Proofs of Reliability

1 Preliminaries

1.1 Problem Overview

We consider the following cloud storage scenario. A client outsources a file F to a CSP. To handle large files, F is first partitioned into subfiles of equal length, which are then individually processed. Let F be divided to n_s subfiles $\mathbf{f}_1, \mathbf{f}_2, \dots \mathbf{f}_{n_s}$. Each $\mathbf{f}_i$ is independently encoded to $\mathbf{y}_i$ through the application of a linear ECC $\mathcal{C}$ of length n, dimension k, and minimum distance d over a finite field $\mathbb{F}_q$, an $(n, k, d)_q$ code for short. This is common practice in most cloud systems [1, 2, 3, 4]. The order of $\mathbb{F}_q$ is $q = 2^\ell$, so that $\mathcal{C}$ operates on ℓ-bit blocks, also referred to as $symbols$.

The SLA between the client and the CSP specifies a reliability degree ω for F. The reliability degree defines the maximum number of tolerated erasures, before F becomes irretrievable. Note that although the application of code $\mathcal{C}$ provides a reliability of $d-1$, the SLA may specify any desired reliability degree $\omega \leq d$. A CSP offering a higher reliability degree can require a higher monetary compensation. To verify compliance to the SLA, the client (or a designated verifier) periodically performs an integrity verification test. This is achieved by executing an interactive *proof-of-reliability* (PoRL) protocol. In a PoRL, the CSP is challenged to prove the retrievability of $n - (d - \omega)$ symbols per subfile. When $\omega = 0$, a PoRL is equivalent to a PoR. A PoRL poses a stronger requirement to the CSP compared to classical PoR protocols. In the latter, it is sufficient to assure the integrity of each subfile, and therefore of F, by retrieving the minimum number of symbols required for retrievability. In a PoRL, the CSP must additionally prove that redundant symbols are stored per subfile. A PoRL assurance implies a PoR assurance, but the converse does not hold. As an additional requirement, the CSP must be able to *efficiently recover* lost symbols, when symbol erasures occur due to hardware failures.

1.2 Entities

– Client C: The client outsources the storage of F to the CSP. File F is divided to n_s, $t\ell$-long subfiles $\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_{n_s}$, where each subfile consists of t symbols that belong to $\mathbb{F}_q$. If the size of F is not a multiple of $t\ell$, then F is padded.

– Cloud service provider (CSP): For every $\mathbf{f}_i$, the CSP stores an n-symbol long verifiable version $\mathbf{y}_i$ that is encoded using a public $(n, k, d)_q$ code $\mathcal{C}$ with generator matrix G and parity check equation matrix H. The CSP must prove to a verifier $\mathcal{V}$ that at least $n - (d - \omega)$ symbols are retrievable, where $\omega \leq d$. The value of ω is specified in the SLA.

– Verifier $\mathcal{V}$: The verifier engages in an interactive protocol with the CSP to check that F is stored with reliability ω. The verifier can be either the client itself, or a separate authorized entity. In case that the verifier is an authorized entity, it shall not know F to perform the verification.

1.3 PORL Functions

Similarly to PoRs [5, 6], a PoRL scheme consists of five polynomial-time algorithms: *KeyGen, Encode, Challenge, ProofGen, ProofVer, NoiseRec* and *BlockRec*. These algorithms operate on subfiles. To simplify notation, we subsequently assume that F consists of a single subfile $\mathbf{f}$.

– *KeyGen*$(1^\tau) \rightarrow K$: The client runs the probabilistic key generation algorithm to generate a master secret key K. On input of the security parameter τ, *KeyGen* outputs K.

– *Encode*$(K, \mathbf{f}) \rightarrow (\mathbf{y})$: The client runs this algorithm to generate the verifiable version of $\mathbf{f}$ that is stored at the CSP. On input of the secret key K and the subfile $\mathbf{f}$, algorithm *Encode* returns the verifiable version $\mathbf{y}$. The CSP stores $\mathbf{y}$.

– *Challenge*$(K) \rightarrow chal$: The verifier runs this probabilistic algorithm to generate a challenge for performing a PORL on f with id fid. The algorithm takes as input the secret key K and the file identifier fid and returns the challenge $chal$ that is sent to the CSP.

– *ProofGen*$(chal) \rightarrow \mathcal{P}$: Upon receiving the challenge $chal$, the CSP runs $ProofGen$ to generate a proof of reliability $\mathcal{P}$ and sends to the verifier V.

– *ProofVer*$(K, chal, \mathcal{P}) \rightarrow b \in \{0, 1\}$: Upon receiving $\mathcal{P}$, the verifier runs *ProofVer* to verify the correctness of the PORL response sent by the CSP. On input of the key K, the challenge $chal$ and $\mathcal{P}$, *ProofVer* outputs a boolean value b with $b = 1$ if $\mathcal{P}$ is valid and $b = 0$ otherwise.

In addition, we define algorithms for the file maintenance phase. These algorithms describe how the CSP can recover lost blocks with input from the verifier.

– *NoiseRec*$(K, i) \rightarrow (m, s_m)$: The verifier runs this algorithm to generate the syndrome s_i needed by the CSP to recover symbol y_i. On input of the coordinate i of the corrupted symbol and the master key K, the algorithm outputs the column coordinate m of the parity check matrix H^T used for symbol recovery and the corresponding syndrome value s_m.

– *SymbolRec*$(i, m, s_m) \rightarrow y_i$: The CSP runs this algorithm to restore a corrupted symbol y_i. On input of the symbol coordinate i, the syndrome value s_m and the column coordinate m of H^T, the CSP computes y_i.

1.4 Adversary Model

We adopt a semi-honest adversary model in which the CSP will attempt to reduce the storage for F only if it can pass the PORL. We further adopt the security properties definitions from [6]. A PORL scheme should satisfy the completeness and soundness properties defined below.

Definition 1 (Completeness)**.** *A PORL scheme is complete if for an honest CSP and verifier $\mathcal{V}$, and for any challenge $chal \leftarrow (K)$:*

$$\Pr(ProofVerif(K, chal, P) \rightarrow 1 | P \leftarrow ProofGen(chal)) = 1$$

That is, when all parties are protocol-compliant, the *ProofVer* algorithm always outputs 1 (no false negatives).

Definition 2 (Soundness)**.** *A PORL scheme is said to be (δ, γ)-sound, if for every adversary $\mathcal{A}$ that provides γ valid proofs of reliability in a row (i.e. succeeds in the soundness game described above) with a non-negligible probability δ, there exists an extractor algorithm $\mathcal{E}$ such that:*

$$\Pr(\mathcal{E}(K) \rightarrow F^* | \mathcal{E}(K) \overset{\text{interact}}{\longleftrightarrow} \mathcal{A}) \geq 1 - \varepsilon,$$

where ϵ is a negligible function in the security parameter τ.

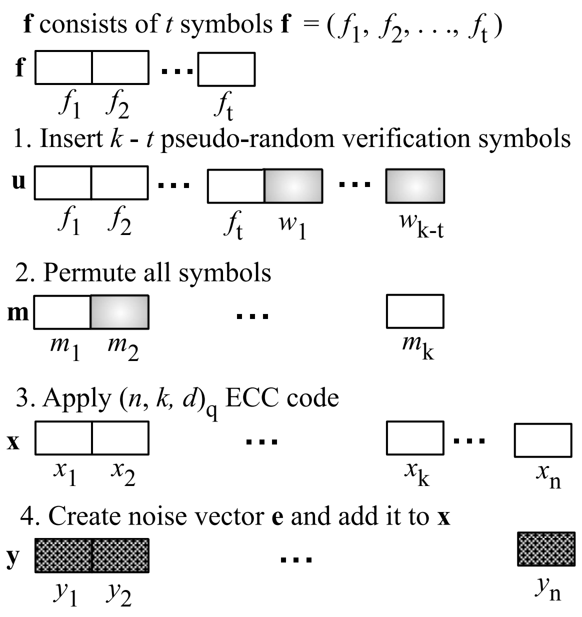
2 A PoRL Construction

In this section, we describe a PoRL construction that verifies the storage integrity of $\mathbf{f}$ with reliability degree ω by a verifier $\mathcal{V}$. Moreover, it enables the efficient maintenance of $\mathbf{y}$ at the CSP.

2.1 Main Idea

The main method for verifying the reliable storage of $\mathbf{f}$ through $\mathbf{y}$, is to prevent the CSP from autonomously recovering erased symbols, so that the retrievability of $\mathbf{y}$ can be verified. If the CSP could perform file maintenance independently, then he could safely delete all redundant symbols generated by the application of the ECC. The erased symbols could be regenerated on the fly when audited by the verifier to prove the storage of $\mathbf{y}$. To prevent this scenario, we force the CSP to interact with the verifier, whenever it needs to recover an erased symbol. Compared to prior methods that also prevent the automated recovery from erasures [7, 8], we achieve the minimum possible communication overhead for data recovery and also minimize the repair bandwidth at the CSP. This cost can be substantial [3, 9].

Consider the partition of a $t\ell$–bit long file $\mathbf{f}$ to t symbols $\mathbf{f} = (f_1, f_2, \ldots, f_t)$, as shown in Figure 1. To avoid storing $\mathbf{f}$ at the client, probabilistic verification is performed by inserting random "verification" blocks. These are equivalent to the sentinels used in [5] and the watchdogs used in [6]. Specifically, the client appends $k - t$ ℓ-bit verification symbols $w_1, w_2, \ldots, w_{k-t}$ to $\mathbf{f}$. In the next step, the client permutes the information and verification symbols to hide the verification symbol locations and creates $\mathbf{m} = (m_1, m_2, \ldots, m_k)$. The client applies an $(n, k, d)_q$ code $\mathcal{C}$ to generate redundancy, extending $\mathbf{m}$ to a codeword $\mathbf{x}$. Since only t blocks in $\mathbf{m}$ carry the data, the storage efficiency of this scheme is t/n.



f consists of t symbols f = (f1, f2, . . ., ft)
f
f1 f2 ft
1. Insert k - t pseudo-random verification symbols
u
f1 f2 ft w1 wk-t
2. Permute all symbols
m
m1 m2 mk
3. Apply (n, k, d)q ECC code
x
x1 x2 xk xn
4. Create noise vector e and add it to x
y
y1 y2 yn

Figure 1: The setup phase of the PORL protocol.

The generator matrix G of $\mathcal{C}$ is selected to be systematic to preserve the verification symbols, so that spot checking can be performed by the verifier. In this case, the CSP can exploit G's structure to differentiate between parity and information symbols. Moreover, parity symbols cannot be used for verification, because they are a function of both the information and verification symbols (for an $(n, k, d)_q$ MDS code, each parity is a function of all k uncoded symbols). A CSP knowing the location of the parity symbols could safely delete them and still pass a PoRL test. To prevent the CSP from passing the PoRL we perform two operations. First, the client selects an $(n, k, d, r)_q$ locally repairable code (LRC) with locality r to generate $\mathbf{x}$ from $\mathbf{m}$. Such codes are already adopted in cloud storage due to their minimal repair bandwidth [9]. The limited code locality is used to create parity symbols that are solely dependent on the $k - t$ verification symbols appended to $\mathbf{f}$. Therefore, the parities from those verification symbols can be regenerated at the verifier and be used for verification. Because the locations of the verification symbols in $\mathbf{m}$ are unknown due to the applied permutation on $\mathbf{u}$, the location of the verification parity symbols among parities is also hidden. A CSP deleting parity symbols will also delete parity verification symbols and therefore fail the PoRL.

Furthermore, to destroy the dependency between the symbols in $\mathbf{x}$ due to the application of a systematic and publicly known G, the client generates noise $\mathbf{e} = (e_1, e_2, \ldots, e_n)$, which consists of n pseudo-random symbols of length ℓ each. The client adds noise e_i to symbol x_i at each codeword coordinate i to generate the verifiable version of $\mathbf{f}$, denoted by $\mathbf{y} = \mathbf{x} + \mathbf{e}$. $\mathbb{F}_q$ is an extension of a binary field, and therefore the addition corresponds to a simple bitwise XOR operation. Besides concealing block dependency, the application of $\mathbf{e}$ randomizes the information blocks so they are not discernible from verification blocks. Permuting the message coordinates and adding noise to a codeword resemble the steps in the McEliece cryptosystem [10], but our goal here is different. There is no restriction on the weight of noise $\mathbf{e}$, $\mathbf{y}$ is not close to a codeword, and to cheat the PoRL test, a CSP would have to be able to find a unique solution to an underdetermined system of $n - k$ linear equations in n variables.

Decoding relies on the fact that $\mathbf{x}$ is orthogonal to all vectors in the dual code, i.e., $\mathbf{x}H^T = 0_{1\times\mu}$, where $0_{1\times\mu}$ represents an all-zero tuple of length μ, and the operations are preformed in the vector space $\mathbb{F}_q^n$. Since $\mathbf{y} = \mathbf{x} + \mathbf{e}$, it follows that $\mathbf{y}H^T = \mathbf{e}H^T = \mathbf{s}$, where the m-tuple of ℓ-bit symbols $\mathbf{s}$ is referred to as a *syndrome* of $\mathbf{y}$. Note that $\mathbf{y}$ is not a codeword of $\mathcal{C}$, but is a codeword of its coset $\mathbf{e}+\mathcal{C}$. Thus, the number of correctable symbol erasures is the same as for $\mathcal{C}$. To prove the possession of $\mathbf{y}$, an untrusted CSP can still store only k symbols from $\mathbf{y}$ instead of the entire $\mathbf{y}$ and regenerate on-the-fly all n symbols of $\mathbf{y}$ from these k symbols, provided he knows the syndrome $\mathbf{s}$. However, $n - k$ symbols are needed to store the syndrome $\mathbf{s} = \mathbf{x}H^T$. Therefore, *the strategy of reducing the symbols of* $\mathbf{y}$ *to* k *and storing the* $n - k$ *symbols of* $\mathbf{s}$ *does not reduce the storage requirement for proving the storage of* $\mathbf{y}$. We emphasize also that the vector $\mathbf{e}$ cannot be inferred from $\mathbf{s} = \mathbf{e}H^T$, as there are 2^k noise vectors $\mathbf{e}$ producing the same syndrome. Even if $\mathbf{e}$ was known to the CSP, storing it would require n symbols, since all symbols in $\mathbf{e}$ are random and independent.

The verifiable version $\mathbf{y}$ contains four types of symbols: (a) information symbols, (b) parity symbols, (c) verification symbols inserted before encoding, we refer to as, information verification symbols, and (d) parity symbols generated from information verification symbols we call parity verification symbols. All symbols are masked by the application of $\mathbf{e}$ and their locations are unknown to the CSP.

Formally, our PoRL protocol consists of three phases: the setup phase, the verification phase, and the data recovery phase. During the setup phase, the client performs a set of transformations to $\mathbf{f}$ to generate a verifiable and maintainable version $\mathbf{y}$. In the verification phase, the verifier executes an interactive protocol with the CSP to verify the storage of $\mathbf{y}$ with reliability r. Finally, in the data recovery phase, the CSP requests input from the verifier to recover a corrupted symbol. We describe each phase in detail.

2.2 Setup phase

In the setup phase, the client transforms a $t\ell$–bit long file $\mathbf{f} = (f_1, f_2, \ldots, f_t)$ to a verifiable version $\mathbf{y}$ that is stored at the CSP. The steps of this phase are as follows.

1. **Key generation:** The client runs the *KeyGen* algorithm to generate a master secret key K. The master secret key is used to generate 3 additional keys. These keys are $K_{ver} = \mathcal{H}_{ver}(K)$, $K_{perm} = \mathcal{H}_{perm}(K)$, and $K_{noise} = \mathcal{H}_{noise}(K)$, with $\mathcal{H}_{check}$, $\mathcal{H}_{perm}$, and $\mathcal{H}_{noise}$ being three cryptographic hash functions. The key K is the only information that needs to be stored at the client.

2. **Verification symbol creation and insertion:** The client creates $k - t$, ℓ-bit verification symbols by applying a pseudo-random function $\phi : \{0,1\}^\tau \times [\![1, k-t]\!] \times \{0,1\}^* \to \{0,1\}^\ell$. Here, a verification symbol is generated by $w_i = \phi(K_{check}, i)$. The file $\mathbf{f}$ is appended with verification symbols $\{w_1, w_2, \ldots, w_{k-t}\}$ to create $\mathbf{u} = (u_1, u_2, \ldots, u_k)$.

3. **Symbol permutation** Let $\Pi : \{0,1\}^\tau \times [\![1,n]\!] \rightarrow [\![1,n]\!]$ be a pseudo-random permutation that shuffles a symbol at position $p \in [\![1,n]\!]$ to position $\Pi(K_{perm}, p)$. The client permutes the symbols in $\mathbf{u}$ by applying Π and generates $\mathbf{m}$.

4. **Application of error correction** The client applies an $(n,k,d,r)_q$ LRC with generator matrix $G^{k\times n}$ to generate $\mathbf{x} = \mathbf{m}G$. Any symbol in $\mathbf{x}$ can be reconstructed by accessing r other symbols.

5. **Noise generation and insertion:** The client applies a pseudo-random function $\phi' : \{0,1\}^\tau \times [\![1,n]\!] \times \{0,1\}^* \rightarrow \{0,1\}^\ell$ to generate n pseudo-random noise symbols $\mathbf{e} = (e_1, e_2, \ldots, e_n)$, where $e_i = \phi'(K_{noise}, i)$. The client generates the verifiable version $\mathbf{y} = \mathbf{x} + \mathbf{e}$. The addition corresponds to a simple bitwise XOR operation.

The client uploads $\mathbf{y}$ to the CSP. The steps of the setup phase are shown in Figure 1.

2.3 Verification Phase

In the verification phase, the verifier V engages in a interactive protocol with the CSP to verify the storage of at least $n-(d-\omega)$ symbols in $\mathbf{y}$. The verification phase follows the steps outlined in the Stealthguard protocol [6]. Briefly, to verify the storage of a verification symbol y_i, the verifier provides to the CSP a nonce η. The CSP constructs v binary matrices $B_1^{a\times b}, B_2^{a\times b}, \ldots, B_v^{a\times b}$, where $a \times b = n$ and v is some parameter of choice. Each element of B_i represents a *witness* for one symbol in $\mathbf{y}$, with all symbols being represented in every matrix. The witnesses for the queried verification symbol are at some fixed position in each matrix, known only to the verifier (because only the verifier knows the permutation in Step 3 of the setup phase). The verifier executes a private information retrieval (PIR) protocol to recover the witnesses corresponding to y_i. As noted in [6], only the witnesses are retrieved from the CSP (not the actual y_i).

Note that the execution of the PIR preserves the privacy of the verification symbol locations in $\mathbf{y}$. The verification process is repeated for γ verification symbols to ensure that $\mathbf{y}$ is stored with the desired reliability degree. The verification symbols queries are randomly selected by the verifier. If any of the γ queries fails, the verifier concludes that $\mathbf{f}$ is not stored with sufficient redundancy. Otherwise, $\mathbf{f}$ is redundantly stored with overwhelming probability. We calculate the value of γ that would satisfy a security parameter τ in Section 3.

2.4 Data Recovery Phase

The data recovery phase is executed when the CSP loses one or more symbols due to some hardware failure. Consider the erasure of symbol $y_i \in \mathbf{y}$. The CSP interacts with the verifier to recover y_i from r other symbols. The recovery of y_i is performed in three steps.

1. **Recovery request:** The CSP provides the coordinate i of the erased symbol y_i to the verifier.

2. **Syndrome creation:** The verifier runs the algorithm *NoiseRec* to generate the *syndrome value* s_m that will allow the CSP to recover y_i. Recall that $\mathbf{x}H^T = 0$ and $\mathbf{y}H^T = \mathbf{x}H^T + \mathbf{e}H^T = \mathbf{e}H^T = \mathbf{s}$. The verifier can recompute the syndrome $\mathbf{s}$ of $\mathbf{y}$

using the public parity check matrix H and $\mathbf{e}$, which is regenerated from K_{noise}. To recover y_i, the verifier selects a column of H^T, say $\mathbf{h}_m$, that has a non-zero entry on the i^{th} row. The verifier computes $s_m = \mathbf{e}\mathbf{h}_m$, and sends (s_m, m) to the CSP.

3. **Symbol recovery:** The CSP recovers y_i by applying the *SymbolRec* algorithm. In particular, the CSP uses the syndrome value s_m provided by the verifier and the m^{th} column of H^T to compute y_i from equation
$$\mathbf{y}\mathbf{h}_m^T = s_m.$$
To recover y_i only the symbols corresponding to the non-zero coordinates of h_m^T need to be retrieved.

3 Security Analysis

In this section, we analyze the security of the PoRL protocol under the completeness and soundness properties defined in Section 1.4.

Theorem 1. *The PoRL is complete.*

Proof. To prove completeness, we show that if the verifier and the CSP are both honest, the CSP should pass a PoRL verification. Suppose that the CSP stores $\mathbf{y}$. In the verification phase, the verifier challenges the CSP by sending a γ queries (challenges) to the CSP. Because the CSP stores $\mathbf{y}$, it can correctly compute $h_i = H(y_i, \eta)$ for all $chal$ and fill the binary matrices $B_1^{a\times b}, B_2^{a\times b}, \ldots, B_v^{a\times b}$. The verifier then performs the PIR on the binary matrices and recovers the v hash bits that correspond to the check symbols y_i. As the hash values at the CSP are generated using the correct check symbol values, the honest CSP passes the verification with probability equal to one. That is,
$$ProofVerif(K, chal, P) = 1.$$
□

Theorem 2. *Let τ be a security parameter for the PoRL and ω be the desired reliability degree. The PoRL is (δ, γ)-sound in the random oracle model, for any $\delta > \delta_{neg}$, $\gamma \geq \gamma_{neg}$, where*
$$\delta_{neg} = \frac{1}{2^\tau},$$
$$\gamma_{neg} = \lceil \frac{ln(2)\tau}{\rho_{neg}} \rceil, \;\; \gamma_{p_{neg}} = \frac{n-k}{n}\gamma_{neg}, \;\; \gamma_{m_{neg}} = \max\{r\gamma_{p_{neg}}, \frac{k}{n}\gamma_{neg}\},$$
$$\rho_{neg} = \frac{2(d-\omega) + 3ln(2)\tau - \sqrt{12(d-\omega)ln(2)\tau + 9\tau^2 ln^2(2)}}{2n} \text{ and } \rho_{neg} < \frac{d-\omega}{n}.$$
Here, $\gamma_{p_{neg}}$ and $\gamma_{m_{neg}}$ denote the minimum number of verification symbols available for querying across parity and information symbols, respectively. If $\gamma_m > \gamma_{m_{neg}}$ and $\gamma_p > \gamma_{p_{neg}}$, it follows that there are at least $\gamma \geq \gamma_{neg}$ verification symbols available for querying during the verification phase. If the CSP succeeds in the soundness game, then there exists an extractor $\mathcal{E}$ that can successfully retrieve $k + \omega$ symbols per subfile with probability no less than $1 - \frac{1}{2^\tau}$, or equivalently, successfully retrieve F (which consists of n_s subfiles) with reliability ω and with probability $1 - \frac{n_s}{2^\tau}$.

Proof. A detailed proof is presented in Appendix A. □

Using theorem 2, we can compute the minimum number of verification symbols that need to be inserted and queried in a file F to guarantee the reliable storage of F. As an example, consider the storage of a file of size 4.78GB divided in 32,768 subfiles. Each subfile $\mathbf{f}_i$ contains 128 symbols of length 256. Assume that $\tau = 60$ and that an (4,556, 4,096, 456, 839)-LRC code with locality $r = 839$ is applied for error correction. Suppose the desired reliability is $\omega = 228$ symbols. According to Theorem 2, to verify that $n - (d - \omega)$ =4,328 symbols are stored for every encoded subfile $\mathbf{y}_i$ with probability no less than $1 - \frac{n_s}{2^\tau} \simeq 1 - \frac{1}{2^{45}}$, at least γ_{neg} =1,714 verification symbols have to be queried in the verification phase.

To ensure the existence of 1,714 verification symbols, the verifiable version of F must contain at least $\gamma_{p_{neg}} = 173$ verification symbol parities, which is approximately 0.0053 per encoded subfile $\mathbf{y}_i$. To guarantee that there are at least 173 verification symbol parities, there should be at least $\gamma_{m_{neg}} = \lceil r\gamma_{p_{neg}} \rceil = 14{,}5147$ information verification symbols inserted in F, which is approximately 4.4295 per encoded subfile $\mathbf{y}_i$. If the adversary successfully responds to 1,714 queries, the probability that 4,328 symbols can be retrieved from each subfile is $1 - \frac{1}{2^{45}}$. Among the $n \cdot n_s$ =149,291,008 coded symbols stored at the CSP for F, the storage overhead due to the insertion of verification symbols is $\gamma_m + \gamma_p$ =145,320 symbols, or approximately 0.0973% of the file size.

4 Confidentiality

A significant portion of the data outsourced to the CSP could be of sensitive nature and therefore, must remain hidden from the CSP and the verifier, if the latter is not implemented in the client. Although we designed the PORL construct with data reliability in mind, the protocol can guarantee the confidentiality of the outsourced file F if the parity matrix H is chosen wisely. Our goal is to hide the information symbols from the CSP and the verifier, while enabling the file verification and maintenance.

Recall that a subfile $\mathbf{f}$ is encoded to $\mathbf{y}$ and stored at the CSP, where $\mathbf{y} = \mathbf{x} + \mathbf{e}$. $\mathbf{e}$ is known to the verifier to perform the maintenance phase. And $\mathbf{x}$ is kept secret to both the verifier and the CSP. Essentially, with H being a pubic information to both the verifier and the CSP, $\mathbf{x}$ is the key to the confidentiality of the file. Once the verifier or the CSP obtained $\mathbf{x}$, it may inverse the encoding operation of ECC and get to $\mathbf{m}$, which contains the plaintext of permuted file symbols and watchdog symbols. Even though $\mathbf{f}$ can't be recovered without knowing the correct order of the symbols, we consider the leakage of $\mathbf{x}$ as a break of confidentiality because further analysis of the file may start from there. A straightforward solution is to encrypt the information symbols $\{f_1, f_2, ..., f_t\}$ at the client before executing the setup phase. In this section, we show that the extra encryption step is not necessary. We first address the file confidentiality with respect to the verifier. And then, we define two levels of confidentiality with respect to the CSP. Later we demonstrate how information confidentiality could be compromised. Afterward, we suggest on the structure of H that preserves confidentiality.

Proposition 1. *The confidentiality of F is preserved to the verifier, if the CSP and the verifier do not collude.*

Proof. A verifier is responsible for performing the file verification and assisting the CSP the file maintenance. For subfile $\mathbf{f}$ of an outsourced file F, the verifier stores the file id fid and the master key K used generate the secret keys in Step 1 of the setup phase. During the verification phase, the verifier receives from the CSP the witnesses that correspond to the information and check symbols that are stored at the CSP. The symbols themselves, are never revealed to the verifier, only symbol witness are sent. A symbol witness is generated by the application of a one-way hash function on the symbol plus the challenge provided by the verifier. So the verifier only knows the hash value of $\mathbf{y}$ not $\mathbf{y}$ itself. To perform the verification, the verifier regenerates the random check symbols that are inserted in Step 2 of the setup phase, as well as the noise used to mask the information and check symbols. The confidentiality of the information symbols is preserved because only the check symbols are verified and due to the one-wayness of the hash function used to create the witnesses for the information symbols. To perform the maintenance, the verifier only calculates the syndrome $\mathbf{s}$ from the noise symbols $\mathbf{e}$ regenerated on the fly from the secret key.

Thus, when the CSP and the verifier do not collude, the verifier can only access $\mathbf{e}$ and the hash of $\mathbf{y}$, so it can not recover $\mathbf{x}$, hence the confidentiality is preserved. Meanwhile, if the CSP and the verifier collude, with $\mathbf{y}$ from the CSP and $\mathbf{e}$ from the verifier, they can solve $\mathbf{x}$ and breach the confidentiality. □

4.1 Breaching the Confidentiality of F at the CSP

Confidentiality of F with respect to the CSP is more complicated, even when the CSP and the verifier do not collude. This is because the CSP stores $\mathbf{y}$, which is dependent on $\mathbf{x}$. Unlike the verifier, who can only access $\mathbf{e}$, which is independent of $\mathbf{x}$. With $\mathbf{y}$ and the publicly known H, the CSP may also access to the syndrome $\mathbf{s}$. It could compute $\mathbf{s} = \mathbf{y}H^T$ as soon as $\mathbf{y}$ is uploaded and stores $\mathbf{s}$ thereafter. From the property of parity check code, we have $\mathbf{s} = \mathbf{y}H^T = \mathbf{x}H^T + \mathbf{e}H^T = \mathbf{e}H^T$. Note that the dimension of H is $(n-k) \times n$. Essentially, $\mathbf{s}$ tells the CSP the value of $n-k$ linear combination of $\mathbf{e}$. These values enables the CSP to breach some info of $\mathbf{e}$ and then solves $\mathbf{x}$ from $\mathbf{y}$.

Proposition 2. *The CSP can never solve more than $n-k$ symbols in $\mathbf{x}$.*

Proof. With $\mathbf{y}$ and $\mathbf{s}$, the process of breaching $\mathbf{x}$ can be seen as the process of solving linear equation systems with $e_i, \forall i \in [\![1, n]\!]$ as the linear independent variables. Yet, because H is of the size $(n-k) \times n$ and $\mathbf{s}$ only contains $n-k$ symbols, they only form $n-k$ linear equations. Yet, to breach a specific x_i, the CSP has to first find the corresponding e_i and then compute $x_i = y_i - e_i$. With $n-k$ linear equations, the CSP can solve at most $n-k$ e_is. Hence, the CSP can solve at most $n-k$ e_is, that is to say it can solve no more than $n-k$ symbols in $\mathbf{x}$. □

Because $\mathbf{x}$ contains n symbols, Proposition 2 guarantees that the CSP can never breach all n symbols of $\mathbf{x}$. However, it is still possible to breach a subset of $\mathbf{x}$. This happens when the a subset $z, z \leq n-k$ of the $n-k$ linear equations contain only z of the linear independent variables. For example, when H is non-binary, if one of the equation is $e_1 - e_2 = s_1$ and another is $e_1 + e_2 = s_2$, the CSP can solve $e_1 = \frac{s_1+s_2}{2}$ and $e_2 = \frac{s_1-s_2}{2}$, hence breach x_1 and x_2. For a binary H, if one of the equation is $e_1 \oplus e_2 \oplus e_3 = s_1$ and another is $e_2 \oplus e_3 = s_2$, it can solve $e_1 = s_1 \oplus s_2$. In the rest of the discussion, we assume H is binary.

Proposition 3. *If every parity check equation of the ECC contains at least two symbols, the CSP can never solve a x_i.*

Proof. Recall that the standard form of the parity check matrix is $H_{std} = [-P^T|I_{n-k}]$. And the linear equations is defined by $\mathbf{e}H^T = \mathbf{s}$. When every parity check equation contains at least two symbols, it means the matrix $-P^T$ has no zero rows. Now, arbitrarily select $z, 1 \leq z \leq n-k$ row(s) from H and form a sub matrix of H, H'. The dimension of H' is $z \times n$. Let's rewrite H' in the form of $H' = [A|B]$, with A as a sub matrix of $-P^T$ and B as a sub matrix of I_{n-k}.

The linear equation system of the error symbols is defined by $\mathbf{e}'H' = 0$, where $\mathbf{e}'$ denotes the subset of error symbols included in these z linear equations. For any column in H', if it's not a zero column, it means the corresponding error symbol is included in the z equations as a variable. Hence, there are z non zero columns in B as it is a sub matrix of I_{n-k} with z rows. Plus there are at least 1 non zero column in A as it is a sub matrix of $-P^T$, which doesn't have zero columns. So, A and B together contains as least $z+1$ non zero columns, which means H' has at least $z+1$ non zeros columns. Therefore, the linear equation system,$\mathbf{e}'H' = 0$, contains at least $z+1$ variables with only z equations. As it is a under-determined linear system, it doesn't have an unique solution. This is true for any $z, z \in [\![1, n-k]\!]$.

Hence, none of e_i can be solve and the confidentiality of $\mathbf{x}$ is preserved. □

5 Storage Overhead Optimization

In this section, we analyze the storage overhead of the PoRL as a function of the LRC code parameters.

5.1 Optimal Code Locality

Assume the application of an $(n, k, d, r)_q$ LRC. Let the code dimension k, the code distance d, the desired reliability ω, the security parameter τ, and the number of subfiles n_s be fixed to some desired values. We are interested in determining the optimal code locality r that maximizes the storage efficiency of the PoRL. We measure storage efficiency via the effective code rate defined as follows:

Definition 3 (Effective code rate (ψ)). *Let γ_m verification symbols be inserted into F before the application of an (n, k, d, r) LRC code $\mathcal{C}$. The effective code rate ψ is defined as:*

$$\psi = \frac{n_s k - \gamma_m}{n_s n} \tag{1}$$

The rate ψ denotes the fraction of information symbols over the total number of symbols stored at the CSP.

Assume the application of an optimal LRC. Such optimal LRCs minimize the code length n to achieve a targeted d, under fixed k and d. The relationship between the code length n and the locality r is given by [9]:

$$n = d + k + \lceil \frac{k}{r} \rceil - 1. \tag{2}$$

From eq. (2), we observe that n decreases with r for fixed k and d. Moreover, from eq. (1), the effective code rate ψ increases with r, due to the decrease of n. However, a larger r potentially increases the number of verification symbols that have to be inserted in the file to guarantee the soundness property with a fixed parameter τ. This is because a larger number of information verification symbols (γ_m) are necessary to generate the desired number of parity verification symbols (γ_p). From the definition of ψ, we observe that the increase of r causes the decrease of both the nominator $k - \frac{\gamma_m}{n_s}$ and the denominator n. Therefore, we analyze the relationship between r and ψ using the following proposition:

Proposition 4. *The effective code rate ψ as a function of r for fixed k, d, ω, n_s and τ is given by:*

$$\psi(r) = \frac{kr - \frac{r\alpha}{n_s}((d-1)r + k)}{(d+k-1)r + k} \tag{3}$$

where

$$\alpha = \frac{2ln(2)\tau}{(2(d-\omega) + 3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega) + 3ln(2)\tau)}} \tag{4}$$

Proof. The proof is provided in the Appendix B. □

Using Proposition 4, we can compute the optimal locality r_{opt}, which maximizes $\psi(r)$. The value of r_{opt} is given in Proposition 5.

Proposition 5. *The optimal locality r_{opt} that maximizes $\psi(r)$ for fixed k, d, ω, n_s and τ is given by:*

$$r_{opt} = \lceil \frac{k}{n_{opt} - d - k + 1} \rceil, \tag{5}$$

$$n_{opt} = \arg\max_{n_i} \{\psi(n_i)\}, \quad n_i : \{\lceil \hat{n}_{opt} \rceil, \lfloor \hat{n}_{opt} \rfloor\}, \tag{6}$$

$$\hat{n}_{opt} = \frac{(d+k-1)n_s - \alpha k + \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha}, \tag{7}$$

$$\alpha = \frac{2ln(2)\tau}{(2(d-\omega) + 3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega) + 3ln(2)\tau)}} \tag{8}$$

Proof. The proof is provided in the Appendix C. □

5.2 Numerical Examples

To demonstrate the code locality/storage efficiency tradeoff, we present several numerical examples. We fixed the security parameter $\tau = 60$, and the reliability degree to $\omega = \frac{d}{2}$, for a file of n_s =32,768 subfiles. We picked three sets of parameters for k and d and computed ψ, γ_{neg} and the fraction of verification symbols per subfile ($\frac{\gamma_m + \gamma_p}{n_s}$), as a function of the code locality r. The numerical results are shown in Figure 2. In every plot of ψ, we mark the optimal effective code rate ψ_{opt} and the corresponding optimal code locality r_{opt}.

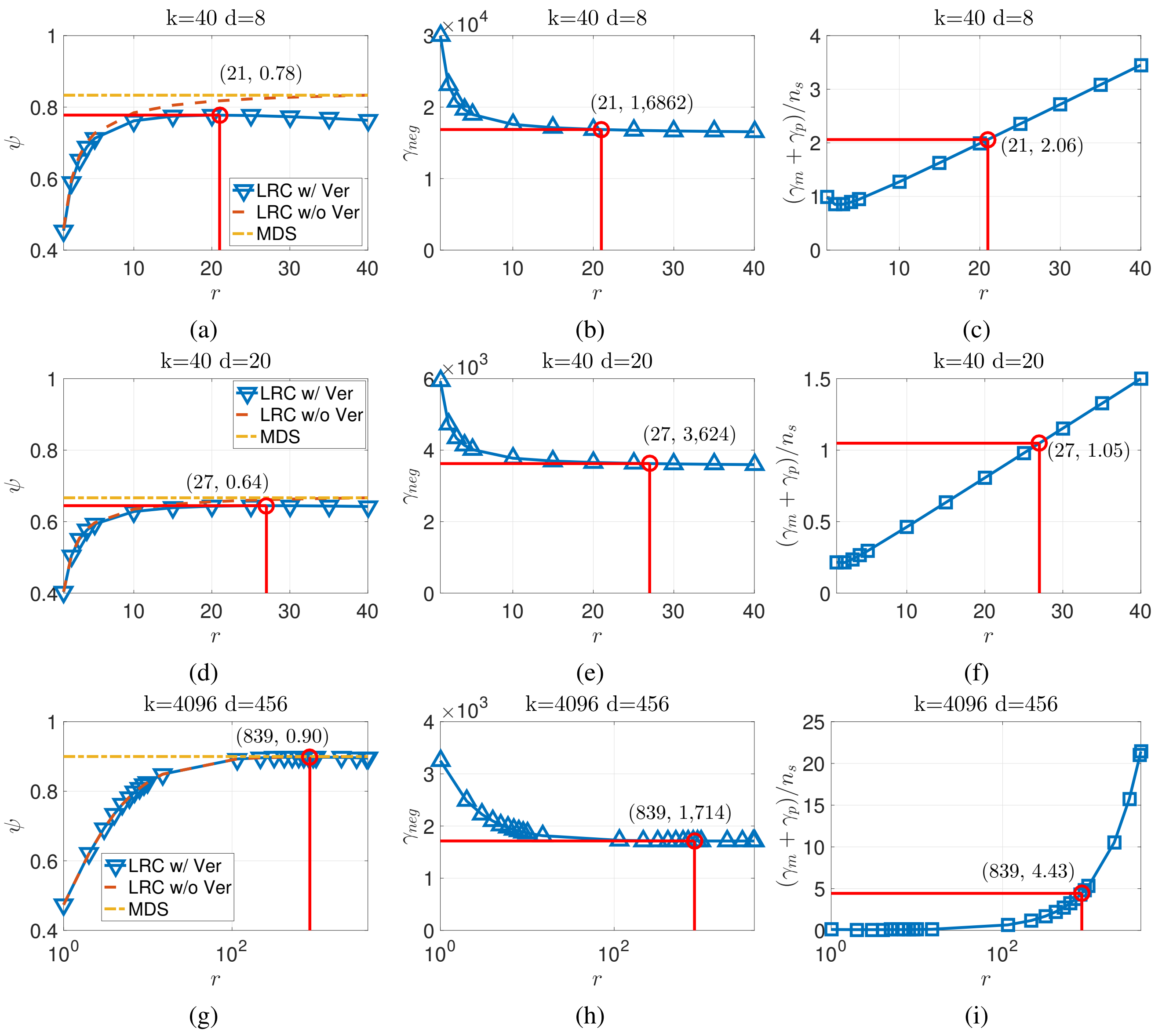


Figure 2: Numeric examples for the effective code rate, number of verification symbols and average number of verification symbols per subfile as a function of the LRC code locality r.

We also show the LRC code rate for different r, if no information verification symbols were inserted. Finally, we show the rate of an MDS code, which minimizes the code length n for achieving d.

In figures 2(a), 2(b), and 2(c), we show the effective code rate ψ, the number of verification symbols, and the average number of verification symbols per split, respectively, as a function of the code locality r, when $k = 40$ and $d = 8$. The maximum effective code rate (0.78) is achieved when $r = 21$. For $r = 21$, 16,862 verification symbols need to be inserted in the file, leading to an average of 2.1 verification symbols per subfile. In Figures 2(d), 2(e), and 2(f), we show the same metrics when the required code distance is increased to $d = 20$, but k is kept equal to 40.

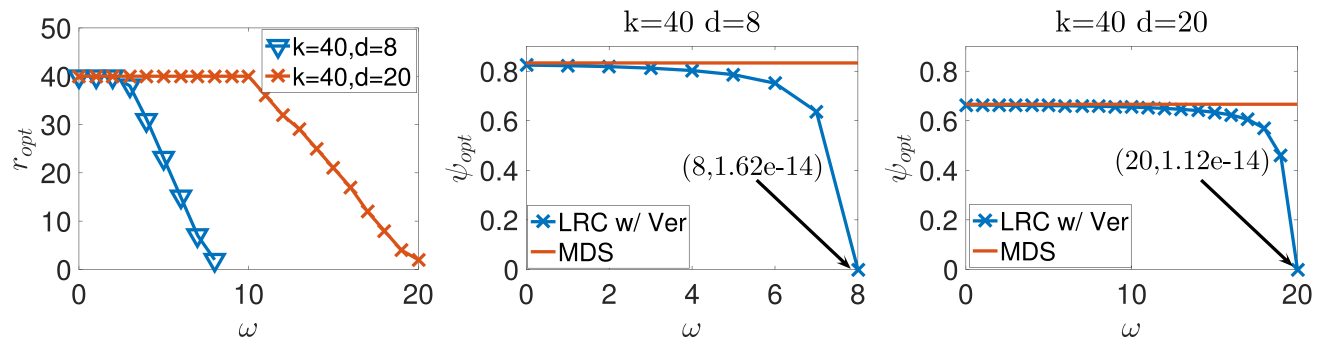
The increase of d has created an interesting phenomenon. The overall maximum effective code rate is reduced. However, this reduction is primarily due to the increase in the number of parities necessary for achieving the desired code distance (refer to eq. (2)) and not the overhead of verification symbols. In fact, the number of verification symbols dropped from 16,862 when $d = 8$ to 3,624 when $d = 20$, yielding a drop by 100% on the average number of verification symbols per subfile. This reduction in overhead is also observed when comparing Figure 2(a) with 2(c). The gap between ψ and the optimal code rate achieved by an MDS code is reduced. This decrease in overhead is primarily due to the higher deletion rate ρ_{adv} allowed to the CSP when $\omega = {}^{d}\!/_{2}$. Fewer verification symbols are needed to detect a larger deletion rate. Another useful observation is that the code locality r_{opt} that maximizes ψ is now increased from 21 in Figure 2 to 27 in Figure 2. This shift in r_{opt} is attributed to the fact that fewer verification parity symbols are needed, and therefore the optimal locality is shifted towards the k.

In figures 2(g), 2(h), and 2(i), we have applied an (4,556, 4,096, 456, $*$) LRC. The optimal effective code rate ψ_{opt} is 0.8981, which is achieved at $r_{opt} = 839$. For comparison, the equivalent MDS code for k =4,096 and d =456 has a code length of 4,552, only four symbol fewer than the LRC. This yields a storage overhead in the order of 0.19% From Figure 2(h), we observe that the verifier has to query 1714 watchdogs across the file to check that at least $k + \omega$ =4,328 symbols are intact in every subfile. The number of verification symbols per subfile is approximately 4.4. Considering there are n =4,556 blocks per split, the overhead for the PoRL verification is negligible.

5.3 Reliability vs. Effective Code Rate

In this section, we analyze the impact of the reliability degree (ω) on the effective code rate. Recall that even though an $(n, k, d, r)_q$ LRC can repair up to d symbols, the reliability level verified by a PoRL is selected to be some value $\omega \leq d$. When k, d, and τ are fixed, a larger ω requires the query of more verification symbols to ensure the retrievability of $n - (d - \omega)$ symbols. The storage overhead for satisfying the PoRL test is therefore increased, as more verification symbols need to be inserted to $\mathbf{f}$ during the the setup phase. To understand the influence of ω on ψ, we fixed τ and n_s to 40 and 32,768 respectively and picked two sets of (k, d), namely $(40, 8)$ and $(40, 20)$. We then computed the r_{opt} that maximizes ψ_{opt} as a function of the reliability degree ω.

In Figure 3(a), we show r_{opt} as a function of ω. At the low ω regime, r_{opt} is approximately k. That is, an MDS code minimizes the storage overhead for satisfying ω. With



(a) (b) (c)

Figure 3: Impact of ω on r_{opt} and ψ_{opt}

the increase of ω, the optimal locality r_{opt} decreased to allow for the generation of more parity verification symbols from a given number of information verification symbols. This matches our intuition. When ω is high, more verification symbol queries are needed to detect the deletion of fewer symbols by the CSP. As γ_m grows linearly with γ_p with slope r, a lower r decreases the required γ_m for generating γ_p. So for high ω, the reduction in storage overhead from inserting fewer verification symbols overcomes the increase in n to satisfy d at lower r.

In figures 3(b) and 3(c), we observe that the effective code rate ψ stayed very close to that of an optimal MDS code for the majority of the ω range. As ω approached d, the effective code rate decreased attaining values close to 0.5 when $\omega = d - 1$ and practically zero when $\omega = d$. For this extreme case, a PoRL must detect the erasure of a single symbol, which can only be achieved if almost all symbols are used for verification, making the PoRL not practical, unless the client/verifier stores $\mathbf{f}$.

References

[1] Apache Software Foundation, "The hadoop distributed file system (HDFS)." http://hadoop.apache.org/docs/stable1/hdfs_design.html, 2016.

[2] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *Procedings of the USENIX ATC Conference*, June 2012.

[3] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *CoRR*, vol. abs/1301.3791, 2013.

[4] M. N. Krishnan, N. Prakash, V. Lalitha, B. Sasidharan, P. V. Kumar, S. Narayanamurthy, R. Kumar, and S. Nandi, "Evaluation of Codes with Inherent Double Replication for Hadoop," in *Proc. Usenix HotStorage*, vol. abs/1406.6783, (Philadelphia, PA), Jun. 2014.

[5] A. Juels and B. S. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *Proceedings of the 14th ACM CCS Conference*, pp. 584–597, 2007.

[6] M. Azraoui, K. Elkhiyaoui, R. Molva, and M. Önen, "Stealthguard: Proofs of retrievability with hidden watchdogs," in *Proceeedings of the European Symposium on Research in Computer Security (ESORICS)*, pp. 239–256, 2014.

[7] K. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*, 2009.

[8] B. Chen, A. K. Ammula, and R. Curtmola, "Towards server-side repair for erasure coding-based distributed storage systems," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 281–288, 2015.

[9] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[10] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *Deep Space Network Progress Report*, vol. 44, pp. 114–116, Jan. 1978.

A Proof of Theorem 2

A.1 Overview

Consider the storage of file $F = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n_s}\}$ at the CSP, encoded to verifiable subfiles $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n_s}$. Suppose the CSP corrupts symbols with a corruption rate ρ_{adv}. We show that if the CSP passes the verification oracle O_{ver} with probability $\delta > \delta_{neg}$, then there is a file extractor algorithm $\mathcal{E}$ that recovers at least $n - (d - \omega)$ symbols from every subfile. We call this a successful retrieval of F with reliability ω.

We first find the maximum ρ_{adv}, denoted by ρ_{neg}, such that if the CSP applies a corruption rate ρ_{neg}, the probability that $\mathcal{E}$ fails to retrieve any subfile with reliability ω is negligible. We then express δ as a function of ρ_{adv} and the number of verification symbols γ queried during the file verification. We compute the minimum number of queries γ_{neg}, such that if the CSP passes O_{ver}, then $\rho_{adv} \leq \rho_{neg}$ with overwhelming probability. In the next step, we compute the minimum number of information verification symbols $\gamma_{m_{neg}}$ and parity verification symbols $\gamma_{p_{neg}}$, that guarantee the presence of at least γ_{neg} verification symbols for querying in O_{ver}. Finally, we show that if $\rho_{adv} \leq \rho_{neg}$, the probability that $\mathcal{E}$ can retrieve F with reliability level ω is $1 - \varepsilon$, where ε is negligible.

A.2 Bounding the Corruption Rate ρ_{neg}

Assume that n, k, d, ω, r, n_s, and τ are fixed to desired values. Let $X_{i,j}$ be a random variable (RV) denoting the probability that the CSP corrupts the j^{th} symbol of the i^{th} subfile. For $X_{i,j}$, we set $X_{i,j} = 1$ if the j^th symbol of $\mathbf{y}_i$ is corrupted by the CSP, and $X_{i,j} = 0$ otherwise. For the verifiable file, it follows that every symbol is corrupted with the same probability ρ_{adv}, i.e., $\Pr(X_{i,j} = 1) = \rho_{adv},\ \forall i, j$. This is because the adversary cannot distinguish between information and verification symbols due to the symbol permutation applied in Step 3 of the Setup phase and the noise inserted in Step 5. Moreover, the adversary cannot distinguish between parity symbols generated from information symbols and those generated by verification symbols, although information and parity symbols are separable due to the systematic nature of the generator matrix G. The latter distinction does not help the CSP in following an intelligent strategy for selecting which symbols it corrupts. As verification are randomly dispersed in a proportional manner between information and parity symbols, the best adversarial strategy is to corrupt symbol at random. This implies that $X_{i,j}$ are independent RVs identically distributed according to the Bernoulli distribution with parameter ρ_{adv}.

Let $P^{\mathcal{E}}_{fail,\mathbf{y}_i}$ denote the probability of failing to retrieve a subfile $\mathbf{y}_i$ with reliability ω using $\mathcal{E}$. This probability is calculated as

$$P^{\mathcal{E}}_{fail,\mathbf{y}_i} = \Pr(N_i \geq d-\omega+1) = \sum_{x=d-\omega+1}^{n} \binom{n}{x} \rho_{adv}^{x}(1-\rho_{adv})^{n-x}. \tag{9}$$

In (9), N_i is an RV denoting the number of corrupted symbols in the i^{th} subfile, which follows the binomial distribution (sum of Bernoulli RVs). $P^{\mathcal{E}}_{fail,\mathbf{y}_i}$ expresses the probability that at least $d-\omega+1$ symbols of $\mathbf{y}_i$ are corrupted. Using the Chernoff bound, $P^{\mathcal{E}}_{fail,\mathbf{y}_i}$ can be bounded to

$$P^{\mathcal{E}}_{fail,\mathbf{y}_i} \leq exp(-\frac{n\rho_{adv}}{3}(1-\frac{d-\omega}{n\rho_{adv}})^2). \tag{10}$$

Let $\mathcal{E}$ fail to retrieve $\mathbf{y}_i$ with negligible probability $\frac{1}{2^\tau}$ (where τ is the desired security parameter). Then it follows that

$$P^{\mathcal{E}}_{fail,\mathbf{y}_i} \leq \frac{1}{2^\tau} \Rightarrow \tag{11}$$

$$\exp(-\frac{n\rho_{adv}}{3}(\frac{d-\omega}{n\rho_{adv}}-1)^2) \leq \frac{1}{2^\tau}. \tag{12}$$

Since $P^{\mathcal{E}}_{fail,\mathbf{y}_i}$ is an increasing function of ρ_{adv}, we want to find the minimum ρ_{adv}, denoted by ρ_{neg}, that limits $P^{\mathcal{E}}_{fail,\mathbf{y}_i}$ below $\frac{1}{2^\tau}$. This is done by solving for ρ_{adv} when the equality holds in (12).

$$\exp(-\frac{n\rho_{neg}}{3}(\frac{d-\omega}{n\rho_{neg}}-1)^2) = \frac{1}{2^\tau}, \quad \rho_{neg} < \frac{d-\omega}{n}. \tag{13}$$

This yields

$$\rho_{neg} = \frac{(2(d-\omega)+3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}{2n} \tag{14}$$

From (14), if $\rho_{adv} \leq \rho_{neg}$, then $P^{\mathcal{E}}_{fail,\mathbf{y}_i} \leq \frac{1}{2^\tau}$. That is, $\mathcal{E}$ can successfully retrieve $\mathbf{y}_i$ with reliability ω and probability $1-\frac{1}{2^\tau}$.

A.3 Bounding the Number of Queries γ

In this part of the proof, we express the probability δ of succeeding in the soundness game as a function of the watchdog queries γ and the corruption rate ρ_{adv}. The adversary returns a valid proof for a single verification symbol query if (a) the verification symbol is stored intact at the CSP or (b) the verification symbol is corrupted, but the q bits of the returned hash match (collide) with the hash of the true verification symbol. Let w_j denote the watchdog being queried. The two events occur with probability

$$P_{w_j}^{CSP} = (1 - \rho_{adv}) + \frac{\rho_{adv}}{2^q}. \tag{15}$$

In (15), the first term is the probability that w_j is not corrupted, whereas the second term is the hash collision probability for a hash of length q, assuming that the watchdog is corrupted. The probability of succeeding in a soundness game with γ independent queries equals the probability of providing γ valid proofs. This is simply the union of γ independent events.

$$P_{success}^{CSP} = (1 - \rho_{adv} + \frac{\rho_{adv}}{2^q})^{\gamma}. \tag{16}$$

From (16), we observe that $P_{success}^{CSP}$ decreases with both γ and ρ_{adv}. We now fix $\rho_{adv} = \rho_{neg}$ and compute the minimum number of queries γ_{neg} for which the adversary's success in the soundness game is bounded by some negligible value to the security parameter τ. That is,

$$P_{success}^{CSP} = (1 - \rho_{neg} + \frac{\rho_{neg}}{2^q})^{\gamma} \leq \frac{1}{2^{\tau}} \tag{17}$$

From (17), we solve for the value of γ for which equality holds and set the solution to γ_{neg}.

$$(1 - \rho_{neg} + \frac{\rho_{neg}}{2^q})^{\gamma_{neg}} = \frac{1}{2^{\tau}}, \tag{18}$$

which leads to

$$\gamma_{neg} = \lceil \frac{-ln(2)\tau}{ln(1 - \rho_{neg} + \frac{\rho_{neg}}{2^q})} \rceil \simeq \lceil \frac{ln(2)\tau}{\rho_{neg}} \rceil. \tag{19}$$

The γ_{neg} denotes the minimum number of verification symbols that need to be queried such that if the adversary succeeds in the soundness game, then $\rho_{adv} \leq \rho_{neg}$ with probability $1 - \frac{1}{2^{\tau}}$, and therefore subfile $\mathbf{y}_i$ can be recovered with reliability ω.

A.4 Bounding $\gamma_{m_{neg}}$ and $\gamma_{p_{neg}}$ for γ_{neg}

To query γ_{neg} verification symbols, at least that many must be inserted to F. In this part of the proof, we treat γ_{neg} as a union of $\gamma_{m_{neg}}$, the number of information verification symbols, and $\gamma_{p_{neg}}$, the number of parity verification symbols. By substituting ρ_{neg} in (19) with (14), we obtain:

$$\begin{aligned} \gamma_{neg} &= \frac{2nln(2)\tau}{(2(d-\omega) + 3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega) + 3ln(2)\tau)}} \\ &= c_0 n, \end{aligned} \tag{20}$$

where

$$c_0 = \frac{2ln(2)\tau}{(2(d-\omega)+3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}. \tag{21}$$

In (20), we have omitted the ceiling function for simplicity. Moreover, c_0 is a constant when d, ω, and τ are fixed to desired values. Thus, γ_{neg} becomes a linear function of n, indicating that the number of verification symbols that need to be inserted is proportional to the code length n. To guarantee that the corruption rate ρ_{adv} for succeeding in the soundness game is bounded by ρ_{neg} even if the adversary selectively targets either the information symbols or the parity symbols, we require that (20) is satisfied by both the number of information verification symbols and the number of parity verification symbols. Therefore, we set

$$\gamma_{m_{neg}} = \frac{k}{n}\gamma_{neg}, \quad \gamma_{p_{neg}} = \frac{n-k}{n}\gamma_{neg}. \tag{22}$$

Moreover, parity verification symbols are generated by information verification symbols when applying the LRC code. To insert $\gamma_{p_{neg}}$ parity verification symbols, at least $r \cdot \gamma_{p_{neg}}$ information verification symbols must be inserted. That is, to satisfy that there are γ_{neg} watchdogs to be queried during the soundness game, at least $\gamma_{m_{neg}} = \max\{\frac{n-k}{n}\gamma_{neg}, r \cdot \gamma_{p_{neg}}\}$ information verification symbols must be inserted in the setup phase, where $\gamma_{p_{neg}} = \frac{n-k}{n}\gamma_{neg}$.

A.5 Proof Summary

We now summarize the proof. By inserting $\gamma_{m_{neg}} = \max\{\frac{n-k}{n}\gamma_{neg}, r \cdot \gamma_{p_{neg}}\}$ information verification symbols in F, we guarantee $\gamma_{p_{neg}} = \frac{n-k}{n}\gamma_{neg}$ parity verification symbols exist in the verifiable version of F. This leads to at least γ_{neg} verification symbols overall, which are indistinguishable from information or parity symbols. For an adversary succeeding in the soundness game with γ_{neg} verification symbol queries, the probability that $\rho_{adv} > \rho_{neg}$ is less than $\frac{1}{2^\tau}$. Hence, the probability that $\mathcal{E}$ fails to retrieve $\mathbf{y}_i$ with reliability rel is $\frac{1}{2^\tau}$. For $\mathcal{E}$ to successfully retrieve F with reliability degree ω, all subfiles $\mathbf{y}_i, \mathbf{y}_2, \ldots \mathbf{y}_{n_s}$ must be successfully retrieved. Thus, F is successfully retrieved with probability $P_s = (1-\frac{1}{2^\tau})^{n_s} \geq 1 - \frac{n_s}{2^\tau}$.

Let $\varepsilon = \frac{n_s}{2^\tau}$, where ε can be made arbitrary small by selecting τ. Then, if the CSP succeeds in the soundness game with $\gamma \geq \gamma_{neg}$ verification symbol queries, $\mathcal{E}$ can successfully retrieve F with reliability ω and with probability no less than $1-\varepsilon$, where ε is negligibly small.

B Proof of Proposition 1

In this proposition, we analytically compute the effective code rate ψ as a function of the code locality r, for fixed k, d, ω, n_s and τ. Recall from Definition 3, that the effective code rate ψ is given by

$$\psi(r) = \frac{n_s k - \gamma_m}{n_s n} \tag{23}$$

For an (n, k, d, r) LRC, the code length n is a function of the locality r when k and d are fixed, and is given by [9] (for the simplicity of the analytic expression of ψ, we ave assumed that $r|k$):

$$n = d + k + \frac{k}{r} - 1. \tag{24}$$

To guarantee the storage of F with reliability ω and with security parameter τ, the client needs to insert γ_m information verification symbols, where γ_m is given in Theorem 2. The γ_m is computed from the necessary number of parity verification symbols to perform a PoRL test and is given by

$$\begin{aligned}
\gamma_m &= r\gamma_p, \quad \gamma_p = \gamma_{neg}\frac{n-k}{n} \text{ with} && (25)\\
\rho_{neg} &= \frac{(2(d-\omega) + 3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}{2n} && (26)
\end{aligned}$$

The substitution of γ_m from (25) to the definition of ψ in (23) yields

$$\begin{aligned}
\psi(r) &= \frac{n_s k - r \times \frac{n-k}{n} \times \frac{ln(2)\tau}{\frac{2(d-\omega)+3ln(2)\tau-\sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}{2n}}}{n_s n}\\
&= \frac{n_s k - r \times (n-k) \times \alpha}{n_s n}\\
\alpha &= \frac{2ln(2)\tau}{(2(d-\omega)+3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}\\
&= \frac{ln(2)\tau}{n\rho_{neg}}\\
&= \frac{\gamma_{neg}}{n}
\end{aligned} \tag{27}$$

Note that α does not depend on n nor r, and thus can be treated as a constant when d, ω, and τ are fixed to desired values. Further substitution of n in (23) by its expression in (24), yields ψ as a function of r

$$\begin{aligned}
\psi(r) &= \frac{n_s k - r \times (d + k + \frac{k}{r} - 1 - k) \times \alpha}{n_s(d + k + \frac{k}{r} - 1)}\\
&= \frac{n_s k - r \times (d + \frac{k}{r} - 1) \times \alpha}{n_s(d + k + \frac{k}{r} - 1)}\\
&= \frac{n_s k r - r\alpha((d-1)r + k)}{n_s((d+k-1)r + k)},\\
\alpha &= \frac{2ln(2)\tau}{(2(d-\omega)+3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega)+3ln(2)\tau)}}.
\end{aligned} \tag{28}$$

C Proof of Proposition 2

In this proposition, we compute the optimal locality r_{opt} that maximizes the effective code rate ψ. To find r_{opt}, we analyze the monotonicity of ψ using the expression that was computed in Proposition 1. Form the expression in (28), it is easier to analyze ψ as function of n as opposed to r. We therefore substitute r with its equivalent expression of n from (24), and find the optimal value of n, denoted as n_{opt} that maximizes ψ. Using (24), we can then compute the corresponding optimal locality ω_{opt} (not that n is a monotonic with r and therefore ψ can be optimized wrt any of the two variables).

To find n_{opt}, we express ψ as a function of n:

$$\psi(n) = \frac{n_s k - \frac{k}{n-d-k+1} \times (n-k) \times \alpha}{n_s n} \tag{29}$$

$$\alpha = \frac{2ln(2)\tau}{(2(d-\omega) + 3ln(2)\tau) - \sqrt{3ln(2)\tau(4(d-\omega) + 3ln(2)\tau)}} \tag{30}$$

By taking the derivative of ψ with respect to n, it follows that

$$\psi'(n) = \frac{\partial \psi}{\partial n} = \frac{\frac{\alpha k}{n_s(d+k-n-1)} - \frac{\alpha k(k-n)}{n_s(d+k-n-1)^2}}{n} - \frac{k - \frac{\alpha k(k-n)}{n_s(d+k-n-1)}}{n^2} \tag{31}$$

Solving for n_{opt} by setting $\psi'(n) = 0$, it follows that

$$\frac{\frac{\alpha k}{n_s(d+k-\hat{n}_{opt}-1)} - \frac{\alpha k(k-\hat{n}_{opt})}{n_s(d+k-\hat{n}_{opt}-1)^2}}{\hat{n}_{opt}} - \frac{k - \frac{\alpha k(k-\hat{n}_{opt})}{n_s(d+k-\hat{n}_{opt}-1)}}{\hat{n}_{opt}^2} = 0 \tag{32}$$

There are two solutions of n_{opt}:

$$\begin{aligned} n_{opt_1} &= \frac{(d+k-1)n_s - \alpha k - \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha} \\ n_{opt_2} &= \frac{(d+k-1)n_s - \alpha k + \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha} \end{aligned} \tag{33}$$

For an (n, k, d, r) LRC, it holds that $r \leq k$ and $k + d \leq n \leq d + 2k - 1$. Moreover, because ω is the reliability degree then $\omega \leq d$. Finally, $\alpha = \frac{\gamma_{neg}}{n}$. Because γ_{neg}, which denotes the total number of watchdog symbols to be queried in one challenge in the verification phase should be bounded in a practical system, α should be a relatively small value compared to n_s. That is, $\alpha < n_s$. Given the parameter constraints, we transform n_{opt_1} as follows:

$$\begin{aligned} n_{opt_1} &= \frac{(d+k-1)n_s - \alpha k - \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha} \\ &= \frac{(d+k-1)(n_s - \alpha) + \alpha(d-1) - \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha} \\ &= (d+k-1) + \frac{\alpha(d-1) - \sqrt{\alpha(d-1)((d+k-1)n_s - \alpha k)}}{n_s - \alpha} \end{aligned} \tag{34}$$

To further analyze (34), we compare $\alpha(d-1)$ with $\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)}$ as follows

$$\begin{aligned}
(\alpha(d-1))^2 - (\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)})^2 && (35)\\
= \alpha(d-1)(\alpha(d-1)-(d+k-1)n_s+\alpha k) && (36)\\
= \alpha(d-1)(\alpha-n_s)(d+k-1) && (37)
\end{aligned}$$

Since $\alpha > 0$, $d-1>0$, $d+k-1>0$ and $\alpha < n_s$, we have $(\alpha(d-1))^2 - (\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)})^2 \leq 0$, which tells:

$$\alpha(d-1) \leq \sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)} \tag{38}$$

Combing (38) and (34), we have

$$n_{opt_1} \leq d+k-1 \tag{39}$$

which contradicts the range of n. Therefore n_{opt_1} is not a valid solution. Next, we exam the validity of n_{opt_2}. To do so, we computing the lower and upper bound of n_{opt_2}. Because $n_{opt_1} < d+k-1$ and $n_{opt_1}+n_{opt_2} = 2(d+k-1)+\frac{2\alpha(d-1)}{n_s-\alpha}$, we obtain that $n_{opt_2} > 2(d+k-1)+\frac{2\alpha(d-1)}{n_s-\alpha} - n_{opt_1} > d+k-1$ as $d-1\geq 0$, $\alpha>0$ and $n_s-\alpha>0$. To compute the upper bound, we examine the difference between n_{opt_2} and n_{opt_1}.

$$n_{opt_2} - n_{opt_1} = \frac{2\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)}}{n_s-\alpha} \tag{40}$$

In LRC, generally $d-1<k$. Thus,

$$\begin{aligned}
n_{opt_2} - n_{opt_1} &= \frac{2\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)}}{n_s-\alpha}\\
&< \frac{2\sqrt{\alpha k(2kn_s-\alpha k)}}{n_s-\alpha}\\
&< \frac{2\sqrt{2\alpha k^2 n_s}}{n_s-\alpha}\\
&= \frac{2k\sqrt{2\alpha n_s}}{n_s-\alpha}
\end{aligned} \tag{41}$$

Because n_s is a relatively large value compare to α, it's reasonable to assume $\frac{2\sqrt{2\alpha n_s}}{n_s-\alpha} \leq 1$. Thus $n_{opt_2}-n_{opt_1} \leq k$. Recall $n_{opt_1} \leq d+k-1$, we have $n_{opt_2} \leq d+2k-1$. By combining the Putting the lower and upper bound on n_{opt_2}, we have $d+k-1 < n_{opt_2} \leq d+2k-1$, which verifies that n_{opt_2} is a valid solution. In conclusion,

$$n_{opt_2} = \frac{(d+k-1)n_s-\alpha k+\sqrt{\alpha(d-1)((d+k-1)n_s-\alpha k)}}{n_s-\alpha} \tag{42}$$

In our calculations so far, we have ignored that n_{opt} and r_{opt} are integer parameters of a linear code. The optimal value of n is either $\lceil n_{opt_2} \rceil$ or $\lfloor n_{opt_2} \rfloor$, depending on which one yields the lower ψ That is,

$$n_{opt} = \arg\max_{n_i} \{\psi(n_i)\}, \quad n_i : \{\lceil n_{opt_2} \rceil, \lfloor n_{opt_2} \rfloor\}. \tag{43}$$

The optimal code locality r_{opt} is calculated according to (2) as $r_{opt} = \lceil \frac{k}{n_{opt}-d-k+1} \rceil$.