Robust Remote Data Checking

Reza Curtmola
Department of Computer Science
New Jersey Institute of Technology
Newark, NJ
crix@njit.edu

Osama Khan Randal Burns
Department of Computer Science
Johns Hopkins University
Baltimore, MD
{okhan, randal}@cs.jhu.edu

ABSTRACT

Remote data checking protocols, such as provable data possession (PDP) [1], allow clients that outsource data to untrusted servers to verify that the server continues to correctly store the data. Through the careful integration of forward error-correcting codes and remote data checking, a system can prove possession with arbitrarily high probability. We formalize this notion in the *robust data possession* guarantee. We distill the key performance and security requirements for integrating forward error-correcting codes into PDP and describe an encoding scheme and file organization for robust data possession that meets these requirements. We give a detailed analysis of this scheme and build a Monte-Carlo simulation to evaluate tradeoffs in reliability, space overhead, and performance. A practical way to evaluate these tradeoffs is an essential input to system design, allowing the designer to choose the encoding and data checking protocol parameters that realize robust data possession.

Categories and Subject Descriptors

H.3.2 [**Information Storage and Retrieval**]: Information Storage; E.4 [**Coding and Information Theory**]: Error Control Codes; E.3 [**Data Encryption**]

General Terms

Security, Reliability, Performance

Keywords

Remote data checking, spot checking, provable data possession, PDP, archival storage, storage security, error-correcting codes

1. INTRODUCTION

Remote or outsourced storage allows clients with limited resources or limited expertise to store and distribute large amounts of data at low costs by using third parties, known as Storage Service Providers (SSPs). An SSP may not always be trusted to store the data with which it has been entrusted. For example, an SSP may try to hide data loss incidents in order to preserve its reputation or it may discard data that are rarely accessed in order to sell the same storage resource multiple times.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
StorageSS'08, October 31, 2008, Fairfax, Virginia, USA.
Copyright 2008 ACM 978-1-60558-299-3/08/10 ...$5.00.

Recent work focuses on adding a *data possession* [1] or *retrievability* [6, 15] guarantee to systems for remote archival/backup. More precisely, a scheme for remote data checking allows a client to periodically challenge the server (SSP) to prove that, at the time of the challenge, the server possesses the same exact data that was initially stored by the client. (The subtle differences between possession and retrievability [3] are not relevant to this paper.) These schemes were designed to meet the unique characteristics of archival storage systems (*e.g.*, operating on very large amounts of data), and are able to provide the data possession guarantee *without returning the data to the client* and *without having the server access all the data*. The latter is achieved through *spot checking* in which the client samples random portions of the data stored at the server. Spot checking allows the client to detect if a fraction of the data stored at the server has been corrupted, which leads to solutions that minimize I/O on the server but which are probabilistic in nature and cannot detect corruption of small parts of the data (*e.g.*, 1 byte).

Integrating error-correcting codes with spot checking enhances the possession guarantee in that the codes recover the data when a small amount of the file has been deleted. To damage a file, an attacker must delete a large and, thus, easily detectable amount of data. This concept was mentioned in early works [6] and plays a more prominent role as the remote storage community moves from the theoretical to the practical [3, 15]. The same pairing of erasure coding with data checking has been used by remote storage systems that distribute or replicate data among many servers [7, 14].

In this paper, we examine the challenges that arise from the integration of Forward Error Correction (FEC) codes with remote data checking schemes that rely on spot checking. We start by putting forth a set of desirable properties of remote data checking schemes, which must be considered when FEC codes are used to improve the data possession guarantee. We then show that the integration of FEC codes is non-trivial and can lead to schemes with significantly different properties and performance characteristics. Indeed, the proposed use of FEC codes in the frameworks of [6, 15] is not optimal and may lead to reduced performance. Independently, Bowers *et al.* [3] proposed an optimized application of FEC codes that is nearly identical to the scheme we present in Section 5. They establish the bounds under which a client is able to retrieve its data from the server. Our focus is more practical. We give a detailed analysis of the effects of adversarial corruption on the data that provides specific guidance as to the selection of encoding parameters and the manner in which client/server possession challenges should be conducted.

We focus on the Provable Data Possession (PDP) [1] framework, as being representative for remote data checking based on spot checking. PDP allows easy and immediate integration with FEC

codes to improve the data possession guarantee: A file F is first encoded using a FEC code and PDP is then applied on the encoded file $\tilde{F}$ (instead of F). The original PDP framework provides the ability to detect if the server corrupts a fraction of F. When combined with an appropriate FEC code, a PDP scheme provides the following *robust data possession guarantee* for the encoded file $\tilde{F}$:

- Protection against corruption of a *large* portion of $\tilde{F}$: The client will *detect* with high probability if the server corrupts more than a δ-fraction of $\tilde{F}$;
- Protection against corruption of a *small* portion of $\tilde{F}$: The client will *recover* the data in F with high probability if the server corrupts at most a δ-fraction of $\tilde{F}$.

On the Adversarial Model. Protection against corruption of a large portion of the data is necessary in order to handle servers that discard a significant fraction of the data. This applies to servers that are financially motivated to sell the same storage resource to multiple clients.

Protection against corruption of a small portion of the data is necessary in order to handle servers that try to hide data loss incidents. This applies to servers that wish to preserve their reputation. Data loss incidents may be accidental (*e.g.*, management errors or hardware failures) or malicious (*e.g.*, insider attacks).

Improving the Data Possession Guarantee. The improvement in the data possession guarantee offered by PDP extends the usability of PDP to different storage applications that require all parts of the data to be protected. Protection against corruption of a large amount of the data inhibits cheating for resource management reasons: Little space can be reclaimed undetectably, making it unattractive to delete data to save on storage costs or sell the same storage multiple times. However, many data are valuable beyond their storage costs, making attacks that corrupt small amounts of data practical. For example, modifying a single bit may destroy an encrypted file or invalidate authentication information. Protection against corruption of a small amount of the data makes PDP suitable for protecting the data itself, not just the storage resource.

Contributions. In this paper, we make the following contributions:

- We identify the requirements that must be considered when choosing an FEC code to improve the data possession guarantee. These are important because different FEC codes result in trade-offs in performance, flexibility and reconfigurability, rate of error correction and output data format.
- We show that remote data checking schemes based on spot checking (and in particular PDP [1]) allow easy and immediate integration with FEC codes to improve the data possession guarantee and we give a specific encoding and file layout scheme that meets the specified requirements.
- We provide a detailed analysis of our scheme that quantifies the probability of the success of an attacker given different encodings, attack strategies, and client checking strategies. This analysis can be used by system designers to choose encoding parameters and data checking disciplines.

Our findings and analysis are general and can be applied to any spot checking scheme that uses FEC codes to improve the data possession guarantee when data are outsourced at untrusted servers.

2. RELATED WORK

In the Introduction, we covered the most closely related work. This includes protocols for provable data possesion [1] and proofs of retrievability [6, 15] in which a remote server can be shown to possess previously stored data through spot checking and without transmitting data back to the checking process. The issue of integrating forward error-correcting codes was initially identified [6] and then specific encodings have been proposed based on Online codes [15] or Reed-Solomon codes [3]. Combining FEC codes with remote data checking schemes is non-trivial and may lead to schemes that are either impractical (*e.g.*, [15] uses an FEC code that covers the entire file) or that lack useful properties (*e.g.*, [6] does not meet the sequentiality requirement identified in Section 4). Bowers et al. [3] show that adding a layer of erasure coding to precompute challenge-response values can enhance proofs of retrievability under certain attack scenarios.

The combination or erasure coding and data checking has also been used to enhance data protection in systems that distribute data sets across multiple servers. These data checking protocols are asymptotically less efficient. Kotla et al. [7] use hierarchical erasure coding in which both the client and server compute FECs. In their model, the server reveals the parameters of the encoding it uses to make storage safe from failures. Based on these parameters, the client chooses complementary erasure coding parameters. Schwarz and Miller [14] present a data checking scheme for distributed erasure-coded data that is based on comparing the unencoded data with the redundant check blocks in a RS encoded file. The scheme only allows sampling within each file block, whereas we focus on schemes that allow sampling across blocks.

Other extensions to remote data checking include extending the data possession guarantee to cover multiple replicas without encoding each replica separately [4] and to efficiently support dynamic data updates [2]. Shah et al. describe techniques for ensuring that remote data checking is privacy preserving [16], *i.e.*, reveals nothing about the content of the data to the auditors, that extend to remote data checking.

3. PRELIMINARIES

Remote data checking (RDC) schemes. A remote data checking scheme allows a client that has stored data at an untrusted server to verify that the server possesses the original data without retrieving it. The data is represented as a file split into blocks. In a *Setup* phase, the client pre-processes the file, generating a piece of verification metadata for each file block, and then stores the file and the verification metadata on the server. The verification metadata can be either distinct from the actual file blocks (*e.g.*, verification tags [1, 15]), or it can be integrated into and be indistinguishable from the file blocks (*e.g.*, sentinels [6]). Later, in a *Challenge* phase, the client randomly samples a small number of file blocks that is independent of f (*i.e.*, *spot checks*); the server provides a *proof of data possession* based on the sampled file blocks and the corresponding verification metadata. By only exchanging an amount of information sub-linear in f, spot checking allows the client to detect with high probability if a fraction of the file stored at the server has been corrupted. Ateniese *et al.* [1] analyze the probabilistic guarantees offered by an RDC scheme based on spot checking.

The details of various RDC schemes (such as PDP [1] or POR [6, 15]) are not important for the purpose of this paper. The simplest RDC scheme stores on the server a message authentication code (MAC) for each file block; the client later retrieves a number of randomly selected file blocks and their corresponding MACs.

Error-correcting codes. We say that a code $\mathcal{C}$ is a (n, k, d) error-correcting code if it encodes a message of k symbols into a codeword of n symbols and has minimum distance d. The minimum distance d is the minimum Hamming distance between any two distinct codewords of $\mathcal{C}$ and reflects the code's ability to handle errors. We will use Reed-Solomon [13] (RS) codes which are Maximum Distance Separable (MDS) codes (*i.e.*, can tolerate as many erasures as their overhead, *e.g.*, $n - k$). We use the notation (n, k) RS code to denote a RS code that can correct up to $n - k$ erasures.

For data checking protocols, we are concerned with erasure correction not error correction. The integrity of data is independently verifiable, *e.g.*, through tags in PDP [1]. Any data error will be detected and the erroneous data will be omitted, converting an error into an erasure.

The rate of a code is $R = k/n$. We consider *systematic* codes, which are codes that embed the unmodified input in the encoded output (*e.g.*, the first k symbols of a codeword are the same k symbols of the original message). We refer to the redundant symbols in the codeword as *check* symbols.

We consider block codes, *i.e.* codes that work on fixed-size symbols. When applied to our constructions, we will assume symbols are blocks of a specified bit length.

Finally, we will be using forward error correction (FEC) codes, which are codes that include redundant symbols in the codewords.

4. STORAGE REQUIREMENTS

Different forward error-correcting codes present trade-offs in performance, flexibility and reconfigurability, rate of error correction, and output data format. All of these factors influence the usability of a coding scheme for storage auditing. The important characteristics include:

- The output of the forward error-correcting encoding should preserve the sequential order of blocks in the original file so that access to contiguous regions of the original file are efficient in the encoded version.
- The computation of forward error-correcting codes should be efficient so that it minimally degrades the performance (throughput and latency) of the storage system.
- The redundancy encoding should minimally expand the file in order to limit space overhead.
- The coding should regionalize data constraints (dependencies among encoded blocks and the original data). This allows for efficient incremental calculation of codes, which increases performance when accessing or updating small portions of a file and when repairing erasures.

5. THE ROBUST DATA POSSESSION GUARANTEE

The data possession guarantee offered by a remote data checking scheme for a file $\mathtt{F}$ can be transformed into a *robust data possession guarantee* for $\mathtt{F}$ by first using an FEC code to encode $\mathtt{F}$ into $\tilde{\mathtt{F}}$, and then using the encoded file $\tilde{\mathtt{F}}$ as input to the RDC scheme. This is a generic transformation that can be applied to any remote data checking scheme matching the description in Section 3.

There are various ways to perform the encoding step, which can lead to remote data checking schemes with significantly different properties and performance characteristics. We will compare two such encodings that meet most or all of our requirements. For efficiency, both use RS codes with fixed parameters applied to subportions of the file. They both also rely on the careful application of permutation and encryption to conceal the dependencies among blocks within each sub-region from the attacker. The first one gives an attacker no information about the constraints among file blocks, but the original file data lack contiguity in the output. The second one gives an attacker limited information about constraints, but outputs the original data unmodified. Our analysis reveals that this extra information does not noticeably decrease the robust possession guarantee.

Before presenting our encoding schemes, we examine alternate encoding strategies that do not work well in practice.

5.1 Impractical Coding Schemes

The requirements of PDP and storage applications renders many applications of FECs impractical for reasons of inefficiency or because the coding does not lead to robust possession guarantees.

Full File Redundancy. Ideally, we would use an FEC code that covers the entire file. This is possible with a RS (n, f) code in which f is the file size (in symbols). This would give an even stronger guarantee than our robust possession guarantee in that this code deterministically corrects up to $f - n$ deletions and not just with high probability.

However, such an FEC code would be inefficient, because its parameters depend on the file size. While very fast for small fields, RS codes become quite inefficient to compute for even modest widths, because they are based on inverting matrices. They may be computed in $\mathrm{O}(n \log n)$ and decoded in $\mathrm{O}(n^2)$. Faster implementations are possible, but these are only asymptotically more efficient and do not apply to practical codeword lengths [9]. Also, each unique n, f pair requires a new encoding matrix of size $f \times n$.

Encoding at the Server. Error correction can be independently applied by the server to protect against accidental data loss. This has the advantage of relieving the client of the burden of error correction encoding. However, applying error correction on the server cannot guarantee the same level of protection in case of malicious data corruption (*e.g.*, insider attacks caused by server compromise, which is a realistic possibility [5,11]). Thus, we apply error correction on the client side. This results in a unified protection scheme, regardless of what happens on the untrusted server.

Rateless Erasure Codes. Rateless erasure codes [9, 10, 12] are codes in which the code messages are computed as an XOR of a random set of input blocks. Thus, they lack the sequentiality property that we desire. Additionally, with rateless erasure codes, the original file can only be reconstructed in the presence of a large fraction of these messages [8]. This means that small portions of the file may not be accessed independently, which is problematic when reading data and when recovering from deletions.

5.2 FEC Codes for PDP

For performance reasons, it is desirable to fix the parameters of the RS encoding. This also fixes the code generation matrix. We divide the file $\mathtt{F}$ into k-block chunks[1] and apply a (n, k) RS code to each chunk, expanding it into a n-block codeword. The first k blocks of the codeword are the original k blocks, followed by $d = n - k$ check blocks. We call a *constraint group* the blocks from the same codeword, *i.e.*, the original k blocks and their corresponding d check blocks. The number of constraint groups in the encoded file $\tilde{\mathtt{F}}$ is the same as the number of chunks in the original file $\mathtt{F}$: $\frac{f}{k}$.

We now describe several encoding schemes that lead to re-

[1]We assume w.l.o.g. that f (the number of blocks in $\mathtt{F}$) is a multiple of k (the size of a chunk). This can always be achieved by padding $\mathtt{F}$ if necessary.

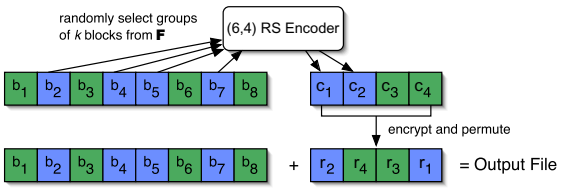

Figure 1: Computation of a $(6, 4)$ RS code with πR.

mote data checking schemes with different properties and performance characteristics. The main difference between these encoding schemes comes from the design choices of how to permute/encrypt the blocks in each constraint group.

Let $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a symmetric-key encryption scheme and π, ψ, ω be pseudo-random permutations (PRPs) defined as:
$-\ \pi : \{0,1\}^{\kappa} \times \{0,1\}^{\log_2(fn/k)} \rightarrow \{0,1\}^{\log_2(fn/k)}$
$-\ \psi : \{0,1\}^{\kappa} \times \{0,1\}^{\log_2(f)} \rightarrow \{0,1\}^{\log_2(f)}$
$-\ \omega : \{0,1\}^{\kappa} \times \{0,1\}^{\log_2(fd/k)} \rightarrow \{0,1\}^{\log_2(fd/k)}$
We use the keys w, z, v, u for the encryption scheme, PRP π, PRP ψ and PRP ω, respectively.

Simple-RS. A simple encoding would take a file $\mathtt{F} = \mathtt{b}_1, \ldots, \mathtt{b}_f$ and generate the encoded file $\tilde{\mathtt{F}} = \mathtt{b}_1, \ldots, \mathtt{b}_f, \mathtt{c}_1, \ldots, \mathtt{c}_{\frac{f}{k}d}$, in which blocks $\mathtt{b}_{ik+1}, \ldots, \mathtt{b}_{(i+1)k}$ are constrained by check blocks $\mathtt{c}_{id+1}, \ldots, \mathtt{c}_{(i+1)d}$, for $0 \leq i \leq \frac{f}{k} - 1$. The blocks of the input file are separated from the check blocks, rather than interleaved, in order to meet the contiguity requirement.

However, with fixed values of k and d, an attacker can *effectively delete* data by deleting a *fixed* number of blocks: Deleting any d blocks of $\tilde{\mathtt{F}}$ drawn from the same constraint group will result in a loss of data from the original file $\mathtt{F}$. Remote data checking schemes based on spot checking can only detect corruption of a δ-fraction of $\tilde{\mathtt{F}}$ and will not detect corruption of d blocks for fixed values of d (*i.e.*, independent of f). Thus, this encoding does not meet the requirement for robust data possession guarantee.

Permute-All (πA). The problem with Simple-RS is that an adversary can distinguish which blocks belong to the same constraint group. The constraints among blocks can be concealed by randomly permuting the blocks of the encoded file. Then encryption is applied to all blocks so that an attacker cannot uncover the constraints among the permuted blocks.

We first generate, like in Simple-RS, the file $\hat{\mathtt{F}} = \mathtt{b}_1, \ldots, \mathtt{b}_f, \mathtt{c}_1, \ldots, \mathtt{c}_{\frac{f}{k}d}$. We then use π and E to randomly permute and then encrypt all the blocks of $\hat{\mathtt{F}}$, obtaining the encoded file $\tilde{\mathtt{F}}$, where $\tilde{\mathtt{F}}[i] = \mathsf{E}_w(\hat{\mathtt{F}}[\pi_z(i)])$, for $1 \leq i \leq fn/k$.

This strategy (also used by Juels and Kaliski in [6]) leads to a robust data possession guarantee (as shown by our analysis in Section 6). However, the scheme has several drawbacks: The resource-intensive nature of permuting the entire encoded file can be rather slow (as acknowledged in [6]); also, the scheme does not meet the sequentiality requirement.

Permute-Redundancy (πR). We can overcome the drawbacks of the πA scheme by observing that it is sufficient to *only permute the check blocks*. We encode the input file $\mathtt{F} = \mathtt{b}_1, \ldots, \mathtt{b}_f$ as follows:

1. Use ψ to randomly permute the blocks of $\mathtt{F}$ to obtain the file $\mathtt{P} = \mathtt{p}_1, \ldots, \mathtt{p}_f$, where $\mathtt{p}_i = \mathtt{b}_{\psi_v(i)}$, $1 \leq i \leq f$. (As explained below, this step is not explicitly required.)
2. Compute check blocks $\mathtt{C} = \mathtt{c}_1, \ldots, \mathtt{c}_{\frac{f}{k}d}$ so that blocks $\mathtt{p}_{ik+1}, \ldots, \mathtt{p}_{(i+1)k}$ are constrained by $\mathtt{c}_{id+1}, \ldots, \mathtt{c}_{(i+1)d}$, for $0 \leq i \leq \frac{f}{k} - 1$.
3. Permute and then encrypt the check blocks to obtain $\mathtt{R} = \mathtt{r}_1, \ldots, \mathtt{r}_{\frac{f}{k}d}$, where $\mathtt{r}_i = \mathsf{E}_w(\mathtt{c}_{\omega_u(i)})$, $1 \leq i \leq \frac{f}{k}d$.
4. Output redundancy encoded file $\tilde{\mathtt{F}} = \mathtt{F}||\mathtt{R}$.

Figure 1 shows the computation of πR and the resulting output file layout. The original file data is output sequentially and unencrypted, followed by permuted and encrypted redundancy. We emphasize the permutation in step 1 is included for ease of exposition and the scheme does not require the blocks of the file $\mathtt{F}$ to be physically permuted. Instead, the check blocks in step 2 are computed directly as a function of the blocks with the corresponding permuted index.

By computing RS codes over the permuted input file, rather than the original input file, an attacker does not know the relationship among blocks of the input file. By permuting the check blocks, the attacker does not know the relationship among the blocks in the redundant portion $\mathtt{R}$ of the output file. By encrypting the check blocks, an attacker cannot find the combinations of input blocks that correspond to output blocks. We note that in the challenge phase, the block dependencies (*i.e.*, constraint groups) remain hidden because the client asks for proof of possession of randomly chosen blocks over the entire encoded file.

However, πR does reveal some information about the structure of the file. An attacker knows that the file is divided into two parts, the original data ($\mathtt{F}$) and the redundancy information ($\mathtt{R}$) and can delete data differentially among these two regions to some advantage. For example, an attacker guarantees damage to a file by deleting all blocks in $\mathtt{R}$ and one block in $\mathtt{F}$. No deterministic attack that deletes the same number of blocks exists for the πA scheme.

The πR scheme meets all the requirements put forth in Section 4: The use of a systematic code, which outputs the original blocks as part of the output, ensures the contiguity and the data independence requirements. RS codes are space optimal because they are Maximum Distance Separable. Also, RS codes with fixed parameters are computationally efficient and ensure that only a constant amount of I/O is required for accessing and repairing small portions of the file, thus meeting the regionalization requirement (although πR does not regionalize in the address space).

6. ANALYSIS

We turn to an analysis of the probability of a successful attack against πA and πR as a function of the RS encoding parameters and the RDC checking discipline. By comparing the results, we identify that an attacker gains no detectable advantage from the πR strategy when compared with πA.

Analysis of πA. We encode a file that contains f blocks with a (n, k) code that corrects for up to d deletions. This produces an encoded file of length $f \cdot \frac{n}{k}$, which has f/k different constraint groups. This file is encrypted and permuted using πA. An attacker deletes x blocks of the file at random and X_i is the number of blocks deleted from constraint group i.

In deleting blocks, an attacker is performing sampling without replacement, which is governed by the hypergeometric distribution. Let A_i be the event that constraint group c_i has more than d blocks deleted from it: $X_i > d$. The probability of A_i is merely the sum of the hypergeometric distributions evaluated for all numbers larger than d:

$$p(A_i) = \sum_{j=d+1}^{n} p(X_i = j) = \sum_{j=d+1}^{n} \frac{\binom{n}{j}\binom{f-n}{x-j}}{\binom{f}{x}}$$

The file is damaged when any of the constraint groups experiences more than d deletions. Thus, the quantity of interest is: $p(\bigcup_{i=1}^{f/k} A_i)$ which can be accurately evaluated through an inclusion/exclusion process:

$$p(\bigcup_{i=1}^{f/k} A_i) = \sum_{i=1}^{f/k} p(A_i) - \sum_{i=1}^{f/k} \sum_{j=i+1}^{f/k} p(A_i \cap A_j) + \sum_{i=1}^{f/k} \sum_{j=i+1}^{f/k} \sum_{l=j+1}^{f/k} p(A_i \cap A_j \cap A_l) - \ldots$$

Because all constraint groups are the same, the outer sums that make up each inclusion/exclusion term need not be computed explicitly. Rather, the probability of intersection can be computed once and multiplied by the number of combinations of groups. However, the computation of $p(A_i \cap A_j)$ involves summing over many terms.

$$p(A_i \cap A_j) = \sum_{u=d+1}^{n} \sum_{v=d+1}^{n} p(A_i = u) \cdot p(A_j = v).$$

Evaluating the i-th term in the inclusion-exclusion argument in this fashion performs $\Theta(n^i)$ work.

Analysis of πR. We change our formulation slightly in this case. The attacker splits her deletions into x_b deletions from the unencoded blocks (F) and x_r deletions from the encrypted and permuted redundancy blocks (R), which produce $X_{b,i}$ and $X_{r,i}$ deletions in constraint group i respectively.

In this formulation, we need to consider all possible combinations of deletions from each side of the encoded file.

$$p(A_i) = \sum_{j=d+1}^{n} \sum_{k=0}^{d} p(X_{b,i} = j - k) \cdot p(X_{r,i} = k)$$

Deletions on F and on R are independent events. The remainder of the formulation (inclusion/exclusion) remains the same. This problem is not substantially harder (a factor of d) than the single file version.

6.1 Monte-Carlo Results

We turn to Monte-Carlo simulation to determine the probability of an attacker's success. Evaluating the inclusion/exclusion series of the hypergeometric distribution is not computationally reasonable. Note that the alternating signs of the inclusion/exclusion terms means that evaluating fewer terms does not provide bounds on the expansion and inclusion/exclusion processes do not always converge quickly. Our Monte-Carlo simulation models an attacker that deletes blocks randomly, but may choose the distribution of those deletions over the two portions of the πR encoding. It then runs one million trials of the attacker, in order to analyze the probability of a successful attack. We implement the simulation in C++ using the Gnu simulation library (gsl). Our presentation of results uses specific parameters for encoding and data checking, but the results are general in that they hold across the wide-range of parameterizations with which we experimented.

We analyze the benefit an attacker realizes from πR when compared with πA. To do so, we identify the attacker's best strategy in πR and then compare the probability of successful attack using that strategy with an attack against πA.

Based on an understanding of the probability of a successful attack, we use the simulation to determine the encoding and data checking parameters that a system can use to achieve its data protection goals.

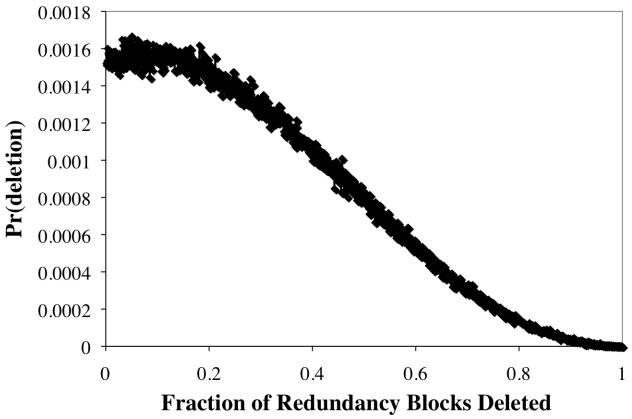

Figure 2: Identify the best attack strategy by varying where blocks are deleted.

Attacker's Best Strategy. An attacker that deletes x blocks can split those blocks between the original file data (F) and the redundancy information (R). Examining the probability of deletion as function of the attacker's choice reveals the best strategy, which is to *distribute the deletions between F and R in proportion to their size*. Figure 2 shows that probability of an attacker damaging a file as a function of this choice for a file of 100,000 blocks unencoded and 108,000 blocks encoded with a $(108, 100)$ RS code in which the attacker deletes 1080 blocks (1% of the data). The attacker maximizes the probability of a successful attack when deleting 5-10% of the data from R. The results are somewhat noisy. The redundancy represents 8% of the data and matches well with this number.

Restricted choice provides the intuition behind the correspondence of the data distribution and the attackers strategy. A successful attack requires $d+1$ blocks to be deleted from a single constraint group. Deleting a block in a constraint group reduces the number of blocks remaining in that constraint group, *restricting* the probability of finding another block from this constraint group. Restricting the probability happens more rapidly in R than in F, because there are fewer blocks in each constraint group. The attacker balances these probabilities by deleting data proportionately between R and F so that the probability of deletion match in each side of the file.

(Near) Equivalence of πA and πR. Figure 3 quantifies the difference in the robust data possession guarantee between the πR and πA encodings. This experiment uses similar configuration parameters: an unencoded file of 100,000 blocks encoded with $(100 + d, 100)$ RS with $d \in (1, 10)$ in which the attacker deletes 1% of the data. The probability of success for an attacker matches closely between the two encodings.

While πR gives some advantage to an attacker, it is minor and quantifiable. A system that uses πR will have to use more redundancy or check more blocks. At the same time, πR has the advantages of meeting several requirements that πA does not: Storing data contiguously and unencrypted.

Realizing the Robust Data Possession Guarantee. This paper's main result shows that the πR encoding realizes the robust possession guarantee. A successful attack against PDP occurs when the attacker damages the file and PDP does not detect this damage. Both need to occur. Thus, the probability of the attack failing is bounded from below by the maximum of the probability of detection and the probability that the file is not damaged.

We use the Monte-Carlo simulation to select the encoding pa-

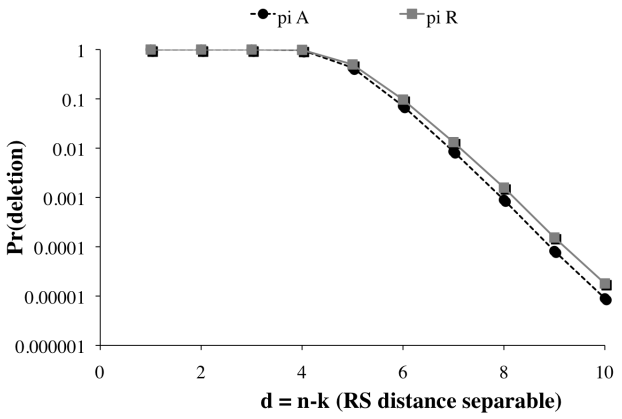

Figure 3: Probability of a succesful attack against the πA and πR encodings for different RS parameters.

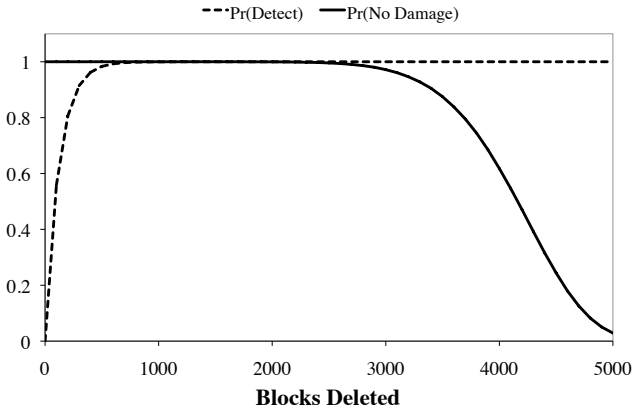

Figure 4: Probability of an attack failing because either no data was damaged or the audit process detected deleted blocks.

rameters and data checking protocol that achieve the data protection goals. To achieve fives 9s of attack resilience, we encode in $(140, 128)$ RS and the auditor checks 1137 blocks of the file. An attacker deletes between zero and all the blocks of an $140,000$ block file. Figure 4 shows that probability of a successful attack is less that 0.00001 in all cases (we truncate the results after the first relevant 5000 block deletions).

There is no deletion attack that allows an attacker to damage a file without getting caught. This analysis is constructive and allows system designers to select specific encodings and data checking disciplines. The designer picks confidence goals, five 9s in our case, and the simulation outputs parameterizations that achieve these goals. System designers may trade-off space-overhead for storage (more redundancy) versus the efficiency of challenges (more checking).

7. REFERENCES

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proc. of ACM CCS '07*.

[2] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In *Proc. of Securecomm 2008*.

[3] K. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. *ePrint Archive Report*, (2008/175), 2008.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese. MR-PDP: Multiple-replica provable data possession. In *Proc. of ICDCS '08*, 2008.

[5] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Proc. of NDSS '03*, 2003.

[6] A. Juels and B. S. Kaliski. PORs: Proofs of retrievability for large files. In *Proc. of ACM CCS '07*, 2007.

[7] R. Kotla, L. Alvisi, and M. Dahlin. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference*, 2007.

[8] M. N. Krohn, M. J. Freedman, and D. Mazières. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy*, 2004.

[9] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. of STOC '97*, 1997.

[10] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *International Workshop on Peer-to-Peer Systems*, 2003.

[11] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *Proc. of PODC '02*, pages 108–117, 2002.

[12] M. Mitzenmacher. Digital fountains: A survey and look forward. In *IEEE Information Theory Workshop*, 2004.

[13] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[14] T. S. J. Schwarz and E. L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *International Conference on Distributed Computing Systems*, 2006.

[15] H. Shacham and B. Waters. Compact proofs of retrievability. *ePrint Archive Report*, (2008/073), 2008.

[16] M. A. Shah, R. Swaminathan, and M. Baker. Privacy-preserving audit and extraction of digital contents. *ePrint Archive Report*, (2008/186).