ, the random variable q^* is indepdendent of $\mathsf{Win}^{(0)}$. It remains to argue that $\mathbb{E}\left[q^*\right] \leq n/\varepsilon_{\mathsf{hk}, \rho}$. For each $i \in [n]$, let X_i be the random variable equal to the number of iterations of Step 4 needed to increase the size of $\mathcal{S}$ from $i - 1$ to i. Then, by linearity of expectation and the fact that $q^* = \sum_{i=1}^{n}$, it is sufficient bound the expectation of each X_i by $1/\varepsilon_{\mathsf{hk}, \rho}$. For that, consider the probability that in a fixed iteration of the loop in Step 4, a matrix is added to the list $\mathcal{S}$. By definition, it is added if $\mathsf{Win}^{(q)} = 1$. The probability of $\mathsf{Win}^{(q)} = 1$ is exactly $\varepsilon_{\mathsf{hk}, \rho}$. Thus, X_i follows a geometric distribution with parameter $\varepsilon_{\mathsf{hk}, \rho}$, which has expectation $1/\varepsilon_{\mathsf{hk}, \rho}$, as desired. This finishes the proof of Claim 8.

Game $\mathcal{G}''$: We slightly modify game $\mathcal{G}'$ into a game $\mathcal{G}''$. Formally, we define $\mathcal{G}''$ as follows.

1. Run Steps 1 to 5 of game $\mathcal{G}'$.
2. Set $\bar{\mathbf{X}} = \mathbf{0}$. Also, initialize an empty set $\mathcal{S}' = \emptyset$ and an empty map $\mathsf{SY}'[\cdot]$.
3. Iterate over the matrices in $\mathcal{S}$. Namely, for each $i \in [n]$, do the following:
   (a) Let $\mathbf{S} \in \mathbb{F}^{n \times L}$ be the ith matrix in $\mathcal{S}$.
   (b) If there is no column $\mathbf{s}$ of $\mathbf{S}$ that is linearly independent to the set $\mathcal{S}'$, then set $\mathsf{Extr} := 0$ and abort the game.
   (c) Otherwise, let $\mathbf{s}$ be such a column, say the jth. Insert $\mathbf{s}$ into $\mathcal{S}'$, and set $\mathsf{SY}' := \mathbf{y}$, where $\mathbf{y}$ is the jth column of $\mathsf{SY}[\mathbf{S}]$.

4. Set $\mathsf{Extr} := 1$.

5. By construction, the vectors contained in $\mathcal{S}'$ are linearly independent. Arrange them as columns into an invertible matrix $\bar{\mathbf{S}} \in \mathbb{F}^{n \times n}$. Similarly, arrange the n vectors in the multi-set $\{\mathsf{SY}'[\mathbf{s}] \mid \mathbf{s} \in \mathcal{S}'\}$ into a matrix $\bar{\mathbf{Y}} \in \mathbb{F}^{k \times n}$. Ensure that for each $\mathbf{s} \in \mathcal{S}'$, if $\mathbf{s}$ is the jth column of $\bar{\mathbf{S}}$, then $\mathbf{y} := \mathsf{SY}'[\mathbf{s}]$ is the jth column of $\bar{\mathbf{Y}}$.

6. Compute $\bar{\mathbf{X}} := \bar{\mathbf{Y}}\bar{\mathbf{S}}^{-1}$. We denote the columns of $\bar{\mathbf{X}}$ by $\bar{\mathbf{X}}_j$ for each $j \in [n]$.

In Claim 9, we bound the probability of $\mathsf{Extr} = 0$ conditioned on the game not aborting and any fixed hash key and randomness. Using this claim, we get for any fixed $\bar{\mathsf{hk}}, \bar{\rho}$ with $\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} > 0$, that

$$\begin{aligned}\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 0 \mid (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] = \quad & \Pr_{\mathcal{G}''}\left[\mathsf{Extr} = 0 \mid \mathsf{Win}^{(0)} = 1 \wedge (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \\ & \cdot \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \mid (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \\ \leq \quad & \frac{n}{\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}|^L} \cdot \varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} = \frac{n}{|\mathbb{F}|^L}.\end{aligned}$$

The same upper bound holds trivially for any $\bar{\mathsf{hk}}, \bar{\rho}$ with $\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} = 0$. This implies that

$$\begin{aligned}\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 0\right] &= \sum_{\bar{\mathsf{hk}},\bar{\rho}} \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 0 \mid (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \cdot \Pr_{\mathcal{G}''}\left[(\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \\ &\leq \sum_{\bar{\mathsf{hk}},\bar{\rho}} \frac{n}{|\mathbb{F}|^L} \Pr_{\mathcal{G}''}\left[(\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] = \frac{n}{|\mathbb{F}|^L}.\end{aligned}$$

Thus, we have

$$\begin{aligned}\varepsilon_0 = \Pr_{\mathcal{G}'}\left[\mathsf{Win}^{(0)} = 1\right] &\leq \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1\right] + \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 0\right] \\ &\leq \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1\right] + \frac{n}{|\mathbb{F}|^L}.\end{aligned}$$

Finally, we will bound the probability that $\mathsf{Win}^{(0)} = 1$ and $\mathsf{Extr} = 1$ in game $\mathcal{G}''$. For that, we introduce the following events in $\mathcal{G}''$.

- Event HColl: This event occurs, if $\mathbf{Y}^{(0)} \neq \bar{\mathbf{X}}\mathbf{S}^{(0)}$ or there is a $j \in J'^{(0)}$, such that $\bar{\mathbf{X}}_j \neq \mathbf{X}_j^{(0)}$.
- Event InCode: This event occurs, if $\mathbf{R}^{(0)}\bar{\mathbf{X}} \in \mathcal{C}^{\equiv P}$.

By the law of total probability, we have

$$\begin{aligned}\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1\right] \leq \quad & \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1 \wedge \mathsf{HColl}\right] \\ + & \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1 \wedge \neg\mathsf{HColl} \wedge \mathsf{InCode}\right] \\ + & \Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1 \wedge \neg\mathsf{HColl} \wedge \neg\mathsf{InCode}\right].\end{aligned}$$

We bound these terms separately in claims Claims 10 to 12. In combination, this will conclude the proof.

Claim 9. Consider the notations and assumptions from the proof of Lemma 36. Let $\bar{\mathsf{hk}} \in \mathsf{HF.Gen}(1^\lambda)$ and $\bar{\rho} \in \{0,1\}^\ell$ be fixed arbitrarily. Then, we have

$$\Pr_{\mathcal{G}''}\left[\mathsf{Extr} = 0 \mid \mathsf{Win}^{(0)} = 1 \wedge (\mathsf{hk}, \rho) = (\bar{\mathsf{hk}}, \bar{\rho})\right] \leq \frac{n}{\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}|^L}.$$

We prove Claim 9. To this end, consider a fixed $\bar{\mathsf{hk}}$ and $\bar{\rho}$ and assume $\mathsf{Win}^{(0)} = 1$. We can bound the probability of $\mathsf{Extr} = 0$ occurring in a fixed iteration of the loop, say the ith. Then, the result will follow using a union bound over all the n iterations. So, consider the ith iteration of the loop, and assume that at the beginning of this ith iteration of the loop, we have $r := |\mathcal{S}'| < n$. Then, $\mathcal{S}'$ is a set of r linearly independent vectors over $\mathbb{F}$, which span a subspace $D \subset \mathbb{F}^n$ of dimension $r < n$. Let q_i be the iteration of the loop in Step 4 of game $\mathcal{G}'$ in which the ith matrix of $\mathcal{S}$ has been added to $\mathcal{S}$. Recall that in this q_ith iteration, $\mathcal{G}(\bar{\mathsf{hk}}, \bar{\rho})$ has been executed, and the only random choices in this game are the challenge matrices $\mathbf{R}^{(q_i)}$ and $\mathbf{S}^{(q_i)}$. As we know that $\mathsf{Win}^{(q_i)} = 1$, we can think of $\mathbf{R}^{(q_i)}, \mathbf{S}^{(q_i)}$ as being sampled uniformly at random from the set $\Gamma \subseteq \mathbb{F}^{P \times k} \times \mathbb{F}^{n \times L}$ of matrices $(\mathbf{R}, \mathbf{S})$ for which $\mathsf{Win} = 1$ in $\mathcal{G}(\bar{\mathsf{hk}}, \bar{\rho})$ with challenges $\mathbf{R}, \mathbf{S}$. This set has size at least one. More precisely, by definition of $\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}}$, it has size $\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}^{P \times k}| \cdot |\mathbb{F}^{n \times L}| > 0$. Then, by what we have discussed so far, the probability of $\mathsf{Extr} = 0$ occurring in the ith iteration of the loop is at most

$$\Pr_{(\mathbf{R}^{(q_i)},\mathbf{S}^{(q_i)}) \xleftarrow{\$} \Gamma}\left[\mathbf{S}^{(q_i)} \in D^L\right] = \frac{|\mathbb{F}^{P\times k}| \cdot |D|^L}{|\Gamma|} = \frac{|\mathbb{F}^{P\times k}| \cdot |\mathbb{F}|^{rL}}{\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}^{P\times k}| \cdot |\mathbb{F}^{n\times L}|}$$
$$= \frac{1}{\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}|^{(n-r)L}} \leq \frac{1}{\varepsilon_{\bar{\mathsf{hk}},\bar{\rho}} \cdot |\mathbb{F}|^{L}},$$

where we used $r < n$. This finishes the proof of Claim 9.

Claim 10. Consider the notations and assumptions from the proof of Lemma 36. Then, there is an algorithm $\mathcal{B}$ with expected running time $\mathbf{ET}(\mathcal{B}) \approx (1+n)\mathbf{T}(\mathcal{A})$ and

$$\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \land \mathsf{Extr} = 1 \land \mathsf{HColl}\right] \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathcal{B},\mathsf{HF}}(\lambda).$$

To prove Claim 10, we will argue that the two sub-events specified in event HColl imply a collision for HF. Then, one can construct a reduction to collision-resistance. Such a reduction gets as input the hashing key hk, runs $\mathcal{G}''$, and outputs the collision if event $\mathsf{Win}^{(0)} = 1$ and $\mathsf{Extr} = 1$ and HColl occurs. In this way, the reduction perfectly simulates $\mathcal{G}''$ for $\mathcal{A}$, and the expected running time of the reduction is polynomial. It remains to argue that $\mathsf{Win}^{(0)} = 1$ and $\mathsf{Extr} = 1$ and HColl implies a collision. The reader may then observe that these collisions can be efficiently found by the reduction. So, assume that these three events occur. First, it is clear that for each $q \in [q^*] \cup \{0\}$, the hash values $h_1^{(q)}, \ldots, h_n^{(q)}$ sent by $\mathcal{A}$ are the same. This is because $\mathcal{A}$ gets the same hk and randomness ρ in every run of $\mathcal{G}$. Thus, we can just denote these hash values by $h_1, \ldots, h_n$. Now, we claim that for each column $j \in [n]$, we have $\mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{X}}_j) = h_j$. To see this, fix an arbitrary $j^* \in [n]$. We have

$$\mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{X}}_{j^*}) = \mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{Y}}\bar{\mathbf{S}}^{-1}_{j^*}) = \left[\mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{Y}}_1) \;\middle|\; \cdots \;\middle|\; \mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{Y}}_n)\right] \bar{\mathbf{S}}^{-1}_{j^*}$$

using the definition of $\bar{\mathbf{X}} := \bar{\mathbf{Y}}\bar{\mathbf{S}}^{-1}$ and the homomorphic property of HF. We continue using the fact that the responses are accepting, namely

$$\left[\mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{Y}}_1) \;\middle|\; \cdots \;\middle|\; \mathsf{HF.Eval}(\mathsf{hk}, \bar{\mathbf{Y}}_n)\right] \bar{\mathbf{S}}^{-1}_{j^*} = \left[\sum_{j=1}^{n} h_j \bar{\mathbf{S}}_{j,1} \;\middle|\; \cdots \;\middle|\; \sum_{j=1}^{n} h_j \bar{\mathbf{S}}_{j,n}\right] \bar{\mathbf{S}}^{-1}_{j^*}$$
$$= [h_1 \mid \cdots \mid h_n] \cdot \bar{\mathbf{S}} \cdot \bar{\mathbf{S}}^{-1}_{j^*} = h_{j^*}.$$

Now that we established this, it is clear that the two sub-events of HColl imply a collision, and the claim follows.

Claim 11. Consider the notations and assumptions from the proof of Lemma 36. Then

$$\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \land \mathsf{Extr} = 1 \land \neg\mathsf{HColl} \land \mathsf{InCode}\right] \leq \frac{1}{|\mathbb{F}|^P}.$$

To prove Claim 11, assume event $\mathsf{Win}^{(0)} = 1 \land \mathsf{Extr} = 1 \land \neg\mathsf{HColl} \land \mathsf{InCode}$ occurs in $\mathcal{G}''$. Then, because of $\neg\mathsf{HColl}$, we know that the columns $\mathbf{X}_j^{(0)}$ for $j \in J'^{(0)}$ are consistent with the columns of $\bar{\mathbf{X}}$. Because of the second condition required for $\mathsf{Win}^{(0)} = 1$, we thus know that $\bar{\mathbf{X}} \notin \mathcal{C}^{\equiv k}$. Thus, the event of interest implies that

$$\bar{\mathbf{X}} \notin \mathcal{C}^{\equiv k} \wedge \mathbf{R}^{(0)}\bar{\mathbf{X}} \in \mathcal{C}^{\equiv P},$$

where $\bar{\mathbf{X}}$ is independent of $\mathbf{R}^{(0)} \in \mathbb{F}^{P\times k}$. Let $\mathbf{H} \in \mathbb{F}^{(n-k)\times n}$ be the parity-check matrix of $\mathbf{G}$. That is, for all $\mathbf{a} \in \mathbb{F}^n$, we have $\mathbf{Ha} = \mathbf{0}$ if and only if $\mathbf{a} \in \mathcal{C}$. Then, we have

$$\bar{\mathbf{X}}\mathbf{H}^\top \neq \mathbf{0} \wedge \mathbf{R}^{(0)}\bar{\mathbf{X}}\mathbf{H}^\top = \mathbf{0}.$$

As all rows of $\mathbf{R}^{(0)}$ are independent, this event occurs with probability at most $1/|\mathbb{F}|^P$.

Claim 12. Consider the notations and assumptions from the proof of Lemma 36. Then

$$\Pr_{\mathcal{G}''}\left[\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1 \wedge \neg\mathsf{HColl} \wedge \neg\mathsf{InCode}\right] \leq \frac{1}{|\mathbb{F}|^L}.$$

To prove Claim 12, assume that event $\mathsf{Win}^{(0)} = 1 \wedge \mathsf{Extr} = 1 \wedge \neg\mathsf{HColl} \wedge \neg\mathsf{InCode}$ occurs in $\mathcal{G}''$. Then, we know that $\mathbf{R}^{(0)}\mathbf{Y}^{(0)} = \mathbf{W}^{(0)}\mathbf{S}^{(0)}$, because $\mathsf{Win}^{(0)} = 1$. Also, we know that $\mathbf{Y}^{(0)} = \bar{\mathbf{X}}\mathbf{S}^{(0)}$ because $\neg\mathsf{HColl}$. This implies that $\mathbf{R}^{(0)}\bar{\mathbf{X}}\mathbf{S}^{(0)} = \mathbf{W}^{(0)}\mathbf{S}^{(0)}$. Because $\neg\mathsf{InCode}$, we also know that $\mathbf{R}^{(0)}\bar{\mathbf{X}} \neq \mathbf{W}^{(0)}$. Thus, we obtain that

$$(\mathbf{R}^{(0)}\bar{\mathbf{X}} - \mathbf{W}^{(0)})\mathbf{S}^{(0)} = \mathbf{0} \wedge \mathbf{R}^{(0)}\bar{\mathbf{X}} - \mathbf{W}^{(0)} \neq \mathbf{0},$$

where $\mathbf{R}^{(0)}\bar{\mathbf{X}} - \mathbf{W}^{(0)}$ and $\mathbf{S}^{(0)} \in \mathbb{F}^{n\times L}$ are independent. This occurs with probability at most $1/|\mathbb{F}|^L$, as all columns of $\mathbf{S}^{(0)}$ are sampled independently. □

Proof of Lemma 22. We prove the lemma using Lemma 36. Except for that, the proof is almost identical to the proof of Lemma 20.

Game $\mathbf{G}_0$: Let $\mathcal{A}$ be an algorithm as in the lemma, running in the code-binding game of $\mathsf{CC}[\mathsf{HF}]$. We call this code-binding game $\mathbf{G}_0$. Recall that in this game, $\mathcal{A}$ receives a commitment key $\mathsf{ck} = \mathsf{hk} \leftarrow \mathsf{HF.Gen}(1^\lambda)$ and gets oracle access random oracles $\mathsf{H}_1, \mathsf{H}_2$. We assume that $\mathcal{A}$ makes at most $Q_{\mathsf{H}_1}, Q_{\mathsf{H}_2}$ queries to random oracles $\mathsf{H}_1, \mathsf{H}_2$, respectively. Then, $\mathcal{A}$ outputs a commitment $\mathsf{com} = \big((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\big)$ and symbols $\mathbf{X}'_j \in \mathbb{F}^k$ for all j in some set $J' \subseteq [n]$. The game $\mathbf{G}_0$ outputs 1, if there is no $\hat{\mathbf{X}} \in \mathcal{C}^{\equiv k}$ such that $\hat{\mathbf{X}}$ is consistent with $(\mathbf{X}'_j)_{j\in J'}$, and all openings verify, i.e. $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 1$ and for all $j \in J'$ it holds that $\mathsf{VerCol}(\mathsf{ck}, \mathsf{com}, j, \mathbf{X}'_j) = 1$. As in the proof of Lemma 20, we assume without loss of generality that $\mathcal{A}$ never queries the same input to the same random oracle twice, and that $\mathcal{A}$ made all queries that algorithm Ver makes to check $\mathcal{A}$'s final output. Also, we assume that whenever $\mathcal{A}$ makes a query $\mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$, it queried $\mathsf{H}_1(h_1, \dots, h_n)$ before. As in the proof of Lemma 20, this increases Q_{H_1} and Q_{H_2} to $\bar{Q}_{\mathsf{H}_1} := Q_{\mathsf{H}_1} + Q_{\mathsf{H}_2} + 1$ and $\bar{Q}_{\mathsf{H}_2} := Q_{\mathsf{H}_2} + 1$, respectively. We have

$$\Pr[\mathbf{G}_0 \Rightarrow 1] = \mathsf{Adv}^{\mathsf{code\text{-}bind}}_{\mathcal{A},\mathsf{CC}[\mathsf{HF}]}(\lambda).$$

Game $\mathbf{G}_1$: In game $\mathbf{G}_1$, we let the game guess the random oracle queries related to $\mathcal{A}$'s final output. Namely, in the beginning of the game, indices $i_1 \leftarrow_\$ [\bar{Q}_{\mathsf{H}_1}]$ and $i_2 \leftarrow_\$ [\bar{Q}_{\mathsf{H}_2}]$ are sampled. Then, $\mathbf{G}_1$ behaves as $\mathbf{G}_0$. Let the i_1th query to H_1 be $\mathsf{H}_1(h_1, \dots, h_n)$ and the i_2th query to H_2 be $\mathsf{H}_2(h'_1, \dots, h'_n, \mathbf{W})$. If $(h_1, \dots, h_n) \neq (h'_1, \dots, h'_n)$, or the i_1th query to H_1 occurs after the i_2th query to H_2, the game aborts. Once $\mathcal{A}$ outputs $\mathsf{com} = \big((h_j)_{j\in[n]}, \mathbf{W}, \mathbf{Y}\big)$ and $\mathbf{X}'_j \in \mathbb{F}^k$ for all $j \in J'$, the game checks if the i_1th query to H_1 was $\mathsf{H}_1(h_1, \dots, h_n)$ and the i_2th query to H_2 was $\mathsf{H}_2(h_1, \dots, h_n, \mathbf{W})$. If not, the game aborts. Otherwise, it continues as $\mathbf{G}_0$ does. One can easily see that

$$\Pr[\mathbf{G}_0 \Rightarrow 1] \leq \bar{Q}_{\mathsf{H}_1}\bar{Q}_{\mathsf{H}_2} \cdot \Pr[\mathbf{G}_1 \Rightarrow 1].$$

Now, we can bound the probability that $\mathbf{G}_1$ outputs 1 using a reduction, which runs in the game given in Lemma 36. Roughly, it embeds its challenges into the i_1th and i_2th random oracle queries to H_1 and H_2, respectively. By Lemma 36, we get that there is an algorithm $\mathcal{B}$ with

$$\Pr[\mathbf{G}_1 \Rightarrow 1] \leq \frac{n}{|\mathbb{F}|^L} + \frac{1}{|\mathbb{F}|^P} + \frac{1}{|\mathbb{F}|^L} + \mathsf{Adv}^{\mathsf{coll}}_{\mathcal{B},\mathsf{HF}}(\lambda).$$

□

J Simulation of Index Samplers

While the analytical results in Section 6.2 provide bounds on the quality of different index samplers, their analysis makes heavy use of bounds, e.g., the union bound. Thus, it is natural to ask whether one can obtain more precise results by analyzing and comparing index samplers other means, e.g., via simulation.

Experiment. We can think of index sampling as the following balls-into-bins experiment. We have N bins and ℓ players. Each player is allowed to throw Q balls into the bins, following some fixed strategy, which is given by the index sampler algorithm $\mathsf{Sample}(1^Q, 1^N)$. More precisely, the players all start with the same state. Further, they are not aware of any identifiers to break symmetry and can not communicate. Each player starts with a random tape and runs $(i_j)_{j \in [Q]} \leftarrow \mathsf{Sample}(1^Q, 1^N)$. Then, it throws its balls into the bins $i_1, \ldots, i_Q$. We want to estimate the probability that less than K bins are non-empty after the experiment.

Setup. For our simulation, we implemented the experiment in C++. We ran the experiment for the three index samplers $\mathsf{Sample}_{\mathsf{wr}}$ (sampling uniformly with replacement), $\mathsf{Sample}_{\mathsf{wor}}$ (sampling uniformly without replacement), and $\mathsf{Sample}_{\mathsf{seg}}$ (segment sampling). We estimated the probability of interest by averaging over 20000 runs of the experiment. This process was repeated for various combinations of ℓ, Q, N, K. When we select such parameter sets, we pay attention to avoid divisibility issues. For example, say we used segment sampling with $Q = 64$ and we want to have at least $K = N/4$ non-empty bins out of N. A first intuition would tell us that for $N = 1152$ we would need less samples to ensure that than for $N = 1280$. However, we would observe the opposite due to a divisibility phenomenon. Namely, for $N = 1152$ we would have to collect at least $K/Q = 4.5$ out of $N/Q = 18$ segments, i.e., 5 out of 18. For $N = 1280$ we would have to collect $K/Q = 5$ out of $N/Q = 20$ segments, i.e., 5 out of 20. Collecting 5 out of 20 requires less samples than 5 out of 18, contradicting our initial intuition. Such phenomenons distract from the actual message we want to convey and the asymptotic behavior of index samples. Therefore, we choose parameters that avoid these divisibility issues. The code of our simulation can be found in

`https://github.com/b-wagn/collectiveBallsInBins`.

Results. We present our some of our simulation results in Figures 4 and 5. We briefly want to discuss them here. First, consider Figure 4. The figure shows how the failure probability p, i.e., the probability of having less than K non-empty bins out of N bins in total, relates to the total number of samples $\ell \cdot Q$. We see that both for collecting quarter and half of the bins, the failure probability rapidly decreases when the number of samples is slightly more than K. For collecting three quarters, we see that we need about $2K$ samples to reach that point, which fits our intuition. Comparing the different samplers, we see that for sampling uniformly range in which the failure probability decreases is smaller than for segment sampling.

Second, consider Figure 5. The figure shows how many samples we need to get the failure probability p below a fixed threshold. Again, we see that segment sampling with a large segment size $Q = 32$ leads to worse results. Namely, to get p below the treshold, we need significantly more samples than for uniform sampling with and without replacement. Segment sampling with a small segment size $Q = 8$ has only a minimal impact. Also, Figure 5 shows that there is almost no difference between sampling with replacement and sampling without replacement. We expect the difference to grow when Q approaches K. For all samplers, Figure 5 suggests that the number of samples is linear in the number of bins N, which is in line with our analytical results in Section 6.2.

Conclusion. Our simulation suggests that sampling without replacement does not perform significantly better than sampling with replacement. As sampling with replacement is much easier to implement efficiently, we may disregard sampling without replacement. Segment sampling with small segment sizes seems to lead only to a minimal loss in quality. Due to its reduced randomness complexity, the improved locality, and ease of implementation, it qualifies a good choice in practice.

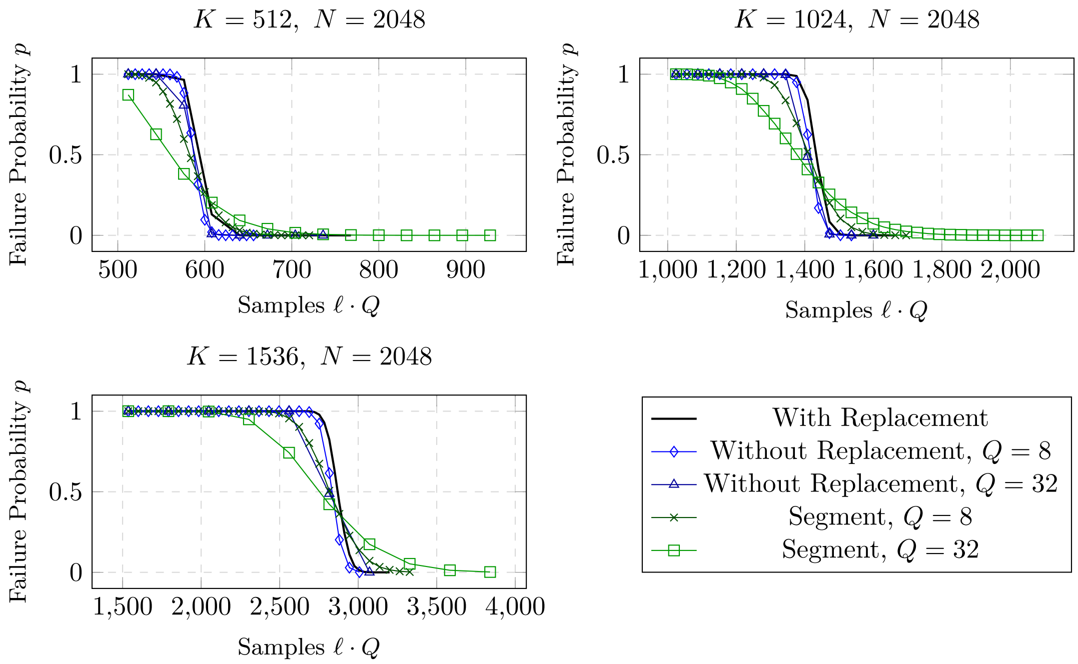

Figure 4: Simulation results for the failure probability p, i.e., the probability of having less than K non-empty bins out of N bins in total after ℓ players threw Q balls into the bins according to the given index sampler.

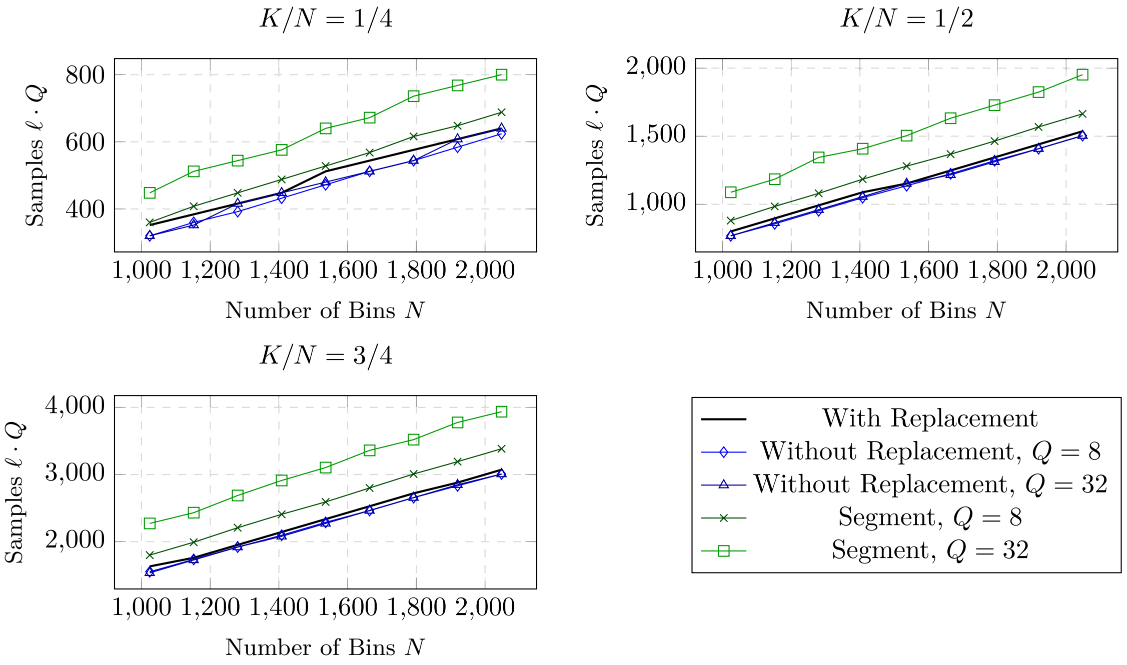

Figure 5: Simulation results for the total number of samples $\ell \cdot Q$ needed to get $p \leq 0.001$, where p is the failure probability, i.e., the probability of having less than K non-empty bins out of N bins in total after ℓ players threw Q balls into the bins according to the given index sampler.

K Script for Parameter Computation

Listing 1: Python script to compute the parameters for different codes. A discussion is given in Section 10.

```
from dataclasses import dataclass

import math

# Statistical Security Parameter for Soundness
SECPAR_SOUND = 40

@dataclass
class Code:
    size_msg_symbol: int  # size of one symbol in the message
    size_code_symbol: int # size of one symbol in the code
    msg_len: int          # number of symbols in the message
    codeword_len: int     # number of symbols in the codeword
    reception: int        # number of symbols needed to reconstruct (worst case)
    samples: int          # number of random samples to reconstruct with high probability

    def interleave(self, ell):
        return Code(
            size_msg_symbol = self.size_msg_symbol * ell,
            size_code_symbol = self.size_code_symbol * ell,
            msg_len = self.msg_len,
            codeword_len = self.codeword_len,
            reception = self.reception,
            samples = self.samples
        )

    def tensor(self, col):
        assert self.size_msg_symbol == col.size_msg_symbol
        assert self.size_code_symbol == col.size_code_symbol
        assert self.size_msg_symbol == self.size_code_symbol

        row_dist = self.codeword_len - self.reception + 1
        col_dist = col.codeword_len - col.reception + 1
        codeword_len = self.codeword_len * col.codeword_len

        '''
        Example:

        D D | o o
        D D | o o
        ----+----
        o o | o o
        o o | o o

        Where D is the data.
        The reception is 8, since 7 is not enough to reconstruct:

        o o | o x
        o o | o x
        ----+----
        o o | o x
        x x | x x

        Given the symbols marked with x, I cannot reconstruct the data.
        '''
        reception = codeword_len - row_dist * col_dist + 1
        '''
        To determine the number of samples, we have multiple options.
        we can use the minimum of all resulting number of samples

        Option 1: use reception and generalized coupon collector
        As reception is a "worst case bound", this may not be tight

        Option 2: use a more direct analysis.
        not being able to reconstruct
        -> there is a row we can not reconstruct
        -> union bound over all rows
        -> for fixed row, assume we can not reconstruct
        -> there is a set of t_r - 1 positions (t_r = reception in rows)
        such that all queries in that row are in that set
        -> we union bounding over all of these sets
        -> for each fixed set, the probability that
        all queries in that row are in that set is
        (1-((n_r - t_r + 1)/(n_r*n_c)))^{number of samples}
        so the total probability of not being able to reconstruct is at most
        n_c * (n_r choose t_r - 1) * (1-((n_r - t_r + 1)/(n_r*n_c)))^{number of samples}
        and (n_r choose t_r - 1) <= (n_r * e / (t_r - 1))^(t_r - 1)

        Option 3: same as Option 2 but reversed roles

        Asymptotic example: Tensor C: F^k -> F^{2k} with itself
        Option 1   -> Omega(k^2 + sec_par) samples
        Option 2/3 -> Omega(k^2 + sec_par * k) samples

        Concretely, Option 2/3 will be tighter, especially for large k
        '''
        samples_via_reception = samples_from_reception(SECPAR_SOUND, reception, codeword_len)

        loge = math.log2(math.e)
        lognc = math.log2(col.codeword_len)
        lognr = math.log2(self.codeword_len)
        logbinomr = (self.reception - 1) * (lognr + loge - math.log2(self.reception - 1))
        loginnerr = math.log2(1.0 - (self.codeword_len - self.reception + 1)/codeword_len)
        logbinomc = (col.reception - 1) * (lognc + loge - math.log2(col.reception - 1))
        loginnerc = math.log2(1.0 - (col.codeword_len - col.reception + 1)/codeword_len)

        samples_direct_via_rows = int(math.ceil(-(lognc + logbinomr + SECPAR_SOUND)/loginnerr))
        samples_direct_via_cols = int(math.ceil(-(lognr + logbinomc + SECPAR_SOUND)/loginnerc))

        samples_direct = min(samples_direct_via_rows, samples_direct_via_cols)
        samples = min(samples_direct, samples_via_reception)

        return Code(
            size_msg_symbol = self.size_msg_symbol,
            msg_len = self.msg_len * col.msg_len,
            size_code_symbol = self.size_code_symbol,
            codeword_len = codeword_len,
            reception = reception,
            samples = samples
        )

    def __eq__(self, other):
        return (
            self.size_msg_symbol == other.size_msg_symbol
            and self.size_code_symbol == other.size_code_symbol
            and self.msg_len == other.msg_len
            and self.codeword_len == other.codeword_len
            and self.reception == other.reception
        )

    def is_identity(self):
        return (
            self.size_msg_symbol == self.size_code_symbol
            and self.msg_len == self.codeword_len
        )


def samples_from_reception(sec_par, reception, codeword_len):
    ’’’
    Compute the number of samples needed to reconstruct
    data with probability at least 1-2^{-sec_par} based on
    the reception efficiency and a generalized coupon collector.
    Note: this may not be the tightest for all schemes (e.g. Tensor)
    ’’’
    # special case: if only one symbol is needed, we are done
    if reception == 1:
        return 1

    # special case: if all symbols are needed: just regular coupon collector
    if reception == codeword_len:
        n = codeword_len
        s = math.ceil((n / math.log(math.e, 2)) * (math.log(n, 2) + sec_par))
        return int(s)

    # generalized coupon collector
    delta = reception - 1
    c = delta/codeword_len
    s = math.ceil(-sec_par / math.log2(c) + (1.0-math.log(math.e,c))*delta)
    return int(s)

# Identity code
def makeTrivialCode(chunksize, k):
    return Code(
        size_msg_symbol = chunksize,
        msg_len = k,
        size_code_symbol = chunksize,
        codeword_len = k,
        reception = k,
        samples = samples_from_reception(SECPAR_SOUND, k, k)
    )

# Reed-Solomon Code
# Polynomial of degree k-1 over field with field element length fsize
# Evaluated at n points
def makeRSCode(fsize, k, n):
    assert k <= n
    assert 2**fsize >= n, ’no such reed-solomon code :(’
    return Code(
        size_msg_symbol = fsize,
        msg_len = k,
        size_code_symbol = fsize,
        codeword_len = n,
        reception = k,
        samples = samples_from_reception(SECPAR_SOUND, k, n)
    )

# tests
assert makeRSCode(5, 2, 4).tensor(makeRSCode(5, 2, 4)).reception == 8
assert makeRSCode(5, 2, 4).reception == 2
```

Listing 2: Python script to compute the parameters for different data availability sampling schemes. A discussion is given in Section 10.

```
#!/usr/bin/env python

import math

# Some constants.
# Sizes of group elements, field elements, and hashes in bits
BLS_FE_SIZE = 48.0 * 8.0
BLS_GE_SIZE = 48.0 * 8.0

# Let’s say we use the SECP256_k1 curve
PEDERSEN_FE_SIZE = 32.0 * 8.0
PEDERSEN_GE_SIZE = 33.0 * 8.0

# Let’s say we use SHA256
HASH_SIZE = 256


from dataclasses import dataclass

from codes import *

@dataclass
class Scheme:
    code: Code             # code that is used
    com_size: int          # size of commitment in bits
    opening_overhead: int # overhead of opening a symbol in the encoding

    def samples(self):
        '''
        i.e. the number of random samples needed to collect
        enough symbols except with small probability
        '''
        return self.code.samples

    def total_comm(self):
        '''
        Compute the total communication in bits.
        '''
        return self.comm_per_query() * self.samples()

    def comm_per_query(self):
        '''
        Compute the communication per query in bits.
        '''
        return math.log2(self.code.codeword_len) + self.opening_overhead + self.code.size_code_symbol

    def encoding_size(self):
        '''
        Compute the size of the encoding in bits.
        '''
        return self.code.codeword_len * (self.opening_overhead + self.code.size_code_symbol)

    def reception(self):
        '''
        Compute the reception of the code.
        '''
        return self.code.reception

    def encoding_length(self):
        '''
        Compute the length of the encoding.
        '''
        return self.code.codeword_len

# Naive scheme
# Put all the data in one symbol, and let the commitment be a hash
def makeNaiveScheme(datasize):
    return Scheme(
        code = Code(
            size_msg_symbol = datasize,
            msg_len = 1,
            size_code_symbol = datasize,
            codeword_len = 1,
            reception = 1,
            samples = 1
        ),
        com_size = HASH_SIZE,
        opening_overhead = 0
    )

# Merkle scheme
# Take a merkle tree and the identity code
def makeMerkleScheme(datasize, chunksize=1024):
    k = math.ceil(datasize / chunksize)
    return Scheme(
        code = makeTrivialCode(chunksize, k),
        com_size = HASH_SIZE,
        opening_overhead = math.ceil(math.log(k, 2))*HASH_SIZE
    )


# KZG Commitment, interpreted as an erasure code commitment for the RS code
# The RS Code is set to have parameters k,n with n = invrate * k
def makeKZGScheme(datasize, invrate=4):
    k = math.ceil(datasize / BLS_FE_SIZE)
    return Scheme(
        code = makeRSCode(
            BLS_FE_SIZE,
            k,
            k * invrate
        ),
        com_size = BLS_GE_SIZE,
        opening_overhead = BLS_GE_SIZE,
    )


# Tensor Code Commitment, where each dimension is expanded with inverse rate invrate.
# That is, data is a k x k matrix, and the codeword is a n x n matrix, with n = invrate * k
# Both column and row code are RS codes.
def makeTensorScheme(datasize, invrate=2):
    m = math.ceil(datasize / BLS_FE_SIZE)
    k = math.ceil(math.sqrt(m))
    n = invrate * k

    rs = makeRSCode(BLS_FE_SIZE, k, n)

    return Scheme(
        code = rs.tensor(rs),
        com_size = BLS_GE_SIZE * k,
        opening_overhead = BLS_GE_SIZE,
    )


# Hash-Based Code Commitment, over field with elements of size fsize,
# parallel repetition parameters P and L. Data is treated as a k x k matrix,
# and codewords are k x n matrices, where n = k*invrate.
def makeHashBasedScheme(datasize, fsize=32, P=8, L=64, invrate=4):
    m = math.ceil(datasize / fsize)
    k = math.ceil(math.sqrt(m))
    n = invrate * k
    rs = makeRSCode(fsize, k, n)

    return Scheme(
        code = rs.interleave(k),
        com_size = n * HASH_SIZE + P * n * fsize + L * k * fsize,
        opening_overhead = 0,
    )

# Homomorphic Hash-Based Code Commitment
# instantiated with Pedersen Hash
# parallel repetition parameters P and L. Data is treated as a k x k matrix,
# and codewords are k x n matrices, where n = k*invrate.
def makeHomHashBasedScheme(datasize, P=2, L=2, invrate=4):
    m = math.ceil(datasize / PEDERSEN_FE_SIZE)
    k = math.ceil(math.sqrt(m))
    n = invrate * k
    rs = makeRSCode(PEDERSEN_FE_SIZE, k, n)

    return Scheme(
        code = rs.interleave(k),
        com_size = n * PEDERSEN_GE_SIZE + P * n * PEDERSEN_FE_SIZE + L * k * PEDERSEN_FE_SIZE,
        opening_overhead = 0,
    )
```

Listing 3: Python script to compute the tables in Section 10.

```
#!/usr/bin/env python

import math
import sys
from tabulate import tabulate

from schemes import *

def makeRow(name,scheme,tex):
        comsize = ’{:.2f}’.format(round(scheme.com_size/8000.0,2))
        encodingsize = ’{:.2f}’.format(round(scheme.encoding_size() / 8000000.0,2))
        commpqsize = ’{:.2f}’.format(round(scheme.comm_per_query() / 8000.0,2))
        reception = scheme.reception()
        encodinglength = scheme.encoding_length()
        samples = scheme.samples()
        commsize = ’{:.2f}’.format(round(scheme.total_comm() / 8000000.0,2))
        if tex:
                row = ["\Inst"+name,comsize,encodingsize,commpqsize,samples,commsize]
        else:
                row = [name,comsize,encodingsize,commpqsize,(reception,encodinglength),samples,commsize]
        return row

#####################################################################

opts = [opt for opt in sys.argv[1:] if opt.startswith("-")]
args = [arg for arg in sys.argv[1:] if not arg.startswith("-")]

if len(args) == 0:
        print("Missing Argument: Datasize in Megabytes.")
        print("Hint: To print the table in LaTeX code, add the option -l.")
        sys.exit(-1)

datasize = int(args[0])*8000000


# Print to LaTeX
tex = "-l" in opts

if tex:
        table = [["Name","|com|","|Encoding|","Comm. p. Q.","Samples","Comm Total"]]
else:
        table = [["Name","|com| [KB]","|Encoding| [MB]","Comm. p. Q. [KB]","Reception","Samples","Comm Total [MB]"]]


scheme = makeNaiveScheme(datasize)
table.append(makeRow("Naive",scheme,tex))

scheme = makeMerkleScheme(datasize)
table.append(makeRow("Merkle",scheme,tex))

scheme = makeKZGScheme(datasize)
table.append(makeRow("RS",scheme,tex))

scheme = makeTensorScheme(datasize)
table.append(makeRow("Tensor",scheme,tex))

scheme = makeHashBasedScheme(datasize)
table.append(makeRow("Hash",scheme,tex))

scheme = makeHomHashBasedScheme(datasize)
table.append(makeRow("HomHash",scheme,tex))

if tex:
        print(tabulate(table,headers=’firstrow’,tablefmt=’latex_raw’,disable_numparse=True))
else:
        print(tabulate(table,headers=’firstrow’,tablefmt=’fancy_grid’))
```

Listing 4: Python script to compute the graphs in Section 10.

```
#!/usr/bin/env python

import math
import sys
import csv
import os

from schemes import *

DATASIZEUNIT = 8000*1000 # Megabytes
DATASIZERANGE = range(1,156,15)

def writeCSV(path,d):
        with open(path, mode="w") as outfile:
                writer = csv.writer(outfile, delimiter=’,’)
                for x in d:
                        writer.writerow([x,d[x]])


# Writes the graphs for a given scheme
# into a csv file
def writeScheme(name,makeScheme):

        commitment = {}
        commpq = {}
        commtotal = {}
        encoding = {}

        for s in DATASIZERANGE:
                datasize = s*DATASIZEUNIT
                scheme = makeScheme(datasize)
                commitment[s] = scheme.com_size / 8000000 # MB
                commpq[s] = scheme.comm_per_query() /8000 # KB
                commtotal[s] = scheme.total_comm() /8000000000 # GB
                encoding[s] = scheme.encoding_size() /8000000000 # GB

        if not os.path.exists("./csvdata/"):
                os.makedirs("./csvdata")

        writeCSV("./csvdata/"+name+"_com.csv",commitment)
        writeCSV("./csvdata/"+name+"_comm_pq.csv",commpq)
        writeCSV("./csvdata/"+name+"_comm_total.csv",commtotal)
        writeCSV("./csvdata/"+name+"_encoding.csv",encoding)


############################################
writeScheme("rs",makeKZGScheme)
writeScheme("tensor",makeTensorScheme)
writeScheme("hash",makeHashBasedScheme)
writeScheme("homhash",makeHomHashBasedScheme)
```