Proof-of-Storage Constructions for Checking Integrity of Cloud Data

A thesis submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the degree of
Doctor of Philosophy in Computer Science

Author:
Binanda Sengupta

Supervisor:
Dr. Sushmita Ruj

Applied Statistics Unit
Indian Statistical Institute
203 Barrackpore Trunk Road
Kolkata, West Bengal

Chapter 1

Introduction

Cloud computing has emerged as a recent technology that enables users with restricted resources to delegate "heavy" tasks to a cloud server with more resources. For example, a smart device having a low-performance processor or a limited amount of storage capacity, cannot accomplish a heavy computation on its own or cannot store a large amount of data (say, in the order of terabytes) in its own storage. A cloud user having such a lightweight device can delegate her computation or storage to the cloud server. Now, she can just download the result of the computation, or she can read (or update) only the required portion of the uploaded data. Cloud service providers (CSPs) offer various such facilities or services to their clients. These services are typically classified into three categories: infrastructure as a service (IaaS) where the clients can utilize infrastructure components (for example, computing servers and storage servers) provided by the CSP, platform as a service (PaaS) where the clients can execute their own applications using the hardware and software tools offered by the CSP, and software as a service (SaaS) where the CSP hosts softwares (or applications) and the clients can execute them remotely as often as needed. The cloud computing environment can further be categorized as private cloud, public cloud and hybrid cloud. A private cloud environment is maintained by a single customer/enterprise where the cloud infrastructure is managed (and often owned) by that customer/enterprise. Only the customer or the members of the enterprise can get services from the cloud infrastructure. On the other hand, a public cloud is owned and managed by a third party cloud; and this third party cloud provides its clients with the services mentioned above. A hybrid cloud consists of public and private clouds.

In the infrastructure-as-a-service public cloud model, the clients can outsource their computational workload to a third party cloud server, or clients having limited storage capacity can store huge amount of data in the server. For storage outsourcing, the public cloud provides a robust infrastructure to the clients (data owners) in order to enable them storing large amount of data on the cloud storage server and accessing their data by downloading them from the cloud server. Several storage service providers like Amazon Simple Storage Service (S3) [4], Google Drive [79] and Dropbox [56] offer this type of storage outsourcing facility to their clients. These cloud storage service providers store clients' data in lieu of monetary benefits. The clients pay these providers for the service and expect that their (untampered) data can be retrieved at any point of time. However, a server can be untrusted and it can try to maximize the benefits with a given amount of storage at its end. We consider the following scenario. Suppose a server has 100 GB of storage capacity and there are two clients each requesting the server for 100 GB of storage. The server may store 50

GB data of each client and claim that it has stored 200 GB data. When a client wants to download her data from the server, she gets only half of her data. On the other hand, some part of the client's data may be permanently lost due to failure of some of the storage nodes, and the service provider might hide this information from the client in order to avoid compensations (or to maintain its reputation). Under these circumstances, the client needs a guarantee that her data have been stored intact by the server. In case of any modification or deletion of data, the server must compensate the client appropriately. Thus, the client (data owner) has to have a mechanism to check the integrity of her data outsourced to the cloud storage server. One possible way is to *audit* the outsourced data. The client performs audits on her data stored on the (possibly untrusted) remote server. If the server passes an audit, then the client is convinced that her data (or some part of the data) are stored untampered. The server might pass an audit without storing the client's data properly, but the probability of such an event is "very small".[1]

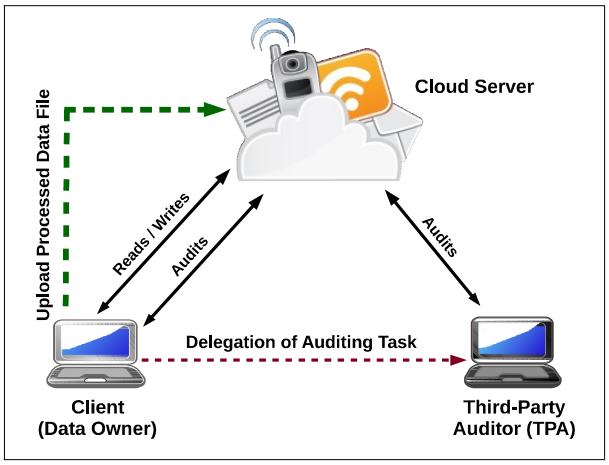
One naive way of auditing a data file is as follows. The client does some preprocessing on her data file before uploading them to the cloud server. This preprocessing phase includes computing a cryptographic authenticator (or tag) corresponding to the data file. The idea of using an authenticator is to prevent the server from modifying the file and still passing an integrity check with high probability. Now, the client uploads the processed file (data file along with the authenticators) to the storage server. During an audit, the client downloads the whole file along with the authenticator to her end and verifies the integrity of the file by checking whether the authenticator matches with the downloaded data file. The authenticator must satisfy the following property: given a set of valid file-authenticator pairs, the server cannot produce another valid file-authenticator pair. However, this solution is inefficient in practice due to the large communication bandwidth required between the client and the cloud server.

1.1 Proofs of Storage

Proofs of storage provide an efficient mechanism to check the availability of the client's data outsourced to a remote storage server. A proof-of-storage protocol is a two-party protocol executed between the client (data owner) and the cloud storage server. In a proof-of-storage scheme, the client can audit her data file stored on the server without accessing the whole file, and still, be able to detect an unwanted modification of the file done by the (possibly untrusted) server. Proof-of-storage schemes can be typically classified as: *provable data possession* (PDP) schemes [9, 58, 161] and *proof-of-retrievability* (POR) schemes [87, 140, 36, 144].

[1] This probability is a negligible function of the security parameter that we define in Section 2.1.1.

Figure 1-1: The entities involved in a proof-of-storage scheme for a data file



A proof-of-storage scheme consists of the following protocols [36, 144]: Init, Read, Write and Audit. The protocol **Init** consists of procedures for setup (key generation and setting system parameters) and outsourcing a data file F to the cloud storage server. The client (data owner) processes the data file F to output another file F'. She then outsources F' to the server. Later, the client executes the protocols **Read** and **Write** with the server to perform *authenticated* reads and writes on F'. In order to check the integrity of her outsourced data file F', the client runs the protocol **Audit**. During an execution of the Audit (*challenge-response*) protocol, the client sends a random challenge Q to the server, and the server responds with proofs of storage T computed on F' and Q. Finally, the client checks the validity of T. If the server passes the Audit protocol, the client is convinced that her data file is correctly stored by the server. These protocols are shown in Figure 1-1.

Depending on the nature of the outsourced data, proof-of-storage schemes are designed for either *static* data or *dynamic* data. For static data, the client cannot change her data after the initial outsourcing (this scenario is suitable mostly for backup or archival data). A proof-of-storage scheme for static data consists of the protocols mentioned above except Write. Dynamic data are more generic in the sense that the client can modify her data as often as needed. A proof-of-storage scheme is *publicly verifiable*[2] if anyone with an access to some public parameters can perform audits. On the other hand, in *privately verifiable* schemes, only the client with some

[2]The existing literature in proofs of storage uses the term "public verifiability" to denote (only) whether a third party auditor having the knowledge of the public parameters can perform an audit on behalf of the client (data owner) [9, 140, 161, 158]. We follow this notion of "public verifiability" in this thesis. This notion implicitly assumes that the client is honest. However, a dishonest client can publish incorrect public parameters in order to get an honest server framed by a third party auditor [95].

secret information can audit her data. In a publicly verifiable proof-of-storage scheme, the client can delegate the auditing task to a third party auditor (TPA) who audits the client's data and lets the client know if she finds any discrepancies. Figure 1-1 illustrates this scenario when the client delegates the auditing task to the third party auditor. We use the term "verifier" to denote an auditor in a proof-of-storage scheme. The client (for a privately verifiable scheme) or a third party auditor (for a publicly verifiable scheme) can act as the verifier. For a publicly verifiable proof-of-storage scheme, the data file the client outsources to the cloud server might have sensitive information that the client does not want to be leaked to a third party auditor (TPA). We assume that a third party auditor (TPA) executes the Audit protocol honestly. However, it might be interested in knowing the content of the file being audited. *Privacy-preserving audits* [158] are employed in this situation where the TPA cannot learn the content of the data file during an audit.

1.1.1 Building Blocks: PDP and POR

The concept of provable data possession (PDP) is introduced by Ateniese et al. [9]. In a PDP scheme, the client computes an authentication tag for each block of her data file and uploads the file along with these authentication tags. Later, the client audits the data file via *spot-checking* where the client verifies the integrity of only a predefined number of blocks sampled randomly. We note that PDP schemes guarantees retrievability of *almost all* data blocks of the file but not that of all blocks. For example, if the server corrupts a single block of the file, then the event that this particular block is not challenged during an audit occurs with a high probability — that makes this corruption undetected by the client.

Although many efficient PDP schemes are available in the literature, they only provide a guarantee of retrievability of almost all blocks of the data file. We briefly mention some situations where this guarantee is not sufficient. The data file may contain some sensitive information (e.g., accounting information) any part of which the client does not want to lose. On the other hand, a corruption in the compression table of an outsourced compressed file might make the whole file unavailable. In a searchable symmetric encryption scheme [48], the client encrypts a database using a symmetric encryption scheme to form an index (metadata for that database) [71] and outsources the encrypted documents along with the index to the server. The size of this index is very small compared to the encrypted database itself. However, loss of the index defeats altogether the purpose of the searchable encryption scheme. Under such circumstances, the client wants a stronger notion than PDP which would guarantee that the server has stored the *entire* file properly and the client can retrieve *all* of her data blocks at any point of time.

To address the issue mentioned above, Juels and Kaliski [87] introduce proofs of retrievability (POR) where the data file outsourced to the server can be retrieved in its entirety by the client. The underlying idea [141] of a POR scheme is to encode the original data file with an error-correcting code, authenticate the blocks of the encoded file with tags and upload them on the storage server. As in PDP schemes, the client can execute an audit protocol to check the integrity of the outsourced data file. The use of error-correcting codes ensures that the server has to corrupt significant number of blocks to make a single block irretrievable; and this corruption is detected during an audit with a very high probability. Thus, if the server passes an audit, *all* data blocks of the file are guaranteed to be available.

Performance of a proof-of-storage (PDP or POR) scheme is evaluated based upon several parameters. Some of these parameters include client-side storage, server-side storage, client-side computation (during Init, Read, Write and Audit), server-side computation (during Read, Write and Audit), client-server communication bandwidth (during Init, Read, Write and Audit).

1.1.2 Overview of Security of a Proof-of-Storage Scheme

We provide a brief overview of the security of a proof-of-storage scheme. We formally define the security notions in the following chapters of this thesis. The untrusted server acts as the *adversary* in this security model. Unless otherwise mentioned, we assume that the untrusted server is *malicious* that exhibits Byzantine behavior (i.e., it can corrupt the client's data in an arbitrary fashion). A secure proof-of-storage scheme must have the following properties [58, 144].

1. **Authenticity** The authenticity property requires that the adversary cannot produce valid proofs of storage T (corresponding to the challenge Q) without storing the data file and its authentication information untampered, except with a "very small" probability.

2. **Freshness** For *dynamic* data, the client can update her outsourced data file. However, the adversary might discard this change and keep an old copy of the data file. As the old copy of the file and its corresponding authentication information constitute a valid pair, the client cannot detect if the adversary is storing the fresh (latest) copy of her data file. The freshness property provides the client with guarantees that the adversary is storing an up-to-date version of the data file. This property is not applicable for *static* data where the client does not modify her data after the initial outsourcing.

3. **Retrievability** Retrievability of the data file requires that, given an "efficient"[3] adversary $\mathcal{A}$

[3] We call an algorithm "efficient" if its running time is polynomial in the security parameter defined in Section 2.1.1.

that can pass the Audit protocol with some probability (that is not "very small"), the data file (or at least the challenged part of the file) can be retrieved "efficiently" by interacting with $\mathcal{A}$. Authenticity and freshness of data restrict the adversary $\mathcal{A}$ to produce valid responses (without storing the data in an authentic and up-to-date fashion) during these interactions only with some "very small" probability.

In proof-of-storage schemes based on POR (or PDP), the retrievability guarantee is achieved for the entire data file (or at least the challenged part of the data file). In this thesis, we often use the terms "a proof-of-storage scheme" and "a secure cloud storage" interchangeably.

We note that, due to the *freshness* property required for dynamic data, constructing an efficient proof-of-storage scheme for dynamic data is more difficult than constructing the same in the static setting. For dynamic data, the client needs an efficient mechanism to update her data file stored on the remote server in such a way that the server cannot pass an audit if it stores an older version of the client's data file. This applies to both dynamic PDP and dynamic POR schemes. Moreover, in order to ensure the retrievability of the entire file at any point of time, POR schemes typically utilize error-correcting codes to encode the data blocks into a codeword. Thus, in a dynamic POR scheme, updating a single data block requires updating several blocks in the codeword. This might make an update in a dynamic POR scheme inefficient (this is explained in Section 2.2.2.2).

Chapter 2

Preliminaries and Background

In this chapter, we discuss some preliminaries and background relevant to this thesis. A version of this chapter has been published in [136]. Some notations and tools specific to a chapter are described in the respective chapter.

2.1 Preliminaries

2.1.1 Notations

We describe the notations we use throughout this thesis in Table 2.1. An algorithm $\mathcal{A}(1^\lambda)$ is called a probabilistic polynomial-time (PPT) algorithm when its running time is polynomial in λ and its output is a random variable that depends on the internal coin tosses of $\mathcal{A}$. A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible in λ if for all positive integers c and for all sufficiently large λ, we have $f(\lambda) < \frac{1}{\lambda^c}$.

2.1.2 Erasure Codes

A $(\tilde{m}, \tilde{n}, d)_\Sigma$-erasure code is an error-correcting code [106] that comprises an encoding algorithm $\texttt{Enc}$: $\Sigma^{\tilde{n}} \to \Sigma^{\tilde{m}}$ (encodes a message consisting of $\tilde{n}$ symbols into a codeword consisting of $\tilde{m} > \tilde{n}$ symbols) and a decoding algorithm $\texttt{Dec}$: $\Sigma^{\tilde{m}} \to \Sigma^{\tilde{n}}$ (decodes a codeword to a message), where Σ is a finite alphabet of symbols and d is the minimum distance (Hamming distance between any two codewords is at least d) of the code. The quantity $\frac{\tilde{n}}{\tilde{m}}$ is called the rate of the code. A *systematic* code is one where the first $\tilde{n}$ symbols of the output of $\texttt{Enc}$ are same as the symbols of the input of $\texttt{Enc}$. A $(\tilde{m}, \tilde{n}, d)_\Sigma$-erasure code can tolerate up to $d-1$ erasures. If $d = \tilde{m}-\tilde{n}+1$, we call the code an $(\tilde{m}, \tilde{n})_\Sigma$-maximum distance separable (MDS) code. For an MDS code, the original message can be reconstructed from any $\tilde{n}$ out of $\tilde{m}$ symbols of the codeword. Reed-Solomon codes [127] and their extensions are examples of non-trivial MDS codes.

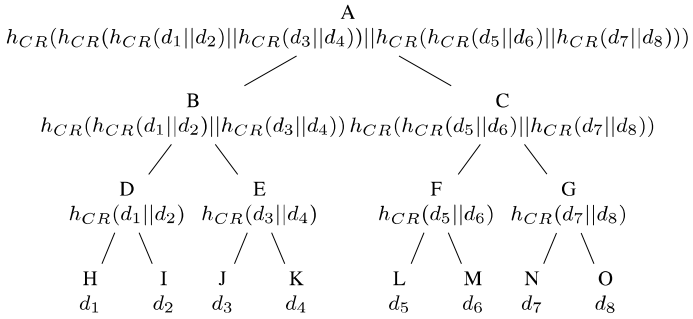
Table 2.1: Notations used

Notation	Description
λ	the security parameter
$\mathcal{A}, \mathcal{B}$	probabilistic polynomial-time (PPT) algorithms
$\mathcal{A}^{\mathcal{O}}$	$\mathcal{A}$ has an access to the oracle $\mathcal{O}$
$a \xleftarrow{R} S$	a is chosen from a set S uniformly at random
$s_1 \| s_2$	the concatenation of two strings s_1 and s_2
$[a, b]$	the set $\{a, a+1, \ldots, b\}$ for two integers a and b (where $a \leqslant b$)
d	the minimum distance of an error-correcting code
$G = \langle g \rangle$	g is a generator of the group G
$\mathbb{GF}(m)$	the finite field (Galois field) of order m
$\mathbb{F}$	a finite field
$\mathrm{v} \in \mathbb{F}^m$	a vector of dimension m over a finite field $\mathbb{F}$
$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
$\mathbb{Z}$	the set of integers
$\mathbb{Z}_m$	the set of congruence classes of integers modulo m (or the ring of integers modulo m)
$\mathbb{Z}_m^*$	the subset of $\mathbb{Z}_m$ whose elements are coprime to m
$\mathbb{F}_p$	the finite field of prime order p (or $\mathbb{GF}(p)$, or $\mathbb{Z}_p$)
sk, ssk	secret keys
pk, psk	public keys
d_M	the metadata for an authenticated data structure M
F	a file to be outsourced to a server
F'	a preprocessed file (before being outsourced to a server)
d_F	the metadata for a file F
`fid`	a file-identifier
$\bar{F}$	a set of files to be outsourced to a server
$\bar{F}'$	a set of preprocessed files (before being outsourced to a server)
$\bar{d}$	the metadata for a set of files
$\overline{\texttt{fid}}$	a set of file-identifiers

2.1.3 Pseudorandom Functions (PRFs)

A pseudorandom function [90, 73, 104] is a family of deterministic functions that take a key and an input, and a random instance of the family is computationally indistinguishable from a truly random function of the input. Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a function, where the $\mathcal{K} = \{0,1\}^\lambda$ is the key space. Then, f is a pseudorandom function if: f_k can be computed in polynomial time in λ and f_k cannot be distinguished (in polynomial time) from a random function given that k is chosen from $\mathcal{K}$ uniformly at random.

Figure 2-1: A Merkle hash tree containing an ordered list of data items $\{d_1, d_2, \ldots, d_8\}$



2.1.4 Collision-Resistant Hash Functions

A collision-resistant hash function h_{CR} takes a bit-string $s \in \{0,1\}^*$ as input and outputs a bit-string of length $O(\lambda)$. A family of functions $\mathcal{H}$ is collision-resistant if no polynomial-time (in λ) algorithm can output, except with a probability negligible in λ, two bit-strings s_1, s_2 such that $h_{CR}(s_1) = h_{CR}(s_2)$, where h_{CR} is chosen from the family $\mathcal{H}$ uniformly at random. We consider a single collision-resistant hash function (e.g., SHA-256) for practical purposes.

2.1.5 Merkle Hash Tree

A Merkle hash tree [108] is a binary tree where data items are stored in the leaf-nodes. The label of a leaf-node is the data item stored in that node. A *collision-resistant* hash function h_{CR} is used to label other nodes present in the tree. The label of an intermediate node v is the output of h_{CR} computed on the labels of its children nodes. Figure 2-1 shows a Merkle hash tree containing an ordered list of data items $\{d_1, d_2, \ldots, d_8\}$ stored at the leaf-nodes. Then, the label of each intermediate node is computed using h_{CR}. The label (hash value) of the root-node A is called the *root-digest* of the Merkle hash tree. The proof of presence of a data item d in the tree consists of d and the labels of the nodes along the *authentication path* (the sequence of siblings of the nodes present on the path from the leaf-node containing d to the root-node A). For example, a proof showing that d_3 is present in the tree consists of $\{d_3, (d_4, l_D, l_C)\}$, where d_4, l_D and l_C are the labels of the nodes K, D and C, respectively. Given such a proof, a verifier computes the label of the root-node A. The verifier outputs `accept` if the label thus computed matches with the root-digest; it outputs `reject`, otherwise. Clearly, the size of a proof is logarithmic in the number of data items stored in the leaf-nodes of the Merkle hash tree.

As the hash function h_{CR} is collision-resistant, it is infeasible (except with some probability

negligible in the security parameter λ) to generate a valid proof for a corrupted data item.

2.1.6 Bilinear Maps

Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p. Let g_1 and g_2 be generators of the groups G_1 and G_2, respectively. A bilinear map (or pairing) [93, 105, 67] is a function $e : G_1 \times G_2 \to G_T$ such that:

1. for all $u \in G_1, v \in G_2, a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$ (bilinear property),
2. e is non-degenerate, that is, $e(g_1, g_2) \neq 1$.

Furthermore, properties 1 and 2 imply that

3. for all $u_1, u_2 \in G_1, v \in G_2$, we have $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$.

If $G_1 = G_2 = G$, the bilinear map is symmetric; otherwise, it is asymmetric. Unless otherwise mentioned, we consider bilinear maps which are symmetric and efficiently computable. Let $\texttt{BLSetup}(1^\lambda)$ be an algorithm which outputs (p, g, G, G_T, e), the parameters of a bilinear map, where g is a generator of G (i.e., $G = \langle g \rangle$).

2.1.7 Discrete Logarithm Assumption

The discrete logarithm problem [107, 21] over a multiplicative group $G = \langle g \rangle$ of prime order p and generated by g is defined as follows.

Definition 2.1 (Discrete Logarithm Problem)**.** *Given $g, h \in G$, the discrete logarithm problem over G is to compute $a \in \mathbb{Z}_p$ such that $h = g^a$.*

We say that the discrete logarithm assumption holds in G if, for any probabilistic polynomial-time adversary $\mathcal{A}(1^\lambda)$, the probability

$$\Pr_{a \xleftarrow{R} \mathbb{Z}_p} [a \leftarrow \mathcal{A}(g, h) : h = g^a]$$

is negligible in λ, where the probability is taken over the internal coin tosses of $\mathcal{A}$ and the random choices of a.

2.1.8 Computational Diffie-Hellman Assumption

The computational Diffie-Hellman problem over a multiplicative group $G = \langle g \rangle$ of prime order p and generated by g is defined as follows.

Definition 2.2 (Computational Diffie-Hellman Problem)**.** *Given $g, g^a, h = g^b \in G$ for some $a, b \in \mathbb{Z}_p$, the computational Diffie-Hellman problem over G is to compute $h^a \in G$.*

We say that the computational Diffie-Hellman assumption holds in the group G if, for any probabilistic polynomial-time adversary $\mathcal{A}(1^\lambda)$, the probability

$$\Pr_{a,b \xleftarrow{R} \mathbb{Z}_p} [h^a \leftarrow \mathcal{A}(g, g^a, h = g^b)]$$

is negligible in λ, where the probability is taken over the internal coin tosses of $\mathcal{A}$ and the random choices of a and b. We note that the hardness of the computational Diffie-Hellman problem in G implies the hardness of the discrete logarithm problem in G.

2.1.9 Digital Signatures

Diffie and Hellman introduce the public-key cryptography and the notion of digital signatures in their seminal paper "New Directions in Cryptography" [51]. Rivest, Shamir and Adleman [129] propose the first digital signature scheme based on the RSA assumption. Following that, several signature schemes are proposed [57, 85, 31].

We define a digital signature scheme as proposed by Goldwasser et al. [76]. A digital signature scheme consists of the following polynomial-time algorithms: a key generation algorithm `KeyGen`, a signing algorithm `Sign` and a verification algorithm `Verify`. The algorithm `KeyGen` takes as input the security parameter λ and outputs a pair of keys (psk, ssk), where ssk is the secret key and psk is the corresponding public key (verification key). The algorithm `Sign` takes a message m from the message space $\mathcal{M}$ and the secret key ssk as input and outputs a signature σ. The algorithm `Verify` takes as input the public key psk, a message m and a signature σ, and outputs `accept` or `reject` depending upon whether the signature is valid or not. Any of these algorithms can be probabilistic in nature. A digital signature scheme has the following properties.

1. *Correctness*: The algorithm `Verify` always accepts a signature generated by an honest

signer, that is,

$$\Pr[\texttt{Verify}(psk, m, \texttt{Sign}(ssk, m)) = \texttt{accept}] = 1.$$

2. *Security*: Let $\text{Sign}_{ssk}(\cdot)$ be the signing oracle and $\mathcal{A}$ be any probabilistic polynomial-time adversary with an oracle access to $\text{Sign}_{ssk}(\cdot)$. The adversary $\mathcal{A}$ makes polynomial number of sign queries to $\text{Sign}_{ssk}(\cdot)$ for different messages and gets back the signatures on those messages. The signature scheme is secure if $\mathcal{A}$ cannot produce, except with some probability negligible in λ, a valid signature on a message not queried previously, that is, for any probabilistic polynomial-time adversary $\mathcal{A}^{\text{Sign}_{ssk}(\cdot)}$, the following probability

$$\Pr[(m, \sigma) \leftarrow \mathcal{A}^{\text{Sign}_{ssk}(\cdot)}(1^{\lambda}) : m \notin Q_s \wedge \texttt{Verify}(psk, m, \sigma) = \texttt{accept}]$$

is negligible in λ, where Q_s is the set of queries made by $\mathcal{A}$ to the signing oracle $\text{Sign}_{ssk}(\cdot)$. The probability is taken over the internal coin tosses of $\mathcal{A}$ and the random choice of ssk.

For example, we mention the algorithms of the BLS signature scheme proposed by Boneh, Lynn and Shacham [31]. Let the algorithm $\texttt{BLSetup}(1^{\lambda})$ output (p, g, G, G_T, e) as the parameters of a bilinear map, where G and G_T are multiplicative cyclic groups of prime order p, g is a generator of G and $e : G \times G \rightarrow G_T$ (see Section 2.1.6). The algorithm `KeyGen` chooses $ssk \xleftarrow{R} \mathbb{Z}_p$ as the secret key, and the public key is set to be $psk = g^{ssk}$. The algorithm `Sign` uses a full-domain hash $H : \{0,1\}^* \rightarrow G$, and it generates a signature $\sigma = H(m)^{ssk}$ on a message $m \in \{0,1\}^*$. Given a message-signature pair (m, σ), the algorithm `Verify` checks $e(\sigma, g) \stackrel{?}{=} e(H(m), psk)$. The algorithm `Verify` outputs `accept` if the equality holds; it outputs `reject`, otherwise. Security of the BLS signature scheme is derived from the computational Diffie-Hellman assumption in the random oracle model.[1]

2.1.10 Message Authentication Codes (MACs)

Message authentication codes (MACs) are symmetric-key counterpart of digital signatures. Both digital signatures and MACs are used extensively for computing authentication tags (or digests) on messages. These digests are publicly (or privately) verifiable when computed using a digital

[1] Random oracle model [22] is a model of computation that assumes the existence of a truly random function, and a cryptographic scheme is proven secure in this model assuming a cryptographic hash function used in the scheme to be a truly random function. On the other hand, the standard model (or plain model) is a model of computation where the security of a cryptographic scheme relies *only* on some complexity assumptions (e.g., discrete logarithm assumption and computational Diffie-Hellman assumption discussed in Section 2.1.7 and Section 2.1.8, respectively).

signature (or MAC) scheme. The algorithms and properties of a MAC scheme are same as those for a digital signature scheme described in Section 2.1.9, except that the signing and verification keys are the same in a MAC scheme. That is, the signer and the verifier need to share the secret key k. Some MAC constructions are based on pseudorandom functions (e.g., XOR MAC [19], CMAC [114]); some of them are based on cryptographic hash functions (e.g., HMAC [115]).

2.1.11 Homomorphic Linear Authenticators

A homomorphic linear authenticator scheme consists of four algorithms (`KeyGen`, `LinTag`, `LinAuth`, `LinVer`) [54, 70]. The algorithm `KeyGen` takes as input the security parameter λ and outputs $K = (sk, pk)$, where sk is the secret key and pk is the corresponding public key (K consists of only sk in the private verification setting). For a finite field $\mathbb{F}$ and a vector of messages $\mathrm{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{F}^n$, the algorithm $\texttt{LinTag}_{sk}(\mathrm{x})$ produces a vector of authenticators $\mathrm{s} = [\sigma_1, \sigma_2, \ldots, \sigma_n]$. Given x, s and a vector of coefficients $\mathrm{c} = [c_1, c_2, \ldots, c_n] \in \mathbb{F}^n$, the algorithm `LinAuth` computes a single authenticator σ for $\mu = \sum_{i \in [1,n]} c_i x_i$ (due to the homomorphic property, the authenticator σ corresponds to the combined message μ). The algorithm `LinVer` takes as input $\bar{K}$ (sk for private verification or pk for public verification), c, μ and σ, and it outputs `accept` or `reject` depending upon whether σ is a valid authenticator for μ or not. There are several constructions of homomorphic linear authenticators (signatures and MACs) designed for network coding [3]. Ateniese et al. [9] introduce homomorphic linear authenticators in the context of proofs of storage.

The security of a homomorphic linear authenticator scheme is defined as follows [54]. Let us consider the following unforgeability game between the challenger and the adversary $\mathcal{A}$.

1. The challenger runs $\texttt{KeyGen}(1^\lambda)$ to output $K = (sk, pk)$.
2. The adversary $\mathcal{A}$ chooses a vector of messages $\mathrm{x} \in \mathbb{F}^n$.
3. The challenger computes a vector of authenticators s by executing $\texttt{LinTag}_{sk}(\mathrm{x})$ and gives it to the adversary $\mathcal{A}$.
4. The adversary $\mathcal{A}$ produces a vector of coefficients $\mathrm{c} \in \mathbb{F}^n$ and a pair (μ', σ').
5. If $\texttt{LinVer}_{\bar{K}}(\mathrm{c}, \mu', \sigma') = \texttt{accept}$ and $\mu' \neq \sum_{i \in [1,n]} c_i x_i$, then the adversary $\mathcal{A}$ wins.

A homomorphic linear authenticator scheme is secure if, for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the unforgeability game described above is negligible in λ.

2.1.12 Bitcoin

Nakamoto introduces a peer-to-peer cryptocurrency known as Bitcoin [111] that does not rely on any trusted server. The users in the Bitcoin network make payment by digitally signing a transaction with their secret keys. The network maintains a Bitcoin blockchain (a public ledger containing valid transactions) in a distributed fashion. A new Bitcoin block[2] is appended to the Bitcoin blockchain roughly in every 10 minutes (an epoch). These Bitcoin blocks are generated by the miners (users trying to mine Bitcoins) in the network who provide a cryptographic proof-of-work to show, in order to claim a mining reward, that they have indeed expended a large amount of computational power. Presently, Bitcoin uses Back's Hashcash [14] as the proof-of-work. The mining scheme in Bitcoin involves solving a cryptographic mining puzzle that is not precomputable. Finding a solution of the puzzle works in the following way: Let $T_1, T_2, \ldots, T_z$ be some of the valid transactions for a certain epoch which are not included in any previous Bitcoin block. The miners try to find a nonce η such that SHA-256$(BH||root_{MHT}||\eta) \leqslant Z$, where Z is a predefined target value (the difficulty level), BH is the hash of the latest Bitcoin block appended to the Bitcoin blockchain and $root_{MHT}$ is the root-digest of the Merkle hash tree built over $T_1, T_2, \ldots, T_z$ (Merkle hash trees are described in Section 2.1.5). Due to the preimage-resistance property of SHA-256, the only way to compute such a nonce η is to search over all possible values of the nonce in a brute-force manner.

2.2 Proofs of Storage

A proof-of-storage protocol is a two-party protocol executed between the client (data owner) and the cloud storage server. The client can perform audits to check integrity of her data outsourced to the remote server. As we have discussed before, proof-of-storage schemes provide guarantees of either provable data possession (PDP) or proofs of retrievability (POR). There is a vast literature available on proof-of-storage schemes. We describe some of the schemes achieving PDP and POR guarantees in Section 2.2.1 and Section 2.2.2, respectively, where we mainly focus on the schemes designed for a single client uploading her data file to a single storage server. Later in Section 2.2.3 and Section 2.2.4, we discuss some proof-of-storage schemes for multiple servers (that store the client's file in a distributed fashion) and for multiple clients (who own a shared or deduplicated data file), respectively.

[2]A block of the data file considered in a proof-of-storage scheme is different from a Bitcoin block. Throughout this thesis, we explicitly mention a block as a "Bitcoin block" in the latter case; otherwise, it is a data block.

We recall that a proof-of-storage scheme is *publicly verifiable* if anyone with an access to some public parameters can perform audits. On the other hand, in *privately verifiable* schemes, only the client with some secret information can audit her data. In a publicly verifiable proof-of-storage scheme, the client can delegate the auditing task to a third party auditor (TPA) who audits the client's data and lets the client know if she finds any discrepancies. Therefore, we use the term "verifier" to denote an auditor for a proof-of-storage scheme. The client (for a privately verifiable proof-of-storage scheme) or a third party auditor (for a publicly verifiable proof-of-storage scheme) can act as the verifier. Proof-of-storage schemes are often classified based on the power of the adversary (the server) considered in these schemes. The adversary can be computationally unbounded (in *information-theoretic* setting) or a polynomial-time algorithm (in *computational* setting). The verifier can perform a limited (or unlimited) number of audits in a *bounded-use* (or *unbounded-use*) proof-of-storage scheme.

2.2.1 Provable Data Possession

Ateniese et al. [9] introduce the notion of *provable data possession* (PDP). The basic idea is as follows. The client splits the data file she wants to outsource to the cloud server into blocks. Then, the client computes an authenticator (tag) for each block of her data file and uploads the file along with the authenticators. During an audit protocol, the client asks the server for proofs (of possession) for some blocks of the file (*challenge*). The server does some computations over the challenge, stored data and authenticators in order to generate proofs (*response*). It sends the response to the client who verifies the integrity of her data based on this response.

PDP schemes guarantee the integrity of *almost all* the blocks of the data file. Although Ateniese et al. introduce provable data possession (PDP) for static data, PDP schemes dealing with dynamic data are later constructed where the client can modify her outsourced data. In Section 2.2.1.1 and Section 2.2.1.2, we discuss some of these PDP schemes for static and dynamic data that are designed for the *single-server model* where the client outsources her data file to a single storage server. Table 2.2 provides a brief overview of these schemes based on some parameters.

2.2.1.1 Provable Data Possession for Static Data

In this section, we discuss some provable data possession schemes for static data as follows.

Table 2.2: Parameters of provable data possession schemes

Provable data possession schemes	Type of outsourced data	Bounded-use/ Unbounded-use	Publicly verifiable	Privacy-preserving	Information-theoretic/ Computational audits	Security model
PDP [9]	Static	Unbounded-use	Yes	No	Computational	Random oracle
Ateniese et al. [11]	Static	Unbounded-use	Yes	No	Computational	Random oracle
Chen et al. [45]	Static	Unbounded-use	Yes	Yes	Computational	Standard
Scalable PDP [12]	Dynamic†	Bounded-use	No	No	Computational	Random oracle
DPDP I [58]	Dynamic	Unbounded-use	Yes	No	Computational	Standard
DPDP II [58]	Dynamic	Unbounded-use	Yes	No	Computational	Standard
Wang et al. [160]	Dynamic	Unbounded-use	Yes	No	Computational	Random oracle
Wang et al. [159]	Dynamic	Unbounded-use	Yes	Yes	Computational	Random oracle
FlexDPDP [60]	Dynamic	Unbounded-use	Yes	No	Computational	Standard

† Scalable PDP scheme supports (block-wise) deletion, modification and append operations; insertion of a block in an arbitrary location of the data file is not supported in this scheme.

"Provable Data Possession at Untrusted Stores" by Ateniese, Burns, Curtmola, Herring, Kissner, Peterson and Song (2007) Ateniese et al. [9, 8] introduce the notion of provable data possession (PDP) where the client audits her data outsourced to a cloud storage server in order to obtain an assurance about the integrity of the data. The client processes her data before uploading them to the remote server. This preprocessing step involves attaching tags to the data blocks. In order to compute these tags, Ateniese et al. introduce homomorphic linear authenticators (see Section 2.1.11) in the context of proofs of storage. The client stores only some metadata at her side. During an audit, the server sends a proof of (data) possession based on the outsourced data, tags and the challenge sent by the client. The client verifies the proof to detect if the server has corrupted her data. Due to the homomorphic property of these tags, the tags of the challenged blocks can be combined into a single tag that corresponds to the data block formed by combining those challenged blocks linearly. Ateniese et al. also introduce the notion of *public verifiability* where anyone having the knowledge of public parameters can audit the client's data.

"Proofs of Storage from Homomorphic Identification Protocols" by Ateniese, Kamara and Katz (2009) An identification protocol enables a prover having the knowledge of a secret key sk to prove its identity to a verifier having the knowledge of the corresponding public key pk. Ateniese et al. [11] provide a general construction of publicly verifiable homomorphic linear authenticators from any identification protocol having certain homomorphic properties. They design a publicly verifiable homomorphic linear authenticator scheme (that is secure in the random oracle model [22]) using a variant of Shoup's identification scheme based on factoring assumption [146], and they exploit this homomorphic linear authenticator scheme in order to construct a *publicly verifiable* proof-of-storage scheme.

"Secure Cloud Storage Meets with Secure Network Coding" by Chen, Xiang, Yang and Chow (2014) Network coding [3, 102] is a technique alternative to the store-and-forward routing technique used in a communication network. Each intermediate node (all nodes except the source and target nodes) in the network combines the incoming packets and outputs another packet. In a secure network coding protocol, the source node runs a procedure `TagGen` in order to authenticate all the packets to be transmitted through the network by attaching an authentication tag to each packet [2, 29, 38]. Every intermediate node runs a procedure `Combine` to generate a tag for the output packet using the tags of the incoming packets.

Chen et al. [45, 46] investigate the relation between a secure cloud storage protocol for static data and a secure network coding protocol. They show that a secure cloud storage protocol for static data can be constructed using any secure network coding protocol. Before outsourcing the data file to the server, the client splits the file into blocks and attaches authentication tags to these blocks (by using the procedure `TagGen` of the secure network coding protocol). Later she challenges the server with some random blocks. The server combines the challenged blocks into one block (and corresponding tags into a single tag) by using the procedure `Combine` of the secure network coding protocol used by an intermediate node to combine the incoming packets. As a concrete instantiation, they construct a *publicly verifiable* secure cloud storage protocol for static data based on the secure network coding protocol proposed by Catalano et al. [38].

2.2.1.2 Provable Data Possession for Dynamic Data

In this section, we discuss some provable data possession schemes for dynamic data as follows.

"Scalable and Efficient Provable Data Possession" by Ateniese, Pietro, Mancini and Tsudik (2008) The client in the scalable PDP scheme [12] precomputes t possible tokens before outsourcing her data file to the server. Each such token is generated as follows. The client chooses a random set of l blocks (using a permutation key) and a random challenge nonce. Then, the client computes a hash function on the nonce concatenated with the l data blocks, and the output of the hash is the token. She encrypts and authenticates each of these tokens using an authenticated encryption scheme and outsources them to the server. Later, for each challenge, the client matches the response of the server with the precomputed response. The scalable PDP scheme [12] is *privately verifiable*, and audits can be performed only for a predefined number of times (bounded-use). It supports (block-wise) deletion, modification and append operations. However, insertion of a data block in an arbitrary location is not supported. Thus, the scheme is *not fully dynamic*.

"Dynamic Provable Data Possession" by Erway, Küpçü, Papamanthou and Tamassia (2009) Erway et al. [58, 59] propose the first *fully dynamic* provable data possession (DPDP) scheme. They construct two *publicly verifiable* dynamic PDP schemes: `DPDP I` (based on rank-based authenticated skip lists) and `DPDP II` (based on rank-based RSA trees). Let there be a key generation algorithm that produces a public key $pk = (N, g)$, where $N = pq$ is a product of two large primes and g is an element of $\mathbb{Z}_N^*$ with large order. Suppose the initial data file F is split into n data blocks $b_1, b_2, \ldots, b_n$. For each $1 \leqslant i \leqslant n$, the client computes a tag $t_i = g^{b_i} \bmod N$ for the i-th data block. In `DPDP I`, the client builds a rank-based authenticated skip list on the tags, and then she uploads the data file, tags and the skip list to the cloud server. Each node in the rank-based authenticated skip list contains a *rank* (in an authenticated fashion) that indicates the number of leaf-nodes reachable from that particular node. This rank information ensures that the server indeed sends the proof for the i-th block (or tag) when it is challenged for that i-th block (or tag). During an audit, the skip-list proof sent by the server is verified against the root-digest of the rank-based authenticated skip list. The `DPDP II` construction is similar to `DPDP I`, except a rank-based RSA tree [120] is used as the authenticated data structure.

"Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing" by Wang, Wang, Ren, Lou and Li (2009) Wang et al. [160, 161] use the publicly verifiable POR scheme proposed by Shacham and Waters [140] (a detailed description of this scheme is given in Section 2.2.2.1) in order to provide a secure cloud storage for dynamic data. To support insertion and deletion operations efficiently, Wang et al. embed the hash of a data block instead of the hash of its index in the corresponding authenticator of the block. Authenticated updates (insertion, deletion or modification operations) on the data blocks are guaranteed by constructing a Merkle hash tree

(Section 2.1.5 provides a brief description of a Merkle hash tree) over the hashes of the data blocks. *Public verifiability* is achieved by making the root-digest of the Merkle hash tree public. During an audit, the server sends to the verifier the hash values of the challenged blocks along with their Merkle proofs, and the verifier checks their validity with respect to the public root-digest. The rest of the verification procedure is similar to that in [140]. Wang et al. extend their basic scheme to support auditing (in a *batch*) multiple data files owned by multiple clients. In order to support batch auditing, they use an aggregate signature scheme [30] based on bilinear maps.

"Privacy-Preserving Public Auditing for Secure Cloud Storage" by Wang, Chow, Wang, Ren and Lou (2011) In the publicly verifiable POR scheme proposed by Shacham and Waters [140], a third party auditor (TPA) can audit the client's data on her behalf. However, as the server sends a linear combination of the challenged blocks as a response, the TPA might learn about the (possibly sensitive) content of those data blocks (by solving a set of linear equations). Wang et al. [159, 158] introduce *privacy-preserving audits* for the publicly verifiable POR scheme of [140] such that the TPA cannot gain any knowledge about the content of the data blocks being challenged. They use a random masking technique where the server masks the linear combination of challenged data blocks with the help of a random number. This enables the TPA to audit the blocks properly but hides the actual content of those blocks from the TPA. This scheme also supports batch audits on multiple data files owned by different clients. They also handle dynamic data using a Merkle hash tree in a similar fashion as shown in [160].

"FlexDPDP: FlexList-based Optimized Dynamic Provable Data Possession" by Esiner, Kachkeev, Braunfeld, Küpçü and Özkasap (2013) Erway et al. [58] propose `DPDP I` where a rank-based authenticated skip list is built over the tags corresponding to the data blocks of fixed size. Esiner et al. [60] show that a variable size update requires changing other blocks that makes the scheme based on a rank-based authenticated skip list inefficient in practice. They introduce FlexList, a flexible length-based authenticated skip list, where a *length* information is stored in each node (instead of rank information), and this indicates the number of bytes of data reachable from that particular node. This enables searching, updating or challenging a specific block (of variable size) containing a particular byte of data — that makes it suitable for variable size updates. Esiner et al. replace the rank-based authenticated skip list in `DPDP I` [58] with a FlexList in order to construct another dynamic PDP scheme (`FlexDPDP`). Later, Esiner et al. [61] analyze `FlexDPDP` rigorously and improve its efficiency by providing optimized algorithms for FlexList.

2.2.2 Proofs of Retrievability

We have described some of the provable data possession (PDP) schemes in Section 2.2.1. However, a PDP scheme cannot convince the client about the retrievability of *all* blocks of the outsourced data file. On the other hand, proofs of retrievability (POR) assure the client that the *entire* file is stored on the server intact. The first paper introducing proofs of retrievability (POR) is by Juels and Kaliski [87]. According to Shacham and Waters [141], the underlying idea is to encode the file with an erasure code (Section 2.1.2 describes erasure codes briefly), compute an authenticator for each block of the encoded file and then upload the encoded file along with these authenticators to the cloud storage server. With this technique incorporated, the server has to delete (or modify) a considerable number of blocks to actually make a data block irretrievable; but then the server can pass an audit only with a probability negligible in the security parameter. This ensures that all blocks of the file are correctly stored at the server's end. The notion of *retrievability* is formalized by defining an extractor algorithm which can extract the entire file by interacting with a server that passes an audit with some non-negligible probability. Although Juels and Kaliski introduce proofs of retrievability for static data, POR schemes with data dynamics are proposed later. We discuss some of the POR schemes (in the single-server model) for static and dynamic data in Section 2.2.2.1 and Section 2.2.2.2, respectively. Table 2.3 provides a brief overview of these schemes based on some parameters. We recall that the adversary, in a proof-of-storage scheme, can be computationally unbounded (in information-theoretic setting) or a polynomial-time algorithm (in computational setting); and the verifier can perform a limited (or unlimited) number of audits in a bounded-use (or unbounded-use) proof-of-storage scheme.

2.2.2.1 Proofs of Retrievability for Static Data

In the static setting, the client does not modify her data files after they are outsourced to the cloud storage server. We discuss some of the POR schemes for static data as follows.

"PORs: Proofs of Retrievability for Large Files" by Juels and Kaliski (2007) Juels and Kaliski [87] propose the first POR scheme for static data. A similar scheme for online memory checking is given by Naor and Rothblum [113]. The scheme by Juels and Kaliski uses *sentinels* (random strings that are independent of the file's content), and the scheme by Naor and Rothblum uses MACs for authentication.

In the POR scheme proposed by Juels and Kaliski [87], the blocks of the encoded file F are encrypted and a large number of random elements (sentinels) are inserted in random locations of F.

Table 2.3: Parameters of proof-of-retrievability schemes

Proof-of-retrievability schemes	Type of outsourced data	Bounded-use/ Unbounded-use	Publicly verifiable	Information-theoretic/ Computational	Security model
Juels and Kaliski [87]	Static	Bounded-use	No	Computational	Standard
Shacham and Waters [140]	Static	Unbounded-use	No	Computational	Standard
	Static	Unbounded-use	Yes	Computational	Random oracle
Dodis et al. [54]	Static	Bounded-use/ Unbounded-use†	Yes/ No†	Information-theoretic/ Computational†	Standard/ Random oracle†
Bowers et al. [33]	Static	Bounded-use	No	Computational	Standard
Benabbas et al. [23]	Static	Unbounded-use	No	Computational	Standard
Xu and Chang [163]	Static	Unbounded-use	No	Computational	Standard
Armknecht et al. [7]	Static	Unbounded-use	No‡	Computational	Standard
Xu et al. [164]	Static	Unbounded-use	No‡	Computational	Standard
Iris [150]	Dynamic	Unbounded-use	No	Computational	Standard
Cash et al. [36]	Dynamic	Unbounded-use	No	Computational	Standard
Shi et al. [144]	Dynamic	Unbounded-use	No	Computational	Standard
	Dynamic	Unbounded-use	Yes	Computational	Standard
Chandran et al. [39]	Dynamic	Unbounded-use	No	Computational	Standard
Guan et al. [81]	Dynamic	Unbounded-use	Yes	Computational	Standard
Etemad and Küpçü [63]	Dynamic	Unbounded-use	Yes	Computational	Standard

† Dodis et al. [54] propose several POR constructions: privately verifiable bounded-use POR in information-theoretic setting — secure in the standard model, privately (or publicly) verifiable unbounded-use POR in computational setting — secure in the standard (or random oracle) model, privately verifiable bounded-use POR in computational setting — secure in the random oracle model.

‡ A third party auditor with some secret information (either shared by the client or generated by the auditor itself) can audit the client's data.

The server cannot distinguish between the encrypted blocks of F and the sentinels. The purpose of introducing sentinels at random locations is as follows. During an audit, the verifier (only the client can be the verifier) checks the authenticity of several sentinels at different locations. If the server modifies a considerable fraction of the blocks, a similar fraction of sentinels are modified as well (as the sentinels are inserted in random locations of F). The server cannot selectively delete non-sentinel blocks as it cannot distinguish them from sentinels. Thus, with high probability, the server cannot pass the audit. On the other hand, once the client challenges the server for some sentinel-locations, they are revealed to the server. Therefore, the future challenges must not include these locations. This makes the POR scheme bounded-use as the number of sentinels is limited.

"Compact Proofs of Retrievability" by Shacham and Waters (2008) Shacham and Waters introduce two types of homomorphic authenticators in order to construct unbounded-use POR schemes for static data [140, 141]. The first one, based on pseudorandom functions (see Section 2.1.3), provides a POR scheme which is privately verifiable (that is, only the client having the secret key can verify a proof) and secure in the standard model. The second one, based on BLS signatures (see Section 2.1.9), gives a POR scheme which is publicly verifiable (that is, anyone having the knowledge of some public parameters can verify a proof) and secure in the random oracle model [22]. We provide a more detailed description of these two schemes as several other proof-of-storage schemes discussed in this chapter are based on them.

- **POR Scheme with Private Verifiability** The client chooses $(\alpha_1, \alpha_2, \ldots, \alpha_s, k)$ as her secret key, where $\alpha_1, \alpha_2, \ldots, \alpha_s \xleftarrow{R} \mathbb{Z}_p$ and $k \xleftarrow{R} \mathcal{K}$ ($\mathcal{K}$ is the key space for a pseudorandom function). Let $f : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$ be a pseudorandom function. Let $p = \Theta(2^\lambda)$ be a large prime, where λ is the security parameter. The client encodes her data file F_0 to be outsourced to the server with an erasure code to form a file F with n blocks $m_1, m_2, \ldots, m_n$, where each block consists of s segments each of which is an element of $\mathbb{Z}_p$, that is, a segment $m_{ij} \in \mathbb{Z}_p$ for all $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant s$. For each $1 \leqslant i \leqslant n$, the client computes an authentication tag for the i-th block as

$$\sigma_i = f_k(i) + \sum_{j=1}^{s} \alpha_j m_{ij} \bmod p \tag{2.1}$$

  and uploads the file F along with the tags $\{\sigma_i\}_{1 \leqslant i \leqslant n}$ to the server.

  During an audit, the client randomly selects a set $I \subset [1, n]$ of indices and sends a challenge set $Q = \{(i, \nu_i)\}_{i \in I}$ to the server, where each coefficient $\nu_i \xleftarrow{R} \mathbb{Z}_p$. Upon receiving Q, the server computes $\sigma = \sum_{(i,\nu_i) \in Q} \nu_i \sigma_i \bmod p$ and $\mu_j = \sum_{(i,\nu_i) \in Q} \nu_i m_{ij} \bmod p$ for all $1 \leqslant j \leqslant s$. The server responds to the client with $(\sigma, \mu_1, \mu_2, \ldots, \mu_s)$. Then the client verifies

the integrity of her data by checking the verification equation

$$\sigma \stackrel{?}{=} \sum_{j=1}^{s} \alpha_j \mu_j + \sum_{(i,\nu_i)\in Q} \nu_i f_k(i) \bmod p \tag{2.2}$$

and outputs 1 or 0 depending on whether the equality holds or not. In this privately verifiable scheme, only the client can perform the verification as verifying Eqn. 2.2 requires the knowledge of the secret key $(\alpha_1, \alpha_2, \ldots, \alpha_s, k)$.

- **POR Scheme with Public Verifiability** Let there be an algorithm $\texttt{BLSetup}(1^\lambda)$ that outputs (p, g, G, G_T, e) as the parameters of a bilinear map, where G and G_T are multiplicative cyclic groups of prime order $p = \Theta(2^{2\lambda})$, g is a generator of G and $e : G \times G \to G_T$ (see Section 2.1.6). The client chooses $x \xleftarrow{R} \mathbb{Z}_p$ as her secret key. Let $\alpha_1, \alpha_2, \ldots, \alpha_s \xleftarrow{R} G$ be random elements of G and $H : \{0,1\}^* \to G$ be the BLS hash (see Section 2.1.9). The public key of the client is $pk = (\alpha_1, \alpha_2, \ldots, \alpha_s, v = g^x)$. The client encodes her data file F_0 to be outsourced to the server with an erasure code to form a file F with n blocks $m_1, m_2, \ldots, m_n$, where each block consists of s segments each of which is an element of $\mathbb{Z}_p$, that is, a segment $m_{ij} \in \mathbb{Z}_p$ for all $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant s$. The client chooses a random file-identifier $\texttt{fid} \xleftarrow{R} \mathbb{Z}_p$. For each $1 \leqslant i \leqslant n$, the client computes a tag for the i-th block as

$$\sigma_i = (H(\texttt{fid}||i) \cdot \prod_{j=1}^{s} {\alpha_j}^{m_{ij}})^x \tag{2.3}$$

and uploads the file F along with the tags $\{\sigma_i\}_{1\leqslant i\leqslant n}$ to the server.

During an audit, the verifier randomly selects a set $I \subset [1, n]$ of indices and sends a challenge set $Q = \{(i, \nu_i)\}_{i\in I}$ to the server, where each coefficient $\nu_i \xleftarrow{R} \mathbb{Z}_p$. Upon receiving Q, the server computes $\sigma = \prod_{(i,\nu_i)\in Q} {\sigma_i}^{\nu_i}$ and $\mu_j = \sum_{(i,\nu_i)\in Q} \nu_i m_{ij} \bmod p$ for all $1 \leqslant j \leqslant s$. The server responds to the verifier with $(\sigma, \mu_1, \mu_2, \ldots, \mu_s)$. Then, the verifier checks integrity of the client's data by checking the verification equation

$$e(\sigma, g) \stackrel{?}{=} e\left(\prod_{(i,\nu_i)\in Q} H(\texttt{fid}||i)^{\nu_i} \cdot \prod_{j=1}^{s} {\alpha_j}^{\mu_j}, v\right) \tag{2.4}$$

and outputs 1 or 0 depending on whether the equality holds or not. In this publicly verifiable scheme, the knowledge of pk would suffice to verify the response from the server.

"Proofs of Retrievability via Hardness Amplification" by Dodis, Vadhan and Wichs (2009) Dodis et al. [54] classify POR schemes based on the power of the adversary considered in these schemes (information-theoretic and computational) and the number of audits that can be performed (bounded-use and unbounded-use). They build the first bounded-use POR scheme in information-theoretic setting where the adversary is computationally unbounded (an adversary in the previous POR schemes [87, 140] is assumed to be a polynomial-time algorithm). Dodis et al. introduce the notion of *POR codes* based on hitting samplers [72, 74] and error-correcting codes, and they show how POR schemes can be instantiated based on these POR codes. They reduce the size of a challenge in the privately verifiable POR scheme of [140] (which is secure in the standard model) from $O(\lambda^2)$ to $O(\lambda)$ by sampling the block-indices to be challenged using a hitting sampler. We note that, for the publicly verifiable POR scheme of [140], these block-indices can be sampled by a random oracle when given a short (λ-bit) seed as input. Furthermore, Dodis et al. [54] replace the inherent Hadamard codes (for choosing the values of coefficients in the challenge set Q) in [140] with more efficient Reed-Solomon codes, thus reducing the overall size of Q to $O(\lambda)$.

"Proofs of Retrievability: Theory and Implementation" by Bowers, Juels and Oprea (2009) Bowers et al. [33] propose a theoretical framework for designing POR schemes. This framework generalizes the earlier POR schemes proposed by Juels and Kaliski [87] and Shacham and Waters [140]. Bowers et al. consider adversaries with high corruption rates and show different trade-offs for different parameter choices in this framework. They also provide a prototype implementation of a variant of the POR scheme described in [87].

"Verifiable Delegation of Computation over Large Datasets" by Benabbas, Gennaro and Vahlis (2011) Benabbas et al. [23] present an efficient verifiable computation scheme for high degree polynomials that is based on algebraic PRFs [112] (with closed form efficiency for polynomials). Here, the client outsources a polynomial of large degree to an untrusted server. Later, she wants to compute the value of the polynomial for an arbitrary input. The server sends to the client the result along with a proof of correctness of the result that the client can verify. Benabbas et al. propose a POR scheme based on this verifiable computation scheme as follows. The client encodes the data file (after preprocessing it using an erasure code) as a polynomial $F(\cdot)$ where each block of the encoded data file represents a coefficient of $F(\cdot)$. The client can delegate the computation of $F(\cdot)$ to the server. During an audit, the client chooses a random point r and sends r to the server. The server responds with the value $F(r)$ along with a proof of correctness. Finally, the client verifies if the proof corresponding to $F(r)$ is correct — the running time of this verification algorithm is sublinear in the degree of $F(\cdot)$. The POR scheme is optimal in terms of

communication bandwidth (during an audit) and client computation. However, the server has to compute over the entire data file to generate $F(r)$.

"Towards Efficient Proofs of Retrievability" by Xu and Chang (2012) As we have discussed, Dodis et al. [54] reduce the size of the challenge set in the POR scheme due to Shacham and Waters [140] from $O(\lambda^2)$ to $O(\lambda)$. Xu and Chang [163] improve the *privately verifiable* scheme of [140] by reducing the size of a response sent by the server to $O(\lambda)$ (the size of a response is $O(s\lambda)$ in [140]). To achieve this, they exploit an efficient commitment scheme for polynomials proposed by Kate et al. [89].

"Outsourced Proofs of Retrievability" by Armknecht, Bohli, Karame, Liu and Reuter (2014) Proof-of-retrievability (POR) schemes with public verifiability enable a third party auditor to audit the data file outsourced by the client (data owner). These schemes employ expensive public-key primitives (e.g., digital signatures) in order to generate authentication tags such that the auditor can check the integrity of data using the client's public key. On the other hand, proof-of-retrievability (POR) schemes with private verifiability use more efficient symmetric-key primitives (e.g., MACs) to generate tags. However, verification of a proof requires the secret key of the client.

Armknecht et al. [7] propose outsourced proofs of retrievability (OPOR) that enable the client to delegate the auditing task to a third party auditor while using a *privately verifiable* POR scheme [140]. Moreover, unlike the previous proof-of-storage schemes, any of the entities (client, auditor and server) involved in OPOR can be malicious, and any two of them can collude as well. Two sets of authentication tags (generated by the client and by the auditor using their respective secret information) along with the data file are outsourced to the server.

During an audit in OPOR, the auditor selects two challenge sets based on a (public) randomized algorithm. The randomness of this algorithm is derived from the hash value of the first block appended to the Bitcoin blockchain with respect to the current time t. The auditor performs two parallel executions of the POR protocol: one for the auditor (using the tags generated by itself) and another on behalf of the client (using the tags generated by the client). The auditor records in a log all information regarding each audit performed. The client can check the log to audit the auditor as often as needed. In case of any discrepancy, an honest auditor reveals its secret information in order to prove its honesty.

"Lightweight Delegatable Proofs of Storage" by Xu, Yang, Zhou and Wong (2016) Xu et al. [164] design a proof-of-storage scheme based on the *privately verifiable* POR scheme due to

Shacham and Waters [140], where the client (data owner) can delegate the auditing task to an auditor and later revoke a particular auditor efficiently. The client generates two sets of tags T_1 and T_2 (computed using two keys sk and vsk, respectively) on the data blocks and uploads them to the server along with the blocks. She uses T_1 to audit the data herself and keeps the corresponding key sk secret. The client shares the other key vsk with a third party auditor, and the auditor can perform audits using the corresponding set of tags T_2. In case the client wants to revoke a particular auditor, she updates the set of tags T_2 efficiently (without downloading the data blocks in order to recompute the tags). She also updates the corresponding key vsk that she can later share with a new auditor. The revoked auditor without the knowledge of the updated vsk cannot perform audits on the client's data. The set of tags T_1 is kept unchanged.

2.2.2.2 Proofs of Retrievability for Dynamic Data

In the previous section, we have described some POR schemes for static data which the clients do not modify once they are uploaded on the cloud server. In this section, we first discuss the challenges in constructing POR schemes for dynamic data where the clients can modify their outsourced data *efficiently*. Then, we describe some of the existing dynamic POR schemes.

To maintain the retrievability of the whole file, erasure codes are employed typically. The blocks of the file are encoded in such a way that the file can be retrieved from a fraction of blocks of the encoded file. The content of each block is now distributed in other $O(n)$ blocks. Therefore, to actually delete a block the server has to delete all the related blocks. This restricts the server from deleting or modifying a block maliciously and still passing the verification with non-negligible probability in λ. However, this advantage comes at the cost of expensive updates. If the client wants to update a single block, she has to update all the related blocks as well. This makes the update process inefficient as n can be very large.

Cash et al. [36] discuss two failed attempts to provide a solution of the problem mentioned above. In the first case, a possible solution might be to encode the file "locally" (i.e., the encoded data file comprises several codewords each of which consists of a small number of blocks). Therefore, an update of a single block in a codeword requires an update of only a few blocks within that particular codeword. However, a malicious server can identify this small set of blocks (within a codeword) whenever the client updates a single block. Thus, the server can delete this small set of blocks without being noticed during an audit. In the second attempt, after encoding the file locally, all of the n blocks are permuted in a pseudorandom fashion. Apparently, the server cannot get any information about the blocks in a codeword. However, during an update the server can identify the related blocks in a codeword. Therefore, the server can again delete these blocks and pass the

verification during an audit. Due to these issues, only a few POR schemes for dynamic data are available in the literature. We describe some dynamic POR schemes as follows.

"Iris: A Scalable Cloud File System with Efficient Integrity Checks" by Stefanov, van Dijk, Juels and Oprea (2012) Stefanov et al. [150] propose an authenticated file system called `Iris` in an enterprise setting where multiple clients can perform efficient reads and writes on the file system in parallel. `Iris` is highly scalable and resilient against a malicious cloud server. `Iris` ensures that the data fetched from the outsourced file system are authentic and fresh (by using MACs and balanced Merkle-tree data structures). `Iris` employs erasure codes over the file-system data blocks in order to achieve POR guarantees for dynamic data. `Iris` assumes a trusted entity called *portal* as a mediator between the enterprise and the storage component, and the communications (between an enterprise client and the cloud server) required for each operation requested by the client must go through the portal. Updating a single data block in `Iris` requires updating only a small fraction of parity blocks. However, as the portal updates the parity blocks aggregated over a long period of time, the cloud server cannot identify the mapping of a particular data block to its corresponding parity blocks. The portal caches error-correcting information and the file-system data (and metadata) accessed recently. The portal also recovers damaged data in case any data corruption is detected during an audit.

"Dynamic Proofs of Retrievability via Oblivious RAM" by Cash, Küpçü and Wichs (2013) Cash et al. [36, 37] propose a *privately verifiable* POR scheme for dynamic data using oblivious RAM [75, 122, 77]. The idea is as follows. The data file the client wants to outsource is split into data blocks, and these blocks are encoded locally using erasure codes to form codewords where each codeword consists of a small number of blocks. Thus, an update on a single block requires updating only related blocks of that particular codeword. This makes the update process much more efficient than that when all the blocks of the data file are encoded to form a single large codeword. However, the challenge is to hide the access-patterns from the server so that the server cannot identify the blocks in a codeword. Cash et al. exploit oblivious RAM (ORAM) to address this problem while keeping the update-complexity low. Oblivious RAM lets the client read the outsourced data in a pseudorandom fashion. It also hides the access-patterns while updating a block of a codeword. During an audit, the client reads blocks from random locations and checks their integrity.

"Practical Dynamic Proofs of Retrievability" by Shi, Stefanov and Papamanthou (2013) The dynamic POR scheme proposed by Cash et al. [36] uses oblivious RAM to hide the access-patterns from the (possibly malicious) server such that the server cannot identify the blocks in a codeword and delete the same without being noticed. Shi et al. [144] argue that hiding the access-patterns is not a necessary requirement for a dynamic POR scheme. According to [144], a dynamic POR scheme needs to satisfy only two properties in order to ensure the availability of the outsourced data file: *authenticated storage* (the client needs an assurance about the authenticity and freshness of her data file) and *retrievability* (the client can extract the entire data file at any point of time). Shi et al. propose a dynamic POR scheme that satisfies these two properties, and it is more efficient (in terms of the cost of computation and communication bandwidth required for performing different operations on the outsourced data) than the scheme by Cash et al. [36] as the additional cost for hiding the access-patterns is now eliminated.

Due to the use of erasure codes in POR schemes, an update in a single block requires updating $O(n)$ other blocks, where n is the total number of blocks present in the data file. To overcome this issue, the encoded copy is not updated for every write operation in [144]. Instead, it is updated (or rebuilt) only when sufficient updates are done on the data file. Thus, the amortized cost of a write operation is reduced dramatically. However, between two such rebuilds this encoded copy stores stale data. Shi et al. introduce a temporary hierarchical log structure which stores values for the intermediate writes. The scheme involves three data structures: an unencoded buffer U which is updated for every write (and blocks are read directly from U), an encoded buffer C which is updated after every n writes, and an encoded hierarchical log structure H which accommodates every block written between two successive rebuilds of C. Audits are performed on H and C.

Shi et al. construct a *privately verifiable* dynamic POR scheme using homomorphic checksums for the data blocks. Updates on these checksums are performed by the client whereas updates on data blocks in H and C are performed by the server. This reduces the communication bandwidth required for a write operation. The use of homomorphic checksums also reduces the audit-bandwidth. They also propose a *publicly verifiable* dynamic POR scheme that uses data structures (and coding techniques) similar to those used in their privately verifiable scheme. However, one or more Merkle hash trees (separate from the Merkle hash tree constructed on U) are maintained to ensure the integrity of the blocks in the hierarchical log structure H (one for the entire log, or one for each of its levels). To enable a third party auditor (TPA) to audit the blocks residing at different levels of this log, the root-digests of these Merkle hash trees are made public.

"Locally Updatable and Locally Decodable Codes" by Chandran, Kanukurthi and Ostrovsky (2014) Similar to the attempt made by Cash et al. [36], Chandran et al. [39] also

aim to reduce the update-complexity in a dynamic POR scheme by encoding the data file "locally". In order to achieve the same, they introduce the notions of "locally updatable and locally decodable codes" (LULDCs, in information-theoretic setting) and "locally updatable and locally decodable-detectable codes" (LULDDCs, in computational setting) where an update in a single bit of the message requires modifying only a few bits of the codeword (while preserving the local decodability property[3]). They construct such codes for "Prefix Hamming Metric" (given parameters δ and t, the adversary is restricted in the sense that it cannot corrupt more than a threshold δ-fraction of the t bits updated most recently). They exploit a hierarchical data structure similar to that used in oblivious RAMs [119, 75]. The i-th level of the data structure is a buffer containing 2^i elements. Each of these buffers is encoded using a locally decodable code. When a buffer becomes full during a write, all the elements are moved down to the next larger buffer. For LULDDCs, each bit of the codeword is authenticated using a constant-size MAC in order to decode the message correctly. Chandran et al. [39] propose an efficient *privately verifiable* POR scheme for dynamic data using similar techniques used in the construction of LULDDCs. Each buffer present in the hierarchical data structure used in the dynamic POR construction is encoded using linear-time encodable and decodable error-correcting codes [149], and each element of a buffer is authenticated using a constant-size MAC. Along with encoded buffers, separate unencoded buffers are maintained for efficient reads.

"Symmetric-Key Based Proofs of Retrievability Supporting Public Verification" by Guan, Ren, Zhang, Kerschbaum and Yu (2015) Guan et al. [81] propose a POR scheme using *indistinguishability obfuscation* ($i\mathcal{O}$) [15, 68]. The construction is similar to the imbalanced signature scheme proposed by Ramchen and Waters [126] where the signing operation is fast but the verification of a signature is expensive. The scheme by Guan et al. exploits the privately verifiable scheme proposed by Shacham and Waters [140] and constructs a *publicly verifiable* scheme with the help of $i\mathcal{O}$. The tag-generation phase (executed by the client) and the proof-generation phase (executed by the server) are now efficient due to the use of symmetric-key primitives (e.g., MACs). However, the verification key is the indistinguishability obfuscation of the verification program (that embeds the secret key used to generate the tags), and this expensive verification is delegated to a third party having more resources. Guan et al. also handle dynamic data using an authenticated data structure which they call a "modified B+ Merkle tree".

[3] Katz and Trevisan [91] introduce "locally decodable codes" (LDCs) where every bit of a message can be probabilistically decoded using a few bits of the corresponding codeword.

"Generic Efficient Dynamic Proofs of Retrievability" by Etemad and Küpçü (2016) Etemad and Küpçü [63] propose a generic framework for constructing a dynamic POR scheme given a black-box access to a PDP scheme for dynamic data and a POR scheme for static data. An efficient memory-checking scheme (e.g., DPDP [58]) is used for the unencoded (and up-to-date) data blocks that enables efficient read operations. These data blocks are encoded (or rebuilt) using erasure codes only after a certain number of write operations. The intermediate writes are temporarily stored in a log structure that comprises of a single (or multiple) buffer(s). The blocks in each buffer are encoded using erasure codes and an authentication tag is associated with each block (similar to a static POR scheme, e.g., [141]). Etemad and Küpçü describe different strategies to design such buffers and optimize the audit-bandwidth by aggregating all the challenged blocks (and their corresponding tags) into a single block (and a single tag). They also show that allowing the client to have some (reasonable) local storage improves the amortized cost of a write.

2.2.3 Proofs of Storage for Multiple Servers

We have discussed several provable data possession (PDP) schemes and proof-of-retrievability (POR) schemes in Section 2.2.2 and Section 2.2.1, respectively. These schemes are designed for the single-server model where the client (data owner) outsources her data file to a single storage server. However, in this model, the storage server is the single point of failure, and the client might lose the entire data file if this storage server collapses somehow. To enhance reliability and fault-tolerance, the client's data file is often disseminated across multiple servers. In case some of the servers fail, the client's data can be retrieved from the other servers. Efficiency of such a distributed cloud storage depends on several factors such as: distribution of file-shares, storage overhead (or redundancy), availability guarantee of the file, repair cost for one (or more) failed server(s), and so on. In this section, we describe some proof-of-storage schemes designed for multiple servers.

2.2.3.1 Proofs of Storage Based on Replication and Error-Correcting Codes

Replication is a popular technique to enhance reliability of data where multiple replicas of the data file are stored on multiple servers. Therefore, the original data file can be retrieved at any point of time if any of these replicas is available. Error-correcting codes (ECCs) are also used to design distributed storage systems where the data file is encoded using an error-correcting code and shares of the encoded file are distributed to different servers. Erasure codes are error-correcting codes that correct erasures (instead of correcting arbitrary corruptions) in the presence of a noisy channel (erasure codes are defined in Section 2.1.2). Error-correcting codes (e.g., Reed-Solomon

codes [127]) are often used in order to achieve storage efficiency and reliability. Several cloud storage providers employ error-correcting codes to enhance robustness against node-failures in their storage systems. Moreover, read operations on the data file are quite efficient for systematic variants of these codes.

"A Cooperative Internet Backup Scheme" by Lillibridge, Elnikety, Birrell, Burrows and Isard (2003) Lillibridge et al. [103] propose a peer-to-peer data backup technique that works in a cooperative fashion among users in the Internet. A user in the Internet distributes her backup data to other peers who store these data at their end. The original data can be reconstructed from these shares in case the user requires them. This scheme uses Reed-Solomon erasure codes in order to disperse the user's data among her peers in the Internet. Moreover, a user can attach cryptographic hashes (e.g., MACs) to the file before sending them to the peers. She can later verify the integrity of her data stored remotely by spot-checking her data based on these hashes.

"Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage" by Schwarz and Miller (2006) Schwarz and Miller [132] exploit *algebraic signatures* along with error-correcting codes to construct a secure distributed storage. As we have discussed earlier, error-correcting codes provide robustness to the distributed storage in case some of the nodes fail or go offline. The data file is split into blocks, and the client generates signatures on these data blocks. The client later checks the integrity of data blocks at random locations. Algebraic signatures (corresponding to the challenged blocks) can be aggregated into a single signature that corresponds to the aggregate block — which reduces the communication overhead during an integrity-check (as the challenged blocks and their signatures are now combined into a single block and a single signature).

"MR-PDP: Multiple-Replica Provable Data Possession" by Curtmola, Khan, Burns and Ateniese (2008) Curtmola et al. [49] extend the proof-of-storage scheme by Ateniese et al. [9] (which is a PDP scheme in the single-server model) for multiple servers. They propose a PDP scheme (`MR-PDP`) where the client generates multiple replicas of her encrypted data file and disseminates these replicas to different storage servers. The client generates these replicas from the same encrypted file by masking the file with different random values. For each of these replicas, `MR-PDP` employs the PDP scheme [9] (that is discussed in Section 2.2.1.1). During an audit, the client checks whether each of the servers has stored the exact replica that has been assigned to it.

"HAIL: A High-Availability and Integrity Layer for Cloud Storage" by Bowers, Juels and Oprea (2009) Bowers et al. [32] propose a distributed secure cloud storage scheme called HAIL (High-Availability and Integrity Layer) that achieves POR guarantees. The data blocks of a file are encoded in two steps: across multiple servers (dispersal code) and within each server (server code). HAIL considers strong adversarial model and enjoys several benefits such as: high reliability, low per-server computation and bandwidth (that is comparable to single-server POR schemes). Moreover, for a dispersal codeword, message authentication codes (MACs) are embedded in the parity blocks themselves — that reduces the storage overhead on the servers.

After the client initially uploads data file F to the servers, the lifetime of the system is split into some time intervals called *epochs*. For a parameter b, the adversary $\mathcal{A}$ is modeled as *mobile* (i.e., in each epoch it can corrupt up to b servers chosen arbitrarily) and fully *Byzantine* (i.e., it can arbitrarily modify or delete any part of the storage in any server it corrupts). An epoch consists of three phases: a *corruption* phase ($\mathcal{A}$ chooses arbitrarily a set of b servers to corrupt), an *audit* phase (the client performs integrity-checks on data stored on each server via spot-checking) and a *remediation* phase (the client checks if some corrupted servers provide incorrect responses above a certain threshold fraction ϵ). In the remediation phase, if the fraction of corruptions exceeds ϵ for some server, the client reads all the file-shares from each server and tries to decode F. If the decoding procedure succeeds, the client computes new file-shares and redistributes these shares to the respective servers. Otherwise, the data file becomes unavailable.

"Cooperative Provable Data Possession for Integrity Verification in Multicloud Storage" by Zhu, Hu, Ahn and Yu (2012) Zhu et al. [166] propose "cooperative PDP" where the client's data blocks (and corresponding authentication tags) are disseminated among various cloud service providers (CSPs). However, the client need not know the internal storage structure of CSPs in order to process her data file before uploading the same. One of the CSPs (which is called the "organizer") interacts with other CSPs to store the data file properly. The storage architecture is hierarchical having different layers of abstraction. The organizer communicates with the client and allocates respective file-shares to other CSPs.

During an audit, the organizer combines the responses from individual CSPs and sends a single response to the client (this aggregation is possible due to the homomorphic property of the authentication tags used in this scheme). Thus, the verification procedure does not require the details of data storage that reduces the burden on the client. Moreover, this construction satisfies the zero-knowledge property in the sense that the privacy of data blocks (and that of authentication tags) is preserved in presence of a third party auditor. Although the storage structure is oblivious to the

client in this scheme, the client has to know a priori the number of CSPs that would store her data. For each data block, the client generates an authentication tag that is based on the particular CSP which would store the block (and the corresponding authentication tag).

Later, Etemad and Küpçü [62] propose a transparent, distributed and replicated dynamic PDP scheme (`DR-DPDP`) based on [58]. In DR-DPDP, the "organizer" decides over the number of servers and manages the distribution of partitions over these servers (and also the parameters for replication). This architecture is transparent to the client and can be changed on-the-fly at any point of time. Moreover, the client need not perform computations depending on the storage architecture.

2.2.3.2 Proofs of Storage Based on Network Coding

A computer network consists of sender (or source) nodes, receiver (or target) nodes and intermediate nodes (or routers). Sender nodes send packets to the target nodes through the network. In the conventional store-and-forward routing, intermediate nodes copy the incoming packets and send them to their respective neighbors. Ahlswede et al. [3] introduce network coding as an alternative to the store-and-forward routing in a network. Here, each intermediate node has computational power, and it encodes the incoming packets to form the output packets. Ahlswede et al. [3] show that the throughput of such a network is optimal in case of multicasting (where packets are sent to a group of nodes simultaneously). In linear network coding [102], each intermediate node in the network outputs a linear combination of the packets it receives. These packets typically are vectors over a finite field $\mathbb{F}$. In random (linear) network coding [83, 84], the coefficients of these linear combinations are chosen uniformly at random from the underlying field $\mathbb{F}$. When the receiver (or target) node accumulates sufficient number of linearly independent packets, it solves a system of linear equations to decode the original file sent by the sender node.

In case a storage node fails in a distributed storage system, a new node replaces the failed node. The amount of communication bandwidth required between the new node and the healthy storage nodes to populate the new node is known as "repair bandwidth". One of the most important challenges in a distributed storage system is to reduce the repair bandwidth. Dimakis et al. [52] introduce network coding in distributed storage systems in order to reduce repair bandwidth. They show that these constructions can achieve a significant reduction in repair bandwidth compared to using Reed-Solomon or other existing error-correcting codes. However, network coding is not systematic — that makes read operations expensive. Thus, these storage systems are mostly suitable for archival data where repairs take place more often than reads. For such a distributed storage system based on network coding, there are proof-of-storage schemes for checking integrity of the remotely stored data. We describe some of them as follows.

"Remote Data Checking for Network Coding-Based Distributed Storage Systems" by Chen, Curtmola, Ateniese and Burns (2010) Chen et al. [44] aim to minimize the combined costs of integrity-checking and repair for a network coding-based distributed storage system. They identify some challenges in constructing such a storage system for *static* data and propose a scheme called `RDC-NC` (remote data checking for network coding) that overcomes these challenges. They employ two verification tags for each coded block of the data file: a challenge tag (in order to check data integrity) and a repair tag (in order to make each repair secure). To prevent a malicious server from storing the same data blocks as stored by some other server (that has the potential to make the file unrecoverable at some point of time without the client's knowledge), the index of each block is embedded into the verification tags corresponding to that block. During a repair, the client uses the repair tags to verify if the blocks of the new server are formed using correct linear combinations of relevant blocks from other healthy servers.

"NC-Audit: Auditing for Network Coding Storage" by Le and Markopoulou (2012) Le and Markopoulou [98] propose `NC-Audit`, a secure distributed cloud storage for *dynamic* data based on network coding. In addition to checking data-integrity and supporting efficient repair of failed nodes, `NC-Audit` also enables the client to update (insert, delete and modify) the outsourced data. Moreover, a third party auditor auditing the client's data gains no knowledge of the content of the client's data. The design of `NC-Audit` is based on `SpaceMac` [97], a homomorphic MAC scheme for network coding, and `NCrypt`, an encryption scheme compatible with `SpaceMac` [98].

"DD-POR: Dynamic Operations and Direct Repair in Network Coding-Based Proof of Retrievability" by Omote and Phuong (2015) Omote and Phuong [116, 117] propose the `DD-POR` (`D2-POR`) scheme for *dynamic* data based on network coding where the client can check the integrity of her data file with the help of authentication tags attached to the data blocks. During a repair, the healthy servers send the coded blocks and tags directly to the new server without any intervention of the client. Unlike `NC-Audit` [98], the new server in `DD-POR` can verify authenticity of the incoming blocks (using the tags associated with them) and discard corrupted blocks. Finally, the new server computes the coded blocks (and tags) that it would store. `DD-POR` uses `InterMac` [99], a multi-source homomorphic MAC scheme, to enable the (possibly untrusted) new server to verify incoming blocks and to compute the coded blocks (and tags) in an authenticated fashion. Moreover, the client can update (insert, delete and modify) her data after the initial outsourcing.

2.2.4 Proofs of Storage for Multiple Data Owners

2.2.4.1 Proofs of Storage for Shared Data

In an enterprise setting, a group of cloud users often outsources to a cloud storage server the data relevant to its members. The members can read and write the shared data. Similar to a proof-of-storage scheme, each data block is authenticated by a tag generated using the secret key of a user. For a publicly verifiable auditing scheme for shared data, these tags are typically signatures computed using the secret keys of the users, and these tags can be verified by a third party auditor (TPA) using the corresponding public keys. Now, given a public key, the TPA can easily identify the user signing a particular block — which enables the TPA to derive private information of the group (e.g., if a particular user performs more updates than others, she might be an important member of the group).

"Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud" by Wang, Li and Li (2012) To overcome the issue mentioned above, Wang et al. [156, 157] propose `Oruta`, a privacy-preserving public auditing scheme for shared data. They construct a new homomorphic authenticable ring signature scheme based on an existing ring signature scheme proposed by Boneh et al. [30], and they use these signatures to authenticate the blocks of the shared data file. This enables the TPA to check the integrity of the shared data without being able to learn the identity of the signer from the authenticator of a data block.

"Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud" by Wang, Li and Li (2012) Ring signatures used in `Oruta` are not suitable for large groups as the size of signatures and the verification time grow linearly with the number of members in a group. Moreover, the identity of a signer cannot be traced in case any discrepancy. Wang et al. [155] propose `Knox`, another auditing scheme for shared data, that is based on homomorphic authenticable group signatures (these signatures are derived from [28, 31]). Both the size of signatures and the verification time are independent of the size of a group. The group manager (the initial owner of the data file) can reveal the identity of the signer if necessary. However, public auditing is not supported in `Knox`.

"Storing Shared Data on the Cloud via Security-Mediator" by Wang, Chow, Li and Li (2013) In another work, Wang et al. [154] introduce an entity called *security-mediator* who generates tags (using its secret key) on the blocks of the data file shared among a group of users. As the tags

are generated by the mediator for all users, the cloud server cannot distinguish the users uploading (or updating) the file. Audits are performed using the public key of the mediator. Moreover, in order to refrain the mediator from knowing the content of data blocks, Wang et al. employ blind signatures [43]. In this case, a user sends a blinded data block to the mediator, and the mediator signs on this blinded block. After receiving the signature, the user unblinds the same to obtain the real signature.

2.2.4.2 Proofs of Storage for Deduplicated Data

Deduplication is a technique that several commercial cloud storage servers often adopt in order to save space by storing a single copy of a particular file uploaded by multiple clients (data owners) [145]. However, deduplication on an encrypted file requires the respective clients to agree on the same encryption key (otherwise, the server might not detect if two files encrypted using different encryption keys are identical). Convergent encryption [55] and message-locked encryption [20] provide a way to extract (deterministically) the same encryption key from the file itself.

"Message-Locked Proofs of Retrievability with Secure Deduplication" by Vasilopoulos, Önen, Elkhiyaoui and Molva (2016) Similar to the possibility of different encryption keys for different owners of a data file, the set of authentication tags for the file, in an auditable deduplicated cloud storage system, need not be the same for all of its owners (since those tags are generated using respective secret keys of the owners). A naive way is to store the all of these sets of tags which might incur a huge storage overhead at the server side. Vasilopoulos et al. [153] propose *message-locked* POR based on any secure message-locked key generation technique, where the secret keys used for encrypting the data file (and generating the tags for the file) are extracted from the file itself. They employ a key server with an assumption that the key server and the storage server do not collude. The key server helps the client to generate the secret keys such that the storage server cannot mount dictionary attacks (where the storage server derives encryption keys from potential candidates of a data file, encrypts them with respective keys and checks whether any of these encryptions matches with the given encrypted file).

"Sharing Proofs of Retrievability across Tenants" by Armknecht, Bohli, Froelicher and Karame (2017) Armknecht et al. [6] propose a POR scheme for a data file F owned by multiple clients in a deduplicated storage system. They exploit the key-homomorphism property of BLS signatures [31] used in the publicly verifiable POR scheme of [140] — authentication tags for each data block generated using different secret keys can be multiplied to get a single tag for

that block (and the corresponding public keys are multiplied to obtain a single public key pk_F that can be used for checking integrity of F). A new owner of an existing file F computes tags for data blocks using her secret key sk and sends them to the server along with Π, a proof of possession of the corresponding public key pk. The server verifies whether Π and the tags are valid. If they are valid, the server includes (pk, Π) in a list `pklog` of owners of F and updates pk_F by multiplying its existing value with pk. For each data block present in F, the server updates the corresponding tag by multiplying its existing value with the new tag. During an audit, an owner of F checks the validity of the entries in `pklog` (sent by the server) and runs an integrity-check using the up-to-date public key pk_F.

2.2.5 Other Schemes for Checking Integrity of Remote Data

Although there are several proof-of-storage schemes available in the literature, we have briefly described only some of them in the current chapter. We conclude this chapter by mentioning some other schemes for checking integrity of remote data as follows.

Deswarte et al. [50] propose two schemes for checking integrity of remote data that are based on challenge-response protocols. Shah et al. [142] propose an auditing scheme for data maintained by online storage services, where an auditor precomputes a list of challenge-response pairs in order to check integrity of data during an audit. Zeng [165] constructs a pairing-based provable data integrity (PDI) scheme that achieves public verifiability. Chang and Xu [40] propose a security model for *remote integrity check* (RIC) where the extractor algorithm has complete access to the server storage (unlike PDP/POR schemes where the extractor can access the server storage only via audits). Paterson et al. [121] interpret the extractors for POR schemes from a coding-theoretic perspective in presence of an adversary that has unbounded computational power.

Ren et al. [128] propose a dynamic proof-of-storage scheme for multiple storage servers, where the data file is split into data blocks and each of these data blocks is encoded using intra-server (erasure coding) and inter-server (network coding) redundancy. An update in a block requires changing only a few codeword symbols. Moreover, the inter-server redundancy achieved using network coding reduces the repair bandwidth required in case any of the servers fails. Armknecht et al. [5] propose `Mirror`, a POR scheme that also provides proofs of data replication. The client (data owner) often prefers to have multiple replicas of her data stored on the cloud server. `Mirror` enables the client to detect, with significant probability, whether the server has stored all the replicas. On the other hand, a dishonest client cannot upload different files on the server by claiming that they are replicas of the same file encrypted with different keys.

He et al. [82] construct `DeyPoS`, a proof-of-storage scheme for dynamic data in a multi-user setting, that achieves secure cross-user deduplication (where a user or data owner proves the ownership of a file already present in the cloud server and thus can skip the uploading process for the file). Wang et al. [162] introduce an *identity-based data outsourcing* (IBDO) system, where all the parties involved in the system are recognized by their respective identities. A client (data owner) can authorize a proxy to outsource some of her data files (specified by the client) to the untrusted cloud storage server. This delegation cannot be misused by any unauthorized third party. Moreover, an authorized proxy cannot outsource other unspecified files on the client's behalf. In this system, the origin and type (along with the integrity) of the outsourced data files are publicly auditable. An auditor can efficiently check integrity of different data files uploaded by multiple clients.

Ateniese et al. [10] propose an *accountable storage* (AS) protocol where the server is accountable for a loss of the client's data. The client initially outsources to the server a set of n data blocks, and she can later provably compute the number of bits the server has corrupted. Moreover, the AS protocol can be integrated with Bitcoin [111] in order to automatically compensate the client with Bitcoins proportional to the damage. Gorke et al. [80] introduce the notion of *proofs of recoverability*. In case a POR protocol fails, the client is able to detect whether the outsourced file is recoverable at all (without downloading the damaged file). In a recent work, Boyd et al. [34] provide generic security models for achieving confidentiality and integrity in an encrypted (and possibly deduplicated) cloud storage.

Bibliography

[1] 1e96a1b27a6cb85df68d728cf3695b0c46dbd44d. Filecoin: A cryptocurrency operated file storage network, July 2014. https://filecoin.io/filecoin-jul-2014.pdf.

[2] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *International Conference on Applied Cryptography and Network Security, ACNS 2009*, pages 292–305, 2009.

[3] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[4] Amazon. Amazon S3. http://aws.amazon.com/s3/.

[5] Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O. Karame. Mirror: Enabling proofs of data replication and retrievability in the cloud. In *USENIX Security Symposium 2016*, pages 1051–1068, 2016.

[6] Frederik Armknecht, Jens-Matthias Bohli, David Froelicher, and Ghassan Karame. Sharing proofs of retrievability across tenants. In *ACM Asia Conference on Computer and Communications Security, ASIACCS 2017*, pages 275–287, 2017.

[7] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O. Karame, Zongren Liu, and Christian A. Reuter. Outsourced proofs of retrievability. In *ACM Conference on Computer and Communications Security, CCS 2014*, pages 831–843, 2014.

[8] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security*, 14(1):12:1–12:34, 2011.

[9] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security, CCS 2007*, pages 598–609, 2007.

[10] Giuseppe Ateniese, Michael T. Goodrich, Vassilios Lekakis, Charalampos Papamanthou, Evripidis Paraskevas, and Roberto Tamassia. Accountable storage. In *International Conference on Applied Cryptography and Network Security, ACNS 2017*, pages 623–644, 2017.

[11] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *Advances in Cryptology - ASIACRYPT 2009*, pages 319–333, 2009.

[12] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *International Conference on Security and Privacy in Communication Networks, SECURECOMM 2008*, page 9, 2008.

[13] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *International Conference on Practice and Theory in Public Key Cryptography, PKC 2011*, pages 17–34, 2011.

[14] Adam Back. Hashcash - a denial of service counter-measure, August 2002. http://www.hashcash.org/papers/hashcash.pdf.

[15] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001*, pages 1–18, 2001.

[16] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 1997.

[17] Paulo S.L.M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *International Conference on Selected Areas in Cryptography, SAC 2006*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin Heidelberg, 2006.

[18] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology - CRYPTO 1994*, pages 216–233, 1994.

[19] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology - CRYPTO 1995*, pages 15–28, 1995.

[20] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology - EUROCRYPT 2013*, pages 296–312, 2013.

[21] Mihir Bellare and Phillip Rogaway. Number-theoretic primitives. https://cseweb.ucsd.edu/~mihir/cse207/w-ntp.pdf.

[22] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security, CCS 1993*, pages 62–73, 1993.

[23] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Advances in Cryptology - CRYPTO 2011*, pages 111–131, 2011.

[24] Elwyn R. Berlekamp. Fast Fourier transform. http://math.berkeley.edu/~berlek/classes/CLASS.110/LECTURES/FFT.

[25] Daniel J. Bernstein and Tanja Lange. eBASH: ECRYPT benchmarking of all submitted hashes. http://bench.cr.yp.to/ebash.html.

[26] Daniel J. Bernstein and Tanja Lange. eBATS: ECRYPT benchmarking of asymmetric systems. https://bench.cr.yp.to/ebats.html.

[27] Alex Biryukov and Ivan Pustogarov. Proof-of-work as anonymous micropayment: Rewarding a Tor relay. In *International Conference on Financial Cryptography and Data Security, FC 2015*, pages 445–455, 2015.

[28] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2004.

[29] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *International Conference on Practice and Theory in Public Key Cryptography, PKC 2009*, pages 68–87, 2009.

[30] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer Berlin Heidelberg, 2003.

[31] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[32] Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *ACM Conference on Computer and Communications Security, CCS 2009*, pages 187–198, 2009.

[33] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *ACM Cloud Computing Security Workshop, CCSW 2009*, pages 43–54, 2009.

[34] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, Mohsen Toorani, and Håvard Raddum. Security notions for cloud storage and deduplication. *IACR Cryptology ePrint Archive*, 2017:1208, 2017.

[35] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium, NDSS 2014*, 2014.

[36] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. In *Advances in Cryptology - EUROCRYPT 2013*, pages 279–295. Springer Berlin Heidelberg, 2013.

[37] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. *Journal of Cryptology*, 30(1):22–57, 2017.

[38] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *International Conference on Practice and Theory in Public Key Cryptography, PKC 2012*, pages 680–696, 2012.

[39] Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In *Theory of Cryptography Conference, TCC 2014*, pages 489–514, 2014.

[40] Ee-Chien Chang and Jia Xu. Remote integrity check with dishonest storage server. In *European Symposium on Research in Computer Security, ESORICS 2008*, pages 223–237, 2008.

[41] Hsi-Cheng Chang and Chiun-Chieh Hsu. Using topic keyword clusters for automatic document clustering. *IEICE Transactions*, 88-D(8):1852–1860, 2005.

[42] Denis Xavier Charles, Kamal Jain, and Kristin E. Lauter. Signatures for network coding. *International Journal of Information and Coding Theory*, 1(1):3–14, 2009.

[43] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology - CRYPTO 1982*, pages 199–203, 1982.

[44] Bo Chen, Reza Curtmola, Giuseppe Ateniese, and Randal C. Burns. Remote data checking for network coding-based distributed storage systems. In *ACM Cloud Computing Security Workshop, CCSW 2010*, pages 31–42, 2010.

[45] Fei Chen, Tao Xiang, Yuanyuan Yang, and Sherman S. M. Chow. Secure cloud storage meets with secure network coding. In *IEEE International Conference on Computer Communications, INFOCOM 2014*, pages 673–681, 2014.

[46] Fei Chen, Tao Xiang, Yuanyuan Yang, and Sherman S. M. Chow. Secure cloud storage meets with secure network coding. *IEEE Transactions on Computers*, 65(6):1936–1948, 2016.

[47] Chitchanok Chuengsatiansup, Michael Naehrig, Pance Ribarski, and Peter Schwabe. PandA: Pairings and arithmetic. In *International Conference on Pairing-Based Cryptography, Pairing 2013*, volume 8365 of *Lecture Notes in Computer Science*, pages 229–250. Springer International Publishing, 2014.

[48] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security, CCS 2006*, pages 79–88, 2006.

[49] Reza Curtmola, Osama Khan, Randal C. Burns, and Giuseppe Ateniese. MR-PDP: Multiple-replica provable data possession. In *IEEE International Conference on Distributed Computing Systems, ICDCS 2008*, pages 411–420, 2008.

[50] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote integrity checking: How to trust files stored on untrusted servers. In *Conference on Integrity and Internal Control in Information Systems, IICIS 2003*, pages 1–11, 2003.

[51] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[52] Alexandros G. Dimakis, Brighten Godfrey, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. In *IEEE International Conference on Computer Communications, INFOCOM 2007*, pages 2000–2008, 2007.

[53] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[54] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography Conference, TCC 2009*, pages 109–127, 2009.

[55] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *IEEE International Conference on Distributed Computing Systems, ICDCS 2002*, pages 617–624, 2002.

[56] Dropbox. Dropbox. https://www.dropbox.com/.

[57] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin Heidelberg, 1985.

[58] C. Christopher Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *ACM Conference on Computer and Communications Security, CCS 2009*, pages 213–222, 2009.

[59] C. Christopher Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security*, 17(4):15:1–15:29, 2015.

[60] Ertem Esiner, Adilet Kachkeev, Samuel Braunfeld, Alptekin Küpçü, and Öznur Özkasap. FlexDPDP: FlexList-based optimized dynamic provable data possession. *IACR Cryptology ePrint Archive*, 2013:645, 2013.

[61] Ertem Esiner, Alptekin Küpçü, and Öznur Özkasap. Analysis and optimization on FlexDPDP: A practical solution for dynamic provable data possession. In *International Conference on Intelligent Cloud Computing, ICC 2014*, pages 65–83, 2014.

[62] Mohammad Etemad and Alptekin Küpçü. Transparent, distributed, and replicated dynamic provable data possession. In *International Conference on Applied Cryptography and Network Security, ACNS 2013*, pages 1–18, 2013.

[63] Mohammad Etemad and Alptekin Küpçü. Generic efficient dynamic proofs of retrievability. In *ACM Cloud Computing Security Workshop, CCSW 2016*, pages 85–96, 2016.

[64] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security, FC 2014*, pages 436–454, 2014.

[65] Jon Fingas. Amazon outage breaks large parts of the internet, February 2017. https://www.engadget.com/2017/02/28/amazon-aws-outage/.

[66] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[67] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, September 2008.

[68] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 40–49, 2013.

[69] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *International Conference on Practice and Theory in Public Key Cryptography, PKC 2010*, pages 142–160, 2010.

[70] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *Advances in Cryptology - ASIACRYPT 2013*, pages 301–320, 2013.

[71] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, page 216, 2003.

[72] Oded Goldreich. A sample of samplers - A computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(20), 1997.

[73] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[74] Oded Goldreich. A sample of samplers: A computational perspective on sampling. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 302–332. 2011.

[75] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[76] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[77] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *International Colloquium on Automata, Languages and Programming, ICALP 2011*, pages 576–587, 2011.

[78] Michael T. Goodrich, Roberto Tamassia, and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *DARPA Information Survivability Conference and Exposition (DISCEX) II*, pages 68–82, 2001.

[79] Google. Google Drive. https://www.google.com/drive/.

[80] Christian A. Gorke, Christian Janson, Frederik Armknecht, and Carlos Cid. Cloud storage file recoverability. In *ACM International Workshop on Security in Cloud Computing, SCC@ASIACCS 2017*, pages 19–26, 2017.

[81] Chaowen Guan, Kui Ren, Fangguo Zhang, Florian Kerschbaum, and Jia Yu. Symmetric-key based proofs of retrievability supporting public verification. In *European Symposium on Research in Computer Security, ESORICS 2015*, pages 203–223, 2015.

[82] Kun He, Jing Chen, Ruiying Du, Qianhong Wu, Guoliang Xue, and Xiang Zhang. DeyPoS: Deduplicatable dynamic proof of storage for multi-user environments. *IEEE Transactions on Computers*, 65(12):3631–3645, 2016.

[83] Tracey Ho, Ralf Koetter, Muriel Médard, David R. Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory, ISIT 2003*, page 442, 2003.

[84] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[85] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[86] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[87] Ari Juels and Burton S. Kaliski. PORs: Proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security, CCS 2007*, pages 584–597, 2007.

[88] Seung-Shik Kang. Keyword-based document clustering. In *International Workshop on Information Retrieval with Asian Languages, IRAL 2003*, pages 132–137, 2003.

[89] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, 2010.

[90] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[91] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *ACM Symposium on Theory of Computing, STOC 2000*, pages 80–86, 2000.

[92] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work, July 2013. http://primecoin.io/bin/primecoin-paper.pdf.

[93] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. In *IMA International Conference on Cryptography and Coding*, pages 13–36, 2005.

[94] Maxwell N. Krohn, Michael J. Freedman, and David Mazières. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy, S&P 2004*, pages 226–240, 2004.

[95] Alptekin Küpçü. Official arbitration with secure cloud storage application. *The Computer Journal*, 58(4):831–852, 2015.

[96] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency, June 2013. https://tools.ietf.org/html/rfc6962.

[97] Anh Le and Athina Markopoulou. Locating Byzantine attackers in intra-session network coding using SpaceMac. In *IEEE International Symposium on Network Coding, NetCod 2010*, pages 1–6, 2010.

[98] Anh Le and Athina Markopoulou. NC-Audit: Auditing for network coding storage. In *IEEE International Symposium on Network Coding, NetCod 2012*, pages 155–160, 2012.

[99] Anh Le and Athina Markopoulou. On detecting pollution attacks in inter-session network coding. In *IEEE International Conference on Computer Communications, INFOCOM 2012*, pages 343–351, 2012.

[100] Dave Lee. Google docs offline for 'significant' number of users, November 2017. http://www.bbc.com/news/technology-42006495.

[101] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis, 2015. http://www.cs.huji.ac.il/~yoadlew/bitcoin.pdf.

[102] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[103] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Michael Burrows, and Michael Isard. A cooperative internet backup scheme. In *USENIX Annual Technical Conference, ATC 2003*, pages 29–41, 2003.

[104] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[105] Ben Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, June 2007. https://crypto.stanford.edu/pbc/thesis.pdf.

[106] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 1977.

[107] Kevin S. McCurley. The discrete logarithm problem. *Proceedings of Symposia in Applied Mathematics*, 42:49–74, 1990.

[108] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO 1987*, pages 369–378, 1987.

[109] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing Bitcoin work for data preservation. In *IEEE Symposium on Security and Privacy, S&P 2014*, pages 475–490, 2014.

[110] Bodo Möller. Algorithms for multi-exponentiation. In *International Conference on Selected Areas in Cryptography, SAC 2001*, pages 165–180, 2001.

[111] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://bitcoin.org/bitcoin.pdf.

[112] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *IEEE Symposium on Foundations of Computer Science, FOCS 1997*, pages 458–467, 1997.

[113] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. In *IEEE Symposium on Foundations of Computer Science, FOCS 2005*, pages 573–584, 2005.

[114] NIST. Recommendation for block cipher modes of operation: The CMAC mode for authentication, May 2005. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.

[115] NIST. The keyed-hash message authentication code (HMAC), July 2008. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

[116] Kazumasa Omote and Tran Thao Phuong. DD-POR: Dynamic operations and direct repair in network coding-based proof of retrievability. In *International Computing and Combinatorics Conference, COCOON 2015*, pages 713–730, 2015.

[117] Kazumasa Omote and Tran Thao Phuong. D2-POR: Direct repair and dynamic operations in network coding-based proof of retrievability. *IEICE Transactions on Information and Systems*, 99-D(4):816–829, 2016.

[118] OpenSSL. Cryptography and SSL/TLS toolkit. https://www.openssl.org/.

[119] Rafail Ostrovsky. Efficient computation on oblivious rams. In *ACM Symposium on Theory of Computing, STOC 1990*, pages 514–523, 1990.

[120] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In *ACM Conference on Computer and Communications Security, CCS 2008*, pages 437–448, 2008.

[121] Maura B. Paterson, Douglas R. Stinson, and Jalaj Upadhyay. A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. *Journal of Mathematical Cryptology*, 7(3):183–216, 2013.

[122] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Advances in Cryptology - CRYPTO 2010*, pages 502–519, 2010.

[123] James S. Plank, Jianqiang Luo, Catherine D. Schuman, Lihao Xu, and Zooko Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *USENIX Conference on File and Storage Technologies, FAST 2009*, pages 253–265, 2009.

[124] James S. Plank and Lihao Xu. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In *IEEE International Symposium on Network Computing and Applications, NCA 2006*, pages 173–180, 2006.

[125] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[126] Kim Ramchen and Brent Waters. Fully secure and fast signing from obfuscation. In *ACM Conference on Computer and Communications Security, CCS 2014*, pages 659–673, 2014.

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

[128] Zhengwei Ren, Lina Wang, Qian Wang, and Mingdi Xu. Dynamic proofs of retrievability for coded cloud storage systems. *IEEE Transactions on Services Computing*, PP(99), 2015. DOI: 10.1109/TSC.2015.2481880.

[129] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, feb 1978.

[130] Meni Rosenfeld. Analysis of Bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011.

[131] Mark Dermot Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *Network and Distributed System Security Symposium, NDSS 2014*, 2014.

[132] Thomas J. E. Schwarz and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *IEEE International Conference on Distributed Computing Systems, ICDCS 2006*, page 12, 2006.

[133] Binanda Sengupta, Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. Retricoin: Bitcoin based on compact proofs of retrievability. In *International Conference on Distributed Computing and Networking, ICDCN 2016*, pages 14:1–14:10, 2016.

[134] Binanda Sengupta, Akanksha Dixit, and Sushmita Ruj. Secure cloud storage with data dynamics using secure network coding. *PrePrint*, 2018.

[135] Binanda Sengupta, Nishant Nikam, Sushmita Ruj, Srinivasan Narayanamurthy, and Siddhartha Nandi. An efficient secure distributed cloud storage for append-only data. In *IEEE International Conference on Cloud Computing, CLOUD 2018*, pages 146–153, 2018.

[136] Binanda Sengupta and Sushmita Ruj. Cloud data auditing using proofs of retrievability. In *Guide to Security Assurance for Cloud Computing*, pages 193–210. Springer International Publishing, 2015.

[137] Binanda Sengupta and Sushmita Ruj. Publicly verifiable secure cloud storage for dynamic data using secure network coding. In *ACM Asia Conference on Computer and Communications Security, ASIACCS 2016*, pages 107–118, 2016.

[138] Binanda Sengupta and Sushmita Ruj. Efficient proofs of retrievability with public verifiability for dynamic cloud storage. *IEEE Transactions on Cloud Computing*, PP(99), 2017. DOI: 10.1109/TCC.2017.2767584.

[139] Binanda Sengupta and Sushmita Ruj. Keyword-based delegable proofs of storage. In *International Conference on Information Security Practice and Experience, ISPEC*, 2018.

[140] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Advances in Cryptology - ASIACRYPT 2008*, pages 90–107, 2008.

[141] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.

[142] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *USENIX Workshop on Hot Topics in Operating Systems, HotOS 2007*, 2007.

[143] Simon Sharwood. Fujitsu's Australian cloud suffers storage crash, outage, August 2017. https://www.theregister.co.uk/2017/08/21/fujitsus_australian_cloud_suffers_storage_crash_outage/.

[144] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *ACM Conference on Computer and Communications Security, CCS 2013*, pages 325–336, 2013.

[145] Youngjoo Shin, Dongyoung Koo, and Junbeom Hur. A survey of secure data deduplication schemes for cloud storage systems. *ACM Computing Surveys*, 49(4):74:1–74:38, 2017.

[146] Victor Shoup. On the security of a practical identification scheme. *Journal of Cryptology*, 12(4):247–260, 1999.

[147] Abhishek Singh, Binanda Sengupta, and Sushmita Ruj. Certificate transparency with enhancements and short proofs. In *Australasian Conference on Information Security and Privacy, ACISP 2017*, pages 381–389, 2017.

[148] N. P. Smart and F. Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. *Discrete Applied Mathematics*, 155(4):538–547, February 2007.

[149] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[150] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Annual Computer Security Applications Conference, ACSAC 2012*, pages 229–238, 2012.

[151] Jim J. Tao. Hybrid and iterative keyword and category search technique, March 2014. US Patent 8667007 B2.

[152] Jim J. Tao. Semantic context based keyword search techniques, March 2017. US Patent 9589050 B2.

[153] Dimitrios Vasilopoulos, Melek Önen, Kaoutar Elkhiyaoui, and Refik Molva. Message-locked proofs of retrievability with secure deduplication. In *ACM Cloud Computing Security Workshop, CCSW 2016*, pages 73–83, 2016.

[154] Boyang Wang, Sherman S. M. Chow, Ming Li, and Hui Li. Storing shared data on the cloud via security-mediator. In *IEEE International Conference on Distributed Computing Systems, ICDCS 2013*, pages 124–133, 2013.

[155] Boyang Wang, Baochun Li, and Hui Li. Knox: Privacy-preserving auditing for shared data with large groups in the cloud. In *International Conference on Applied Cryptography and Network Security, ACNS 2012*, pages 507–525, 2012.

[156] Boyang Wang, Baochun Li, and Hui Li. Oruta: Privacy-preserving public auditing for shared data in the cloud. In *IEEE International Conference on Cloud Computing*, pages 295–302, 2012.

[157] Boyang Wang, Baochun Li, and Hui Li. Oruta: Privacy-preserving public auditing for shared data in the cloud. *IEEE Transactions on Cloud Computing*, 2(1):43–56, 2014.

[158] Cong Wang, Sherman S. M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2):362–375, 2013.

[159] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *IEEE International Conference on Computer Communications, INFOCOM 2010*, pages 525–533, 2010.

[160] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *European Symposium on Research in Computer Security, ESORICS 2009*, pages 355–370, 2009.

[161] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):847–859, 2011.

[162] Yujue Wang, Qianhong Wu, Bo Qin, Wenchang Shi, Robert H. Deng, and Jiankun Hu. Identity-based data outsourcing with comprehensive auditing in clouds. *IEEE Transactions on Information Forensics and Security*, 12(4):940–952, 2017.

[163] Jia Xu and Ee-Chien Chang. Towards efficient proofs of retrievability. In *ACM Symposium on Information, Computer and Communications Security, ASIACCS 2012*, pages 79–80, 2012.

[164] Jia Xu, Anjia Yang, Jianying Zhou, and Duncan S. Wong. Lightweight delegatable proofs of storage. In *European Symposium on Research in Computer Security, ESORICS 2016*, pages 324–343, 2016.

[165] Ke Zeng. Publicly verifiable remote data integrity. In *International Conference on Information and Communications Security, ICICS 2008*, pages 419–434, 2008.

[166] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Mengyang Yu. Cooperative provable data possession for integrity verification in multicloud storage. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2231–2244, 2012.