Remote Data Checking Using Provable Data Possession[⋆]

Giuseppe Ateniese[*], Randal Burns[*], Reza Curtmola[†], Joseph Herring[*], Osama Khan[*], Lea Kissner[‡], Zachary Peterson[♯], and Dawn Song[§]

We introduce a model for *provable data possession* (PDP) that can be used for remote data checking: A client that has stored data at an untrusted server can verify that the server possesses the original data without retrieving it. The model generates probabilistic proofs of possession by sampling random sets of blocks from the server, which drastically reduces I/O costs. The client maintains a constant amount of metadata to verify the proof. The challenge/response protocol transmits a small, constant amount of data, which minimizes network communication. Thus, the PDP model for remote data checking is lightweight and supports large data sets in distributed storage systems. The model is also robust in that it incorporates mechanisms for mitigating arbitrary amounts of data corruption.

We present two provably-secure PDP schemes that are more efficient than previous solutions. In particular, the overhead at the server is low (or even constant), as opposed to linear in the size of the data. We then propose a generic transformation that adds robustness to any remote data checking scheme based on spot checking. Experiments using our implementation verify the practicality of PDP and reveal that the performance of PDP is bounded by disk I/O and not by cryptographic computation. Finally, we conduct an in-depth experimental evaluation to study the tradeoffs in performance, security, and space overheads when adding robustness to a remote data checking scheme.

Categories and Subject Descriptors: H.3.2 [**Information Storage and Retrieval**]: Information Storage.; E.3 [**Data Encryption**]:

General Terms: Security, Reliability, Performance

Additional Key Words and Phrases: Remote data checking, provable data possession, PDP, homomorphic verifiable tags, archival storage, cloud storage security, robust auditing, erasure coding

1 Introduction

Clients with limited storage resources or that desire to outsource the management of a data center distribute data to storage service providers (SSPs) that agree by contract to preserve the data and to keep it readily available for retrieval. Verifying the authenticity of data stored remotely on untrusted servers has emerged as a critical issue. It arises in peer-to-peer storage systems [Kubiatowicz et al. 2000], network file systems [Li et al. 2004; Kallahalla et al. 2003], long-term archives [Maniatis et al. 2005], web-service object stores [Yumerefendi and Chase 2007], and database systems [Maheshwari et al. 2000]. Such systems prevent storage servers from misrepresenting or modifying data by providing authenticity checks when accessing data.

However, archival storage requires guarantees about the authenticity of data on storage, namely that storage servers possess data. It is insufficient to detect data corruption when

[*] Dept. of Comp. Sci., Johns Hopkins University - {ateniese, randal, jrh, okhan}@cs.jhu.edu

[†] Dept. of Comp. Sci., New Jersey Institute of Technology - crix@njit.edu

[‡] Google, Inc. - chialea@gmail.com

[♯] Comp. Sci. Dept., Naval Postgraduate School - znpeters@nps.edu

[§] Comp. Sci. Division, University of California, Berkeley - dawnsong@cs.berkeley.edu

[⋆] Portions of this article were previously published in [Ateniese et al. 2007] and [Curtmola et al. 2008].

accessing the data, because it may be too late to recover lost or damaged data. Archival storage servers retain tremendous amounts of data, little of which are accessed. They also hold data for long periods of time during which there may be exposure to data loss from administration errors as the physical implementation of storage evolves, *e.g.*, backup and restore, data migration to new systems, and changing memberships in peer-to-peer systems.

In this scenario, it is desirable to audit that the SSP meets its contractual obligations. SSPs have many motivations to fail these obligations; *e.g.*, an SSP may try to hide data loss incidents in order to preserve its reputation or it may discard data that are rarely accessed so that it may resell the same storage. *Remote data checking* (RDC) allows an auditor to challenge a server to provide a *proof of data possession* in order to validate that the server possesses the data that were originally stored by a client. We say that an RDC scheme seeks to provide a *data possession guarantee*.

Archival network storage presents unique performance demands. Given that file data are large and are stored at remote sites, accessing an entire file is expensive in I/O costs to the storage server and in transmitting the file across a network. Reading an entire archive, even periodically, greatly limits the scalability of network stores. Furthermore, I/O incurred to establish data possession interferes with on-demand bandwidth to store and retrieve data. We conclude that clients need to be able to verify that a server has retained file data *without retrieving the data from the server* and *without having the server access the entire file*.

A scheme for auditing remote data should be both *lightweight* and *robust*. Lightweight means that it does not unduly burden the SSP; this includes both overhead (*i.e.*, computation and I/O) at the SSP and communication between the SSP and the auditor. This goal can be achieved by relying on *spot checking*, in which the auditor randomly samples small portions of the data and checks their integrity, thus minimizing the I/O at the SSP. Spot checking allows the client to detect if a fraction of the data stored at the server has been corrupted, but it cannot detect corruption of small parts of the data (*e.g.*, 1 byte). Robust means that the auditing scheme incorporates mechanisms for mitigating arbitrary amounts of data corruption. Protecting against large corruptions ensures the SSP has committed the contracted storage resources: Little space can be reclaimed undetectably, making it unattractive to delete data to save on storage costs or sell the same storage multiple times. Protecting against small corruptions protects the data itself, not just the storage resource. Many data have value well 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.

Previous solutions do not meet all these requirements for proving data possession. Some schemes [Golle et al. 2002] provide a weaker guarantee by enforcing *storage complexity*: The server has to store an amount of data at least as large as the client's data, but not necessarily the same exact data. Moreover, most previous techniques require the server to access the entire file, which is not feasible when dealing with large amounts of data, or require storage on the client linear with the size of the data, which does not conform with the notion of storage outsourcing. A notable exception is the work of Schwarz and Miller [Schwarz and Miller 2006], which meets most of the requirements for proving data possession, but provides a less formal security analysis.

We introduce a model for provable data possession (PDP) that allows remote data checking, *i.e.*, provides proof that a third party stores a file. The model is unique in that it is lightweight, *i.e.* by using spot checking it allows the server to access small portions of the file to generate the proof; all previous techniques must access the entire file. Within this model, we give the first provably-secure scheme for remote data checking. The client stores a small $O(1)$ amount of metadata to verify the server's proof. Also, the scheme uses $O(1)$ network bandwidth[1]. The challenge and the response are each slightly more than 1 Kilobit. We also present a more efficient version of this scheme that proves data possession using a single modular exponentiation at the server, even though it provides a weaker possession guarantee. Concurrently with this work, another model for proofs of retrievability (PoRs) [Juels and Kaliski 2007] was proposed to perform remote data checking.

Both our schemes use *homomorphic verifiable tags*. Because of the homomorphic property, tags computed for multiple file blocks can be combined into a single value. The client pre-computes tags for each block of a file and then stores the file and its tags with a server. At a later time, the client can verify that the server possesses the file by generating a random challenge against a randomly selected set of file blocks. The server retrieves the queried blocks and their corresponding tags, using them to generate a proof of possession. The client is thus convinced of data possession, without actually having to retrieve file blocks.

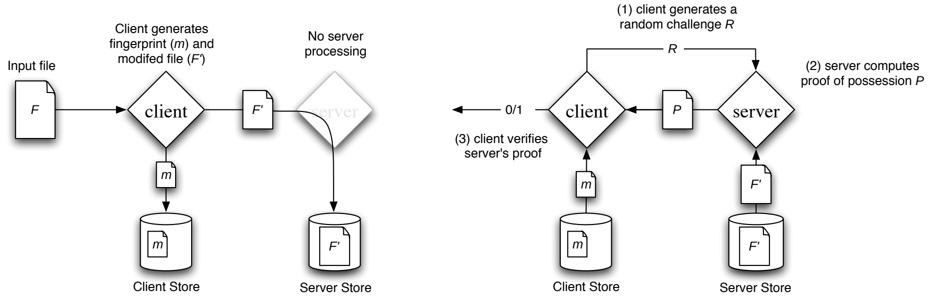
Our PDP schemes provide *data format independence*, which is a relevant feature in practical deployments (more details on this in the remarks of Section 3.3), and put no restriction on the number of times the client can challenge the server to prove data possession. Also, a variant of our main PDP scheme offers *public verifiability* (described in Section 3.3).

To enhance possession guarantees in our model, we define the notion of *robust auditing*, which integrates forward error-correcting codes (FECs) with remote data checking. Attacks that corrupt small amounts of data do no damage, because the corrupted data may be recovered by the FEC. Attacks that do unrecoverable amounts of damage are easily detected, because they must corrupt many blocks of data to overcome the redundancy. We identify the requirements that guide the design, implementation, and parameterization of robust auditing schemes. Important issues include the choice of an FEC code, the organization or layout of the output data, and the selection of encoding parameters. The forces on this design are subtle and complex. The integration must maintain the security of remote data checking regardless of the adversary's attack strategy and regardless of the access pattern to the original data. The integration must also maximize the encoding rate of data and the I/O performance of the file on remote storage, and minimize storage overhead for redundancy and the I/O complexity of auditing remote data. Identifying specific encodings that preserve security and performance is challenging. Indeed, several of the proposed use of FEC codes [Juels and Kaliski 2007; Shacham and Waters 2008] is not optimal and may result in poor I/O and encoding performance.

We propose a generic transformation that meets the specified requirements and that encodes a file using FECs in order to add robustness to any RDC scheme based on spot checking. We provide a detailed analysis of the reliability of the resulting encoding that measures the probability of a successful attack against a robust auditing scheme.

We implement one of our PDP schemes (E-PDP) and show experimentally that probabilistic possession guarantees make it practical to verify possession of large data sets. With sampling, E-PDP verifies a 64MB file in about 0.4 seconds as compared to 1.8 seconds without sampling. Further, I/O bounds the performance of E-PDP; it generates proofs as quickly as the disk produces data and it is 185 times faster than the previous secure protocol on 768 KB files. Finally, we provide an in-depth evaluation of robust auditing that

[1] Storage and network overhead are constant in the size of the file, but depend on the chosen security parameter.


Input file
F
Client generates fingerprint (m) and modifed file (F')
client
m
F'
No server processing
server
m
F'
Client Store
Server Store
(1) client generates a random challenge R
R
(2) server computes proof of possession P
0/1
client
P
server
(3) client verifies server's proof
m
F'
m
F'
Client Store
Server Store

(a) Pre-process and store

(b) Verify file possession by server

Fig. 1: Protocol for provable data possession.

studies tradeoffs in performance, security, and space overhead as a function of encoding parameters and the audit strategy. For reasonable parameters, robustness improves the possession guarantee by 8 orders of magnitude (*i.e.*, from 10^{-2} to 10^{-10}). When used together, the analysis and experimental evaluation are constructive: They allow system designers to identify the specific configurations that realize performance and security goals.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we describe a framework for provable data possession, emphasizing the features and parameters that are relevant for PDP. In Section 3, we introduce homomorphic verifiable tags, followed by definitions for PDP schemes and then we give our constructions (S-PDP and E-PDP). Section 4 presents generic mechanisms to incorporate robustness in remote data checking schemes. We support our theoretical claims with experiments that show the practicality of our schemes in Section 5. We review related work in Section 6 and conclude in Section 7.

2 Provable Data Possession (PDP)

We describe a framework for provable data possession. A PDP protocol (Fig. 1) checks that an outsourced storage site retains a file, which consists of f blocks. The client C (data owner) pre-processes the file, generating a small piece of metadata that is stored locally, transmits the file to the server S, and may delete its local copy. The server stores the file and responds to challenges issued by the client. Storage at the server is $\Omega(f)$ and storage at the client is $\mathrm{O}(1)$, conforming to our notion of an outsourced storage relationship.

As part of pre-processing, the client may alter the file to be stored at the server. The client may encrypt, encode or expand the file, or may include additional metadata to be stored at the server. Before deleting its local copy of the file, the client may execute a data possession challenge to make sure the server has successfully stored the file.

At a later time, an auditor issues a challenge to the server to establish that the server has retained the file. The auditor requests that the server compute a function of the stored file, which it sends back to the client. Using its local metadata, the auditor verifies the response.

We will assume for ease of exposition that the client (data owner) is the same entity as the auditor. However, our solutions can be easily extended to a setting where these two may be separate entities (*e.g.*, if business requirements require separation, or if data privacy is a concern and the auditor should not have access to the plain data ([Shah et al. 2008]).

Adversarial model. Although the server S must answer challenges from the client C (failure to do so represents a data loss), it is not trusted to store the file and may try to convince the client it possesses (*i.e.*, stores) the file even if the file is totally or partially corrupted. 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). Later, in Section 4 we show how to incorporate robustness in order to mitigate arbitrary amounts of data corruption.

Requirements and Parameters. The important performance parameters of PDP include:

- *Computation complexity:* The computational cost to pre-process a file (at C), to generate a proof of possession (at S) and to verify such a proof (at C);
- *Block access complexity:* The number of file blocks accessed to generate a proof of possession (at S);
- *Communication complexity:* The amount of data transferred (between C and S).

For a scalable solution, the amount of computation and block accesses at the server should be minimized, because the server may be involved in concurrent interactions with many clients. *We stress that in order to minimize bandwidth, an efficient* PDP*scheme cannot consist of retrieving entire file blocks.* While relevant, the computation complexity at the client is of less importance, even though our schemes minimize that as well.

To meet these performance goals, our PDP schemes sample the server's storage, accessing a random subset of blocks. In doing so, the PDP schemes provide a probabilistic guarantee of possession; a deterministic guarantee cannot be provided without accessing all blocks. In fact, as a special case of our PDP scheme, the client may ask proof for all the file blocks, making the data possession guarantee deterministic. Sampling proves data possession with high probability based on accessing few blocks in the file, which radically alters the performance of proving data possession. Interestingly, when the server corrupts a fraction of the file, the client can detect server misbehavior with high probability by asking proof for a constant amount of blocks, independently of the total number of file blocks. As an example, for a file with $f = 10,000$ blocks, if S has corrupted 1% of the blocks, then C can detect server misbehavior with probability greater than 99% by asking proof of possession for only 460 randomly selected blocks. For more details, see Section 4.2.1.

3 Provable Data Possession Schemes

3.1 Preliminaries

The client C wants to store on the server S a file $\mathtt{F}$ which is a finite ordered collection of f blocks: $\mathtt{F} = (\mathtt{b}_1, \ldots, \mathtt{b}_f)$. We denote the output x of an algorithm $\mathcal{A}$ by $\mathcal{A} \to x$. We denote by $|x|$ the length of x (in bits).

Homomorphic Verifiable Tags (HVTs). We introduce the concept of a homomorphic verifiable tag that will be used as a building block for our PDP schemes.

Given a message $\mathtt{b}$ (corresponding to a file block), we denote by $\mathtt{T}_{\mathtt{b}}$ its homomorphic verifiable tag. The tags will be stored on the server together with the file $\mathtt{F}$. Homomorphic verifiable tags act as verification metadata for the file blocks and, besides being unforgeable, they also have the following properties:

– *Blockless verification*: Using HVTs the server can construct a proof that allows the client to verify if the server possesses certain file blocks, even when the client does not have access to the actual file blocks.

– *Homomorphic tags*: Given two values $\mathtt{T}_{\mathtt{b}_i}$ and $\mathtt{T}_{\mathtt{b}_j}$, anyone can combine them into a value $\mathtt{T}_{\mathtt{b}_i+\mathtt{b}_j}$ corresponding to the sum of the messages $\mathtt{b}_i + \mathtt{b}_j$.

In our construction, an HVT is a pair of values $(\mathtt{T}_{i,\mathtt{b}}, \mathtt{W}_i)$, where $\mathtt{W}_i$ is a random value obtained from an index i and $\mathtt{T}_{i,\mathtt{b}}$ is stored at the server. The index i can be seen as a *one-time* index because it is never reused for computing tags (a simple way to ensure that every tag uses a different index i is to use a global counter for i). The random value $\mathtt{W}_i$ is generated by concatenating the index i to a secret value, which ensures that $\mathtt{W}_i$ is different and unpredictable each time a tag is computed. HVTs and their corresponding proofs have a fixed constant size and are (much) smaller than the actual file blocks.

We emphasize that techniques based on *aggregate signatures* [Boneh et al. 2003], *multi-signatures* [Micali et al. 2001; Okamoto 1988], batch RSA [Fiat 1990], batch verification of RSA [Harn 1998; Bellare et al. 1998], condensed RSA [Mykletun et al. 2004], etc. would all fail to provide *blockless verification*, which is needed by our PDP scheme. Indeed, the client should have the ability to verify the tags on *specific* file blocks even though he *does not* possess any of those blocks.

3.2 Definitions

We start with the definition of a provable data possession scheme and protocol, followed by the security definition that captures the data possession property.

DEFINITION 3.1. (PROVABLE DATA POSSESSION SCHEME (PDP)) *A PDP scheme is a collection of four polynomial-time algorithms* (KeyGen, $\mathsf{TagBlock}$, $\mathsf{GenProof}$, $\mathsf{CheckProof}$) *such that:*

$\mathsf{KeyGen}(1^k) \rightarrow (\mathrm{pk}, \mathrm{sk})$ *is a probabilistic key generation algorithm that is run by the client to setup the scheme. It takes a security parameter k as input, and returns a pair of matching public and secret keys* $(\mathrm{pk}, \mathrm{sk})$.

$\mathsf{TagBlock}(\mathrm{pk}, \mathrm{sk}, b) \rightarrow T_b$ *is a (possibly probabilistic) algorithm run by the client to generate the verification metadata. It takes as inputs a public key* pk*, a secret key* sk *and a file block* b*, and returns the verification metadata* T_b.

$\mathsf{GenProof}(\mathrm{pk}, F, \mathsf{chal}, \Sigma) \rightarrow \mathcal{V}$ *is run by the server in order to generate a proof of possession. It takes as inputs a public key* pk*, an ordered collection* F *of blocks, a challenge* chal *and an ordered collection* Σ *which is the verification metadata corresponding to the blocks in* F*. It returns a proof of possession* $\mathcal{V}$ *for the blocks in* F *that are determined by the challenge* chal.

$\mathsf{CheckProof}(\mathrm{pk}, \mathrm{sk}, \mathsf{chal}, \mathcal{V}) \rightarrow \{\text{“}success\text{”}, \text{“}failure\text{”}\}$ *is run by the client in order to validate a proof of possession. It takes as inputs a public key* pk*, a secret key* sk*, a challenge* chal *and a proof of possession* $\mathcal{V}$*. It returns whether* $\mathcal{V}$ *is a correct proof of possession for the blocks determined by* chal.

We construct a PDP protocol from a PDP scheme in two phases, Setup and $\mathsf{Challenge}$:

Setup: The client C is in possession of the file $\mathtt{F}$ and runs $\mathsf{KeyGen}(1^k) \rightarrow (\mathrm{pk}, \mathrm{sk})$, followed by $\mathsf{TagBlock}(\mathrm{pk}, \mathrm{sk}, \mathtt{b}_i) \rightarrow \mathtt{T}_{\mathtt{b}_i}$, for all $1 \leq i \leq f$. C stores the pair $(\mathrm{sk}, \mathrm{pk})$. C then sends $\mathrm{pk}, \mathtt{F}$ and $\Sigma = (\mathtt{T}_{\mathtt{b}_1}, \ldots, \mathtt{T}_{\mathtt{b}_f})$ to S for storage and may delete $\mathtt{F}$ and Σ.

$\mathsf{Challenge}$: C generates a challenge chal that, among other things, indicates the specific blocks for which C wants a proof of possession. C then sends chal to S. S runs $\mathsf{GenProof}(\mathrm{pk}, \mathtt{F}, \mathsf{chal}, \Sigma) \rightarrow \mathcal{V}$ and sends to C the proof of possession $\mathcal{V}$. Finally, C

can check the validity of the proof $\mathcal{V}$ by running $\mathsf{CheckProof}(\mathrm{pk}, \mathrm{sk}, \mathsf{chal}, \mathcal{V})$.

In the Setup phase, C computes tags for each file block and stores them together with the file at S. In the $\mathsf{Challenge}$ phase, C requests proof of possession for a subset of the blocks in $\mathtt{F}$. This phase can be executed an unlimited number of times in order to ascertain whether S still possesses the selected blocks. We note that $\mathsf{GenProof}$ and $\mathsf{CheckProof}$ may receive different input values for chal, as these algorithms are run by S and C, respectively.

We state the security for a PDP protocol using a game that captures the data possession property. Intuitively, the Data Possession Game captures that an adversary cannot successfully construct a valid proof without possessing all the blocks corresponding to a given challenge, unless it guesses all the missing blocks.

Data Possession Game:

- **Setup**: The challenger runs $\mathsf{KeyGen}(1^k) \rightarrow (\mathrm{pk}, \mathrm{sk})$, sends pk to the adversary and keeps sk secret.
- **Query**: The adversary makes tagging queries adaptively: It selects a block $\mathtt{b}_1$ and sends it to the challenger. The challenger computes the verification metadata $\mathsf{TagBlock}(\mathrm{pk}, \mathrm{sk}, \mathtt{b}_1) \rightarrow \mathtt{T}_{\mathtt{b}_1}$ and sends it back to the adversary. The adversary continues to query the challenger for the verification metadata $\mathtt{T}_{\mathtt{b}_2}, \ldots, \mathtt{T}_{\mathtt{b}_f}$ on the blocks of its choice $\mathtt{b}_2, \ldots, \mathtt{b}_f$. As a general rule, the challenger generates $\mathtt{T}_{\mathtt{b}_j}$ for some $1 \leq j \leq f$, by computing $\mathsf{TagBlock}(\mathrm{pk}, \mathrm{sk}, \mathtt{b}_j) \rightarrow \mathtt{T}_{\mathtt{b}_j}$. The adversary then stores all the blocks as an ordered collection $\mathtt{F} = (\mathtt{b}_1, \ldots, \mathtt{b}_f)$, together with the corresponding verification metadata $\mathtt{T}_{\mathtt{b}_1}, \ldots, \mathtt{T}_{\mathtt{b}_f}$.
- **Challenge**: The challenger generates a challenge chal and requests the adversary to provide a proof of possession for the blocks $\mathtt{b}_{i_1}, \ldots, \mathtt{b}_{i_c}$ determined by chal, where $1 \leq i_j \leq f, 1 \leq j \leq c, 1 \leq c \leq f$.
- **Forge**: The adversary computes a proof of possession $\mathcal{V}$ for the blocks indicated by chal and returns $\mathcal{V}$.

If $\mathsf{CheckProof}(\mathrm{pk}, \mathrm{sk}, \mathsf{chal}, \mathcal{V}) =$ "*success*", then the adversary has won the Data Possession Game.

DEFINITION 3.2. *A PDP protocol* ($\mathsf{Setup}, \mathsf{Challenge}$) *built on a PDP scheme* ($\mathsf{KeyGen}, \mathsf{TagBlock}, \mathsf{GenProof}, \mathsf{CheckProof}$) *guarantees data possession if for any (probabilistic polynomial-time) adversary $\mathcal{A}$ the probability that $\mathcal{A}$ wins the Data Possession Game on a set of file blocks is negligibly close to the probability that the challenger can extract those file blocks by means of a (probabilistic polynomial-time) knowledge extractor $\mathcal{E}$.*

In our security definition, the notion of a knowledge extractor is similar with the standard one introduced in the context of proofs of knowledge [Bellare and Goldreich 1992]. If the adversary is able to win the Data Possession Game, then $\mathcal{E}$ can execute $\mathsf{GenProof}$ repeatedly until it extracts the selected blocks. On the other hand, if $\mathcal{E}$ cannot extract the blocks, then the adversary cannot win the game with more than negligible probability. We refer the reader to [Juels and Kaliski 2007] for a more generic and extraction-based security definition for proofs of retrievability (PoR) and to [Naor and Rothblum 2005] for the security definition of sub-linear authenticators.

3.3 Efficient and Secure PDP Schemes

In this section we present our PDP constructions: The first (S-PDP) provides a data possession guarantee, while the second (E-PDP) achieves better efficiency at the cost of weakening the data possession guarantee.

We start by introducing some additional notation used by the constructions. Let $p = 2p' + 1$ and $q = 2q' + 1$ be safe primes and let $N = pq$ be an RSA modulus. Let g be a generator of QR_N, the unique cyclic subgroup of $\mathbb{Z}_N^*$ of order $p'q'$ (*i.e.*, QR_N is the set of quadratic residues modulo N). We can obtain g as $g = a^2$, where $a \overset{R}{\leftarrow} \mathbb{Z}_N^*$ such that $\gcd(a \pm 1, N) = 1$. All exponentiations are performed modulo N, and for simplicity we sometimes omit writing it explicitly. Let $h : \{0,1\}^* \rightarrow QR_N$ be a secure deterministic hash-and-encode function[2] that maps strings uniformly to QR_N.

The schemes are based on the KEA1 assumption which was introduced by Damgard in 1991 [Damgard 1992] and subsequently used by several others, most notably in [Hada and Tanaka 1998; Bellare and Palacio 2004a; 2004b; Krawczyk 2005; Dent 2006a]. In particular, Bellare and Palacio [Bellare and Palacio 2004a] provided a formulation of KEA1, that we follow and adapt to work in the RSA ring:

KEA1-r (**K**nowledge of **E**xponent **A**ssumption): For any adversary **A** that takes input (N, g, g^s) and returns group elements (C, Y) such that $Y = C^s$, there exists an "extractor" $\bar{\mathbf{A}}$ which, given the same inputs as **A**, returns x such that $C = g^x$.

Recently, KEA1 has been shown to hold in generic groups (*i.e.*, it is secure in the generic group model) by A. Dent [Dent 2006b] and independently by Abe and Fehr [Abe and Fehr 2007]. In private communication, Yamamoto has informed us that Yamamoto, Fujisaki, and Abe introduced the KEA1 assumption in the RSA setting in [Yamamoto et al. 2005][3].

Later in this section, we also show an alternative strategy which does not rely on the KEA1-r assumption, at the cost of increased network communication.

Overview of S-PDP. We first give an overview of our provable data possession scheme that supports sampling. In the Setup phase, the client computes a homomorphic verifiable tag $\mathtt{T}_{i,\mathtt{b}_i}$ for each file block $\mathtt{b}_i$. To maintain constant storage, the client generates the random values $\mathtt{W}_i$ (used to obtain $\mathtt{T}_{i,\mathtt{b}_i}$) by concatenating the block index i to a secret value v; thus, TagBlock has an extra parameter, i. Each value $\mathtt{T}_{i,\mathtt{b}_i}$ is a function of the index i of the block $\mathtt{b}_i$. This binds the tag on a block to that specific block and prevents using the tag to obtain a proof for a different block. The values $\mathtt{T}_{i,\mathtt{b}_i}$ and the file $\mathtt{F}$ are stored at the server. The extra storage at the server is the overhead for allowing thin clients that only store a small, constant amount of data, regardless of the file size.

In the Challenge phase, the client asks the server for proof of possession of c file blocks whose indices are randomly chosen using a pseudo-random permutation keyed with a fresh randomly-chosen key for each challenge. This *spot checking* technique prevents the server from anticipating which blocks will be queried in each challenge. C also generates a fresh (random) challenge $g_s = g^s$ to ensure that S does not reuse any values from a previous Challenge phase. The server returns a proof of possession that consists of two values: $\mathtt{T}$ and

[2] Here, h is modeled as a random oracle. In practice, h is computed by squaring the output of the full-domain hash function for the provably secure FDH signature scheme [Bellare and Rogaway 1993; 1996] based on RSA. We refer the reader to [Bellare and Rogaway 1993] for ways to construct an FDH function out of regular hash functions, such as SHA-1. Alternatively, h can be the deterministic encoding function used in RSA-PSS [Bellare and Rogaway 1998].

[3] Their assumption, named NKEA1, is the same as ours, KEA1-r, except that we restrict g to be a generator of the group of quadratic residues of order $p'q'$. As noted in their paper [Yamamoto et al. 2005], if the order is not known then the extractor returns an x such that $C = \pm g^x$.

```
KeyGen($1^k$): Generate $\mathrm{pk} = (N, g)$ and $\mathrm{sk} = (e, d, v)$, such that $ed \equiv 1 (\bmod\ p'q')$, $e$ is a large secret prime such that $e > \lambda$ and $d > \lambda$ , $g$ is a generator of $QR_N$ and $v \overset{R}{\leftarrow} \{0,1\}^{\kappa}$. Output $(\mathrm{pk}, \mathrm{sk})$.
TagBlock($\mathrm{pk}, \mathrm{sk}, \mathtt{b}, i$):
1.  Let $(N, g) = \mathrm{pk}$ and $(d, v) = \mathrm{sk}$. Generate $\mathtt{W}_i = v||i$.
    Compute $\mathtt{T}_{i,\mathtt{b}} = \left(h(\mathtt{W}_i) \cdot g^{\mathtt{b}}\right)^d \bmod N$.
2.  Output $\mathtt{T}_{i,\mathtt{b}}$.
GenProof($\mathrm{pk}, \mathtt{F} = (\mathtt{b}_1, \ldots, \mathtt{b}_f), \mathsf{chal}, \Sigma = (\mathtt{T}_{1,\mathtt{b}_1}, \ldots, \mathtt{T}_{f,\mathtt{b}_f})$):
1.  Let $(N, g) = \mathrm{pk}$ and $(c, k_1, k_2, g_s) = \mathsf{chal}$.
    For $1 \le j \le c$:
    —compute the indices of the blocks for which the proof is generated: $i_j = \pi_{k_1}(j)$
    —compute coefficients: $a_j = \theta_{k_2}(j)$.
2.  Compute $\mathtt{T} = \mathtt{T}^{a_1}_{i_1,\mathtt{b}_{i_1}} \cdot \ldots \cdot \mathtt{T}^{a_c}_{i_c,\mathtt{b}_{i_c}} = \left(h(\mathtt{W}_{i_1})^{a_1} \cdot \ldots \cdot h(\mathtt{W}_{i_c})^{a_c} \cdot g^{a_1\mathtt{b}_{i_1}+\ldots+a_c\mathtt{b}_{i_c}}\right)^d \bmod N$. // (note that $\mathtt{T}_{i_j,\mathtt{b}_{i_j}}$ is the $i_j$-th value in $\Sigma$).
3.  Compute $\rho = H(g_s^{a_1\mathtt{b}_{i_1}+\ldots+a_c\mathtt{b}_{i_c}} \bmod N)$.
4.  Output $\mathcal{V} = (\mathtt{T}, \rho)$.
CheckProof($\mathrm{pk}, \mathrm{sk}, \mathsf{chal}, \mathcal{V}$):
1.  Let $(N, g) = \mathrm{pk}$, $(e, v) = \mathrm{sk}$, $(c, k_1, k_2, s) = \mathsf{chal}$ and $(\mathtt{T}, \rho) = \mathcal{V}$.
2.  Let $\tau = \mathtt{T}^e$. For $1 \le j \le c$:
    —compute $i_j = \pi_{k_1}(j),\ \mathtt{W}_{i_j} = v||i_j,\ a_j = \theta_{k_2}(j)$, and $\tau = \frac{\tau}{h(\mathtt{W}_{i_j})^{a_j}} \bmod N$
    (As a result, one should get $\tau = g^{a_1\mathtt{b}_{i_1}+\ldots+a_c\mathtt{b}_{i_c}} \bmod N$)
3.  If $H(\tau^s \bmod N) = \rho$, then output "success". Otherwise output "failure".
```

```
We construct a PDP protocol from a PDP scheme in two phases, Setup and Challenge:
Setup: The client $C$ runs KeyGen($1^k$) $\rightarrow (\mathrm{pk}, \mathrm{sk})$, stores $(\mathrm{sk}, \mathrm{pk})$ and sets $(N, g) = \mathrm{pk}$, $(e, d, v) = \mathrm{sk}$. $C$ then runs TagBlock($\mathrm{pk}, (d, v), \mathtt{b}_i, i$) $\rightarrow \mathtt{T}_{i,\mathtt{b}_i}$ for all $1 \le i \le f$ and sends $\mathrm{pk}, \mathtt{F}$ and $\Sigma = (\mathtt{T}_{1,\mathtt{b}_1}, \ldots, \mathtt{T}_{f,\mathtt{b}_f})$ to $S$ for storage. $C$ may delete $\mathtt{F}$ and $\Sigma$ from local storage.
Challenge: $C$ requests proof of possession for $c$ distinct blocks of the file $\mathtt{F}$ (with $1 \le c \le f$):
1.  $C$ generates the challenge $\mathsf{chal} = (c, k_1, k_2, g_s)$, where $k_1 \overset{R}{\leftarrow} \{0,1\}^{\kappa}$, $k_2 \overset{R}{\leftarrow} \{0,1\}^{\kappa}$, $g_s = g^s \bmod N$ and $s \overset{R}{\leftarrow} \mathbb{Z}_N^*$. $C$ sends $\mathsf{chal}$ to $S$.
2.  $S$ runs GenProof($\mathrm{pk}, \mathtt{F}, \mathsf{chal}, \Sigma = (\mathtt{T}_{1,\mathtt{b}_1}, \ldots, \mathtt{T}_{f,\mathtt{b}_f})$) $\rightarrow \mathcal{V}$ and sends to $C$ the proof of possession $\mathcal{V}$.
3.  $C$ sets $\mathsf{chal} = (c, k_1, k_2, s)$ and checks the validity of the proof $\mathcal{V}$ by running CheckProof($\mathrm{pk}, (e, v), \mathsf{chal}, \mathcal{V}$).
```

Fig. 2: S-PDP: a PDP scheme that guarantees data possession.

ρ. $\mathtt{T}$ is obtained by combining into a single value the individual tags $\mathtt{T}_{i,\mathtt{b}_i}$ corresponding to the requested blocks. ρ is obtained by raising the challenge g_s to a function of the requested blocks. The value $\mathtt{T}$ contains information about the indices of the blocks requested by the client (in the form of the $h(\mathtt{W}_i)$ values). C can remove all the $h(\mathtt{W}_i)$ values from $\mathtt{T}$ because it has both the key for the pseudo-random permutation (used to determine the indices of the requested blocks) and the secret value v (used to generate the values $\mathtt{W}_i$). C finally verifies the validity of the server's proof by checking if a certain relation holds between $\mathtt{T}$ and ρ.

Details of S-PDP. Let κ, ℓ, λ be security parameters (λ is a positive integer) and let H be a cryptographic hash function. In addition, we make use of a pseudo-random function (PRF) θ and a pseudo-random permutation (PRP) π with the following parameters:

- $\theta : \{0,1\}^{\kappa} \times \{0,1\}^{\log_2(f)} \rightarrow \{0,1\}^{\ell}$;
- $\pi : \{0,1\}^{\kappa} \times \{0,1\}^{\log_2(f)} \rightarrow \{0,1\}^{\log_2(f)}$

We write $\theta_k(x)$ to denote θ keyed with key k applied on input x. Our S-PDP scheme is described in Fig. 2. The purpose of including the a_j coefficients in the values for ρ and $\mathtt{T}$ computed by S is to ensure that S possesses each one of the requested blocks. These coefficients are determined by a PRF keyed with a fresh randomly-chosen key for each challenge, which prevents S from storing combinations (*e.g.*, sums) of the original blocks instead of the original file blocks themselves. Also, we are able to maintain constant communication cost because tags on blocks can be combined into a single value.

In Appendix A [Ateniese et al. 2010], we prove:

THEOREM 3.3. *Under the RSA and KEA1-r assumptions,* S-PDP *guarantees data possession in the random oracle model.*

Regarding efficiency, we remark that each challenge requires a small, constant amount of communication between C and S (the challenge and the response are each slightly more than 1 Kilobit). In terms of server block access, the demands are c accesses for S, while in terms of computation we have c exponentiations for both C and S. When S corrupts a fraction of the file blocks, c is a relatively small, constant value (for more details, see Section 4.2.1). Since the size of the file is $O(f)$, accommodating the additional tags does not change (asymptotically) the storage requirements for the server.

In our analysis we assume w.l.o.g. that the indices for the blocks picked by the client in a challenge are different. One way to achieve this is to implement π using the techniques proposed by Black and Rogaway [Black and Rogaway 2002]. In a practical deployment, our protocol can tolerate collisions of these indices.

Notice that the server may store the client's file $\mathtt{F}$ however it sees fit, as long as it is able to recover the file when answering a challenge. For example, it is allowed to compress $\mathtt{F}$ (*e.g.*, if all the blocks of $\mathtt{F}$ are identical, then only storage slightly larger than one full block may be needed). Alternatively, w.l.o.g., one could assume that $\mathtt{F}$ has been optimally compressed by the client and the size of $\mathtt{F}$ is equal to $\mathtt{F}$'s information entropy function.

A concrete example of using S-PDP. For a concrete example of using S-PDP, we consider a 1024-bit modulus N and a 4 GB file $\mathtt{F}$ which has $f = 1,000,000$ 4KB blocks. During Setup, C stores the file and the tags at S. The tags require additional storage of 128 MB. The client stores about 3 Kbytes (N, e, d each have 1024 bits and v has 128 bits). During the Challenge phase, C and S use AES for π (used to select the random block indices i), HMAC for θ (used to determine the random coefficients a) and SHA1 for H.[4]

In a challenge, C sends to S four values which total 168 bytes (c has 4 bytes, k_1 has 16 bytes, k_2 has 20 bytes, g_s has 1024 bits). Assuming that S corrupts at least 1% of $\mathtt{F}$, then C can detect server misbehavior with probability over 99% by asking proof for $c = 460$

[4] Clearly, these are simplifications. As such, instantiating correctly a random oracle requires more careful considerations. One strategy is to use SHA-384 (the truncated version of SHA-512 with strengthened Merkle-Damgard transform) and apply it multiple times over the input and an index and then concatenate the resulting blocks.

randomly selected blocks (see Section 4.2.1 for details on how to derive this number). The server's response contains two values which total 148 bytes ($\mathtt{T}$ has 1024 bits, ρ has 20 bytes). We emphasize that the server's response to a challenge consists of a small, constant value; in particular, the server does not send back to the client any of the file blocks.

A more efficient scheme, with weaker guarantees (E-PDP). Our S-PDP scheme provides the guarantee that S possesses each one of the c blocks for which C requested proof of possession in a challenge. We now describe a more efficient variant of S-PDP, which we call E-PDP, that achieves better performance at the cost of offering weaker guarantees. E-PDP differs from S-PDP only in that all the coefficients a_j are equal to 1:

- In GenProof (steps 2 and 3) the server computes $\mathtt{T} = \mathtt{T}_{i_1,\mathtt{b}_1} \cdot \ldots \cdot \mathtt{T}_{i_c,\mathtt{b}_c}$ and $\rho = H(g_s^{\mathtt{b}_{i_1}+\ldots+\mathtt{b}_{i_c}} \bmod N)$.
- In CheckProof (step 2) the client computes $\tau = \dfrac{\mathtt{T}^e}{h(\mathtt{W}_{i_1}) \cdot \ldots \cdot h(\mathtt{W}_{i_c})} \bmod N$.

The E-PDP scheme reduces the computation on both the server and the client to one exponentiation (see Server Computation details in Section 5.1, the server computes ρ as one exponentiation to a value whose size in bits is slightly larger than $|\mathtt{b}_i|$).

We emphasize that E-PDP only guarantees possession of the sum of the blocks $\mathtt{b}_{i_1} + \ldots + \mathtt{b}_{i_c}$ and not necessarily possession of each one of the blocks for which the client requests proof of possession. In practice, this guarantee may be sufficient if the client adopts a probabilistic approach for multiple audits (as we assume in this paper) and chooses appropriate parameters for E-PDP to reduce the server's ability to cheat. For example, if the server pre-computes and stores the sums of all possible combinations of c out the f blocks ($\binom{f}{c}$ values), then the server could successfully pass any challenge with 100% probability. However, for reasonable parameters *e.g.* $f = 1000$ and $c = 101$, the server would need to pre-compute and store $\approx 10^{140}$ values and might be better off trying to factor N. It is advisable to set c as a prime number to prevent the server from storing combinations of sums of blocks, with each sum having a number of blocks that is a divisor of c. As pointed out, for a similar setting, in [Shacham and Waters 2008] (Appendix B), the server may also try to cheat with a lower success probability by storing no more than f blocks. However, when c is a prime (or any odd integer) their attack is not directly applicable to our scheme since it requires knowledge of the order of the group, which is unknown in our case (but it is known for their scheme). Nevertheless, this does not exclude that there are other techniques that could work against E-PDP since, we stress again, it only guarantees possession of the sum of the blocks. The client could further reduce the server's ability to cheat by choosing different values for c over the duration of multiple audits.

An alternative strategy (No KEA1-r Assumption). Instead of the value ρ, S could send the sum of the queried blocks as an integer (S does not know the order of QR_N) and let C verify the proof of possession using this value. Thus, we will not have to rely on the KEA1-r assumption. However, network communication will increase to slightly more than the size of a file block. In addition, notice that any solution based on proofs of knowledge of the sum would require even more bandwidth than just sending the sum itself. Again, this is because S does not know the order of QR_N and would have to work with large integers.

Public verifiability. The variant of the protocol described above that does not rely on KEA1-r can be further modified in order to offer the *public verifiability* property, which allows anyone, not just the data owner, to challenge the server for data possession. The advantages of having public verifiability are akin to those of public-key over symmetric-key cryptography. We describe next the P-PDP scheme, which offers public verifiability.

The size of the each file block and the coefficients a_i's (the outputs of the pseudo-random function θ) are now limited so that the sum of any c blocks will be less than $\lambda/2$ (recall that $e > \lambda$). The following changes should be applied to the S-PDP protocol in Fig. 2:

—the client (data owner) makes e public (along with N and g).

—the values $\mathtt{W}_i$ are generated as $\mathtt{W}_i = w_v(i)$, where w is a PRF such that $w : \{0,1\}^\kappa \times \{0,1\}^{\log_2(f)} \rightarrow \{0,1\}^\ell$ and $v \overset{R}{\leftarrow} \{0,1\}^\kappa$.

—after the initial Setup phase, the client (data owner) publishes v (the key for the PRF w).

—in GenProof and in CheckProof, the challenge chal does not contain the values g_s and s anymore (also, in the Challenge phase these values are not used anymore).

—in GenProof the server computes $M = a_1\mathrm{b}_{i_1} + \ldots + a_c\mathrm{b}_{i_c}$ instead of ρ and returns $\mathcal{V} = (\mathtt{T}, M)$.

—in CheckProof, step 3, the client now checks whether $g^M = \tau$ and, in addition, the client checks whether $|M| < \lambda/2$. (This test ensures that, in the proof in Appendix A [Ateniese et al. 2010], the condition $\gcd(e, 2(M^* - M)) = 1$ is verified.)

The communication cost of this variant is slightly larger than the size of a file block. We leave as an open problem devising publicly-verifiable PDP schemes where the size of challenges and responses is less than the size of a single file block. In Appendix A [Ateniese et al. 2010], we prove:

THEOREM 3.4. *Under the RSA assumption, the* P-PDP *scheme guarantees data possession in the random oracle model.*

Remark 1 (Data Format Independence). Our PDP schemes put no restriction on the format of the data, in particular files stored at the server do not have to be encrypted. This feature is very relevant since we anticipate that PDP schemes will have the biggest impact when used with large public repositories (*e.g.*, digital libraries, astronomy/medical/legal repositories, archives etc.).

Remark 2 (Prime-order Group Variant). Alternatively, our PDP schemes can potentially be modified to work within a group of a publicly-known prime order q. In this case, however, file blocks (seen as integers) must be less than q, otherwise the server could simply store them reduced modulo q. In a prime-order setting, network communication is further reduced (particularly in the elliptic curve setting), but pre-processing becomes more expensive given the small size of the file blocks. In contrast, the RSA setting allows us to work with arbitrarily large file blocks.

4 Robust Auditing of Outsourced Data

In this section, we propose a generic transformation to add robustness to any remote data checking scheme based on spot checking. Toward robustness, we integrate forward error-correcting codes (FECs) with remote data checking in order to strengthen the guarantee achieved by an audit. We start by identifying the requirements that guide the design, implementation, and parameterization of robust auditing schemes. We then define the notion of robust auditing and present the generic transformation that provides protection against arbitrary small amounts of data corruption. Finally, we analyze the reliability of our proposed data layout and then give practical guidelines for the selection of its parameters.

4.1 Requirements: Integrating FECs with RDC

We provide a brief review of forward error correction and then analyze the requirements and tradeoffs that drive the design of a scheme that integrates FECs with RDC.

Forward Error Correction. Forward error correcting (FEC) codes are classified by their parameters. An (n, k, d) FEC code takes k input symbols, outputs n output symbols and has distance $d = n - k$. We refer to the redundant symbols as *check* symbols.

Codes are further characterized by the number of erasures and errors from which they can recover. Our application is concerned with erasures only, because each block has integrity and authenticity metadata so that the block is either present and correct, or missing/deleted. Corrupted blocks are detected and treated as erasures. An important class of codes are minimum-distance separable (MDS) codes that can recover from d erasures. The Reed-Solomon (RS) codes that we use are MDS codes. We denote by (n, k) a RS code that can correct up to $n - k$ erasures.

For performance reasons, it is typical to choose small values for n, k, d so that the input file is divided into segments of k blocks each. Blocks from the same encoding segment are *constrained* with respect to each other and a segment is called a *constraint group*. Constraint groups are independent, *i.e.*, they can be computed in parallel.

Requirements for Integrating FECs with RDC. We identify the following requirements:

Reliability. Maximizing reliability is, perhaps, the most obvious requirement. Reliability here refers to minimizing the probability of an adversary successfully deleting some (or all) original data, regardless of the deletion strategy. The measure of reliability is independent of the spot checking scheme.

Sequentiality. We consider *systematic* codes that embed the unmodified input within the output, because they efficiently support sequential I/O to the original file. In contrast, schemes that permute the original file in the encoded output [Juels and Kaliski 2007] turn sequential reads to the original file into random reads in the encoded file. Throughput for sequential data exceeds that of random data by more than an order of magnitude in modern disk drives. The sequentiality requirement is critical even when retrieving the entire file, *e.g.*, in an archival object store: The data may be streamed from disk sequentially to the network; no buffering, data reorganization, or random I/O are needed.

Space overhead. In introducing redundancy, FECs expand the size of the input data, which increases storage costs and reduces the space available for other data. However, reliability is derived from redundancy and, thus, from space overhead. In order to offer the maximum amount of reliability with a minimum amount of space overhead, we use MDS (maximum distance separable) FEC codes.

Encoding throughput. The performance of erasure coding varies widely depending upon the specific code and its parameters. In general, performance decreases with increasing code width and redundancy. Also, MDS codes are less efficient than non-MDS codes. In practice, coding parameters can be chosen so that other aspects of RDC limit throughput.

Access pattern robustness. Accessing the original data should not compromise the robustness of the scheme. This is of concern for schemes that rely on using the encryption and permutation of blocks to hide constraints among the blocks of a file [Juels and Kaliski 2007]. In this case, sequential access to the original file would reveal the original order of blocks in the permuted file and a deleting attacker would gain knowledge which could be used to perform a targeted attack on constraint groups. Thus, for security reasons, in a scheme that lacks access pattern robustness, accessing a small sequential portion of the original file would require retrieving the entire encoded file.

Design Tradeoffs. The obvious application of FEC uses codes in which the redundancy covers the entire input file. So, for a file of size f, one would use a systematic, MDS (n, f, d) code. This encoding resists the deletion of any fraction d/n of the data blocks. Having the redundancy cover the entire input provides many benefits. It maximizes reliability and minimizes space overhead. The scheme also fulfills sequentiality and has access pattern robustness, because the file layout is known to the attacker.

While such codes are theoretically possible, they are impractical owing to poor encoding performance. Reed-Solomon codes and their variants are the only MDS codes for arbitrary values of (n, k, d) and their performance is unacceptable ($O(n \log n)$ to encode). Furthermore, one needs to construct a new code for each different file size used and the encoder needs space in $O(n^2)$.

Practical applications of RS-codes break input files into fix-sized segments and then encode each segment separately. For our application, acceptable data rates are only achieved for $n \leq 2^8$, *i.e.*, encoding rates that compare well with disk I/O transfer rates. However, this makes the file less reliable, because each segment may be attacked individually. An attacker may corrupt a fixed number of blocks $d + 1$ from any segment to damage the file. The reliability reduction can be overcome by securely concealing the constraint groups from the attacker. Our scheme pursues this approach.

Other approaches that use redundancy covering the entire file do not meet other requirements. Rateless codes, such as fountain codes [Byers et al. 1998] or online codes [Maymounkov 2003], present a very efficient alternative to MDS codes (O(n) encoding and decoding time) in which all blocks are constrained. However, they do not achieve access pattern robustness and sequentiality.

4.2 Robust Auditing of Outsourced Data

A *robust* auditing scheme incorporates mechanisms for mitigating arbitrary amounts of data corruption. We consider a notion of mitigation that includes the ability to both efficiently detect data corruption and be impervious to data corruption. When data corruption is detected, the owner can act in a timely fashion (*e.g.*, data can be restored from other replicas). Even when data corruption is not detected, a robust auditing scheme ensures that no data will be lost. More formally, we define a robust auditing scheme as follows:

DEFINITION 4.1. *A* robust auditing scheme *$\mathcal{RA}$ is a tuple $(\mathcal{C}, \mathcal{T})$, where $\mathcal{C}$ is a remote data checking scheme for a file $\mathtt{F}$ and $\mathcal{T}$ is a transformation that yields $\tilde{\mathtt{F}}$ when applied on $\mathtt{F}$. We say that $\mathcal{RA}$ provides δ-*robustness *when:*
– the auditor will detect *with high probability if the server corrupts more than a δ-fraction of $\tilde{\mathtt{F}}$ (**protection against corruption of a*** **large** ***portion of $\tilde{F}$**)*
– the auditor will recover *the data in $\mathtt{F}$ with high probability if the server corrupts at most a δ-fraction of $\tilde{\mathtt{F}}$ (**protection against corruption of a*** **small** ***portion of $\tilde{F}$**)*

In essence, by adding robustness to a RDC scheme $\mathcal{C}$, we seek to improve the original data possession guarantee offered by $\mathcal{C}$. Whereas a RDC scheme offers a data possession guarantee, a robust auditing scheme offers a *robust data possession guarantee*.

RDC schemes based on spot checking meet the lightweight requirement; they also provide partial robustness, in that they efficiently detect with high probability when a considerable portion of the data is corrupted (*e.g.*, over 1%). To fully meet the robustness

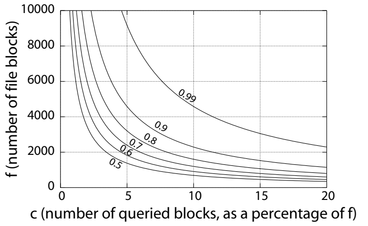


Fig. 3: P_X, the probability of server misbehavior detection. We show P_X as a function of f and c for $x = 1\%$ of f (where x is the number of blocks corrupted by the server).

requirement, we combine spot checking with data encoding (using erasure codes). Note that spot checking is a general technique that is beneficial for auditing, regardless of the details of a specific auditing scheme (*e.g.*, auditing based on retrieving the data objects [Kotla et al. 2007], or auditing based on retrieving fingerprints of the data objects [Ateniese et al. 2007; Juels and Kaliski 2007]). The notion of robustness we introduce here is relevant for any auditing scheme based on spot checking. Although spot checking has been previously considered for auditing, we are the first to consider the robustness dimension of spot checking-based auditing and the various tradeoffs related to it.

In this section, we present our auditing schemes based on spot checking and analyze the guarantees they provide. We start with BSCA (Basic Spot Checking Audit), which performs simple spot checking of the original data and was used in the PDP schemes presented in Section 3. BSCA serves as a comparison baseline for our Robust Spot Checking Audit (RSCA) schemes, which improve the basic scheme through the use of erasure encoding.

4.2.1 ***Basic Spot Checking Audit (BSCA)***. The server stores an f-block file, out of which it corrupts x blocks. The client C spot checks by randomly selecting for audit c different blocks over the entire file. This "sampling" mechanism, used in our PDP schemes in Section 3.3, greatly reduces the workload on the server S.

Let X be a discrete random variable defined as the number of blocks chosen by C that match the blocks corrupted by S. We compute P_X, the probability that at least one of the blocks picked by C matches one of the blocks corrupted by S. We have:

$$P_X = P\{X \geq 1\} = 1 - P\{X = 0\} = 1 - \frac{f-x}{f} \cdot \frac{f-1-x}{f-1} \cdot \frac{f-2-x}{f-2} \cdot \ldots \cdot \frac{f-c+1-x}{f-c+1}$$

It follows that: $1 - \left(1 - \frac{x}{f}\right)^c \leq P_X \leq 1 - \left(1 - \frac{x}{f-c+1}\right)^c$

P_X indicates the probability that, if S corrupts x blocks of the file, then C detects server misbehavior after a challenge in which it asks proof for c blocks. Fig. 3 plots P_X for different values of f and c.

However, when the server only corrupts a small portion of the file (*e.g.*, one block), an auditor using the BSCA scheme would have to dramatically increase the number of audited blocks in order to achieve detection with high probability. This would render impractical the whole concept of lightweight audit through spot checking. To conclude, BSCA does not provide satisfactory audit guarantees when a small number of blocks is corrupted.

4.2.2 ***Robust Spot Checking Audit (RSCA)***. The data possession guarantee offered by a remote data checking scheme for a file `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. The intuition behind encoding the file is that encoding complements spot checking and extends the robustness of the auditing mechanism to be impervious to small amounts of data corruption. This generic transformation can be applied to any remote data checking scheme based on spot checking.

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 sub-portions 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 sequentially 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.

For performance reasons, it is desirable to fix the parameters of the RS encoding. This also fixes the code generation matrix. We divide the f-block file $\mathtt{F}$ into k-block chunks 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 remote 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, and for PRPs π, ψ, ω, respectively.

Simple-RS. A simple encoding takes a file $\mathtt{F} = \mathtt{b}_1, \ldots, \mathtt{b}_f$ and generates the encoded file $\tilde{\mathtt{F}} = \mathtt{b}_1, \ldots, \mathtt{b}_f, \mathtt{c}_1, \ldots, \mathtt{c}_{\frac{f}{k}d}$, with blocks $\mathtt{b}_{ik+1}, \ldots, \mathtt{b}_{(i+1)k}$ 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 sequentiality requirement.

However, with fixed values of k and d, an attacker can *effectively corrupt* data by deleting a *fixed* number of blocks: Deleting any $d{+}1$ 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 $\tilde{\mathtt{F}}$ (as per Def. 4.1) 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. Encryption is then

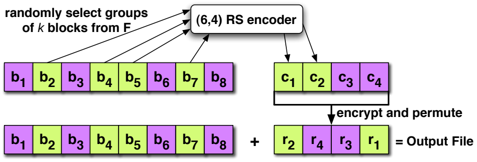


Fig. 4: Computation of a $(6,4)$ code with πR. The file has two constraint groups (different colors).

applied to all blocks so that constraints among the permuted blocks cannot be uncovered.

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 [Juels and Kaliski 2007]) leads to a robust data possession guarantee (as shown by our analysis in Section 4.3). However, the scheme has several drawbacks: The resource-intensive nature of permuting the entire encoded file can be rather slow (as acknowledged in [Juels and Kaliski 2007]); also, the scheme does not meet the sequentiality requirement. Moreover, the scheme does not achieve access pattern robustness, because sequentially accessing the original data (*i.e.*, data in $\mathtt{F}$) reveals dependencies among constraint groups in $\tilde{\mathtt{F}}$.

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 $\tilde{\mathtt{F}} = \mathtt{F}||\mathtt{R}$.

Fig. 4 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 that 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. 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 corrupt data differentially among these two regions to some advantage.

For example, an attacker guarantees damage to a file by deleting all blocks in R and one block in F. No deterministic attack that corrupts the same number of blocks exists for πA.

The πR scheme meets all the requirements put forth in Section 4.1: The use of a systematic code, which outputs the original blocks as part of the output, ensures the sequentiality and access pattern robustness 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. However, the encryption step required for robustness breaks the *data format independence* feature described in Section 3.3 (still, the use of a systematic code partially achieves this feature). We leave as an open problem the design of schemes that are robust and also fully meet data format independence.

4.3 Scheme 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.

An attack is successful if: (a) the attacker causes damage to the original data and (b) the attack is not detected by the auditing mechanism. Both need to happen. Clearly, an attacker that does not cause damage to the original data is not successful. Also, an attacker whose actions are detected is not successful, because the data owner is able to repair the damage in a timely fashion (*e.g.*, from other replicas). Thus:

$$P(attack) = P(damage) \cdot (1 - P(detect)) \qquad (1)$$

We analyze next $P(damage)$ and $P(detect)$ and their dependency on the attacker's deletion strategy and the client's auditing strategy. In general, the auditing strategy is fixed and is publicly known. This allows the attacker to adapt her deletion strategy in order to maximize $P(attack)$.

In what follows, we assume the attacker corrupts x blocks and the auditor checks c blocks, out of the $\frac{fn}{k}$-block file $\tilde{\mathtt{F}}$.

4.3.1 ***Probability of data damage:*** $P(damage)$. An attacker causes damage to the original file F if it corrupts $d + 1$ blocks that belong to the same constraint group in the encoded file $\tilde{\mathtt{F}}$.

Analysis of πA and πR. We encode an f-block file with a (n, k) code that corrects for up to d corruptions. This produces an encoded file of length $f\frac{n}{k}$ blocks, which has f/k different constraint groups. In πA, all the file blocks are encrypted and permuted and the attacker deletes x blocks of the file at random. In πR, only the check blocks are encrypted and permuted and the attacker splits her deletions between the unencoded blocks (F) and the encrypted and permuted redundancy blocks (R).

In Appendix B [Ateniese et al. 2010] we derive formulas for $P(damage)$ for both πA and πR. However, the formulas are not in closed form and evaluating the inclusion/exclusion series of the hypergeometric distribution is not computationally reasonable (the alternating signs of the inclusion/exclusion terms do not allow to bound the expansion by evaluating fewer terms and inclusion/exclusion processes do not always converge fast).

Monte-Carlo Results. Thus, we turn to Monte-Carlo simulation to determine $P(damage)$. Our Monte-Carlo simulation models an attacker that corrupts blocks randomly, but may

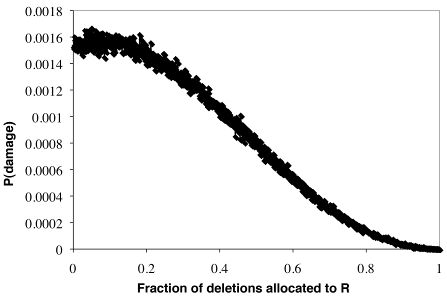


Fig. 5: Identification of the best strategy to damage F by varying where blocks are corrupted.

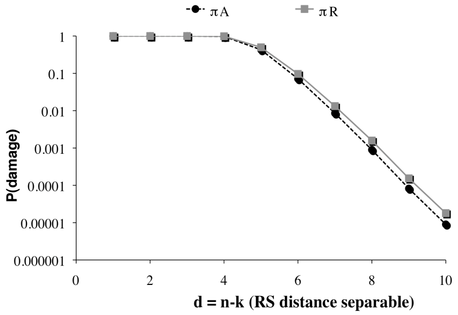


Fig. 6: Probability of damaging F against the πA and πR encodings for different RS parameters.

choose the distribution of those corruptions 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 a 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.

Attacker's Best Strategy. An attacker that corrupts x blocks can split those blocks between the original file data (F) and the redundancy information (R). Examining the probability of deletion as a 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*. Fig. 5 shows the 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 corrupts 1080 blocks (1% of the data). The attacker maximizes $P(damage)$ when it allocates 5-10% of the total deletions to R. Although the results are somewhat noisy, they match well with the fact that R represents 7.4% of the total encoded file.

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 corrupted 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. We now quantify the difference in $P(damage)$ 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 corrupts 1% of the data. Fig. 6 shows that $P(damage)$ for an attacker matches closely between the two encodings.

While πR gives some advantage to an attacker, it is minor and quantifiable. A system

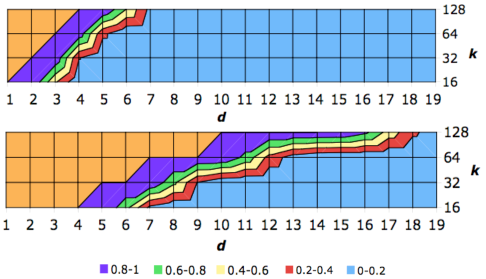


Fig. 7: Surface map of $P(damage)$ when adversary corrupts 1% (top) and 5% (bottom) of data from $\mathtt{F}$ and $\mathtt{R}$ (k and d are given in number of blocks). The areas on the far right and on the far left correspond to $P(damage) = 0$ and $P(damage) = 1$, respectively.

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: Sequentiality and access pattern robustness.

Visualizing Damage Probability in πR. Fig. 7 shows the damage probability in πR for different encoding parameters. We measure this probability at two deletion levels: 1% and 5%. As is expected, the amount of reliability goes up with the distance of the code d (due to the increased redundancy). However, it also goes up with decreasing k. Smaller k translates to more constraint groups and more storage overhead (while holding d constant).

4.3.2 ***Probability of attack detection:*** $P(detect)$. In Section 4.3.1, we established that the attacker's best strategy to damage the original file is to distribute her deletions between $\mathtt{F}$ and $\mathtt{R}$ in proportion to their size. In this section, we examine the probability of attack detection, $P(detect)$. The auditor must distribute the audited (checked) blocks between $\mathtt{F}$ and $\mathtt{R}$. Because detecting attacks in $\mathtt{F}$ and in $\mathtt{R}$ are not mutually exclusive events, we have:

$$P(detect_{\tilde{\mathtt{F}}}) = P(detect_{\mathtt{F}}) + P(detect_{\mathtt{R}}) - P(detect_{\mathtt{F}}) \cdot P(detect_{\mathtt{R}}) \qquad (2)$$

Ideally, the auditor should distribute the checked blocks to match the attacker's strategy of distributing deletions between $\mathtt{F}$ and $\mathtt{R}$ (recall that the attacker corrupts x out of f blocks). However, the auditor does not know the attacker's deletion strategy *a priori*. Thus, the auditor assumes that the attacker has maximized her $P(damage)$ and checks the blocks accordingly, by distributing the c checked blocks between $\mathtt{F}$ and $\mathtt{R}$ in proportion to their size. More precisely, if $\tilde{\mathtt{F}} = \mathtt{F}||\mathtt{R}$, where $\mathtt{F}$ has f blocks, $\mathtt{R}$ has $\frac{f}{k}d$ blocks, and $\tilde{\mathtt{F}}$ has $f + \frac{f}{k}d$ blocks, then $\mathtt{F}$ represents a fraction $\frac{k}{n}$ of $\tilde{\mathtt{F}}$ and $\mathtt{R}$ represents a fraction $\frac{d}{n}$ of $\tilde{\mathtt{F}}$. Thus, the auditor checks $c_{\mathtt{F}} = \frac{k}{n}c$ blocks from $\mathtt{F}$ and $c_{\mathtt{R}} = \frac{d}{n}c$ blocks from $\mathtt{R}$.

Given this checking strategy, we determine if the attacker can adapt her deletion strategy in order to increase her chances to evade detection (*i.e.*, minimize $P(detect_{\tilde{\mathtt{F}}})$). We have:

$$P(detect_{\mathtt{F}}) \geq 1 - \left(1 - \frac{x_{\mathtt{F}}}{f}\right)^{c_{\mathtt{F}}} \; ; \; P(detect_{\mathtt{R}}) \geq 1 - \left(1 - \frac{x - x_{\mathtt{F}}}{\frac{f}{k}d}\right)^{c_{\mathtt{R}}}$$

in which $x_{\mathtt{F}}$ is the number of blocks corrupted from $\mathtt{F}$. We replace in Eq. 2 and take the first derivative to determine the value of $x_{\mathtt{F}}$ for which $P(detect_{\tilde{\mathtt{F}}})$ is minimized. This yields $x_{\mathtt{F}} = \frac{k}{n}x$. Thus, when the auditor distributes the checked blocks between $\mathtt{F}$ and $\mathtt{R}$ according to their size, the attacker's best strategy to minimize her probability of attack detection is to *also* distribute her deletions between $\mathtt{F}$ and $\mathtt{R}$ according to their size.

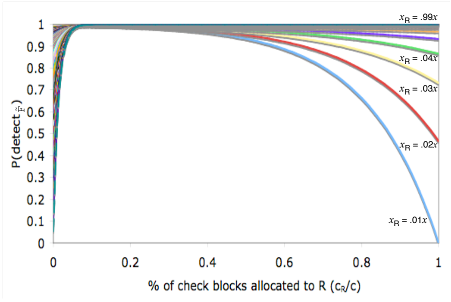


Fig. 8: Charting all possible adversarial deletion strategies given a priori knowledge of the checking strategy.

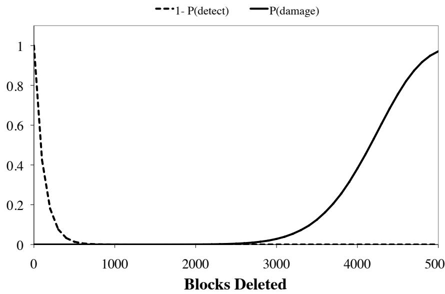


Fig. 9: $1 - P(detect)$ and $P(damage)$ as a function of the number of corruptions by the attacker.

In Fig. 8, we analyze all possible deletion strategies, for all possible checking strategies given a 100,000 block unencoded file expanded using a (108,100) RS code. The adversary corrupts 1080 blocks in all. The figure confirms that by checking proportionately to the size of $\mathtt{F}$ and $\mathtt{R}$, the auditor maximizes the minimum $P(detect_{\tilde{\mathtt{F}}})$ the adversary can achieve by varying her deletion strategy when it has a priori knowledge of the checking strategy. We conclude that the auditor should choose this checking strategy.

4.3.3 ***Probability of a successful attack:*** $P(attack)$. We use our analysis to show that the πR encoding realizes the robust possession guarantee. We give an example in which we set a robust possession target: no attack should succeed with $P(attack) > 10^{-10}$.

We used the parameter selection guidelines that we outline in the next section to identify the specific parameterization that meets the robust possession goal. We selected the solution that minimized the number of blocks spot-checked during audit for which the space overhead was $< 10\%$ and k was equal to 128. This produces an $(140, 128, 12)$-RS code and an auditor that checks 1188 blocks for an input file $128,000$ blocks.

For this configuration, Fig. 9 allows us to visualize robustness. An attacker chooses the number of blocks to corrupt, between 0 blocks and the entire file. We show only 0 to 5000. For small deletions, $P(damage)$ is essentially zero. For large deletions, $1 - P(detect)$ is negligible. In the interesting region from 500–3000 blocks corrupted, both quantities take low values (always $< 10^{-4}$) and their product $P(attack)$ remains below 10^{-10}.

4.4 Parameter Selection

We use our analytical and experimental results to define specific parameterizations of the encoding scheme. At the highest level, the parameter selection process chooses values for the encoding discipline (*i.e.*, the RS-encoding parameters (n, k, d)) and for the checking discipline (*i.e.*, the number of blocks c to check in each audit) that meet application requirements on the encoding rate, space overhead, audit performance, and the robust possession guarantee. This is an optimization process for which we will search the parameter space for the best solution. Applications can minimize (or maximize) one of the user requirements and constrain the values of all other requirements. A simple hill-climbing approach suffices because all of the user requirements are monotonic in their dependent parameters. We can enhance this search at times by using binary search on parameter values.

Parameter selection is best described through an example. Thus, we describe the solution used in Section 4.3.3. In this case, we find the parameterization on (n, k, d) and c for a file of 128,000 blocks that maximizes the audit performance (minimizes c) subject to the robust possession goal that $P(attack) < 10^{-10}$. The output is a list of parameter values that meet the robust possession goal. The user may then select a specific encoding and audit discipline from this list using any criteria.

The system searches the parameter space for configurations of (n, k, d) and c that meet the target, using our Monte-Carlo simulator to evaluate $P(damage)$ and our analysis for $P(detect)$ to evaluate $P(attack)$ (Eq. 1). There are three degrees of freedom in this search and we select k, d, and c: for RS codes $n - k = d$.

We observe that for specific values of n and k, there will be a minimum value of c that realizes the robust possession goal. This is always true: For arbitrary k and $d = 0$, the auditor can check all blocks to get a 100% guarantee that no blocks were damaged. An inefficient search executes:

```
for all $k \in k_{min} \dots k_{max}$ do
    for all $d \in d_{min} \dots d_{max}$ do
        find $c$ such that $P(attack) < 10^{-10}$
```

To speed the discovery of the appropriate value of c, we can use binary search to implement `find`, because $P(attack)$ is monotone in parameter c. We can also prune the search for d using branch and bound principles.

Other requirements and reasonable values further constrain the search and define $k_{min}, k_{max}, d_{min}$ and d_{max}. To realize acceptable encoding rates, $k + d < 256$ so that we can use the Cauchy variant of Reed-Solomon encoding [Plank and Xu 2006]. At the same time, k will tend to be as large as possible in order to increase reliability. In many cases, small values do not need to be searched. Many applications will limit the amount of space overhead (d/k) in order to meet cost or capacity limitations. Often k or n can be fixed a priori, because the storage devices have natural alignment, *e.g.*, in a RAID array it is beneficial to have n be evenly divisible by the number of disks. Finally, lightweight auditing mandates that c should be small. Restricting c renders many combinations of k and d infeasible, which can be determined quickly.

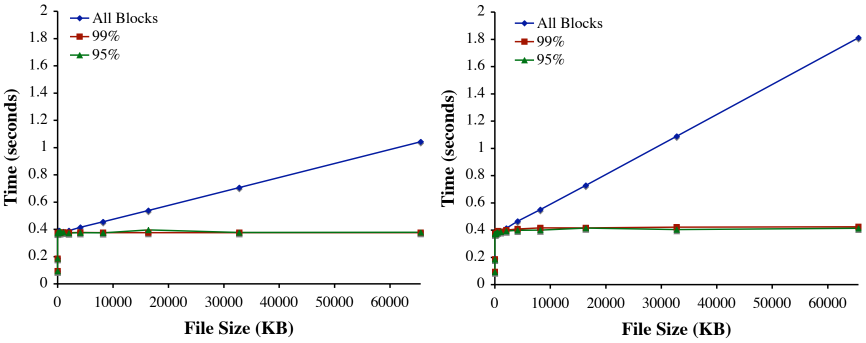
In this example, we place further requirements on the solution: Space overhead should be $< 10\%$ and $k = 128$. These constraints may be used to select from all possible configurations. However, they are better used to prune the search space. In this case, we can fix $k = 128$ and restrict d to be 12 or fewer. This still outputs many solutions: $d \in [0, 12]$ and the corresponding values for c. We selected parameters $(140, 128, 12)$-RS code and an auditor that checks 1188 blocks, *i.e.*, the lowest value of c and the highest value of d.

This example was one specific search and similar search processes can be used to optimize many different user requirements. Some users may wish to maximize reliability subject to constraints on c, space overhead, and encoding rate. We demonstrate the relationship between encoding rate and RS-parameters in Fig. 14. Alternatively, one may wish to minimize storage costs (space overhead) subject to reliability and audit performance.

5 System Implementation and Performance Evaluation

5.1 PDP Schemes

We measure the performance of E-PDP and the benefits of sampling based on our implementation of E-PDP. As a basis for comparison, we have also implemented the scheme of Deswarte *et al.* [Deswarte et al. 2003] and Filho *et al.* [Filho and Baretto 2006] (B-PDP),



(a) Server (in cache)

(b) Server (on disk)

Fig. 10: Performance of sampling at multiple confidence levels.

and the more efficient scheme in [Golle et al. 2002] (MHT-SE) suggested by David Wagner (these schemes are described in Appendix C [Ateniese et al. 2010] and briefly in Section 6).

We conducted experiments on an Intel 2.8 GHz Pentium IV system with a 512 KB cache, an 800 MHz EPCI bus, and 1024 MB of RAM. The system runs Red Hat Linux 9, kernel version 2.4.22. Algorithms use the crypto library of OpenSSL version 0.9.8b with a modulus N of size 1024 bits and files have 4KB blocks. Experiments that measure disk I/O performance do so by storing files on an ext3 file system on a Seagate Barracuda 7200.7 (ST380011A) 80GB Ultra ATA/100 drive. All experimental results represent the mean of 20 trials. Because results varied little across trials, we do not present confidence intervals.

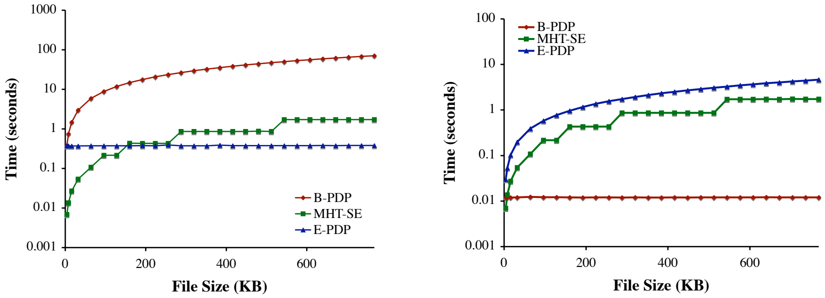
Sampling. To quantify the performance benefits of sampling for E-PDP, we compare the client and server performance for detecting 1% corrupted data at 95% and 99% confidence (Fig. 10). These results are compared with using E-PDP over all blocks of the file at large file sizes, up to 64 MB. We measure both the computation time only (in memory) as well as the overall time (on disk), which includes I/O costs.

Examining all blocks uses time linear in the file size for files larger than 4MB. This is the point at which the computation becomes bound from either memory or disk throughput. Larger inputs amortize the cost of the single exponentiation required by E-PDP. This is also the point at which the performance of sampling diverges. The number of blocks needed to achieve the target confidence level governs performance.

For larger files, E-PDP generates data as fast as it can be accessed from memory and summed, because it only computes a single exponentiation. In E-PDP, the server generates $\sum_{i=1}^{c} \mathtt{b}_i$, which it exponentiates. The maximum size of this quantity in bits is $|\mathtt{b}_i| + \log_2(c)$; its maximum value is $c \cdot 2^{|\mathtt{b}_i|}$. Thus, the cryptographic costs grows logarithmically in the file size. The linear cost of accessing all data blocks and computing the sum dominate this logarithmic growth.

Comparing results when data are on disk versus in cache shows that disk throughput bounds E-PDP's performance when accessing all blocks. Except the first blocks of a file, I/O and the challenge computation occur in parallel. Thus, E-PDP generates proofs faster than the disk can deliver data: 1.0 second versus 1.8 seconds for a 64 MB file. Because I/O bounds performance, no protocol can outperform E-PDP by more than the startup costs. While faster storage may remove the I/O bound today, assuming that over time increases in processor speeds will exceed those of disk bandwidth, then the I/O bound will hold.

Sampling breaks the linear scaling relationship between time to generate a proof of data



(a) Challenge time compared

(b) Pre-processing time compared

Fig. 11: Computation performance.

possession and the file size. At 99% confidence, E-PDP can build a proof of possession for any file, up to 64 MB in size in about 0.4 seconds. Disk I/O incurs about 0.04 seconds of additional runtime for larger file sizes over the in-memory results. Sampling performance characterizes the benefits of E-PDP. Probabilistic guarantees make it practical to use public-key cryptography constructs to verify possession of very large data sets.

Server Computation. The next experiments look at the worst-case performance of generating a proof of possession, which is useful for planning purposes to allow the server to allocate enough resources. For E-PDP, this means sampling every block in the file, while for MHT-SE this means computing the entire hash tree. We compare the computation complexity of E-PDP with other algorithms, which do not support sampling. All schemes perform an equivalent number of disk and memory accesses.

In step 3 of the GenProof algorithm of S-PDP, S has two ways of computing ρ: Either sum the values $a_j \mathtt{b}_{i_j}$ (as integers) and then exponentiate g_s to this sum or exponentiate g_s to each value $a_j \mathtt{b}_{i_j}$ and then multiply all values. We observed that the former choice takes considerable less time, as it only involves one exponentiation to a $(|\mathtt{b}_i| + \ell + \log_2(c))$-bit number, as opposed to c exponentiations to a $(|\mathtt{b}_i| + \ell)$-bit number (typically, $\ell = 160$).

Fig. 11(a) shows the computation time as a function of file size used at the server when computing a proof for B-PDP, MHT-SE and E-PDP. Note the logarithmic scale. Computation time includes the time to access the memory blocks that contain file data in cache. We restrict this experiment to files of 768 KB or less, because of the amount of time consumed by B-PDP.

E-PDP radically alters the complexity of data possession protocols and even outperforms protocols that provide weaker guarantees, specifically MHT-SE. For files of 768 KB, E-PDP is more than 185 times faster than B-PDP and more than 4.5 times as fast as MHT-SE. These performance ratios become arbitrarily large for larger file sizes. For B-PDP performance grows linearly with the file size, because it exponentiates the entire file. For MHT-SE, performance also grows linearly, but in disjoint clusters which represent the height of the Merkle-tree needed to represent a file of that size.

Pre-Processing. In preparing a file for outsourced storage, the client generates its local metadata. In this experiment, we measure the processor time for metadata generation only. This does not include the I/O time to load data to the client or store metadata to disk, nor does it include the time to transfer the file to the server. Fig. 11(b) shows the pre-processing time as a function of file size for B-PDP, MHT-SE and E-PDP.

E-PDP exhibits slower pre-processing performance. The costs grow linearly with the

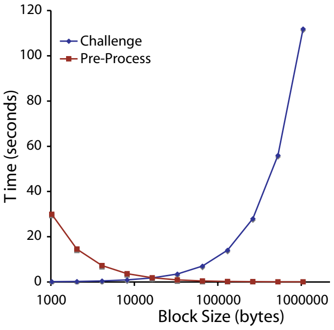


Fig. 12: E-PDP pre-processing versus challenge trade-offs with block size for a 1 MB file.

file size at 162 KB/s. E-PDP performs an exponentiation on every block of the file in order to create the per-block tags. For MHT-SE, preprocessing performance mirrors challenge performance, because both protocol steps perform the same computation. It generates data at about 433 KB/s on average.

The preprocessing performance of B-PDP differs from the challenge phase even though both steps compute the exact same signature. This is because the client has access to $\phi(N)$ and can reduce the file modulo $\phi(N)$ before exponentiating. In contrast, the security of the protocol depends on $\phi(N)$ being a secret that is unavailable to the server.

E-PDP also exponentiates data that was reduced modulo $\phi(N)$ but does not reap the same speed up, because it must do so for every block. This creates a natural trade-off between preprocessing time and challenge time by varying the block size; *e.g.*, the protocol devolves to B-PDP for files of a single block. Fig. 12 shows this trade-off and indicates that the best balance occurs at natural file system and memory blocks sizes of 4-64 KB (note that in this example all file blocks are checked in a challenge). We choose a block size of 4K in order to minimize the server's effort.

Given the efficiency of computing challenges, pre-processing represents the limiting performance factor for E-PDP. The rate at which clients can generate data to outsource bounds the overall system performance perceived by the client. However, there are several mitigating factors. (1) Outsourcing data is a one time task, as compared to challenging outsourced data, which will be done repeatedly. (2) The process is completely parallelizable. Each file can be processed independently at a different processor. A single file can be parallelized trivially if processors share key material.

5.2 Encoding Schemes for Robustness

In this section, we will study how the I/O performance and the encoding rate are affected by the choice of encoding scheme and encoding parameters respectively. The I/O experiments help us compare the disk read performance for the three proposed schemes: Simple-RS, πA and πR. Encoding rate experiments allow us to determine the parameterizations that meet the encoding rate goals defined as a user requirement.

I/O Performance. The sequentiality of the data layout, achieved by both the Simple-RS and the πR schemes, has significant implications for I/O read performance. In Fig. 13, we study the effects of the three encoding schemes (Simple-RS, πA, and πR) on the rate at which data can be read from the disk. Measuring this rate is useful for both parameter selection and for comparing the I/O performance of the three encoding schemes. For each encoding scheme, we consider two typical data access patterns in a storage system:

	Sequential	Random
Simple-RS	53,650	463
πA	458	458
πR	53,650	463

Fig. 13: I/O read performance for the three schemes (Simple-RS, πA, and πR) under two different workloads (sequential and random). The throughput values are in KB/sec.

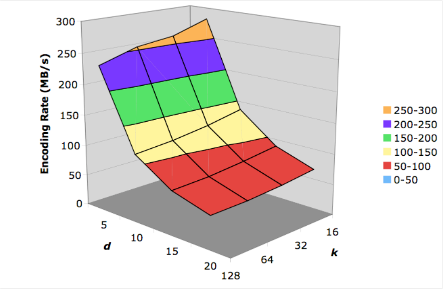


Fig. 14: Sensitivity of encoding parameters to encoding rate (k and d are given in number of blocks).

sequential and random.

We measure I/O performance on the original file data, *i.e.*, I/Os are performed to locations in the original file $\mathtt{F}$. Our system maps these logical offsets to the disk locations in the corresponding encoded file $\tilde{\mathtt{F}}$. This mapping is not important for Simple-RS and πR, which systematically embed the input file in the output. It does matter for πA, which permutes the blocks of $\mathtt{F}$ into $\tilde{\mathtt{F}}$. We used Iozone v3.308 [Iozone] to measure the performance. The experiment was run on a dual core Intel Pentium 4 running at 3 GHz, with 1 GB memory The L1 cache size was 16KB while the L2 cache size was 2048KB. The hard disk was a Western Digital SATA II 80GB hard disk running at 7200 RPM.

This experiment shows the benefit of a systematic layout, such as πR, on sequential read performance. Permuting the blocks as in πA decreases performance for this workload by more than two orders of magnitude. The permutation turns the sequential workload in $\mathtt{F}$ into a random workload in $\tilde{\mathtt{F}}$.

Encoding Performance. We examine the effect of the encoding parameters on performance of encoding files. This experiment was conducted on a 2 GHz dual core AMD Opteron 2212 processor with a HyperTransport bus running at 1 GHz and 4 GB of memory at 667MHz. The L1 cache is 64 KB and the L2 cache is 1 MB. The hard disk is a 250GB 7.2K RPM Serial ATA 3Gbps 3.5-in Cabled Hard Drive. Jerasure v1.0 [Plank et al. 2008] was used to perform Cauchy Reed Solomon encoding (we use Reed Solomon encoding based on Cauchy matrices [Plank and Xu 2006], which was shown to be twice as fast as classical Reed Solomon encoding based on Vandermonde matrices [Plank 2005]).

Fig. 14 shows how the encoding rate varies with k and d. The encoding rate is far more sensitive to the distance d than to width of the code k. Therefore, for the purpose of encoding performance, at a given space overhead, it is preferable to have smaller, more numerous constraint groups (and hence smaller code distances).

As described in Section 4.4, encoding rates serve as an input to parameter selection, often allowing us to prune the parameter search space. These results allow us to define the specific parameterizations that meet encoding goals. Typically, encoding performance requirements take a lower bound.

6 Related Work

Deswarte *et al.* [Deswarte et al. 2003] and Filho *et al.* [Filho and Baretto 2006] provide techniques to verify that a remote server stores a file using RSA-based hash functions.

Unlike other hash-based approaches, it allows a client to perform multiple challenges using the same metadata. In this protocol, communication and client storage complexity are both $O(1)$. The limitation of the algorithm lies in the computational complexity at the server, which must exponentiate the entire file, accessing all of the file's blocks. Further, RSA over the entire file is extremely slow — 20 seconds per Megabyte for 1024-bit keys on a 3.0 GHz processor [Filho and Baretto 2006]. In fact, these limitations led us to study algorithms that allowed for sub-file access (sampling). We implement this protocol for comparison with our PDP scheme and refer to it as B-PDP (basic PDP). A description of B-PDP is provided in Appendix C [Ateniese et al. 2010]. Shah *et al.* [Shah et al. 2007] use a similar technique for third-party auditing of data stored at online service providers and put forth some of the challenges associated with auditing online storage services.

Schwarz and Miller [Schwarz and Miller 2006] propose a scheme that allows a client to verify the storage of m/n erasure-coded data across multiple sites even if sites collude. The scheme can also be used to verify storage on a single server and relies on a special construct, called an "algebraic signature": A function that fingerprints a block and has the property that the signature of the parity block equals the parity of the signatures of the data blocks. The parameters of the scheme are comparable with our PDP schemes and the authors propose performance optimizations to achieve checking throughputs of hundreds of Mbytes/sec. However, the scheme receives a less formal security analysis.

Sebe *et al.* [Sebe et al. 2004] give a protocol for remote file integrity checking, based on the Diffie-Hellman problem in $\mathbb{Z}_N$. The client has to store N bits per block, where N is the size of an RSA modulus, so the total storage on the client is $O(n)$ (which does not conform to our notion of an outsourced storage relationship). Indeed, the authors state that this solution only makes sense if the size of a block is much larger than N. Moreover, the protocol requires the server to access the entire file. Similar techniques were proposed by Yamamoto *et al.* [Yamamoto et al. 2007], in the context of checking data integrity through batch verification of homomorphic hash functions.

Related to provable data possession is the enforcement of storage complexity, which shows that a server retains *an amount of information at least as large as the file* received from the client; the server does not necessarily retain the original file. To the best of our knowledge, Golle *et al.* [Golle et al. 2002] were the first to propose a scheme that enforces storage complexity. Golle *et al.* also briefly mention a scheme suggested by David Wagner, based on Merkle hash trees, which lowers the computational requirements for the server at the expense of increased communication. We implement Wagner's suggestion for comparison with our PDP scheme and refer to it as MHT-SE. A description of MHT-SE is provided in Appendix C [Ateniese et al. 2010].

Oprea *et al.* [Oprea et al. 2005] propose a scheme based on tweakable block ciphers that allows a client to detect the modification of data blocks by an untrusted server. The scheme does not require additional storage at the server and if the client's data has low entropy then the client only needs to keep a relatively low amount of state. However, verification requires the entire file to be retrieved, which means that the server file access and communication complexity are both linear with the file size per challenge. The scheme is targeted for data retrieval. It is impractical for verifying data possession.

Simultaneously with PDP, Juels and Kaliski have introduced a similar notion, that of *proof of retrievability* (PoR) [Juels and Kaliski 2007], which allows a client to be convinced that it can retrieve a file previously stored at the server. The main PoR scheme uses disguised blocks (called sentinels) hidden among regular file blocks in order to detect data corruption by the server. Although comparable in scope with PDP, their PoR scheme can only be applied to encrypted files and can handle a limited number of queries, which has to be fixed a priori. In contrast, our PDP schemes can be applied to public databases (*e.g.*, digital libraries, astronomy/medical/legal repositories, archives, etc.) and put no restriction on the number of challenges that can be executed. Shacham and Waters [Shacham and Waters 2008] give two PoR protocols based on homomorphic authenticators. The first is based on bilinear maps and achieves public verifiability, whereas the second is based on pseudo-random functions, more efficient, but is only privately verifiable.

The issue of integrating forward error correcting codes was initially introduced by Juels and Kaliski [Juels and Kaliski 2007]. They discuss breaking the file into chunks of size k and using an (n, k, d)-error correcting code on each chunk. The resulting output will be encrypted and permuted, ensuring that dependencies among contrained blocks (in the same chunk) remain hidden. While secure, this scheme results in very poor encoding and sequential I/O performance. The output file must be written randomly and, thus, one block at a time. The resulting file layout does not support sequential I/O, because sequential blocks in the original file have no spatial relationship in the resulting output. This is the πA scheme that permutes all blocks, which we implemented and evaluated for comparison.

Shacham and Waters [Shacham and Waters 2008] propose using Online codes [Maymounkov 2003] in a similar fashion. However, they improve upon the strategy of Juels and Kaliski by using tweakable ciphers, which avoids the permutation step.

Bowers et al. [Bowers et al. 2009b] describe an integration of Reed-Solomon codes with a systematic file layout that is similar to our file layout. It was identified independently and at roughly the same time (as our initial paper [Curtmola et al. 2008]). Both their scheme and ours meet the requirements we put forth. While they establish the bounds under which a client is able to retrieve data from the server, their treatment does not include practical guidance as to how to configure and use FECs with RDC, nor does it include our analysis, system implementation or evaluation.

Other extensions to remote data checking include extending the data possession guarantee to multiple servers based on replication without encoding each replica separately [Curtmola et al. 2008], based on erasure coding [Wang et al. 2009; Bowers et al. 2009a] and based on network coding [Chen et al. 2010], and to efficiently support dynamic data updates [Ateniese et al. 2008; Erway et al. 2009].

7 Conclusion

We focused on the problem of auditing if an untrusted server stores a client's data. We introduced a model for provable data possession, in which it is desirable to minimize the file block accesses, the computation on the server, and the client-server communication. Our solutions for PDP fit this model: They incur a low (or even constant) overhead at the server and require a small, constant amount of communication per challenge. Key components of our schemes are the support for spot checking, which ensures the schemes remain lightweight, and the homomorphic verifiable tags, which allow to verify data possession without having access to the actual data file. We also define the notion of robust auditing, which integrates RDC with FEC to mitigate arbitrarily small file corruptions and propose a generic transformation for adding robustness to any spot checking-based RDC scheme.

Experiments show that our schemes make it practical to verify possession of large data sets. Previous schemes that do not allow sampling are not practical when PDP is used to prove possession of large amounts of data, as they impose a significant I/O and computational burden on the server.

REFERENCES

ABE, M. AND FEHR, S. 2007. Perfect NIZK with adaptive soundness. In *Proc. of Theory of Cryptography*.

ATENIESE, G., BURNS, R., CURTMOLA, R., HERRING, J., KHAN, O., KISSNER, L., PETERSON, Z., AND SONG, D. 2010. Remote data checking using provable data possession. e-appendix.

ATENIESE, G., BURNS, R., CURTMOLA, R., HERRING, J., KISSNER, L., PETERSON, Z., AND SONG, D. 2007. Provable data possession at untrusted stores. In *Proc. of ACM CCS '07*.

ATENIESE, G., PIETRO, R. D., MANCINI, L. V., AND TSUDIK, G. 2008. Scalable and efficient provable data possession. In *Proc. of Securecomm*.

BELLARE, M., GARAY, J., AND RABIN, T. 1998. Fast batch verification for modular exponentiation and digital signatures. In *Proc. of EUROCRYPT '98*. LNCS. 236–250.

BELLARE, M. AND GOLDREICH, O. 1992. On defining proofs of knowledge. In *Proc. of CRYPTO '92*.

BELLARE, M. AND PALACIO, A. 2004a. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *Proc. of CRYPTO '04*. Lecture Notes in Computer Science. Springer, 273–289.

BELLARE, M. AND PALACIO, A. 2004b. Towards plaintext-aware public-key encryption without random oracles. In *Proc. of ASIACRYPT '04*. LNCS, vol. 3329. Springer, 48–62.

BELLARE, M. AND ROGAWAY, P. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *First Conference on Computer and Communications Security*. ACM, 62–73.

BELLARE, M. AND ROGAWAY, P. 1996. The exact security of digital signatures - How to sign with RSA and Rabin. In *EUROCRYPT*. 399–416.

BELLARE, M. AND ROGAWAY, P. 1998. PSS: Provably secure encoding method for digital signatures. IEEE P1363a: Provably secure signatures.

BLACK, J. AND ROGAWAY, P. 2002. Ciphers with arbitrary finite domains. In *Proc. of CT-RSA*. Springer-Verlag.

BONEH, D., GENTRY, C., LYNN, B., AND SHACHAM, H. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proc. of EUROCRYPT '03*. LNCS, vol. 2656. Springer, 416–432.

BOWERS, K. D., JUELS, A., AND OPREA, A. 2009a. HAIL: a high-availability and integrity layer for cloud storage. In *Proc. of ACM CCS '09*.

BOWERS, K. D., JUELS, A., AND OPREA, A. 2009b. Proofs of retrievability: Theory and implementation. In *Proc. of the 2009 ACM workshop on Cloud computing security (CCSW '09)*.

BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. 1998. A digital fountain approach to reliable distribution of bulk data. In *Proc. of ACM SIGCOMM*. 56–67.

CHEN, B., CURTMOLA, R., ATENIESE, G., AND BURNS, R. 2010. Remote data checking for network coding-based distributed storage systems. In *Proc. of ACM Cloud Computing Security Workshop (CCSW '10)*.

CURTMOLA, R., KHAN, O., AND BURNS, R. 2008. Robust remote data checking. In *Proc. of ACM StorageSS*.

CURTMOLA, R., KHAN, O., BURNS, R., AND ATENIESE, G. 2008. MR-PDP: Multiple-replica provable data possession. In *Proc. of ICDCS*.

DAMGARD, I. 1992. Towards practical public key systems secure against chosen ciphertext attacks. In *Proc. of CRYPTO '91*, J. Feigenbaum, Ed. Vol. 576. Springer, 445–456.

DENT, A. W. 2006a. The Cramer-Shoup encryption scheme is plaintext aware in the standard model. In *Proc. of EUROCRYPT '06*. LNCS, vol. 4004. Springer, 289–307.

DENT, A. W. 2006b. The hardness of the DHK problem in the generic group model. Cryptology ePrint Archive, Report 2006/156. `http://eprint.iacr.org/2006/156`.

DESWARTE, Y., QUISQUATER, J.-J., AND SAIDANE, A. 2003. Remote integrity checking. In *Proc. of IICIS*.

ERWAY, C., KUPCU, A., PAPAMANTHOU, C., AND TAMASSIA, R. 2009. Dynamic provable data possession. In *Proc. of ACM CCS*.

FIAT, A. 1990. Batch RSA. In *Proc. of CRYPTO '89*. Springer-Verlag, 175–185. LNCS.

FILHO, D. L. G. AND BARETTO, P. S. L. M. 2006. Demonstrating data possession and uncheatable data transfer. IACR ePrint archive. Report 2006/150, `http://eprint.iacr.org/2006/150`.

GOLLE, P., JARECKI, S., AND MIRONOV, I. 2002. Cryptographic primitives enforcing communication and storage complexity. In *Financial Cryptography*. 120–135.

HADA, S. AND TANAKA, T. 1998. On the existence of 3-round zero-knowledge protocols. In *Proc. of CRYPTO*.

HARN, L. 1998. Batch verifying multiple RSA digital signatures. *Electronics Letters 34,* 12, 1219–1220.

Iozone. Iozone filesystem benchmark. `http://www.iozone.org/`.

JUELS, A. AND KALISKI, B. S. 2007. PORs: Proofs of retrievability for large files. In *Proc. of ACM CCS*.

KALLAHALLA, M., RIEDEL, E., SWAMINATHAN, R., WANG, Q., AND FU, K. 2003. Plutus: Scalable secure file sharing on untrusted storage. In *Proc. of FAST*.

KOTLA, R., ALVISI, L., AND DAHLIN, M. 2007. Safestore: a durable and practical storage system. In *USENIX Annual Technical Conference*.

KRAWCZYK, H. 2005. HMQV: A high-performance secure Diffie-Hellman protocol. In *Proc. of CRYPTO '05*.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS '00*. ACM.

LI, J., KROHN, M., MAZIÈRES, D., AND SHASHA, D. 2004. Secure untrusted data repository (SUNDR). In *Proceedings of OSDI*.

MAHESHWARI, U., VINGRALEK, R., AND SHAPIRO, W. 2000. How to build a trusted database system on untrusted storage. In *Proc. of OSDI*.

MANIATIS, P., ROUSSOPOULOS, M., GIULI, T., ROSENTHAL, D., BAKER, M., AND MULIADI, Y. 2005. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computing Systems 23,* 1, 2–50.

MAYMOUNKOV, P. 2003. Online codes. Tech. Rep. TR2003-883, NYU.

MICALI, S., OHTA, K., AND REYZIN, L. 2001. Accountable-subgroup multisignatures: extended abstract. In *Proc of ACM CCS '01*. 245–254.

MILLER, G. L. 1976. Riemann's hypothesis and tests for primality. *JCSS 13,* 3, 300–317.

MYKLETUN, E., NARASIMHA, M., AND TSUDIK, G. 2004. Authentication and integrity in outsourced databases. In *Proceedings of NDSS*. The Internet Society.

NAOR, M. AND ROTHBLUM, G. N. 2005. The complexity of online memory checking. In *Proc. of FOCS*.

OKAMOTO, T. 1988. A digital multisignature schema using bijective public-key cryptosystems. *ACM Transactions on Computer Systems 6,* 4, 432–441.

OPREA, A., REITER, M. K., AND YANG, K. 2005. Space-efficient block storage integrity. In *Proc. of NDSS*.

PLANK, J. S. 2005. Erasure codes for storage applications. Tutorial Slides, presented at *FAST 2005*.

PLANK, J. S., SIMMERMAN, S., AND SCHUMAN, C. D. 2008. Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2. Tech. Rep. CS-08-627, University of Tennessee. August.

PLANK, J. S. AND XU, L. 2006. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *Proc. of IEEE NCA '06*.

SCHWARZ, T. S. J. AND MILLER, E. L. 2006. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Proceedings of ICDCS '06*. IEEE Computer Society.

SEBE, F., MARTINEZ-BALLESTE, A., DESWARTE, Y., DOMINGO-FERRER, J., AND QUISQUATER, J.-J. 2004. Time-bounded remote file integrity checking. Tech. Rep. 04429, LAAS. July.

SHACHAM, H. AND WATERS, B. 2008. Compact proofs of retrievability. In *Proc. of Asiacrypt 2008*.

SHAH, M., BAKER, M., MOGUL, J. C., AND SWAMINATHAN, R. 2007. Auditing to keep online storage services honest. In *Proc. of HotOS XI*. Usenix.

SHAH, M. A., SWAMINATHAN, R., AND BAKER, M. 2008. Privacy-preserving audit and extraction of digital contents. *ePrint Archive Report* 2008/186.

SHAMIR, A. 1983. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst. 1,* 1, 38–44.

WANG, C., WANG, Q., REN, K., AND LOU, W. 2009. Ensuring data storage security in cloud computing. In *Proc. of IWQos Workshop*.

YAMAMOTO, G., FUJISAKI, E., AND ABE, M. 2005. An efficiently-verifiable zero-knowledge argument for proofs of knowledge. Tech. Rep. ISEC2005-48, IEICE. July.

YAMAMOTO, G., ODA, S., AND AOKI, K. 2007. Fast integrity for large data. In *Proc. of SPEED '07*.

YUMEREFENDI, A. Y. AND CHASE, J. 2007. Strong accountability for network storage. In *Proc. of FAST '07*.

A Proofs

Proof of Theorem 3.3. Under the KEA1-r assumption, we reduce the security of our S-PDP scheme to the security of the RSA problem and the security of integer factoring. We model both hash functions $h(\cdot)$ and $H(\cdot)$ as random oracles. However we do not use the ”full power” of the random oracle model and indeed a scheme that does not use any random oracles can be easily derived from ours but at the cost of increased computational cost and bandwidth requirement.

We assume there exists an adversary $\mathcal{B}$ that wins the Data Possession Game on a challenge picked by $\mathcal{A}$ and show that $\mathcal{A}$ will be able to extract the blocks determined by the challenge. If $\mathcal{B}$ can break the data possession guarantee of the S-PDP scheme, we show how to construct an adversary $\mathcal{A}$ that uses $\mathcal{B}$ in order to either break RSA or factor the product of two large primes.

For the RSA problem, $\mathcal{A}$ is given (N, e, y), with $y \overset{R}{\leftarrow} \mathbb{Z}_N^*$, and needs to find a value $b \equiv y^{1/e} \bmod N$. We assume w.l.o.g. that e is a large prime number. $\mathcal{A}$ will play the role of the challenger in the Data Possession Game and will interact with $\mathcal{B}$.

We first look at the case when in GenProof and CheckProof all the coefficients $a_1, \ldots, a_c$ are equal to 1. This corresponds to the case where the server proves it possesses the sum of the requested blocks. We then generalize the proof to the case where the coefficients are random and pairwise distinct, which corresponds to the case where the server proves it possesses each individual block.

$\mathcal{A}$ simulates a PDP environment for $\mathcal{B}$ as follows:

Setup: $\mathcal{A}$ computes $g = y^2 \bmod N$, sets the public key $\mathrm{pk} = (N, g)$ and sends pk to $\mathcal{B}$. $\mathcal{A}$ generates the secret value $v \overset{R}{\leftarrow} \{0,1\}^\kappa$.

Query: $\mathcal{B}$ makes tagging queries adaptively: $\mathcal{B}$ selects a block m_1 and is also allowed to select an index i_1. $\mathcal{B}$ sends m_1 and i_1 to $\mathcal{A}$. $\mathcal{A}$ generates $\mathtt{T}_{i_1,m_1}$ and sends it back to $\mathcal{B}$. $\mathcal{B}$ continues to query $\mathcal{A}$ for the tags $\mathtt{T}_{i_2,m_2}, \ldots, \mathtt{T}_{i_f,m_f}$ on the blocks $m_2, \ldots, m_f$ and indices $i_1, \ldots, i_f$ of its choice. The only restriction is that $\mathcal{B}$ cannot make tagging queries for two different blocks using the same index.

$\mathcal{A}$ answers $\mathcal{B}$'s tagging oracle queries as follows:

– when $\mathcal{A}$ receives a tagging query for a block m and index i, with $1 \leq i \leq f$:

—if a previous tagging query has been made for the same m and i, then $\mathcal{A}$ retrieves the recorded tuple $(m, i, r_i, \mathtt{W}_i)$ and returns $\mathtt{T}_{i,m} = r_i$.

—else, $\mathcal{A}$ picks $r_i \overset{R}{\leftarrow} QR_N$, computes $\mathtt{W}_i = v||i$, records the tuple $(m, i, r_i, \mathtt{W}_i)$ and returns $\mathtt{T}_{i,m} = r_i$.

– when $\mathcal{A}$ receives a hash query for a value x:

—if a previous hash query was made for a value x, then $\mathcal{A}$ retrieves the recorded tuple (x, ω_x) and returns $h(x) = \omega_x$.

—else $\mathcal{A}$ picks $\omega \overset{R}{\leftarrow} QR_N$, records the tuple (x, ω_x) and returns $h(x) = \omega_x$.

$\mathcal{A}$'s view of the hash values $h(\mathtt{W}_i)$, for $1 \leq i \leq f$, is: $h(\mathtt{W}_i) = r_i^e \cdot g^{-m_i} \bmod N$. Clearly $\mathcal{B}$ could query the random oracle on values $\mathtt{W}_i$ with only negligible probability. Notice that in the random oracle model $h(\mathtt{W}_i)$ behaves simply as a PRF under the secret v.

Challenge: $\mathcal{A}$ generates the challenge $\mathsf{chal} = (g_s, i_1, \ldots, i_c)$, where $g_s = g^s \bmod N$, $s \overset{R}{\leftarrow} \mathbb{Z}_N^*$ and $i_1, \ldots, i_c$ are the indices of the blocks for which $\mathcal{A}$ requests proof of possession (with $1 \leq i_j \leq f, 1 \leq j \leq c, 1 \leq c \leq f$). $\mathcal{A}$ sends chal to $\mathcal{B}$.

Forge: $\mathcal{B}$ generates a proof $\mathcal{V} = (\mathtt{T}, \rho)$ about the blocks $m_{i_1}, \ldots, m_{i_c}$ determined by $i_1, \ldots, i_c$, where $\mathtt{T} = \mathtt{T}_{\{i_1,\ldots,i_c\}, m_{i_1}+\ldots+m_{i_c}}$. Note that $\mathcal{V}$ is a valid proof that passes $\mathsf{CheckProof}(\mathrm{pk}, \mathrm{sk}, \mathsf{chal}, \mathcal{V})$. $\mathcal{B}$ returns $\mathcal{V}$ to $\mathcal{A}$ and $\mathcal{A}$ checks the validity of $\mathcal{V}$. Let $M = m_{i_1} + \ldots + m_{i_c}$.

As H is a random oracle, with overwhelming probability we can extract the pre-image value ρ_p that $\mathcal{B}$ utilized to calculate ρ. (By the definition of a random oracle, $\mathcal{B}$ can guess a valid value of ρ with only negligible probability.)

$\mathcal{A}$ has given $\mathcal{B}$ both g, g^s and $\mathcal{B}$ has implicitly returned $\tau = \frac{\mathtt{T}^e}{\prod_{j=1}^{c} h(\mathtt{W}_{i_j})}, \rho_p$ by returning $\mathtt{T}, \rho$. Because $\tau^s = \rho_p$, by KEA-1r, $\mathcal{A}$ can utilize the extractor $\bar{\mathbf{B}}$ to extract a value M^* such that $g^{M^*} = \tau$ (if $-g^{M^*} = \tau$ then $\mathcal{A}$ sets $\mathtt{T} = -\mathtt{T} \bmod N$).

If $M^* = M$, then $\mathcal{A}$ was able to successfully extract the correct message M. We analyze next the case when $M^* \neq M$. Note that M^* is the "full-domain" value utilized by this calculation. (If the extractor $\bar{\mathbf{B}}$ is able to extract a value $M' \neq M^*$ such that $g^{M'} = g^{M^*} \bmod N$, this will allow to compute a multiple of $\phi(N)$, from which the factorization of N can be efficiently computed [Miller 1976].)

From $\tau = g^{M^*}$ we get $\mathtt{T}^e = (\prod_{j=1}^{c} h(\mathtt{W}_{i_j})) \cdot g^{M^*}$, where clearly $g^{M^*} \neq g^M$, and thus:

$$\begin{aligned}
\mathtt{T} &= \left(\left(\prod_{j=1}^{c} h(\mathtt{W}_{i_j}) \right) \cdot g^{M^*} \right)^d \\
&= \left(\left(\prod_{j=1}^{c} (r_{i_j}^e \cdot g^{-m_{i_j}}) \right) \cdot g^{M^*} \right)^d \\
&= \left(\prod_{j=1}^{c} r_{i_j} \right) \cdot \left(g^{M^*-M} \right)^d
\end{aligned}$$

$\mathcal{A}$ computes:

$$z = \frac{\mathtt{T}}{\prod_{j=1}^{c} r_{i_j}} = \left(g^{M^*-M} \right)^d$$

We have $z^e = g^{M^*-M} = y^{2(M^*-M)}$. Notice that $\gcd(e, 2(M^* - M)) = 1$ with overwhelming probability (this holds because e is a large prime number unknown to $\mathcal{B}$). Applying Shamir's "trick" [Shamir 1983], $\mathcal{A}$ uses the extended Euclidian algorithm to efficiently compute integers u and v such that $u \cdot e + v \cdot 2(M^* - M) = 1$ and outputs $y^{1/e} = y^u z^v$.

Note that the interactions of $\mathcal{A}$ with $\mathcal{B}$ are indistinguishable to $\mathcal{B}$ from interactions with an honest challenger in the Data Possession Game, as $\mathcal{A}$ chooses all parameters according to our protocol (and in particular note that $\mathcal{B}$ does not learn the value e by interacting with $\mathcal{A}$).

The proof generalizes to the case where the coefficients $a_1, \ldots, a_c$ are random and pairwise distinct. Indeed, in this case it is enough to apply the same simulation shown above and in addition to notice that at the end of the simulation $\mathcal{A}$ will be able to extract $\bar{M} = a_1 m_{i_1} + \ldots + a_c m_{i_c}$. We now have to show that our protocol constitutes a proof of knowledge of the blocks $m_{i_1}, \ldots, m_{i_c}$ when $a_1, \ldots, a_c$ are pairwise distinct. We show that a knowledge extractor $\mathcal{E}$ may extract the file blocks $m_{i_1}, \ldots, m_{i_c}$. Note that each time $\mathcal{E}$ runs the PDP protocol, $\mathcal{E}$ obtains a linear equation of the form $\bar{M} = a_1 m_{i_1} + \ldots + a_c m_{i_c}$. By choosing independent coefficients $a_1, \ldots, a_c$ in c executions of the protocol on the same blocks $m_{i_1}, \ldots, m_{i_c}$, $\mathcal{E}$ obtains c independent linear equations in the variables $m_{i_1}, \ldots, m_{i_c}$. $\mathcal{E}$ may then solve these equations to obtain the file blocks $m_{i_1}, \ldots, m_{i_c}$.

Proof of Theorem 3.4. The proof of Theorem 3.4 follows directly from the proof of Theorem 3.3. The main difference is that we do not need to use the KEA1-r extractor since the message is given to $\mathcal{A}$ directly. In addition, we allow the adversary $\mathcal{B}$ to select only messages m_i of a certain size and to check the validity of tags after each tag query.

Recall that we model h as a random oracle but now h is not just computed by the client over local and private values. (While mapping elements of arbitrary size into QR_N is sound in the random oracle model, in practice we instantiate the random oracle by squaring the output of a full-domain hash, the latter being effectively a square root of the output of h. This is not an issue since extending the publicly-verifiable scheme and its proof to work within the larger group $\mathbb{Z}_N^*$ is straightforward.)

When $\mathcal{B}$ makes a tagging query for a block m_i and index i, with $1 \leq i \leq f$, then $\mathcal{A}$ picks $r_i \overset{R}{\leftarrow} QR_N$, computes $\mathtt{W}_i = w_v(i)$ and returns $(\mathtt{T}_{i,m}, \mathtt{W}_i)$, where $\mathtt{T}_{i,m} = r_i$. $\mathcal{B}$ may verify the tag by checking whether the relation $(\mathtt{T}_{i,m_i})^e = h(\mathtt{W}_i) \cdot g^{m_i}$ holds. Indeed, when $\mathcal{B}$ makes a hash query for a value $\mathtt{W}_i$, $\mathcal{A}$ will return $h(\mathtt{W}_i) = r_i^e \cdot g^{-m_i} \bmod N$.

Now, if $\mathcal{B}$ releases a sum M^* that passes $\mathsf{CheckProof}$ such that $M^* \neq M$, then we can clearly solve the RSA instance since $gcd(e, 2(M^* - M)) = 1$ given that e is a prime bigger than $|M^* - M|$ (both $|M^*|$ and $|M|$ are smaller than $\lambda/2$ and $e > \lambda$).

B Analysis of Probability of Data Damage: $P(damage)$

Analysis of πA. We encode an f-block file with a (n, k) code that corrects for up to d corruptions. This produces an encoded file of length $f\frac{n}{k}$, which has f/k different constraint groups. The 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 corrupted 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 corrupted 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 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 ($\mathtt{F}$) and x_r deletions from the encrypted and permuted redundancy blocks ($\mathtt{R}$), 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 $\mathtt{F}$ and on $\mathtt{R}$ 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.

C Implemented PDP Schemes

As a basis of comparison, we have implemented the following two PDP schemes in addition to our E-PDP scheme:

Basic RSA-based PDP Scheme (B-PDP)**[Deswarte et al. 2003; Filho and Baretto 2006].** Let N be an RSA modulus and let $g \in \mathbb{Z}_N^*$.

Setup: C stores $a = g^{\mathtt{F}} \bmod N$. C sends $\mathtt{F}$ to S.

Challenge:

1. C generates a random value $r \overset{R}{\leftarrow} \mathbb{Z}_n^*$ and sends g^r to S.
2. S computes $b = (g^r)^{\mathtt{F}} \bmod N$ and sends b to C.
3. C computes a^r and checks if $b = a^r \bmod N$.

Merkle Hash Tree-based Storage Enforcing Scheme (MHT-SE)**[Golle et al. 2002].** Let ψ be a one-way length-increasing transformation and let h be a cryptographic hash function. In a binary tree, we denote by $PATH(i)$ the set of nodes on the path between the root of the tree and the i-th leaf.

Setup: C applies the transformation ψ on the file $\mathtt{F}$ and obtains the expanded file $\mathtt{F}' = \psi(\mathtt{F})$. C stores h_{root} as the root of the Merkle hash tree which is computed using h on the blocks of $\mathtt{F}'$.

Challenge:

1. C randomly picks the index i of a block $\mathtt{F}'_i$ of the expanded file $\mathtt{F}'$ and sends i to S.
2. Let H_i be the set of hashes corresponding to the nodes that “hang” off $PATH(i)$ in the Merkle hash tree determined by $\mathtt{F}'$. S sends H_i and $\mathtt{F}'_i$ back to C.
3. C uses H_i and $\mathtt{F}'_i$ to recompute the root of the tree and compares this value with the stored value for h_{root}.